1. **INTRODUCTION**

The tasks of managing and using cars well, cracking theft and robbery of motor vehicles, as well as maintaining the normal order of urban transport have become increasingly heavy.

Currently, it has become an important issue for the public security department to tom static management into dynamic change management and to tumor manual management into automation.

There are urgent needs to employ **Intelligent Transportation System (ITS)** so as to make effective management. **ITS** can perform efficient and reliable management to ambient vehicles under various circumstances.

As one of the core technologies of ITS, Vehicle Feature Recognition Technology is an important link to police enforcement system, automated highway toll collection system, Urban Traffic Surveillance System and Intelligent Parking Management System, etc.

Thus, employing image processing technology to recognize the vehicle license plate number of various kinds of vehicles is not only an important issue for information process technology, but also a research issue which is of great importance in modem transportation management.

**1.1. ABSTRACT**

**Automatic license plate recognition (ALPR)** is the extraction of vehicle license plate information from an image. The system model uses already captured images for this recognition process.

First the recognition system starts with character identification based on number plate extraction, Splitting characters and template matching. The proposed model has low complexity and less time consuming in terms of number plate segmentation and character recognition.

This can improve the system performance and make the system more efficient by taking relevant samples. at the same time compared their advantages and disadvantages, which provide the basis for license plate recognition.

Basically, for the identification of the license plate take character reorganization. And before than that localize the area of the license plate.

By using this process, we can identify the number plate of the vehicle. These are used for the identification of the vehicle when we lost our vehicle.

**1.2. COMPANY PROFILE**

PANTECH SOLUTIONS was founded in the year 2004 is an AS 9100 and ISO 9001-2000 company based in Chennai. PANTECH is derive solutions in embedded system, advanced Digital Signal Processing (DSP) and Information Technology (IT).

PANTECH specialize in image processing and speech processing algorithms development, simulation, optimization, platform porting, customization and system integration. PANTECH manages all aspects of website development. PANTECH has worked with different client fields of Art, Architecture, Publishing media healthcare and Hotel industry.

The strength of our company lies in our ability to respond quickly to customers, create new markets, rapidly develop new products, and dominate emergent technologies. The key to this process is personal commitment, the employees’ sense of identity with enterprise and its mission.

In the knowledge-creating, inventing new knowledge and strategic planning. It is a way of behaving, indeed a way of being, in which everyone is a knowledge worker-that is to say an entrepreneur. PANTECH provides its clients with technology that embraces vision and budget. In planning a company’s future, a solution provider with the most advanced technology is looked for.

**1.3. OBJECTIVE**

For the detection of license number plate detection using character reorganization.

The system model uses already captured images for this recognition process.

First the recognition system starts with character identification based on number plate extraction, Splitting characters and template matching.

Using this process, we can identify the authentication person. And we can track the vehicle also.

Using this we can find out our vehicle number plate and get our vehicle.

For that purpose, -we are using morphological filters for adding color. Take character reorganization and segmentation. Then we can get the output.

**1.4. RELATED WORK**

Security has always been a major concern for mankind. Today we have video surveillance cameras in schools, hospitals and every other public place to make us feel secured. According to a survey by HIS it is estimated that there were around 245 million security cameras installed and functioning back on days, which is like having one security camera for every 30 people on this planet. With the advancement in technology especially in Image processing and Machine Learning, it is possible to make these cameras smarter by training them to process information from the Video feed.

License plate recognition systems have been widely used in parking lots. In order to identify license plates easily, traditional license plate recognition systems used in the parking lot have a fixed light source and a shooting angle. For particularly tilting angles, such as license plate images taken with super wide-angle lenses or fisheye lenses, the deformation of the license plate can be particularly severe, resulting in poor recognition of traditional license plate recognition systems.

In this paper, we propose a three-stage license plate recognition system based on localization that can be used for various shooting angles and more oblique images. Experimental results show that the proposed architecture can identify license plates with bevel angles over 0~60 degrees and achieve map rates of up to 91%. Compared with the approach using, the proposed method with character reorganization. has made significant progress in identifying characters that are inclined above 45 degrees.

However, the application of the license plate recognition system faces a completely different situation on the intersection monitor. The license plate recognition system used in the parking lot cannot meet the requirements of intersection monitors, such as various shooting angles, skewed license plates, changes in light and shadow, etc. For example, considering the high-resolution wide-angle image, the proportion of license plates and license plate characters in the image is very small. Furthermore, for particularly tilting angles, such as license plate images taken with super wide-angle lenses or fisheye lenses, the deformation of the license plate can be particularly severe, resulting in poor recognition of traditional license plate recognition systems.

**2. SYSTEM ANALYSIS**

**2.1. EXISTING METHODOLOGY**

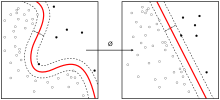
**Local Binary Pattern (LBP)** is a simple yet very efficient texture operator which labels the pixels of an image by thresholding the neighborhood of each pixel and considers the result as a binary number. Due to its discriminative power and computational simplicity, LBP texture operator has become a popular approach in various applications. It can be seen as a unifying approach to the traditionally divergent statistical and structural models of texture analysis. Perhaps the most important property of the LBP operator in real-world applications is its robustness to monotonic gray-scale changes caused, for example, by illumination variations. Another important property is its computational simplicity, which makes it possible to analyze images in challenging real-time settings.

The basic idea for developing the LBP operator was that two-dimensional surface textures can be described by two complementary measures: local spatial patterns and gray scale contrast. The original LBP operator (Ojala et al. 1996) forms labels for the image pixels by thresholding the 3 x 3 neighborhood of each pixel with the center value and considering the result as a binary number. The histogram of these 28 = 256 different labels can then be used as a texture descriptor. This operator used jointly with a simple local contrast measure provided very good performance in unsupervised texture segmentation (Ojala and Pietikäinen 1999). After this, many related approaches have been developed for texture and color texture segmentation.

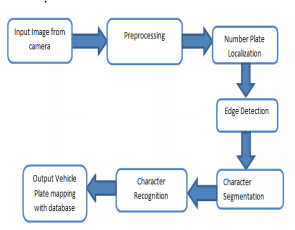
The LBP operator was extended to use neighborhoods of different sizes (Ojala et al. 2002). Using a circular neighborhood and bilinearly interpolating values at non-integer pixel coordinates allow any radius and number of pixels in the neighborhood. The gray scale variance of the local neighborhood can be used as the complementary contrast measure. In the following, the notation (P,R) will be used for pixel neighborhoods which means P sampling points on a circle of radius of R. for an example of LBP computation.

A **Support Vector Machine (SVM)** is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples. In two-dimensional space this hyperplane is a line dividing a plane in two parts where in each class lay in either side.

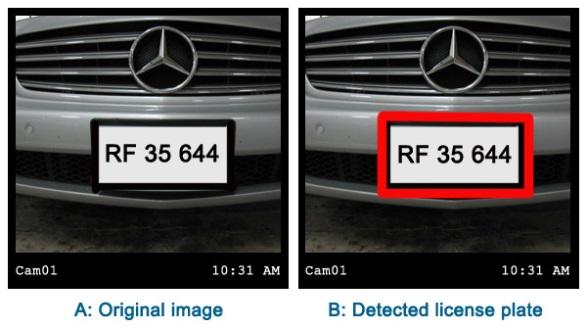
ore formally, a support-vector machine constructs a [hyperplane](https://en.wikipedia.org/wiki/Hyperplane) or set of hyperplanes in a [high-](https://en.wikipedia.org/wiki/High-dimensional_space) or infinite-dimensional space, which can be used for [classification](https://en.wikipedia.org/wiki/Statistical_classification), [regression](https://en.wikipedia.org/wiki/Regression_analysis), or other tasks like outliers detection. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class (so-called functional margin), since in general the larger the margin, the lower the [generalization error](https://en.wikipedia.org/wiki/Generalization_error) of the classifier.



**2.2. PROPOSED METHOD:**



Pre-processing is a common name for operations with images at the lowest level of abstraction -- both input and output are intensity images. The aim of pre-processing is an improvement of the image data that suppresses unwanted distortions or enhances some image features important for further processing

**2.3. EXPERIMENTAL METHOD**

**2.4. KERNEL MACHINE**

Whereas the original problem may be stated in a finite-dimensional space, it often happens that the sets to discriminate are not [linearly separable](https://en.wikipedia.org/wiki/Linear_separability) in that space. For this reason, it was proposedthat the original finite-dimensional space be mapped into a much higher-dimensional space, presumably making the separation easier in that space.

To keep the computational load reasonable, the mappings used by SVM schemes are designed to ensure that [dot products](https://en.wikipedia.org/wiki/Dot_product) of pairs of input data vectors may be computed easily in terms of the variables in the original space, by defining them in terms of a [kernel function](https://en.wikipedia.org/wiki/Positive-definite_kernel) selected to suit the problem. The hyperplanes in the higher-dimensional space are defined as the set of points whose dot product with a vector in that space is constant, where such a set of vectors is an orthogonal (and thus minimal) set of vectors that defines a hyperplane. -

The vectors defining the hyperplanes can be chosen to be linear combinations with parameters of images of [feature vectors](https://en.wikipedia.org/wiki/Feature_vector)  that occur in the data base. With this choice of a hyperplane, the points in the [feature space](https://en.wikipedia.org/wiki/Feature_space) that are mapped into the hyperplane are defined by the relation.

Note that if becomes small as grows further away from, each term in the sum measures the degree of closeness of the test point to the corresponding data base point. In this way, the sum of kernels above can be used to measure the relative nearness of each test point to the data points originating in one or the other of the sets to be discriminated. Note the fact that the set of points mapped into any hyper plane can be quite convoluted as a result, allowing much more complex discrimination between sets that are not convex at all in the original space.

**3. SYSTEM REQUIRMENT**

**3.1 HARDWARE REQUIREMENT:**

* 1 GM RAM
* 80 GB Hard Disk
* Above 2GHz Processor
* Windows OS

**3.2 SOFTWARE REQUIREMENT:**

* Windows 7 OS
* Python 3.8 (64- BIT)
* IDLE (Python 3.8 64-bit)

**4. THE AUTOMATIC LICENSE NUMBER PLATE RECOGNITION SYSTEM**

Automatic License Number Plate Recognition Systems are available in all shapes and sizes:

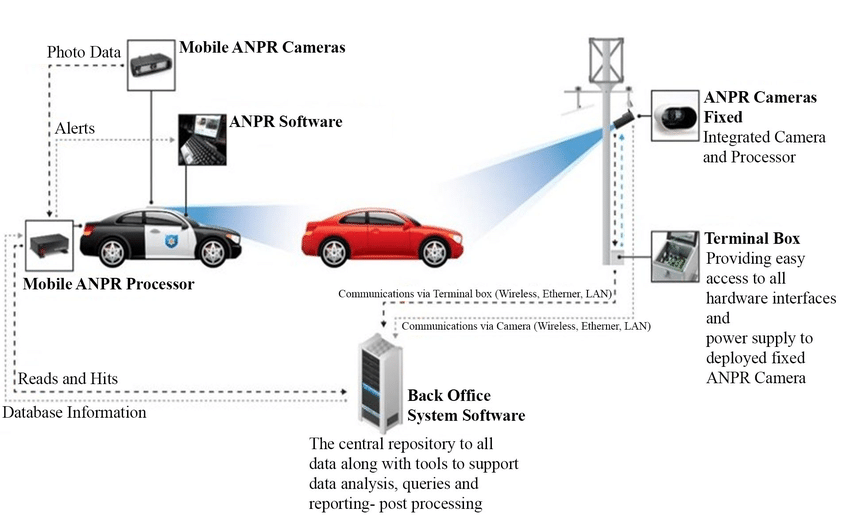
1. ANPR executed in measured lighting situations with predictable number plate types can utilize basic techniques for image processing.
2. More advanced ANPR systems use dedicated object detectors, like HOG + Linear SVM, SSDs, YOLO, and Faster R-CNN to localize license number plates in images.
3. State-of-the-art ANPR software uses Recurrent Neural Networks (RNNs) and Long Short-Term Memory networks (LSTMs) in order to aid in better OCRing of the text from the number plates themselves.
4. Even more advanced ANPR systems utilize specialized neural network architectures in order to pre-process and clean images before they are OCRed, thereby developing the accuracy of ANPR.

The fact that makes Automatic License Number Plate Recognition more complicated may require operating in real-time.

For instance, let us consider an ANPR system that is mounted on a toll road. It has to be able to detect the number plate of each vehicle passing by OCR the characters on the plate, and then store this data in a database so the vehicle's owner can be billed for the toll.

Few compounding factors make ANPR extremely challenging, involving finding a set of data we can utilize in order to train a custom model for ANPR. Large, robust datasets of ANPR that are utilized to train state-of-the-art models are tightly guarded and hardly (if ever) released publicly:

1. These datasets consist of sensitive identifying details associated with the vehicle, driver, and location.
2. The datasets of ANPR are tedious to curate, needing an unbelievable time investment and staff hours to interpret.
3. The contracts of ANPR with local and federal governments tend to be extremely reasonable. It is often not the trained model that is valuable; however, instead of the dataset that a specified company has curated.
4. For the same cause, we will observe ANPR industries acquired not for their ANPR system but for the data itself.



**5. DIGITAL IMAGE PROCESSING**

The identification of objects in an image and this process would probably start with image processing techniques such as noise removal, followed by (low-level) feature extraction to locate lines, regions and possibly areas with certain textures.

The clever bit is to interpret collections of these shapes as single objects, e.g. cars on a road, boxes on a conveyor belt or cancerous cells on a microscope slide. One reason this is an AI problem is that an object can appear very different when viewed from different angles or under different lighting. Another problem is deciding what features belong to what object and which are background or shadows etc.

The human visual system performs these tasks mostly unconsciously, but a computer requires skillful programming and lots of processing power to approach human performance. Manipulation of data in the form of an image through several possible techniques.

An image is usually interpreted as a two-dimensional array of brightness values and is most familiarly represented by such patterns as those of a photographic print, slide, television screen, or movie screen. An image can be processed optically or digitally with a computer.

**6. BASICS OF IMAGE PROCESSING**

**FUNDAMENTALS OF DIGITAL IMAGE:**

**IMAGE:** An image is a two-dimensional picture, which has a similar appearance to some subject usually a physical object or a person.

Image is a two-dimensional, such as a photograph, screen display, and as well as a three-dimensional, such as a statue. They may be captured by optical devices—such as cameras, mirrors, lenses, telescopes, microscopes, etc. and natural objects and phenomena, such as the human eye or water surfaces.

The word image is also used in the broader sense of any two-dimensional figure such as a map, a graph, a pie chart, or an abstract painting. In this wider sense, images can also be rendered manually, such as by drawing, painting, carving, rendered automatically by printing or computer graphics technology, or developed by a combination of methods, especially in a pseudo-photograph.



*Fig: Color image to Gray scale Conversion Process*

An image is a rectangular grid of pixels. It has a definite height and a definite width counted in pixels. Each pixel is square and has a fixed size on a given display. However different computer monitors may use different sized pixels. The pixels that constitute an image are ordered as a grid (columns and rows); each pixel consists of numbers representing magnitudes of brightness and color.

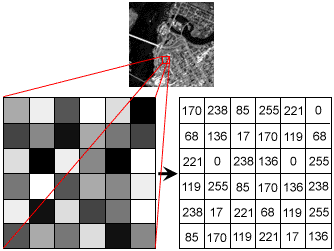
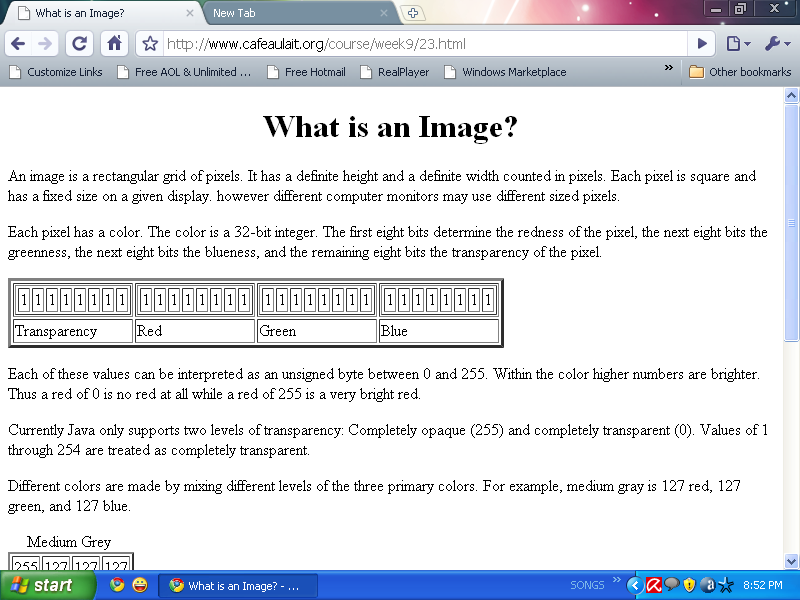


Fig: Gray Scale Image Pixel Value Analysis

Each pixel has a color. The color is a 32-bit integer. The first eight bits determine the redness of the pixel, the next eight bits the greenness, the next eight bits the blueness, and the remaining eight bits the transparency of the pixel.

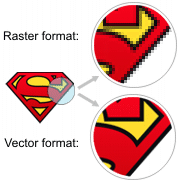


*Fig: BIT Transferred for Red, Green and Blue plane (24bit=8bit red;8-bit green;8bit blue)*

**IMAGE FILE SIZES:**

Image file size is expressed as the number of bytes that increases with the number of pixels composing an image, and the color depth of the pixels. The greater the number of rows and columns, the greater the image resolution, and the larger the file. Also, each pixel of an image increases in size when its color depth increases, an 8-bit pixel (1 byte) stores 256 colors, a 24-bit pixel (3 bytes) stores 16 million colors, the latter known as true color. Image compression uses algorithms to decrease the size of a file. High resolution cameras produce large image files, ranging from hundreds of kilobytes to megabytes, per the camera's resolution and the image-storage format capacity. High resolution digital cameras record 12-megapixel (1MP = 1,000,000 pixels / 1 million) images, or more, in true color. For example, an image recorded by a 12 MP camera; since each pixel uses 3 bytes to record true color, the uncompressed image would occupy 36,000,000 bytes of memory, a great amount of digital storage for one image, given that cameras must record and store many images to be practical. Faced with large file sizes, both within the camera and a storage disc, image file formats were developed to store such large images.

**IMAGE FILE FORMATS:**

Image file formats are standardized means of organizing and storing images. This entry is about digital image formats used to store photographic and other images. Image files are composed of either pixel or vector (geometric) data that are rasterized to pixels when displayed (with few exceptions) in a vector graphic display. Including proprietary types, there are hundreds of image file types. The PNG, JPEG, and GIF formats are most often used to display images on the Internet.

*Fig: Horizontal and Vertical Process*

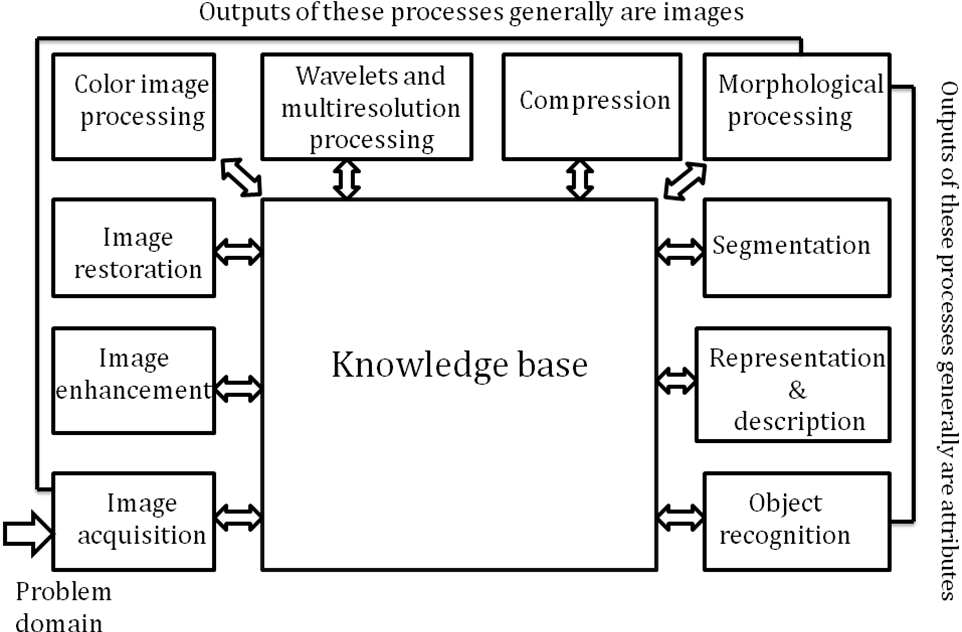
In addition to straight image formats, Metafile formats are portable formats which can include both raster and vector information. The metafile format is an intermediate format. Most Windows applications open metafiles and then save them in their own native format.

**IMAGE PROCESSING:**

Digital image processing, the manipulation of images by computer, is relatively recent development in terms of man’s ancient fascination with visual stimuli. In its short history, it has been applied to practically every type of images with varying degree of success. The inherent subjective appeal of pictorial displays attracts perhaps a disproportionate amount of attention from the scientists and also from the layman. Digital image processing like other glamour fields, suffers from myths, mis-connect ions, mis-understandings and mis-information. It is vast umbrella under which fall diverse aspect of optics, electronics, mathematics, photography graphics and computer technology. It is truly multidisciplinary endeavor ploughed with imprecise jargon.

Several factors combine to indicate a lively future for digital image processing. A major factor is the declining cost of computer equipment. Several new technological trends promise to further promote digital image processing. These include parallel processing mode practical by low-cost microprocessors, and the use of Charge Coupled Devices (CCDs) for digitizing, storage during processing and display and large low cost of image storage arrays.

**6.1 FUNDAMENTAL STEPS IN DIGITAL IMAGE PROCESSING**



*Fig: Basics steps of image Processing*

**IMAGE ACQUISITION:**

Image Acquisition is to acquire a digital image. To do so requires an image sensor and the capability to digitize the signal produced by the sensor. The sensor could be monochrome or color TV camera that produces an entire image of the problem domain every 1/30 sec. the image sensor could also be line scan camera that produces a single image line at a time. In this case, the objects motion past the line.



*Fig: Digital camera*

Scanner produces a two-dimensional image. If the output of the camera or other imaging sensor is not in digital form, an analog to digital converter digitizes it. The nature of the sensor and the image it produces are determined by the application.

*Fig: Digital Scanner*

**IMAGE ENHANCEMENT:**

Image enhancement is among the simplest and most appealing areas of digital image processing. Basically, the idea behind enhancement techniques is to bring out detail that is obscured, or simply to highlight certain features of interesting an image. A familiar example of enhancement is when we increase the contrast of an image because “it looks better.” It is important to keep in mind that enhancement is a very subjective area of image processing.



*Fig: Image enhancement process for Gray Scale Image and Color Image using Histogram Bits*

**IMAGE RESTORATION:**

Image restoration is an area that also deals with improving the appearance of an image. However, unlike enhancement, which is subjective, image restoration is objective, in the sense that restoration techniques tend to be based on mathematical or probabilistic models of image degradation.



*Fig: Noise image- Image Enhancement*

Enhancement, on the other hand, is based on human subjective preferences regarding what constitutes a “good” enhancement result. For example, contrast stretching is considered an enhancement technique because it is based primarily on the pleasing aspects it might present to the viewer, whereas removal of image blur by applying a deblurring function is considered a restoration technique.

**COLOR IMAGE PROCESSING:**

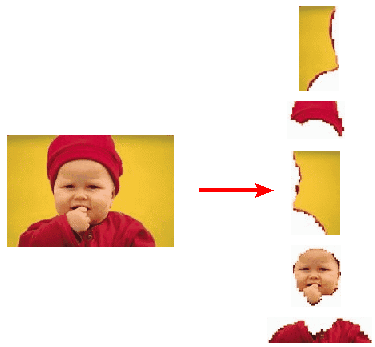
The use of color in image processing is motivated by two principal factors. First, color is a powerful descriptor that often simplifies object identification and extraction from a scene. Second, humans can discern thousands of color shades and intensities, compared to about only two dozen shades of gray. This second factor is particularly important in manual image analysis.



*Fig: gray Scale image -Color Image*

**SEGMENTATION:**

Segmentation procedures partition an image into its constituent parts or objects. In general, autonomous segmentation is one of the most difficult tasks in digital image processing. A rugged segmentation procedure brings the process a long way toward successful solution of imaging problems that require objects to be identified individually.



*Fig: Image Segment Process*

On the other hand, weak or erratic segmentation algorithms almost always guarantee eventual failure. In general, the more accurate the segmentation, the more likely recognition is to succeed.

Digital image is defined as a two dimensional function f(x, y), where x and y are spatial (plane) coordinates, and the amplitude of ‘f’ at any pair of coordinates (x, y) is called intensity or grey level of the image at that point. The field of digital image processing refers to processing digital images by means of a digital computer. The digital image is composed of a finite number of elements, each of which has a particular location and value. The elements are referred to as picture elements, image elements, pels, and pixels. Pixel is the term most widely used.

**IMAGE COMPRESSION:**

Digital Image compression addresses the problem of reducing the amount of data required to represent a digital image. The underlying basis of the reduction process is removal of redundant data. From the mathematical viewpoint, this amounts to transforming a 2D pixel array into a statically uncorrelated data set. The data redundancy is not an abstract concept but a mathematically quantifiable entity. If n1 and n2 denote the number of information-carrying units in two data sets that represent the same information, the relative data redundancy  [2] of the first data set (the one characterized by n1) can be defined as,



Where  called as compression ratio [2]. It is defined as,

= 

In image compression, three basic data redundancies can be identified and exploited: Coding redundancy, interpixel redundancy, and psychovisual redundancy. Image compression is achieved when one or more of these redundancies are reduced or eliminated. The image compression is mainly used for image transmission and storage. Image transmission applications are in broadcast television; remote sensing via satellite, air-craft, radar, or sonar; teleconferencing; computer communications; and facsimile transmission. Image storage is required most commonly for educational and business documents, medical images that arise in computer tomography (CT), magnetic resonance imaging (MRI) and digital radiology, motion pictures, satellite images, weather maps, geological surveys, and so on.

**IMAGE COMPRESSION MODEL:**

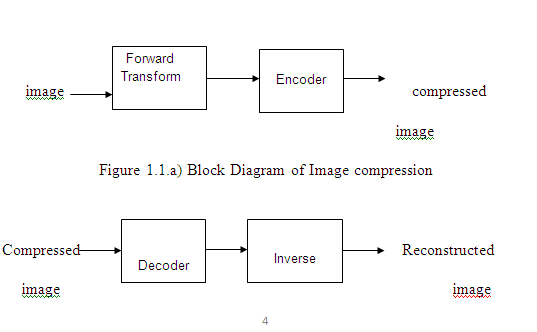


Figure 1.1.b) Decompression Process for Image

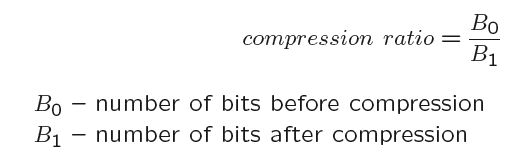
**IMAGE COMPRESSION TYPES:**

There are two types’ image compression techniques.

1. Lossy Image compression

2. Lossless Image compression

**COMPRESSION RATIO:**



**1. LOSSY IMAGE COMPRESSION:**

Lossy compression provides higher levels of data reduction but result in a less than perfect reproduction of the original image. It provides high compression ratio. lossy image compression is useful in applications such as broadcast television, videoconferencing, and facsimile transmission, in which a certain amount of error is an acceptable trade-off for increased compression performance. Originally, PGF has been designed to quickly and progressively decode lossy compressed aerial images. A lossy compression mode has been preferred, because in an application like a terrain explorer texture data (e.g., aerial orthophotos) is usually mid-mapped filtered and therefore lossy mapped onto the terrain surface. In addition, decoding lossy compressed images is usually faster than decoding lossless compressed images.

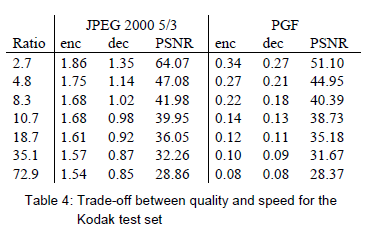
In the next test series, we evaluate the lossy compression efficiency of PGF. One of the best competitors in this area is for sure JPEG 2000. Since JPEG 2000 has two different filters, we used the one with the better trade-off between compression efficiency and runtime. On our machine the 5/3 filter set has a better trade-off than the other. However, JPEG 2000 has in both cases a remarkable good compression efficiency for very high compression ratios but also a very poor encoding and decoding speed. The other competitor is JPEG. JPEG is one of the most popular image file formats.



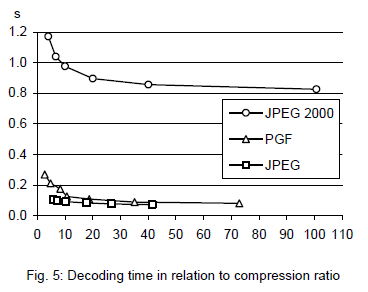
It is very fast and has a reasonably good compression efficiency for a wide range of compression ratios. The drawbacks of JPEG are the missing lossless compression and the often-missing progressive decoding. Fig. 4 depicts the average rate-distortion behavior for the images in the Kodak test set when fixed (i.e., nonprogressive) lossy compression is used. The PSNR of PGF is on average 3% smaller than the PSNR of JPEG 2000, but 3% better than JPEG.

These results are also qualitative valid for our PGF test set and they are characteristic for aerial ortho-photos and natural images. Because of the design of PGF we already know that PGF does not reach the compression efficiency of JPEG 2000. However, we are interested in the trade-off between compression efficiency and runtime. To report this trade-off, we show in Table 4 a comparison between JPEG 2000 and PGF and in Fig. 5, we show for the same test series as in Fig. 4 the corresponding average decoding times in relation to compression ratios. Table 4 contains for seven different compression ratios (mean values over the compression ratios of the eight images of the Kodak test set) the corresponding average encoding and decoding times in relation to the average PSNR values. In case of PGF the encoding time is always slightly longer than the corresponding decoding time.

The reason for that is that the actual encoding phase (cf. Subsection 2.4.2) takes slightly longer than the corresponding decoding phase. For six of seven ratios the PSNR difference between JPEG 2000 and PGF is within 3% of the PSNR of JPEG 2000. Only in the first row is the difference larger (21%), but because a PSNR of 50 corresponds to an almost perfect image quality the large PSNR difference corresponds with an almost undiscoverable visual difference. The price they pay in JPEG 2000 for the 3% more PSNR is very high. The creation of a PGF is five to twenty times faster than the creation of a corresponding JPEG 2000 file, and the decoding of the created PGF is still five to ten times faster than the decoding of the JPEG 2000 file. This gain in speed is remarkable, especially in areas where time is more important than quality, maybe for instance in real-time computation.



In Fig. 5 we see that the price we pay in PGF for the 3% more PSNR than JPEG is low: for small compression ratios (< 9) decoding in PGF takes two times longer than JPEG and for higher compression ratios (> 30) it takes only ten percent longer than JPEG. These test results are characteristic for both natural images and aerial ortho-photos. Again, in the third test series we only use the ‘Lena’ image. We run our lossy coder with six different quantization parameters and measure the PSNR in relation to the resulting compression ratios. The results (ratio: PSNR) are:



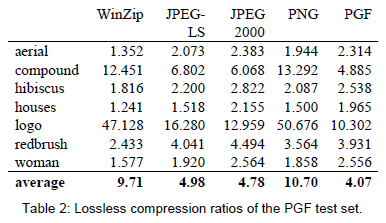
**2. LOSSLESS IMAGE COMPRESSION:**

Lossless Image compression is the only acceptable amount of data reduction. It provides low compression ratio while compared to lossy. In Lossless Image compression techniques are composed of two relatively independent operations:

(1) devising an alternative representation of the image in which its interpixel redundancies are reduced and

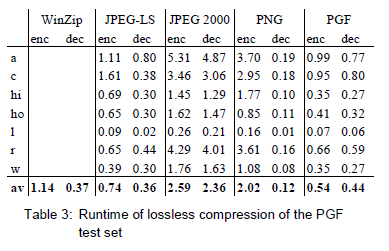
(2) coding the representation to eliminate coding redundancies.

Lossless Image compression is useful in applications such as medical imaginary, business documents and satellite images. Table2 summarizes the lossless compression efficiency and Table 3 the coding times of the PGF test set. For WinZip we only provide average runtime values, because of missing source code we have to use an interactive testing procedure with runtimes measured by hand. All other values are measured in batch mode.



In Table 2 it can be seen that in almost all cases the best compression ratio is obtained by JPEG 2000, followed by PGF, JPEG-LS, and PNG. This result is different to the result in [SEA+00], where the best performance for a similar test set has been reported for JPEG-LS. PGF performs between 0.5% (woman) and 21.3% (logo) worse than JPEG 2000. On average it is almost 15% worse. The two exceptions to the general trend are the ‘compound’ and the ‘logo’ images. Both images contain for the most part black text on a white background. For this type of images, JPEG-LS and in particular WinZip and PNG provide much larger compression ratios. However, in average PNG performs the best, which is also reported in [SEA+00].

These results show, that as far as lossless compression is concerned, PGF performs reasonably well on natural and aerial images. In specific types of images such as ‘compound’ and ‘logo’ PGF is outperformed by far in PNG.

 Table 3 shows the encoding (enc) and decoding (dec) times (measured in seconds) for the same algorithms and images as in Table 2. JPEG 2000 and PGF are both symmetric algorithms, while WinZip, JPEG-LS and in particular PNG are asymmetric with a clearly shorter decoding than encoding time. JPEG 2000, the slowest in encoding and decoding, takes more than four times longer than PGF. This speed gain is due to the simpler coding phase of PGF. JPEG-LS is slightly slower than PGF during encoding, but slightly faster in decoding images.

WinZip and PNG decode even more faster than JPEG-LS, but their encoding times are also worse. PGF seems to be the best compromise between encoding and decoding times.

Our PGF test set clearly shows that PGF in lossless mode is best suited for natural images and aerial ortho photos. PGF is the only algorithm that encodes the three Mega Byte large aerial ortho photo in less than second without a real loss of compression efficiency. For this particular image the efficiency loss is less than three percent compared to the best. These results should be underlined with our second test set, the Kodak test set.

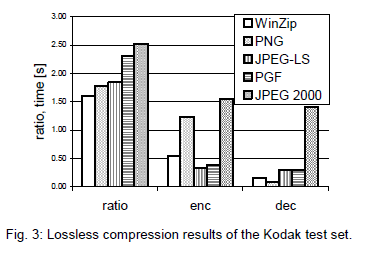


Fig. 3 shows the averages of the compression ratios (ratio), encoding (enc), and decoding (dec) times over all eight images. JPEG 2000 shows in this test set the best compression efficiency followed by PGF, JPEG-LS, PNG, and WinZip. In average PGF is eight percent worse than JPEG 2000. The fact that JPEG 2000 has a better lossless compression ratio than PGF does not surprise, because JPEG 2000 is more quality driven than PGF. However, it is remarkable that PGF is clearly better than JPEG-LS (+21%) and PNG (+23%) for natural images. JPEG-LS shows in the Kodak test set also a symmetric encoding and decoding time behavior. It is encoding and decoding times are almost equal to PGF. Only PNG and WinZip can faster decode than PGF, but they also take longer than PGF to encode.

If both compression efficiency and runtime is important, then PGF is clearly the best of the tested algorithms for lossless compression of natural images and aerial ortho photos. In the third test we perform our lossless coder on the ‘Lena’ image.

To digitally process an image, it is first necessary to reduce the image to a series of numbers that can be manipulated by the computer. Each number representing the brightness value of the image at a particular location is called a picture element, or pixel. A typical digitized image may have 512 × 512 or roughly 250,000 pixels, although much larger images are becoming common.

Once the image has been digitized, there are three basic operations that can be performed on it in the computer. For a point operation, a pixel value in the output image depends on a single pixel value in the input image. For local operations, several neighboring pixels in the input image determine the value of an output image pixel. In a global operation, all of the input image pixels contribute to an output image pixel value.

Correspondingly, these combinations attempt to strike a winning tradeoff: be flexible and hence bring tolerance toward intraclass variation, while also being discriminative enough to be robust to background clutter and interclass similarity. An important feature of our contour-based recognition approach is that it affords us substantial flexibility to incorporate additional image information.

Specifically, we extend the contour-based recognition method and propose a new hybrid recognition method which exploits shape tokens and SIFT features as recognition cues. Shape-tokens and SIFT features are largely orthogonal, where the former corresponds to shape boundaries and the latter to sparse salient image patches. Here, each learned combination can comprise features that are either 1) purely shape-tokens, 2) purely SIFT features, or 3) a mixture of shape-tokens and SIFT features. The number and types of features to be combined together are learned automatically from training images, and represent the more discriminative ones based on the training set. Consequently, by imparting these two degrees of variability (in both the number and the types of features) to a combination, we empower it with even greater flexibility and discriminative potential.

**7. CLASSIFICATION OF IMAGES**

There are 3 types of images used in Digital Image Processing. They are

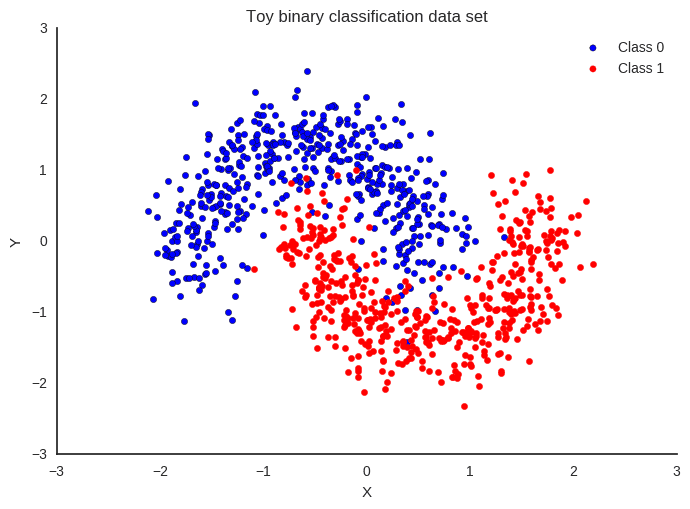
1. **Binary Image**
2. **Gray Scale Image**
3. **Color Image**

**1. BINARY IMAGE:**

A binary image is a [digital image](http://en.wikipedia.org/wiki/Digital_image) that has only two possible values for each [pixel](http://en.wikipedia.org/wiki/Pixel).  Typically, the two colors used for a binary image are black and white though any two colors can be used.  The color used for the object(s) in the image is the foreground color while the rest of the image is the background color.

Binary images are also called bi-level or two-level. This means that each pixel is stored as a single bit (0 or 1). This name black and white, monochrome or monochromatic are often used for this concept, but may also designate any images that have only one sample per pixel, such as [grayscale images](http://en.wikipedia.org/wiki/Grayscale).

Binary images often arise in [digital image processing](http://en.wikipedia.org/wiki/Digital_image_processing) as [masks](http://en.wikipedia.org/w/index.php?title=Mask_(image_processing)&action=edit&redlink=1) or as the result of certain operations such as [segmentation](http://en.wikipedia.org/wiki/Segmentation_(image_processing)), [thresholding](http://en.wikipedia.org/wiki/Thresholding_(image_processing)), and [dithering](http://en.wikipedia.org/wiki/Dither). Some input/output devices, such as [laser printers](http://en.wikipedia.org/wiki/Laser_printer), [fax machines](http://en.wikipedia.org/wiki/Fax), and bi-level [computer displays](http://en.wikipedia.org/wiki/Visual_display_unit), can only handle bi-level images.

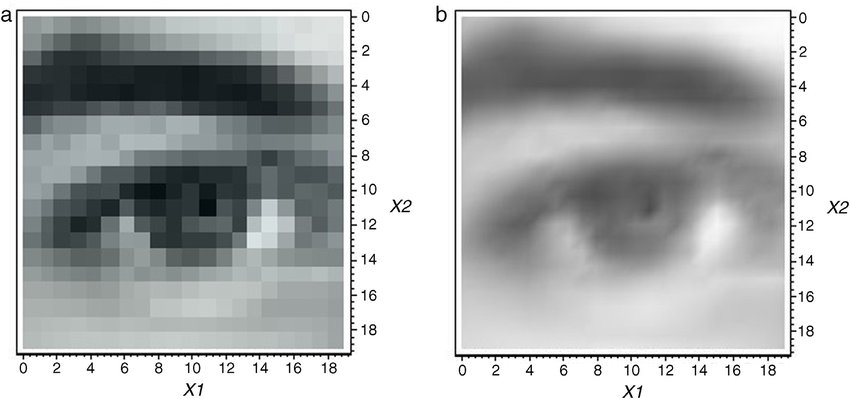


**2. GRAY SCALE IMAGE:**

A Grayscale Image is [digital image](http://en.wikipedia.org/wiki/Digital_image) is an image in which the value of each [pixel](http://en.wikipedia.org/wiki/Pixel) is a single [sample](http://en.wikipedia.org/wiki/Sample_(signal)), that is, it carries only [intensity](http://en.wikipedia.org/wiki/Luminous_intensity) information. Images of this sort, also known as [black-and-white](http://en.wikipedia.org/wiki/Black-and-white), are composed exclusively of shades of [gray](http://en.wikipedia.org/wiki/Gray)(0-255), varying from black(0) at the weakest intensity to white(255) at the strongest.

Grayscale images are distinct from one-bit [black-and-white](http://en.wikipedia.org/wiki/Black-and-white) images, which in the context of computer imaging are images with only the two [colors](http://en.wikipedia.org/wiki/Color), [black](http://en.wikipedia.org/wiki/Black), and [white](http://en.wikipedia.org/wiki/White) (also called bi-level or [binary images](http://en.wikipedia.org/wiki/Binary_image)). Grayscale images have many shades of gray in between. Grayscale images are also called [monochromatic](http://en.wikipedia.org/wiki/Monochromatic), denoting the absence of any [chromatic](http://en.wikipedia.org/wiki/Chromaticity) variation.

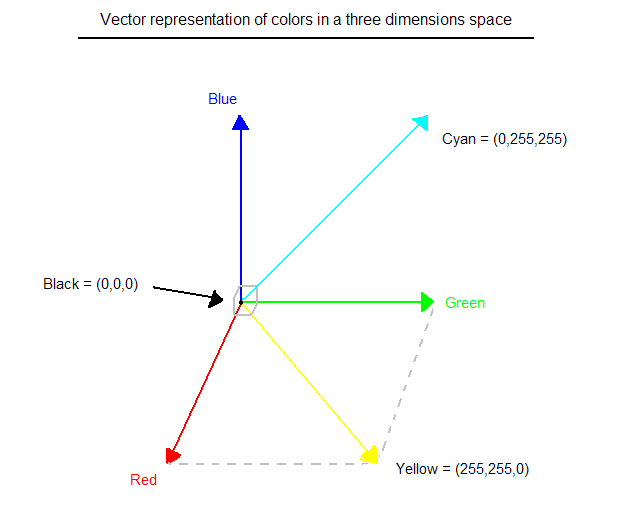
Grayscale images are often the result of measuring the intensity of light at each pixel in a single band of the [electromagnetic spectrum](http://en.wikipedia.org/wiki/Electromagnetic_spectrum) (e.g. [infrared](http://en.wikipedia.org/wiki/Infrared), [visible light](http://en.wikipedia.org/wiki/Visible_spectrum), [ultraviolet](http://en.wikipedia.org/wiki/Ultraviolet), etc.), and in such cases they are monochromatic proper when only a given [frequency](http://en.wikipedia.org/wiki/Frequency) is captured. But also, they can be synthesized from a full color image; see the section about converting to grayscale.



**3. COLOUR IMAGE:**

A (digital) color image is a [digital image](http://en.wikipedia.org/wiki/Digital_image) that includes [color](http://en.wikipedia.org/wiki/Color) information for each [pixel](http://en.wikipedia.org/wiki/Pixel). Each pixel has a particular value which determines its appearing color. This value is qualified by three numbers giving the decomposition of the color in the three primary colors Red, Green and Blue. Any color visible to human eye can be represented this way. The decomposition of a color in the three primary colors is quantified by a number between 0 and 255. For example, white will be coded as R = 255, G = 255, B = 255; black will be known as (R,G,B) = (0,0,0); and say, bright pink will be : (255,0,255).

In other words, an image is an enormous two-dimensional array of color values, pixels, each of them coded on 3 bytes, representing the three primary colors. This allows the image to contain a total of 256x256x256 = 16.8 million different colors. This technique is also known as RGB encoding, and is specifically adapted to human vision



*Fig.1 Hue Saturation Process of RGB SCALE Image*

From the above figure, colors are coded on three bytes representing their decomposition on the three primary colors. It sounds obvious to a mathematician to immediately interpret colors as vectors in a three-dimension space where each axis stands for one of the primary colors. Therefore, we will benefit of most of the geometric mathematical concepts to deal with our colors, such as norms, scalar product, projection, rotation or distance.

**8. CONVERSIONS**

**8.1 COLOR TO GRAY CONVERSION:**

Color Image to Grayscale Image Conversion, Image processing is a vital research area and the utilization of images increases in various applications. On different research areas, scientists are working on such as image compression, image restoration, image segmentation etc. to enhance the existing image processing techniques and invent new method of solving image processing problems. The latest image processing applications such as medical image processing, satellite image processing, and molecular image processing uses various image processing techniques.

Conversion of color image to grayscale image is one of the image processing applications used in different fields effectively. In publication organizations’ printing, a color image is expensive compared to a grayscale image. Thus, color images have converted to grayscale image to reduce the printing cost for low priced edition books. Similarly, color deficient viewer requires good quality of grayscale image to perceive the information, as the normal people perceive the color picture. Likewise, various image processing applications require conversion of color image to grayscale image for different purpose.

Conversion of a color image to a grayscale image requires more knowledge about the color image. A pixel color in an image is a combination of three colors Red, Green, and Blue (RGB). The RGB color values are represented in three dimensions XYZ, illustrated by the attributes of lightness, chroma, and hue.

Quality of a color image depends on the color represented by the number of bits the digital device could support. The basic color image represented by 8 bits, the high color image represented using 16 bits, the true color image represented by 24 bits, and the deep color image is represented by 32 bits. The number of bits decides the maximum number of different colors supported by the digital device.

If each Red, Green, and Blue occupies 8 bits then the combination of RGB occupies 24 bit and supports 16,777,216 different colors. The 24 bit represents the color of a pixel in the color image. The grayscale image has represented by luminance using 8 bits value. The luminance of a pixel value of a grayscale image ranges from 0 to 255.

The conversion of a color image into a grayscale image is converting the RGB values (24 bits) into grayscale value (8 bits). Various image processing techniques and software applications converts color image to grayscale image. However, the image processing techniques or applications are unable to handle the disparity in the chromaticity and the luminance.

In the literature, several linear and non-linear techniques had discussed for converting color image to grayscale image. The recent techniques handle these disparities much better than the earlier techniques. Nevertheless, the techniques involve several computational procedures such as conversion of RGB space to XYZ space then approximations then mapping or other related techniques. Grayscale mappings of color images that are constructed by approximating spectral uniformity are often inadequate.

The recent technique used to convert from color image to gray image highly consumes computational time and memory. Thus, a new algorithm proposed to convert color image to grayscale image in a minimum amount of time. There are several issues related to conversion of color image to grayscale image and different solutions to address these issues have addressed in the literature. The software such as Adobe Photoshop devised custom non-linear projections and required users to set image dependent parameters by trial and error.

The following writings discuss recent six prime research works focusing on the conversion of color image to grayscale image. A technique proposed has utilized the L\*a\*b luminance chrominance representation. The proposed technique introduces an additive correction term for spatial chrominance variations. The first step of this algorithm computes high pass filtered versions of all three channels, and the high-pass content from the two chrominance channels combined into a single signal that represents high frequency chrominance information.

The experiment shows that the is luminant colors handled perfectly. The cost of setting up and solving the optimization problem is proportional to the size of the image. The proposed technique is highly resource (time and memory) consumable. In addition, the technique has not provided large improvements for scenes with high dynamic luminance range like natural scenes.

A technique proposed for re-coloring of images for color-deficient viewers without introducing visual artifacts. The mapping of color to grayscale preserves contrasts and maintains luminance consistency. The quadratic objective function has defined for contrast preservation. Further, constraints added to enforce luminance consistency within narrow chrominance bands.

The technique performed well for certain images and as standard for other images. Another technique proposed enhances the contrast and converts color to grayscale. The proposed technique used Gaussian pairing technique for image sampling, dimensionality reduction, and sampling color differences. The predominant component analysis used for analyzing color differences. The technique has satisfied Continuous mapping, Global consistency, Grayscale preservation, Luminance ordering, Saturation ordering, and Hue ordering.

The process controlled by three parameters: the degree of image enhancement; the typical size of relevant image features in pixels; and the proportion of image pixels assumed to be outliers. First, the algorithm converts the RGB values into YPQ color space. Further, to analyze the distribution of color contrasts between image features, color differences between pixels considered using Gaussian pairing. Dimensionality reduction by predominant component analysis performed to find the color axis that best represents the chromatic contrasts lost when the luminance channel supplies the color to grayscale mapping. Next, has combined luminance and chrominance information. The final step used saturation to calibrate luminance while adjusting its dynamic range and compensating for image noise. The decolorize algorithm is effective at enhancing contrast.

The algorithm avoids the noise, contouring, and halo artifacts. However, tuning on parameters required individually to suit each image. A recent technique demonstrated color to grayscale conversion based on the experimental background of the Coloroid system observations. A survey of the coloroid system to and from CIE XYZ system formulas completed. Observations based on the Coloroid system discussed. The seven basic Coloroid hues fixed. Relative gray-equivalent differences of the basic hue pairs calculated. Proposed two formulas based on the CIE Lab color space and the Coloroid color space for building the gradient field. Further, the inconsistency of gradient field corrected.

Finally, 2D integration applied to get the grayscale image. From the demonstration noted that the iso-luminant colors and bluish colors transformed to grayscale more realistic. The technique preserves overall appearance of the color image. A most recent work converted the color image and video to grayscale. The proposed technique converted the image and video perceptually accurate. First, H-K (Helmholtz-Kohlrausch) phenomenon predicted by a chromatic lightness term that corrects perceived lightness based on the color’s chromatic component.

The color image converted to linear RGB by inverse gamma mapping, then transformed to CIEL-UV color space. Its apparent chromatic object lightness channel calculated. Lightness channel to grayscale values mapped using reference white chromatic values. Gamma mapping applied to move from a linear space to a gamma-corrected space. Local contrast increased in the grayscale image to represent better the local contrast of original image. The work carried out using CIELab and CIEL-UV color spaces.

This two-step approaches a good compromise between a fully automatic technique (first step) and user control (second step) making this approach well suited for natural images, photographs, artistic reproductions as well as business graphics. The main limitation of the approach is the locality of the second step. It cannot restore chromatic contrast between non-adjacent regions. END The steps 1 to 3 calculate the luminance and chrominance values of the source color image. In the step 4 sum of chrominance value calculated. Steps 5 to 16 the RGB values are approximated using RGB components. Step 17 calculates the average of the four values R4, G4, B4 and UV. The I1 represents the resulted gray color image. The color image, 3D plot of the RGB values of the color image, and 3D plot of the reduced RGB values of the color image have shown and compared.

Comparison of source image RGB and reduced RGB values. Source Color Image 3D plot of the source color image RGB values 3D plot of the reduced RGB values The 3D plot of the source image and reduced RGB values compared. The RGB values of the source image ranging from 0 to 255 were reduced to a range of 0 to 85. The reduction enhances the color range and helps to calculate the grayscale in a better way. In the above picture observed that the color ranges are in the 3D plot of the source image RGB values are enhanced many colors are highly visible in the 3D plot of the approximated and reduced RGB values without any major changes in the colors and structure. The reduction process of RGB values retained the major values of the RGB at most of the points observed. The reductions made at the RGB color level so that the resultant grayscale image to retain the luminance and chrominance property at the maximum.

The proposed algorithm tested on thirty-four number of different eight-bit color images published in the recent research publications. The color values of the color images are in the range of 0 to 255 for each Red, Green, and Blue. In the images, seven are jpg in format and twenty-seven are png in type. The results of the experiment carefully examined. The difference between the results of the proposed technique and recent techniques identified and discussed. The result revealed that the proposed algorithm yields grayscale image with better luminance and chrominance property for most of the cases and as standard for other few cases. However, there is a minimum amount of loss in the grayscale image due to reduction the algorithm preserved contrasts, sharpness, shadow, and structure of the color image in the reproduced grayscale image.

Experiment Results Color Image Grayscale Image using luminance components In the first grayscale image the letters A, B, and C are sharp as in the color image whereas it is not so in other earlier techniques. In the second image, the shades of the different color reproduced as sharp as in the color image in the grayscale image than the recent algorithms. The conversion of color image to grayscale image using the proposed algorithm uses approximation of RGB values using luminance RGB components approximated RGB reduced by three, added with chrominance value and average of these four value results perceptually and structurally good quality of grayscale images.

First, the luminance and chrominance values are calculated. Further, the RGB values of the source color image reduced. Finally, the reduced RGB values have added chrominance values. The resulted grayscale images confirm that the luminance and chrominance properties and structure of the color images 198 retained well in the grayscale image. The results confirm that the iso-luminant images have handled as handled by other recent techniques. The proposed alg-orithm is helpful for different applications where good quality of grayscale image is highly required. The proposed algorithm results best quality of grayscale images using RGB reduction and chrominance addition in a short amount of time.

**8.2 GRAY TO BLACK AND WHITE:**

A binary image is the type of image where each pixel is black or white. You can also say the pixels as 0 or 1 value. Here 0 represents black and 1 represents a white pixel.

I often find peoples calling grayscale image as black and white. Now let’s understand what is grayscale and what is a black and white or binary image with the example of my own picture.

Many peoples think the grayscale image as black and white

Many of us will think the below image as black and white:



But this is a wrong concept. Real binary image or pure black and white image look like you can see below:



Each of the pixels of the above binary image is either black or white. Or we cay each pixel value is 0 or 1.

So you have seen the same image as grayscale and binary and got the idea of a binary image.

**8.3 CONVERT RGB TO BINARY:**

Now I am going to show you how you can convert RGB to Binary Image or convert a colored image to black and white.

Here we are just going to write a few lines of Python code and it will convert our RGB image into a binary image.

To convert an RGB image into a binary type image, we need OpenCV. So first of all, if we don’t have OpenCV installed, then we can install it via pip:

pip install opencv-python

Now we can continue writing Python code.

At the top, we have to import the OpenCV Python library:

import cv2

After that, read our image as grayscale. Grayscale is a simplified image and it makes the process simple. Below is the code to get grayscale data of the image:

img = cv2.imread('imgs/mypic.jpg',2)

Then set the threshold value for our grayscale image:

ret, bw\_img = cv2.threshold(img,127,255,cv2.THRESH\_BINARY)

Now show the image:

cv2.imshow("Binary Image",bw\_img)

In the end, we have to use waitKey and destroyAllWindows method to keep our window always open until we press any key or close our window and also destroy all windows. Below is the Python code that will do that:

cv2.waitKey(0)

cv2.destroyAllWindows()

The complete and final Python code to convert an RGB or colored image into the binary is given below:

1. import cv2
2. img = cv2.imread('imgs/mypic.jpg',2)
3. ret, bw\_img = cv2.threshold(img,127,255,cv2.THRESH\_BINARY)
4. cv2.imshow("Binary Image",bw\_img)
5. cv2.waitKey(0)
6. cv2.destroyAllWindows()

now you can run and test the above code on your system. You have to pass your own image to the imread() method. After you run the code, you will able to see the binary image of the given image which path you have given to the imread() method.

**9. MORPHOLOGICAL FILTER**

The idea of the morphological filter is shrunk and let grow process. The word “shrink” means using median filter to round off the large structures and to remove the small structures and in grow process, remaining structures are grown back by the same amount.

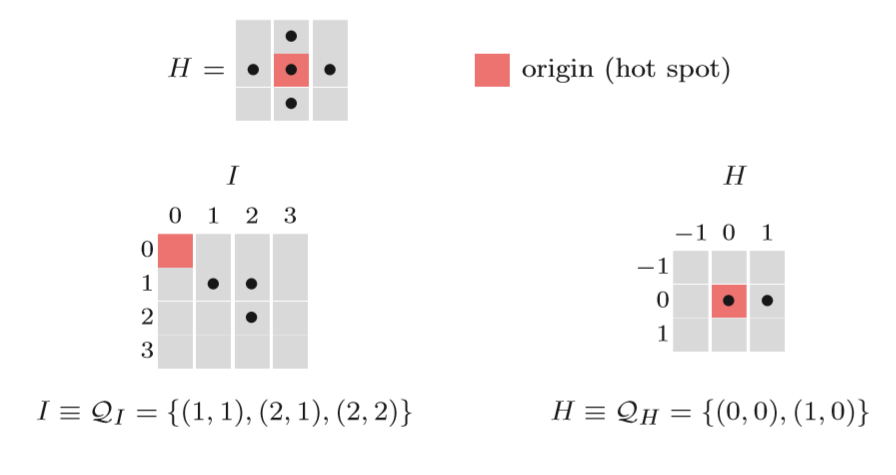
The morphological operation of the binary image is described first and will talk in the following outline.

Outlines are

* The structuring element of a binary filter,
* Dilation and Erosion
* Composite Operation

**THE STRUCTURING ELEMENT:**

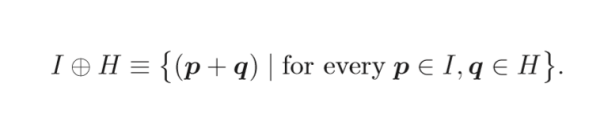
In morphological filter, each element in the matrix is called “structuring element” instead of coefficient matrix in the linear filter. The structuring elements contain only value 0 and 1. And the hot spot of the filter is the dark shade element.

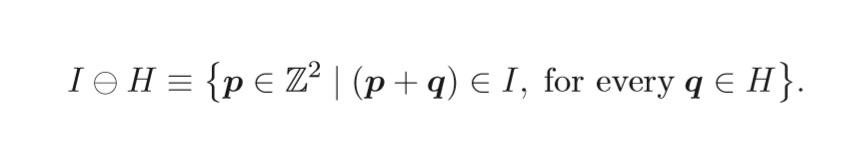


The binary image is described as sets of two-dimensional coordinate point. This is called “Point Set” Q and point set consist of the coordinate pair p = (u,v) of all foreground pixels. Some operations of point set are similar to the operation in others image. For inverting binary image is complement operation and combining two binary image use union operators. Shifting binary image, I by some coordinate vector d by adding vector d to point p. Or reflection of binary image I by multiply -1 to point p.

**DILATION AND EROSION:**

* **Dilation** is a morphological operator which works for the grow process as I mentioned before. The equation of this operator is defined as



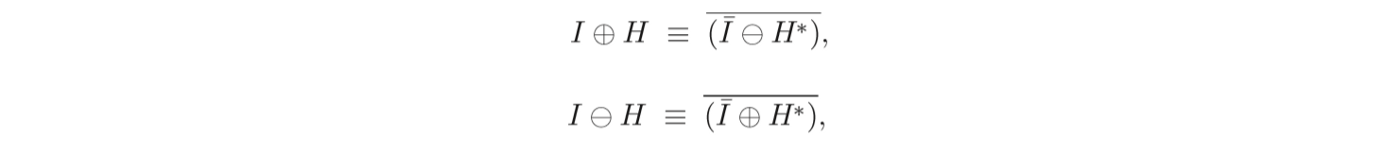
* **Erosion** is a morphological operator which works for the shrink process as I mentioned before as well and the equation is defined as

**Properties of dilation and erosion are**

* **Com-mutative:** only in dilation
* **Associative:** only in dilation

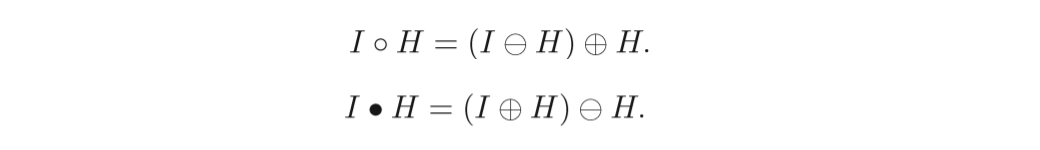
**Note that**: in erosion is in contrast to dilation, not have commutative property.

In addition, erosion and dilation are duels, for a dilation of the foreground can be accomplished by an erosion of background and subsequent of the result in two different properties but work similarity



**COMPOSITE OPERATION:**

In morphological process, dilation and erosion work together in composite operation. There are common ways to represent the order of these two operations, opening and closing. Opening denotes an erosion followed by dilation and closing work in opposite way.



The opening and closing also are dual in sense that opening the foreground is equal to closing the background.



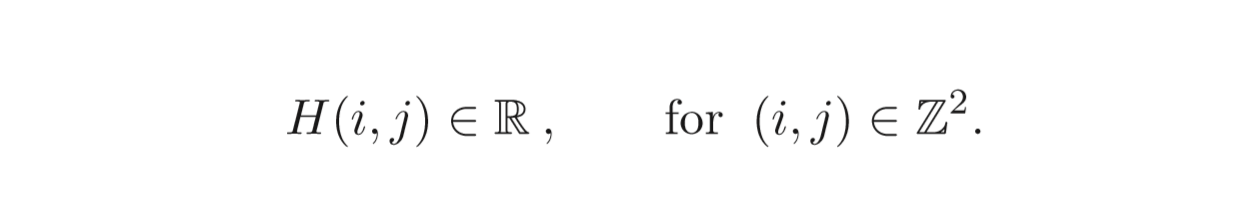
Morphological Filter can also apply to gray-scale image, but in the different definition. It is a generalization with MIN and MAX operators. I will describe in following outline.

Outlines are

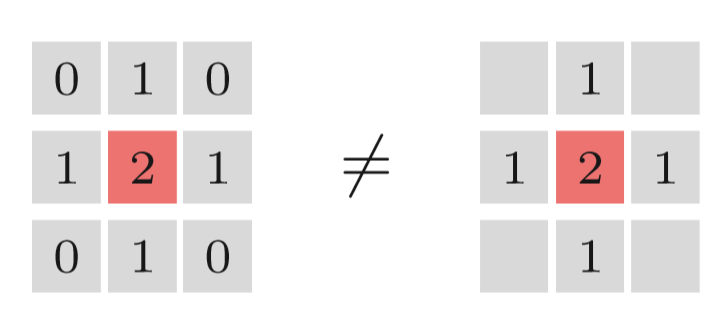
* Structuring Elements,
* Dilation and Erosion,
* Opening and Closing

**STRUCTURING ELEMENT:**

In Gray-scale morphology, structuring elements are defined as real-value 2D functions instead of point sets



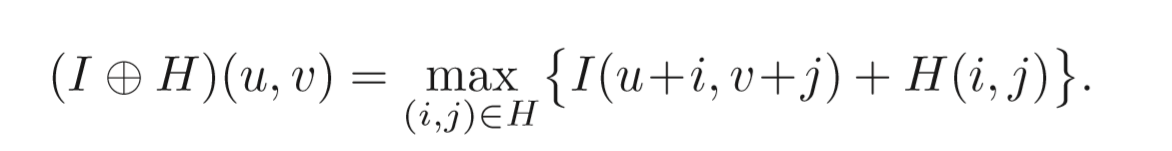
The value in H can be negative or zero value. But it contrasts to linear convolution, zero elements are used to compute the result. And if you do not want to use the elements in some location, you can put no element in that location.



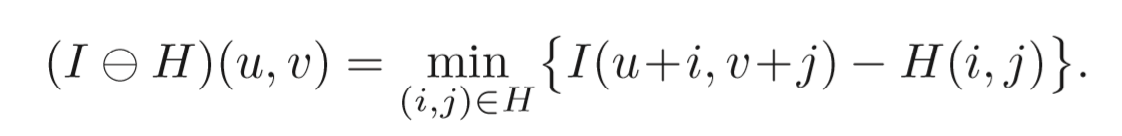
**DILATION AND EROSION:**

The result of dilation and erosion in Gray-scale morphology is contributed from maximum and minimum operation.

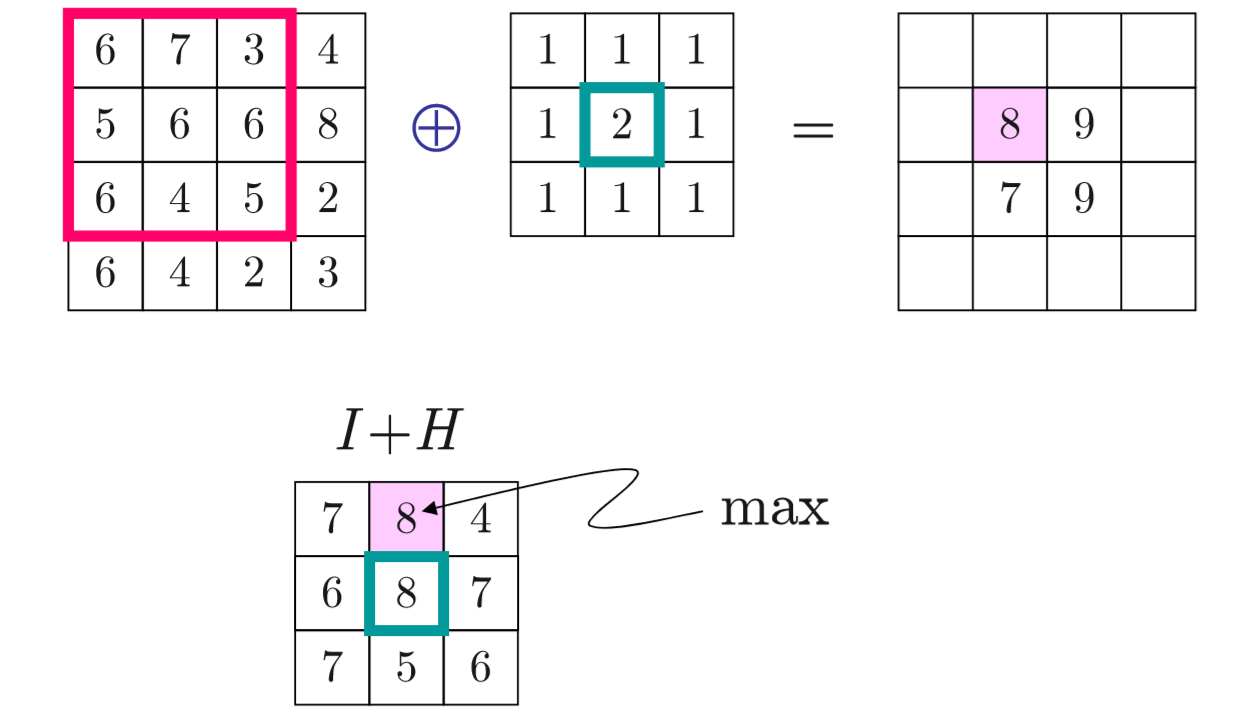
For dilation, the result is the maximum value of the value in H add to the current sub-image.



For erosion, the result is the minimum value of the difference.



These operations can cause the negative value, so we need to clamping the result after calculation.



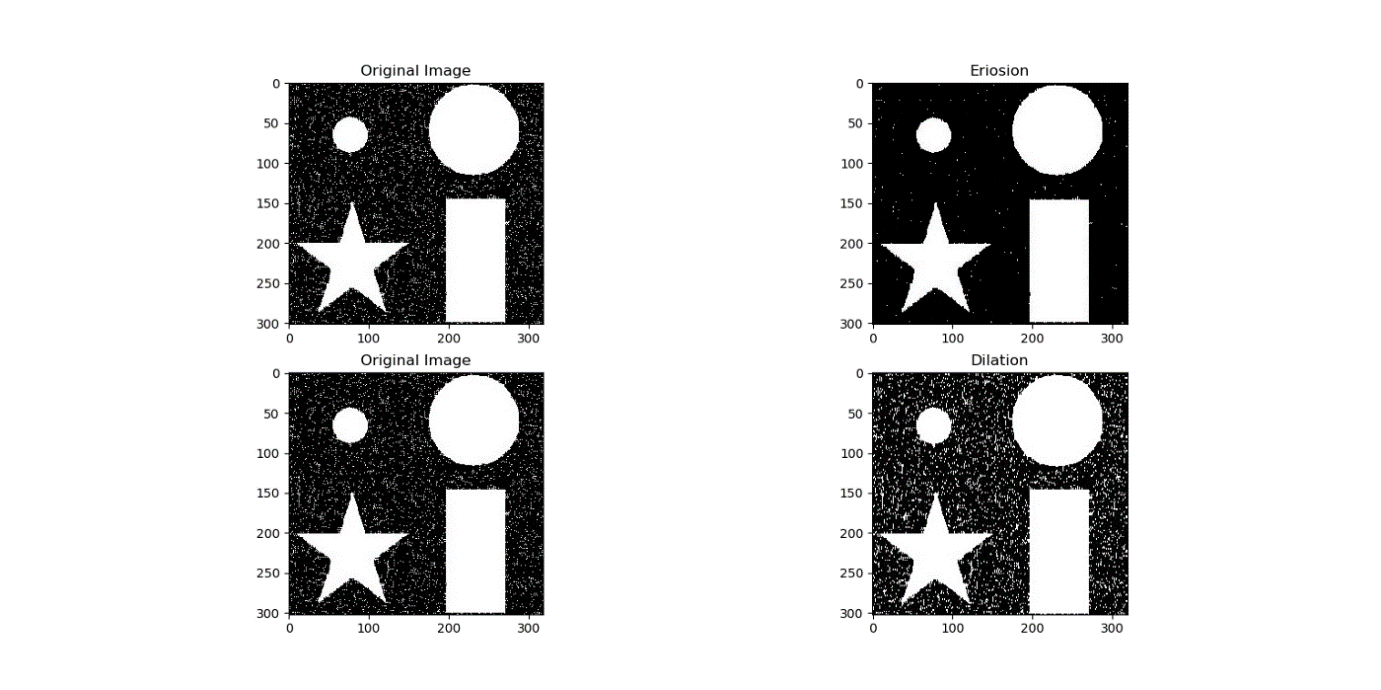
Example of dilation in gray-scale morphology

**OPENING AND CLOSING**

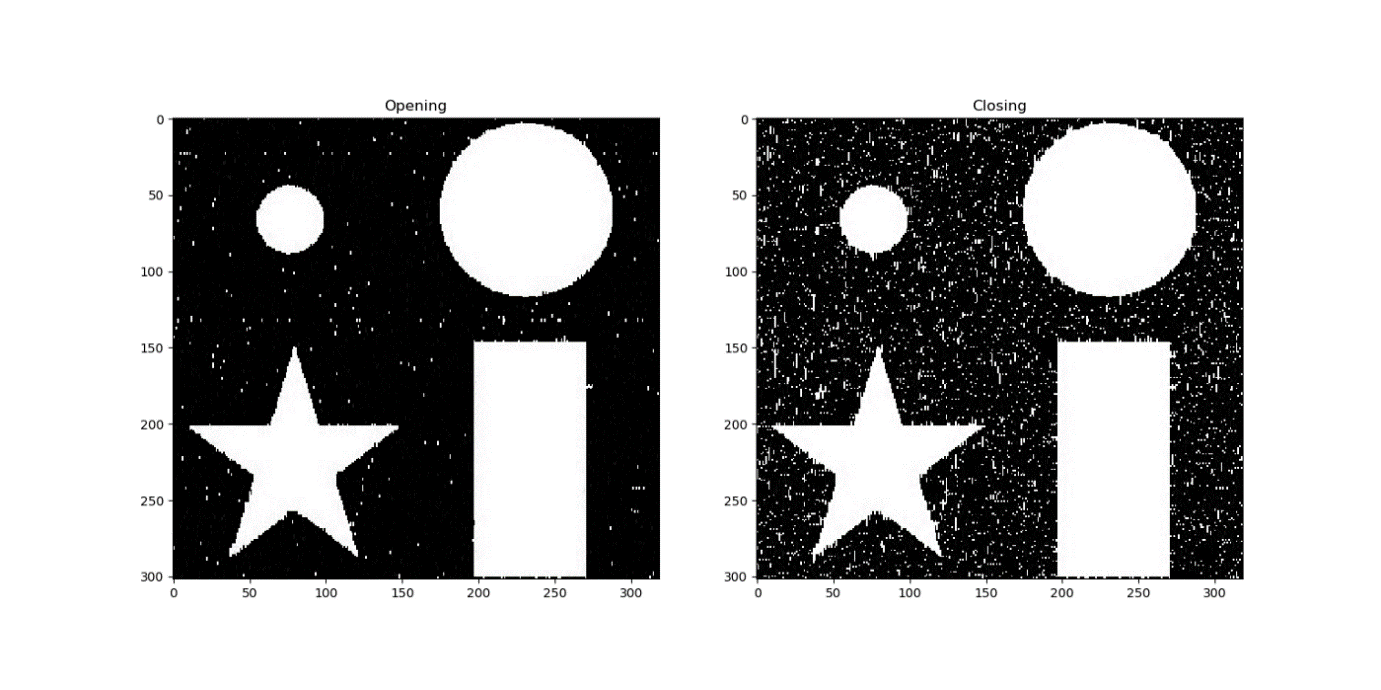
Opening and closing in Gray-scale morphology work in the same way as in binary morphology. The difference is just the operator in dilation and erosion.

For implementation in Python 3 using OpenCV module, you can use the function cv2.erode(input,size) and cv2.dilate(input,size)

This is the result of the program, erosion and dilation, opening and closing.



The result of erosion and dilation of the program.



The result of opening and closing from the program.

**ADVANTAGES:**

* Low complexity
* High accuracy.

**APPLICATIONS:**

* Tracking analysis,
* Security analysis

**10. EDGE DETECTION**

Edge detection includes a variety of mathematical methods that aim at identifying points in a [digital image](https://en.wikipedia.org/wiki/Digital_image) at which the [image brightness](https://en.wikipedia.org/wiki/Luminous_intensity) changes sharply or, more formally, has discontinuities. The points at which image brightness changes sharply are typically organized into a set of curved line segments termed edges.

The same problem of finding discontinuities in one-dimensional signals is known as [step detection](https://en.wikipedia.org/wiki/Step_detection) and the problem of finding signal discontinuities over time is known as [change detection](https://en.wikipedia.org/wiki/Change_detection). Edge detection is a fundamental tool in [image processing](https://en.wikipedia.org/wiki/Image_processing), [machine vision](https://en.wikipedia.org/wiki/Machine_vision) and [computer vision](https://en.wikipedia.org/wiki/Computer_vision), particularly in the areas of [feature detection](https://en.wikipedia.org/wiki/Feature_detection_(computer_vision)) and [feature extraction](https://en.wikipedia.org/wiki/Feature_extraction).  
Then detect the vehicle using the data base.



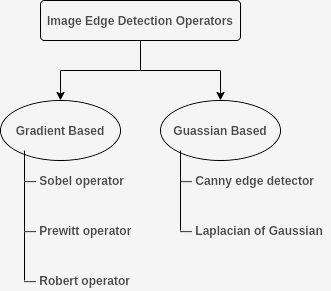
**Edge Detection** is a method of segmenting an image into regions of discontinuity. It is a widely used technique in digital image processing like 

* pattern recognition
* image morphology
* feature extraction

Edge detection allows users to observe the features of an image for a significant change in the Gray level. This texture indicating the end of one region in the image and the beginning of another. It reduces the amount of data in an image and preserves the structural properties of an image.

**Edge Detection Operators** are of two types:

* **Gradient –** based operator which computes first-order derivations in a digital image like, Sobel operator, Prewitt operator, Robert operator
* **Gaussian –** based operator which computes second-order derivations in a digital image like, Canny edge detector, Laplacian of Gaussian

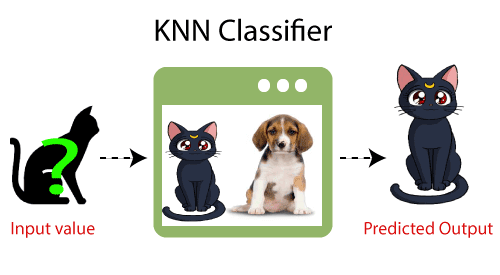


**11. K-NEAREST NEIGHBOR ALGORITHM**

K-Nearest Neighbor is one of the simplest Machine Learning algorithms based on Supervised Learning technique. It assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories. It stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.

K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems. K-NN is a **non-parametric algorithm**, which means it does not make any assumption on underlying data.

It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset. KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

**Example:** Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog. So, for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the cats and dogs’ images and based on the most similar features it will put it in either cat or dog category.

**WHY DO WE NEED A K-NN ALGORITHM?**

Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x1, so this data point will lie in which of these categories. To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the category or class of a particular dataset. Consider the below diagram:

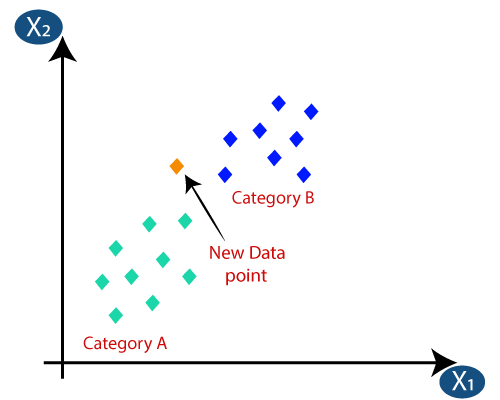


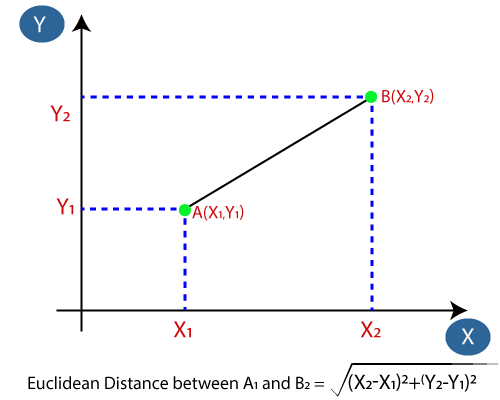
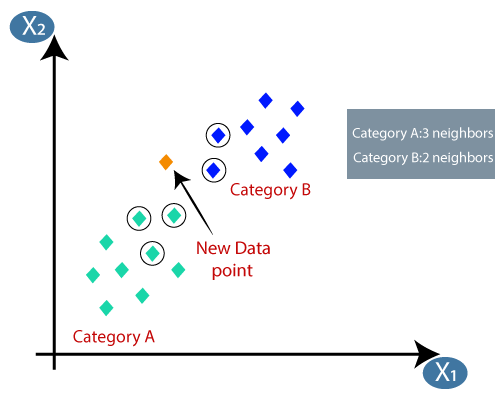
**HOW DOES K-NN WORK?**

The K-NN working can be explained based on the below algorithm:

* **Step-1:** Select the number K of the neighbors
* **Step-2:** Calculate the Euclidean distance of **K number of neighbors**
* **Step-3:** Take the K nearest neighbors as per the calculated Euclidean distance.
* **Step-4:** Among these k neighbors, count the number of the data points in each category.
* **Step-5:** Assign the new data points to that category for which the number of the neighbor is maximum.
* **Step-6:** Our model is ready.

Suppose we have a new data point, and we need to put it in the required category. Consider the below image:



* Firstly, we will choose the number of neighbors, so we will choose the k=5.
* Next, we will calculate the **Euclidean distance** between the data points. The Euclidean distance is the distance between two points, which we have already studied in geometry. It can be calculated as:
* By calculating the Euclidean distance we got the nearest neighbors, as three nearest neighbors in category A and two nearest neighbors in category B. Consider the below image:
* As we can see the 3 nearest neighbors are from category A, hence this new data point must belong to category A.

**HOW TO SELECT THE VALUE OF K IN THE K-NN ALGORITHM?**

Below are some points to remember while selecting the value of K in the K-NN algorithm:

* There is no particular way to determine the best value for "K", so we need to try some values to find the best out of them. The most preferred value for K is 5.
* A very low value for K such as K=1 or K=2, can be noisy and lead to the effects of outliers in the model.
* Large values for K are good, but it may find some difficulties.

**ADVANTAGES OF KNN ALGORITHM:**

* It is simple to implement.
* It is robust to the noisy training data
* It can be more effective if the training data is large.

**DISADVANTAGES OF KNN ALGORITHM:**

* Always needs to determine the value of K which may be complex some time.
* The computation cost is high because of calculating the distance between the data points for all the training samples.

**STEPS TO IMPLEMENT THE K-NN ALGORITHM:**

* Data Pre-processing step
* Fitting the K-NN algorithm to the Training set
* Predicting the test result
* Test accuracy of the result (Creation of Confusion matrix)
* Visualizing the test set result.

**12.TEXT DETECTION AND EXTRACTION USING OPENCV AND OCR**

**OpenCV** (Open-source computer vision) is a library of programming functions mainly aimed at real-time computer vision. [OpenCV](https://www.geeksforgeeks.org/opencv-python-tutorial/) in python helps to process an image and apply various functions like resizing image, pixel manipulations, object detection, etc. In this article, we will learn how to use contours to detect the text in an image and save it to a text file.

**REQUIRED INSTALLATIONS:**

pip install opencv-python

pip install pytesseract

[OpenCV](https://www.geeksforgeeks.org/opencv-python-tutorial/) package is used to read an image and perform certain image processing techniques. Python-tesseract is a wrapper for Google’s Tesseract-OCR Engine which is used to recognize text from images.

**APPROACH:**   
After the necessary imports, a sample image is read using the **imread** function of opencv.

**APPLYING IMAGE PROCESSING FOR THE IMAGE:**

The colorspace of the image is first changed and stored in a variable. For color conversion we use the function cv2.cvtColor(input\_image, flag). The second parameter flag determines the type of conversion. We can chose among **cv2.COLOR\_BGR2GRAY** and **cv2.COLOR\_BGR2HSV**. cv2.COLOR\_BGR2GRAY helps us to convert an RGB image to gray scale image and cv2.COLOR\_BGR2HSV is used to convert an RGB image to HSV (Hue, Saturation, Value) color-space image.

Here, we use **cv2.COLOR\_BGR2GRAY**. A threshold is applied to the converted image using cv2.threshold function.

There are 3 types of thresholding: 

1. Simple Thresholding
2. Adaptive Thresholding
3. Otsu’s Binarization

**13. SOFTWARE USED**

**13.1 PYTHON**

Python is a high-level, cross-platform, and open-sourced programming language released under a GPL-compatible license. [Python Software Foundation](https://www.python.org/psf-landing/) (PSF), a non-profit organization, holds the copyright of Python.

Guido Van Rossum conceived Python in the late 1980s. It was released in 1991 at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to the ABC language. He named this language after a popular comedy show called 'Monty Python's Flying Circus' (and not after Python-the snake).

In the last few years, its popularity has increased immensely. According to stackoverflow.com's recent survey, Python is in the top three [Most Loved Programming Language in 2020](https://insights.stackoverflow.com/survey/2020#technology-most-loved-dreaded-and-wanted-languages-loved).

**PYTHON FEATURES:**

* Python is an object-oriented, high-level language, interpreted, dynamic and multipurpose programming language.
* Python is easy to learn yet powerful and versatile scripting language which makes it attractive for Application Development.
* Python's syntax and dynamic typing with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas.
* Python supports multiple programming pattern, including object-oriented programming, imperative and functional programming, or procedural styles.
* Python is not intended to work on special area such as web programming. That is why it is known as multipurpose because it can be used with web, enterprise, 3D CAD etc.
* We don't need to use data types to declare variable because it is dynamically typed so we can write a=10 to declare an integer value in a variable.
* Python makes the development and debugging fast because there is no compilation step included in python development and edit-test-debug cycle is very fast.

**PYTHON ADVANTAGES:**

**1) Easy to Use:**

Python is easy to very easy to use and high-level language. Thus, it is programmer-friendly language.

**2) Expressive Language:**

Python language is more expressive. The sense of expressive is the code is easily understandable.

**3) Interpreted Language:**

Python is an interpreted language i.e., interpreter executes the code line by line at a time. This makes debugging easy and thus suitable for beginners.

**4) Cross-platform language:**

Python can run equally on different platforms such as Windows, Linux, Unix, Macintosh etc. Thus, Python is a portable language.

**5) Free and Open Source:**

Python language is freely available (www.python.org).The source-code is also available. Therefore, it is open source.

**6) Object-Oriented language:**

Python supports object-oriented language. Concept of classes and objects comes into existence.

**7) Extensible:**

It implies that other languages such as C/C++ can be used to compile the code and thus it can be used further in your python code.

**8) Large Standard Library:**

Python has a large and broad library.

**9) GUI Programming:**

Graphical user interfaces can be developed using Python.

**10) Integrated:**

It can be easily integrated with languages like C, C++, JAVA etc.

**PYTHON APPLICATIONS:**

Python as a whole can be used in any sphere of development.

Let us see what the major regions are where Python proves to be handy.

**1) Console Based Application**

Python can be used to develop console-based applications. For example: IPython.

**2) Audio or Video based Applications**

Python proves handy in multimedia section. Some of real applications are: TimPlayer, cplay etc.

**3) 3D CAD Applications**

Fandango is a real application which provides full features of CAD.

**4) Web Applications**

Python can also be used to develop web-based application. Some important developments are: PythonWikiEngines, Pocoo, PythonBlogSoftware etc.

**5) Enterprise Applications**

Python can be used to create applications which can be used within an Enterprise or an Organization. Some real time applications are: OpenErp, Tryton, Picalo etc.

**6) Applications for Images**

Using Python several applications can be developed for image. Applications developed are: VPython, Gogh, imgSeek etc.

There are several such applications which can be developed using Python6.

**PYTHON TOOLS AND FRAMEWORKS:**

The following lists important tools and frameworks to develop different types of Python applications:

* **Web Development**: [Django](https://www.djangoproject.com/), [Pyramid](http://www.pylonsproject.org/), [Bottle](http://bottlepy.org/), [Tornado](http://tornadoweb.org/), [Flask](http://flask.pocoo.org/), [web2py](http://www.web2py.com/)
* **GUI Development**: [tkInter](https://wiki.python.org/moin/TkInter" \t "_blank), [PyGObject](https://wiki.gnome.org/Projects/PyGObject" \t "_blank), [PyQt](http://www.riverbankcomputing.co.uk/software/pyqt/intro" \t "_blank), [PySide](https://wiki.qt.io/PySide" \t "_blank), [Kivy](https://kivy.org/" \t "_blank), [wxPython](http://www.wxpython.org/" \t "_blank)
* **Scientific and Numeric**: [SciPy](https://www.scipy.org/), [Pandas](https://pandas.pydata.org/), [IPython](http://ipython.org/" \t "_blank)
* **Software Development**: [Buildbot](https://buildbot.net/" \t "_blank), [Trac](http://trac.edgewall.org/), [Roundup](http://roundup.sourceforge.net/)
* **System Administration**: [Ansible](https://www.ansible.com/), [Salt](https://www.saltstack.com/), [OpenStack](https://www.openstack.org/)

**PYTHON EXAMPLE:**

Python code is simple and easy to run. Here is a simple Python code that will print "Welcome to Python".

A simple python example is given below.

1. >>> a="Welcome To Python"
2. >>> print a
3. Welcome To Python
4. >>>

Explanation:

* Here we are using IDLE to write the Python code. Detail explanation to run code is given in Execute Python section.
* A variable is defined named "a" which holds "Welcome To Python".
* "print" statement is used to print the content. Therefore "print a" statement will print the content of the variable. Therefore, the output "Welcome To Python" is produced.

**PYTHON 3.4 EXAMPLE:**

In python 3.4 version, you need to add parenthesis () in a string code to print it.

1. >>> a=("Welcome To Python Example")
2. >>> print a
3. Welcome To Python Example
4. >>>

**HOW TO EXECUTE PYTHON:**

There are three different ways of working in Python:

**1) Interactive Mode:**

You can enter python in the command prompt and start working with Python.

Press Enter key and the Command Prompt will appear like:

Now you can execute your Python commands.

**2) Script Mode:**

Using Script Mode, you can write your Python code in a separate file using any editor of your Operating System.

Save it by .py extension.

Now open Command prompt and execute it by :

**3) Using IDE: (Integrated Development Environment)**

You can execute your Python code using a Graphical User Interface (GUI).

All you need to do is:

Click on Start button -> All Programs -> Python -> IDLE(Python GUI)

You can use both Interactive as well as Script mode in IDE.

**1) Using Interactive mode:**

Execute your Python code on the Python prompt and it will display result simultaneously.

**2) Using Script Mode:**

i) Click on Start button -> All Programs -> Python -> IDLE(Python GUI)

ii) Python Shell will be opened. Now click on File -> New Window.

A new Editor will be opened . Write your Python code here.

Run then code by clicking on Run in the Menu bar.

Run -> Run Module

Result will be displayed on a new Python shell

**PROS AND CONS OF SCRIPT MODE:**

The script mode has few advantages and disadvantages as well.

Let's understand the following advantages of running code in script mode.

* We can run multiple lines of code.
* Debugging is easy in script mode.
* It is appropriate for beginners and also for experts.

Let's see the disadvantages of the script mode.

* We have to save the code every time if we make any change in the code.
* It can be tedious when we run a single or a few lines of code.

**13.2 PYTHON - IDLE**

IDLE (Integrated Development and Learning Environment) is an integrated development environment (IDE) for Python. The Python installer for Windows contains the IDLE module by default.

IDLE is not available by default in Python distributions for Linux. It needs to be installed using the respective package managers. Execute the following command to install IDLE on Ubuntu:

**$ sudo apt-get install idle**

IDLE can be used to execute a single statement just like Python Shell and also to create, modify, and execute Python scripts. IDLE provides a fully-featured text editor to create Python script that includes features like syntax highlighting, autocompletion, and smart indent. It also has a debugger with stepping and breakpoints features.

**FEATURES OF IDLE:**

IDLE has the following features:

* coded in 100% pure Python, using the [tkinter](https://docs.python.org/3/library/tkinter.html" \l "module-tkinter" \o "tkinter: Interface to Tcl/Tk for graphical user interfaces) GUI toolkit
* cross-platform: works mostly the same on Windows, Unix, and macOS
* Python shell window (interactive interpreter) with colorizing of code input, output, and error messages
* multi-window text editor with multiple undo, Python colorizing, smart indent, call tips, auto completion, and other features
* search within any window, replace within editor windows, and search through multiple files (grep)
* debugger with persistent breakpoints, stepping, and viewing of global and local namespaces
* configuration, browsers, and other dialogs

**MENUS:**

IDLE has two main window types, the Shell window and the Editor window. It is possible to have multiple editor windows simultaneously. On Windows and Linux, each has its own top menu. Each menu documented below indicates which window type it is associated with.

Output windows, such as used for Edit => Find in Files, are a subtype of editor window. They currently have the same top menu but a different default title and context menu.

On macOS, there is one application menu. It dynamically changes according to the window currently selected. It has an IDLE menu, and some entries described below are moved around to conform to Apple guidelines.

**13.3 OPENCV**

**INTRODUCTION TO COMPUTER VISION**

Using software to parse the world’s visual content is as big of a revolution in computing as mobile was 10 years ago and will provide a major edge for developers and businesses to build amazing products.

Computer Vision is the process of using machines to understand and analyze imagery (both photos and videos). While these types of algorithms have been around in various forms since the 1960’s, recent advances in [Machine Learning](https://blog.algorithmia.com/introduction-to-machine-learning/), as well as leaps forward in data storage, computing capabilities, and cheap high-quality input devices, have driven major improvements in how well our software can explore this kind of content.

**WHAT IS COMPUTER VISION?**

Computer Vision is the broad parent’s name for any computations involving visual content – that means images, videos, icons, and anything else with pixels involved. But within this parent idea, there are a few specific tasks that are core building blocks:

* In object classification, you train a model on a dataset of specific objects, and the model classifies new objects as belonging to one or more of your training categories.
* For object identification, your model will recognize a specific instance of an object – for example, parsing two faces in an image and tagging one as Tom Cruise and one as Katie Holmes.

A classical application of computer vision is handwriting recognition for digitizing handwritten content (we’ll explore more use cases below). Outside of just recognition, other methods of analysis include:

* Video motion analysis uses computer vision to estimate the velocity of objects in a video, or the camera itself.
* In image segmentation, algorithms partition images into multiple sets of views.
* Scene reconstruction creates a 3D model of a scene inputted through images or video.
* In image restoration, noise such as blurring is removed from photos using Machine Learning based filters.

Any other application that involves understanding pixels through software can safely be labeled as computer vision.

**HOW COMPUTER VISION WORKS:**

One of the major open questions in both Neuroscience and Machine Learning is: how exactly do our brains work, and how can we approximate that with our own algorithms? The reality is that there are very few working and comprehensive theories of brain computation; so despite the fact that Neural Nets are supposed to “mimic the way the brain works,” nobody is quite sure if that’s actually true. Jeff Hawkins has an [entire book on this topic called On Intelligence](https://www.amazon.com/Intelligence-Understanding-Creation-Intelligent-Machines/dp/0805078533).

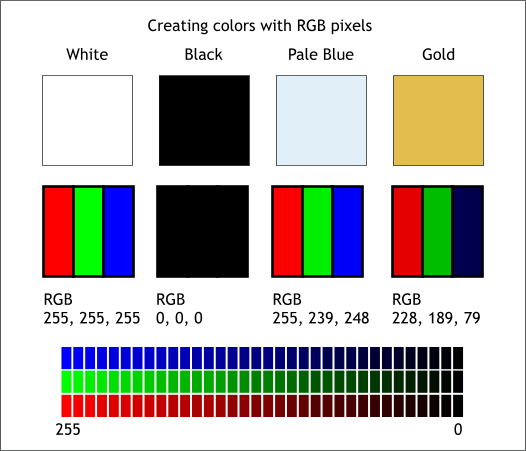
The same paradox holds true for computer vision – since we’re not decided on how the brain and eyes process images, it’s difficult to say how well the algorithms used in production approximate our own internal mental processes. For example, [studies have shown](https://www.technologyreview.com/s/508376/in-a-frogs-eye/) that some functions that we thought happen in the brain of frogs actually take place in the eyes. We’re a far cry from amphibians, but similar uncertainty exists in human cognition.

Machines interpret images very simply: as a series of pixels, each with their own set of color values. Consider the simplified image below, and how grayscale values are converted into a simple array of numbers:



Think of an image as a giant grid of different squares, or pixels (this image is a very simplified version of what looks like either Abraham Lincoln or a Dementor). Each pixel in an image can be represented by a number, usually from 0 – 255. The series of numbers on the right is what software sees when you input an image. For our image, there are 12 columns and 16 rows, which means there are 192 input values for this image.

When we start to add in color, things get more complicated. Computers usually read color as a series of 3 values – red, green, and blue (RGB) – on that same 0 – 255 scale. Now, each pixel actually has 3 values for the computer to store in addition to its position. If we were to colorize President Lincoln (or Harry Potter’s worst fear), that would lead to 12 x 16 x 3 values, or 576 numbers.



For some perspective on how computationally expensive this is, consider this tree:

* Each color value is stored in 8 bits.
* 8 bits x 3 colors per pixel = 24 bits per pixel.
* A normal sized 1024 x 768 image x 24 bits per pixel = almost 19M bits, or about 2.36 megabytes.

That’s a lot of memory to require for one image, and a lot of pixels for an algorithm to iterate over. But to train a model with meaningful accuracy – especially when you’re talking about [Deep Learning](https://blog.algorithmia.com/introduction-to-deep-learning/) – you’d usually need tens of thousands of images, and the more the merrier. Even if you were to use [Transfer Learning](https://en.wikipedia.org/wiki/Transfer_learning) to use the insights of an already trained model, you’d still need a few thousand images to train yours on.

With the sheer amount of computing power and storage required just to train deep learning models for computer vision, it’s not hard to understand why advances in those two fields have driven Machine Learning forward to such a degree.

**BUSINESS USE CASES FOR COMPUTER VISION:**

Computer vision is one of the areas in Machine Learning where core concepts are already being integrated into major products that we use every day. [Google is using maps](https://research.googleblog.com/2017/05/updating-google-maps-with-deep-learning.html) to leverage their image data and identify street names, businesses, and office buildings. Facebook is using computer vision to identify people in photos, and do a number of things with that information.

But it’s not just tech companies that are leverage Machine Learning for image applications. Ford, the American car manufacturer that has been around [literally since the early 1900’s](https://en.wikipedia.org/wiki/Ford_Motor_Company), is [investing heavily in autonomous vehicles (AVs)](https://media.ford.com/content/fordmedia/fna/us/en/news/2016/08/16/ford-targets-fully-autonomous-vehicle-for-ride-sharing-in-2021.html). Much of the underlying technology in AVs relies on analyzing the multiple video feeds coming into the car and using computer vision to analyze and pick a path of action.

Another major area where computer vision can help is in the medical field. Much of diagnosis is image processing, like reading x-rays, MRI scans, and other types of diagnostics. [Google has been working with medical research teams](https://research.google.com/teams/brain/healthcare/) to explore how deep learning can help medical workflows and have made significant progress in terms of accuracy. To paraphrase from their research page:

“Collaborating closely with doctors and international healthcare systems, we developed a state-of-the-art computer vision system for reading retinal fundus images for diabetic retinopathy and determined our algorithm’s performance is on par with U.S. board-certified ophthalmologists. We’ve recently published some of our research in the [Journal of the American Medical Association](https://research.google.com/pubs/archive/45732.pdf) and summarized the highlights in a [blog post](https://research.googleblog.com/2016/11/deep-learning-for-detection-of-diabetic.html).”

But aside from the groundbreaking stuff, it’s getting much easier to integrate computer vision into your own applications. A number of high-quality third party providers like Clarifai offer [a simple API for tagging and understanding images](https://www.clarifai.com/), while Kairos [provides functionality around facial recognition](https://www.kairos.com/). We’ll dive into the open-source packages available for use below.

**COMPUTER VISION ON ALGORITHMIA:**

Algorithmia makes it easy to deploy computer vision applications as scalable microservices. Our marketplace has a few algorithms to help get the job done:

* [SalNet](https://algorithmia.com/algorithms/deeplearning/SalNet) automatically identifies the most important parts of an image
* [Nudity Detection](https://algorithmia.com/algorithms/sfw/NudityDetectioni2v) detects nudity in pictures
* [Emotion Recognition](https://algorithmia.com/algorithms/deeplearning/EmotionRecognitionCNNMBP) parses emotions exhibited in images
* [DeepStyle](https://demos.algorithmia.com/deep-style/) transfers next-level filters onto your image
* [Face Recognition](https://algorithmia.com/algorithms/cv/FaceRecognition)…recognizes faces.
* [Image Memorability](https://algorithmia.com/algorithms/deeplearning/LargescaleImageMemorability) judges how memorable an image is.

A typical workflow for your product might involve passing images from a security camera into Emotion Recognition and raising a flag if any aggressive emotions are exhibited or using Nudity Detection to block inappropriate profile pictures on your web application.

**PACKAGES AND FRAMEWORKS:**

[**OpenCV**](https://opencv.org/)– “OpenCV was designed for computational efficiency and with a strong focus on real-time applications. Adopted all around the world, OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 14 million. Usage ranges from interactive art, to mines inspection, stitching maps on the web or through advanced robotics.”

[**SimpleCV**](http://simplecv.org/) – “SimpleCV is an open-source framework for building computer vision applications. With it, you get access to several high-powered computer vision libraries such as OpenCV – without having to first learn about bit depths, file formats, color spaces, buffer management, eigenvalues, or matrix versus bitmap storage.”

[**Mahotas**](http://mahotas.readthedocs.io/en/latest/)– “Mahotas is a computer vision and image processing library for Python. It includes many algorithms implemented in C++ for speed while operating in numpy arrays and with a very clean Python interface. Mahotas currently has over 100 functions for image processing and computer vision and it keeps growing.

**NUMPY:**

NumPy, which stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on arrays can be performed. This tutorial explains the basics of NumPy such as its architecture and environment. It also discusses the various array functions, types of indexing, etc. An introduction to Matplotlib is also provided. All this is explained with the help of examples for better understanding.

**AUDIENCES:**

This tutorial has been prepared for those who want to learn about the basics and various functions of NumPy. It is specifically useful for algorithm developers. After completing this tutorial, you will find yourself at a moderate level of expertise from where you can take yourself to higher levels of expertise.

**PREREQUISITES:**

* You should have a basic understanding of computer programming terminologies. A basic understanding of Python and any of the programming languages is a plus.
* NumPy is a Python package. It stands for 'Numerical Python'. It is a library consisting of multidimensional array objects and a collection of routines for processing of array.

Numeric, the ancestor of NumPy, was developed by Jim Hugunin. Another package Numarray was also developed, having some additional functionalities. In 2005, Travis Oliphant created NumPy package by incorporating the features of Numarray into Numeric package. There are many contributors to this open-source project.

**OPERATIONS USING NUMPY:**

Using NumPy, a developer can perform the following operations −

* Mathematical and logical operations on arrays.
* Fourier transforms and routines for shape manipulation.
* Operations related to linear algebra. NumPy has in-built functions for linear algebra and random number generation.

**14. SOURCE CODE**

# Main.py

import cv2

import numpy as np

import os

import DetectChars

import DetectPlates

import PossiblePlate

#module level variables##################################################

SCALAR\_BLACK = (0.0, 0.0, 0.0)

SCALAR\_WHITE = (255.0, 255.0, 255.0)

SCALAR\_YELLOW = (0.0, 255.0, 255.0)

SCALAR\_GREEN = (0.0, 255.0, 0.0)

SCALAR\_RED = (0.0, 0.0, 255.0)

showSteps = True

###################################################################################################

def main():

blnKNNTrainingSuccessful = DetectChars.loadKNNDataAndTrainKNN() # attempt KNN training

if blnKNNTrainingSuccessful == False: # if KNN training was not successful

print("\nerror: KNN traning was not successful\n") # show error message

return # and exit program

# end if

imgOriginalScene = cv2.imread("1.png") # open image

if imgOriginalScene is None: # if image was not read successfully

print("\nerror: image not read from file \n\n") # print error message to std out

os.system("pause") # pause so user can see error message

return # and exit program

# end if

listOfPossiblePlates = DetectPlates.detectPlatesInScene(imgOriginalScene) # detect plates

listOfPossiblePlates = DetectChars.detectCharsInPlates(listOfPossiblePlates) # detect chars in plates

# resizeimage.resize\_cover(imgOriginalScene, [200, 100])

cv2.imshow("imgOriginalScene", imgOriginalScene) # show scene image

if len(listOfPossiblePlates) == 0: # if no plates were found

print("\nno license plates were detected\n") # inform user no plates were found

else: # else

# if we get in here list of possible plates has at leat one plate

# sort the list of possible plates in DESCENDING order (most number of chars to least number of chars)

listOfPossiblePlates.sort(key = lambda possiblePlate: len(possiblePlate.strChars), reverse = True)

# suppose the plate with the most recognized chars (the first plate in sorted by string length descending order) is the actual plate

licPlate = listOfPossiblePlates[0]

cv2.imshow("imgPlate", licPlate.imgPlate) # show crop of plate and threshold of plate

cv2.imshow("imgThresh", licPlate.imgThresh)

if len(licPlate.strChars) == 0: # if no chars were found in the plate

print("\nno characters were detected\n\n") # show message

return # and exit program

# end if

drawRedRectangleAroundPlate(imgOriginalScene, licPlate) # draw red rectangle around plate

print("\nlicense plate read from image = " + licPlate.strChars + "\n") # write license plate text to std out

print("----------------------------------------")

writeLicensePlateCharsOnImage(imgOriginalScene, licPlate) # write license plate text on the image

cv2.imshow("imgOriginalScene", imgOriginalScene) # re-show scene image

cv2.imwrite("imgOriginalScene.png", imgOriginalScene) # write image out to file

# end if else

cv2.waitKey(0) # hold windows open until user presses a key

return

# end main

###################################################################################################

def drawRedRectangleAroundPlate(imgOriginalScene, licPlate):

p2fRectPoints = cv2.boxPoints(licPlate.rrLocationOfPlateInScene) # get 4 vertices of rotated rect

## cv2.line(imgOriginalScene, tuple(p2fRectPoints[0]), tuple(p2fRectPoints[1]), SCALAR\_RED, 2) # draw 4 red lines

## cv2.line(imgOriginalScene, tuple(p2fRectPoints[1]), tuple(p2fRectPoints[2]), SCALAR\_RED, 2)

## cv2.line(imgOriginalScene, tuple(p2fRectPoints[2]), tuple(p2fRectPoints[3]), SCALAR\_RED, 2)

## cv2.line(imgOriginalScene, tuple(p2fRectPoints[3]), tuple(p2fRectPoints[0]), SCALAR\_RED, 2)

# end function

###################################################################################################

def writeLicensePlateCharsOnImage(imgOriginalScene, licPlate):

ptCenterOfTextAreaX = 0 # this will be the center of the area the text will be written to

ptCenterOfTextAreaY = 0

ptLowerLeftTextOriginX = 0 # this will be the bottom left of the area that the text will be written to

ptLowerLeftTextOriginY = 0

sceneHeight, sceneWidth, sceneNumChannels = imgOriginalScene.shape

plateHeight, plateWidth, plateNumChannels = licPlate.imgPlate.shape

intFontFace = cv2.FONT\_HERSHEY\_SIMPLEX # choose a plain jane font

fltFontScale = float(plateHeight) / 30.0 # base font scale on height of plate area

intFontThickness = int(round(fltFontScale \* 1.5)) # base font thickness on font scale

textSize, baseline = cv2.getTextSize(licPlate.strChars, intFontFace, fltFontScale, intFontThickness) # call getTextSize

# unpack roatated rect into center point, width and height, and angle

( (intPlateCenterX, intPlateCenterY), (intPlateWidth, intPlateHeight), fltCorrectionAngleInDeg ) = licPlate.rrLocationOfPlateInScene

intPlateCenterX = int(intPlateCenterX) # make sure center is an integer

intPlateCenterY = int(intPlateCenterY)

ptCenterOfTextAreaX = int(intPlateCenterX) # the horizontal location of the text area is the same as the plate

if intPlateCenterY < (sceneHeight \* 0.75): # if the license plate is in the upper 3/4 of the image

ptCenterOfTextAreaY = int(round(intPlateCenterY)) + int(round(plateHeight \* 1.6)) # write the chars in below the plate

else: # else if the license plate is in the lower 1/4 of the image

ptCenterOfTextAreaY = int(round(intPlateCenterY)) - int(round(plateHeight \* 1.6)) # write the chars in above the plate

# end if

textSizeWidth, textSizeHeight = textSize # unpack text size width and height

ptLowerLeftTextOriginX = int(ptCenterOfTextAreaX - (textSizeWidth / 2)) # calculate the lower left origin of the text area

ptLowerLeftTextOriginY = int(ptCenterOfTextAreaY + (textSizeHeight / 2)) # based on the text area center, width, and height

# write the text on the image

cv2.putText(imgOriginalScene, licPlate.strChars, (ptLowerLeftTextOriginX, ptLowerLeftTextOriginY), intFontFace, fltFontScale, SCALAR\_YELLOW, intFontThickness)

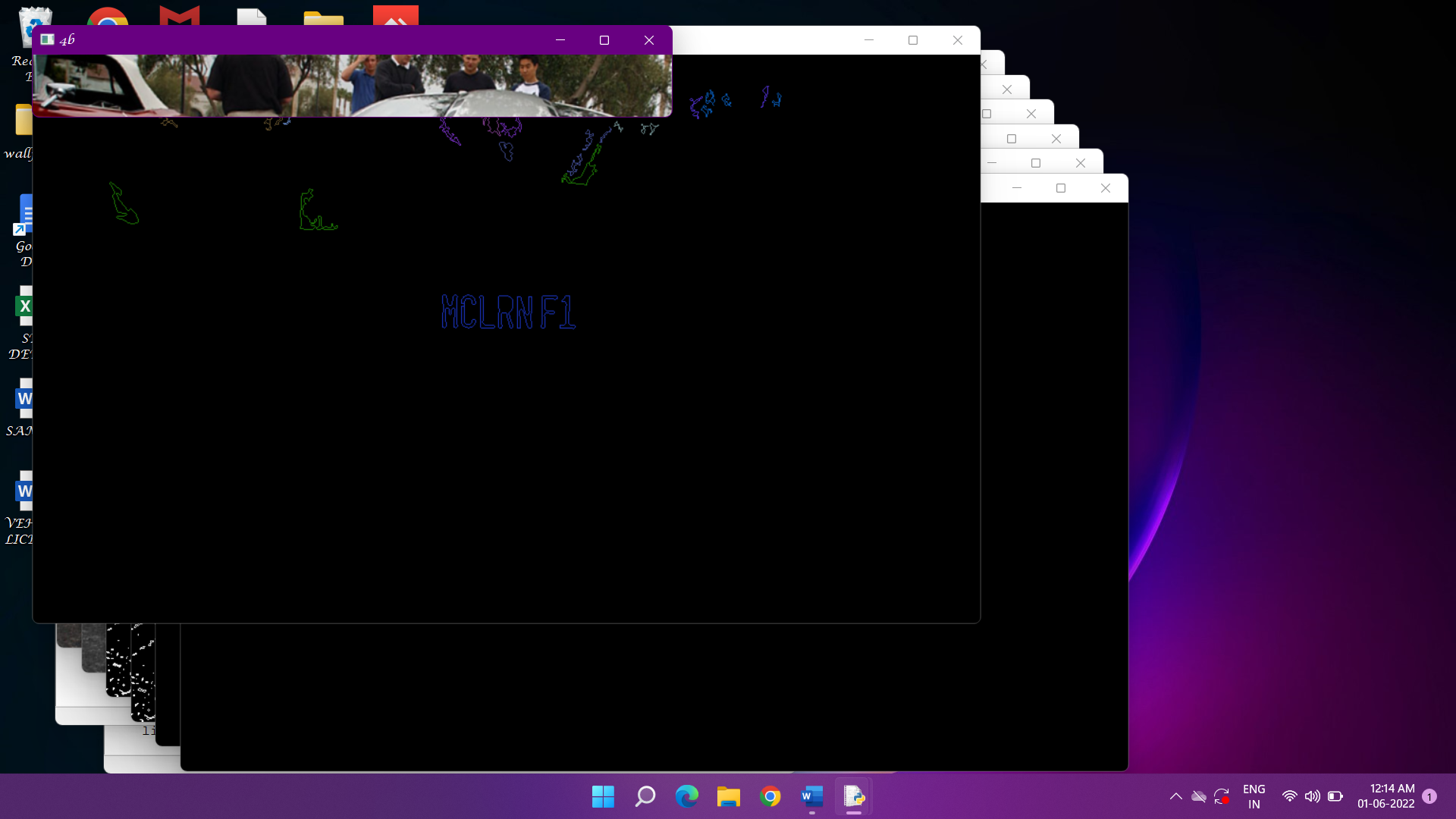
# end function

###################################################################################################

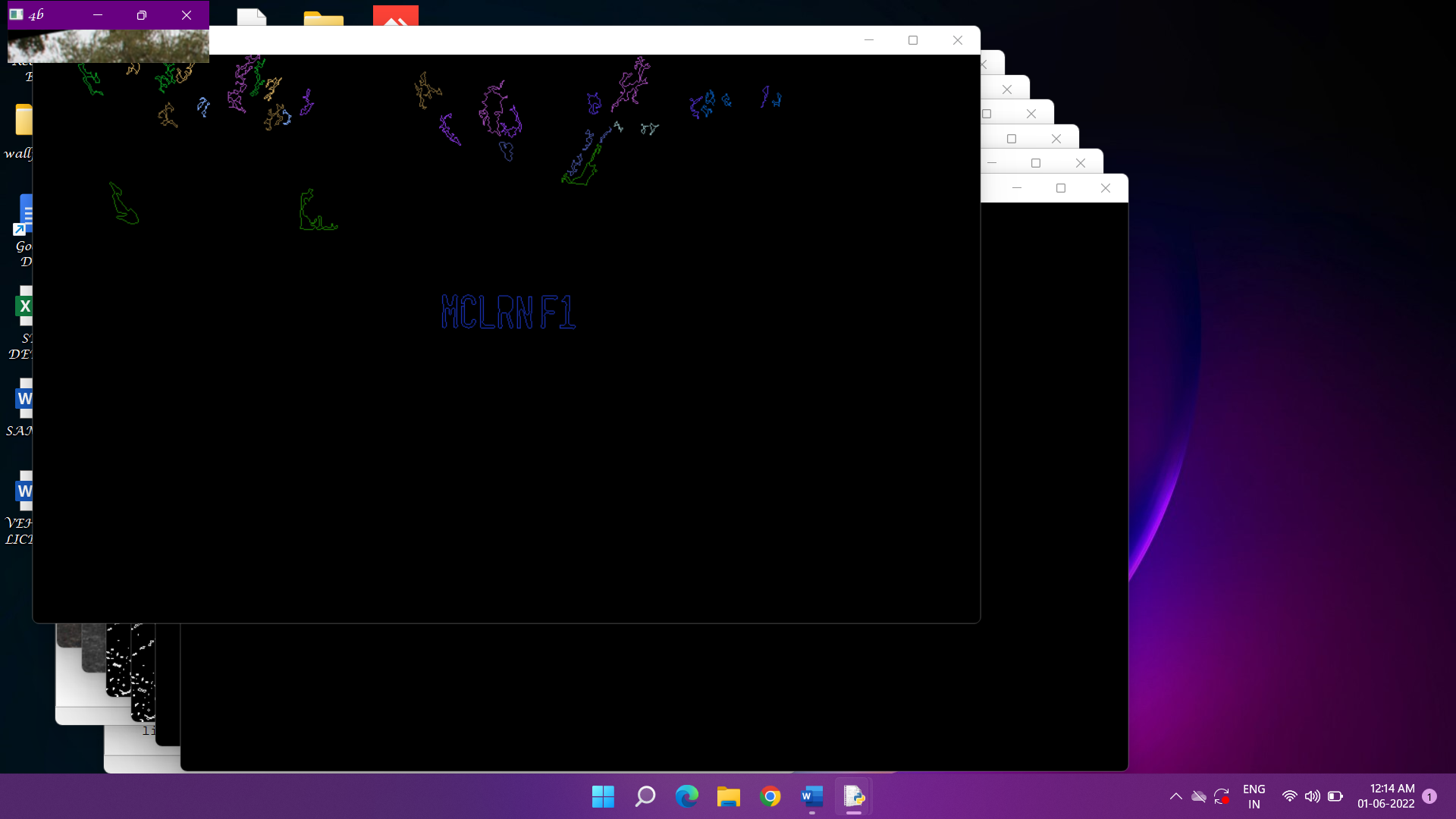
if \_\_name\_\_ == "\_\_main\_\_":

main()

**15. SCREENSHOTS**

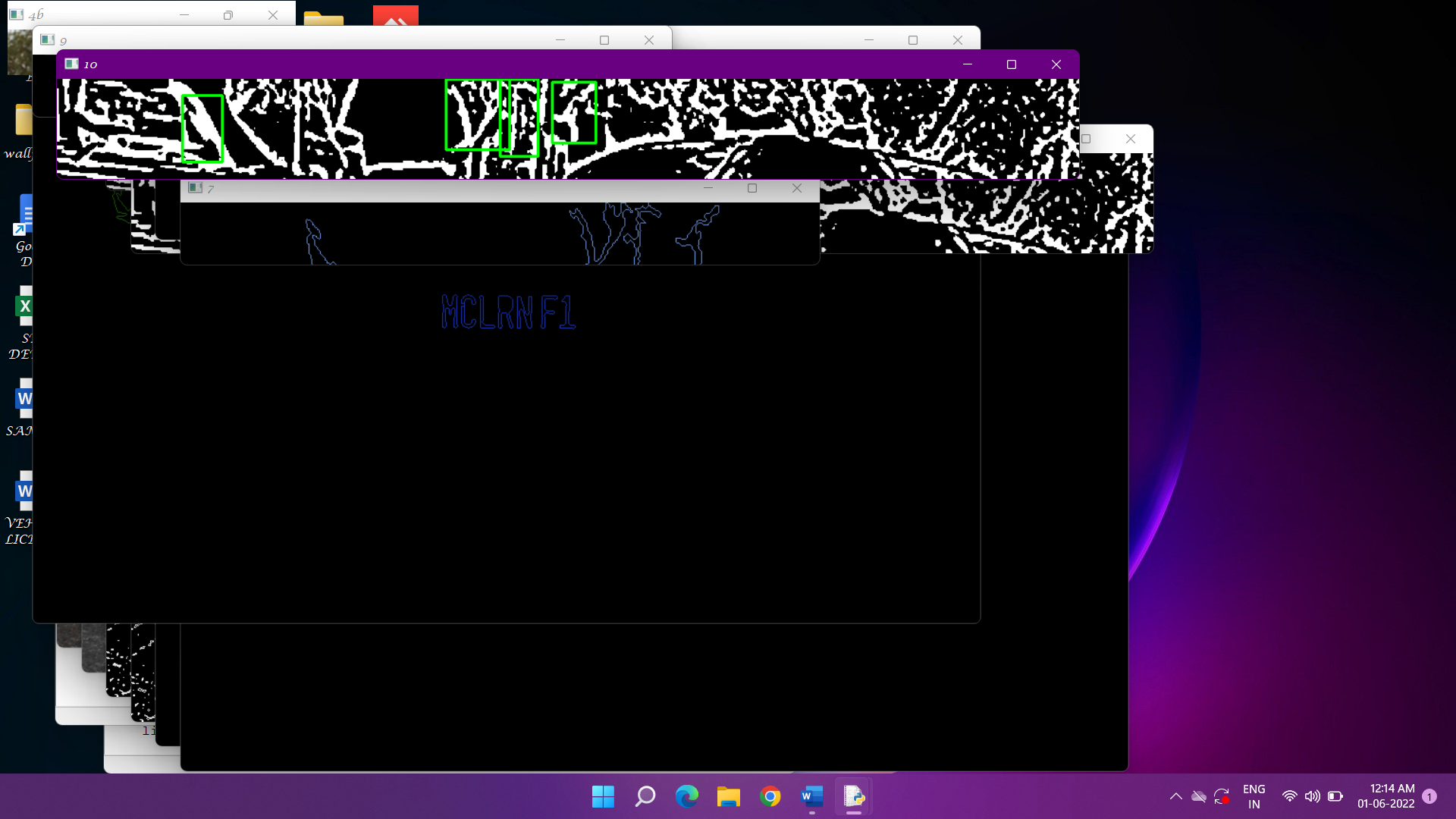
**1.**

**2.**

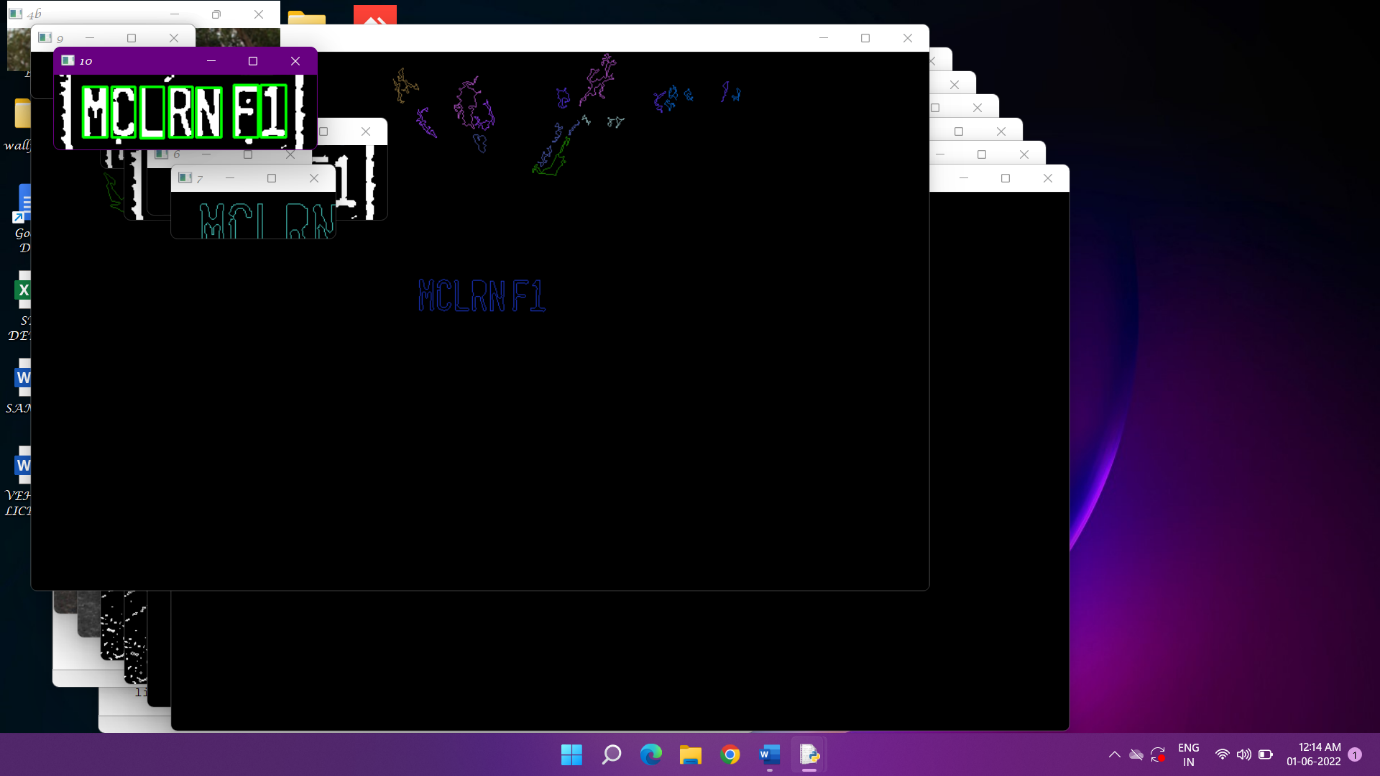
****

**-**

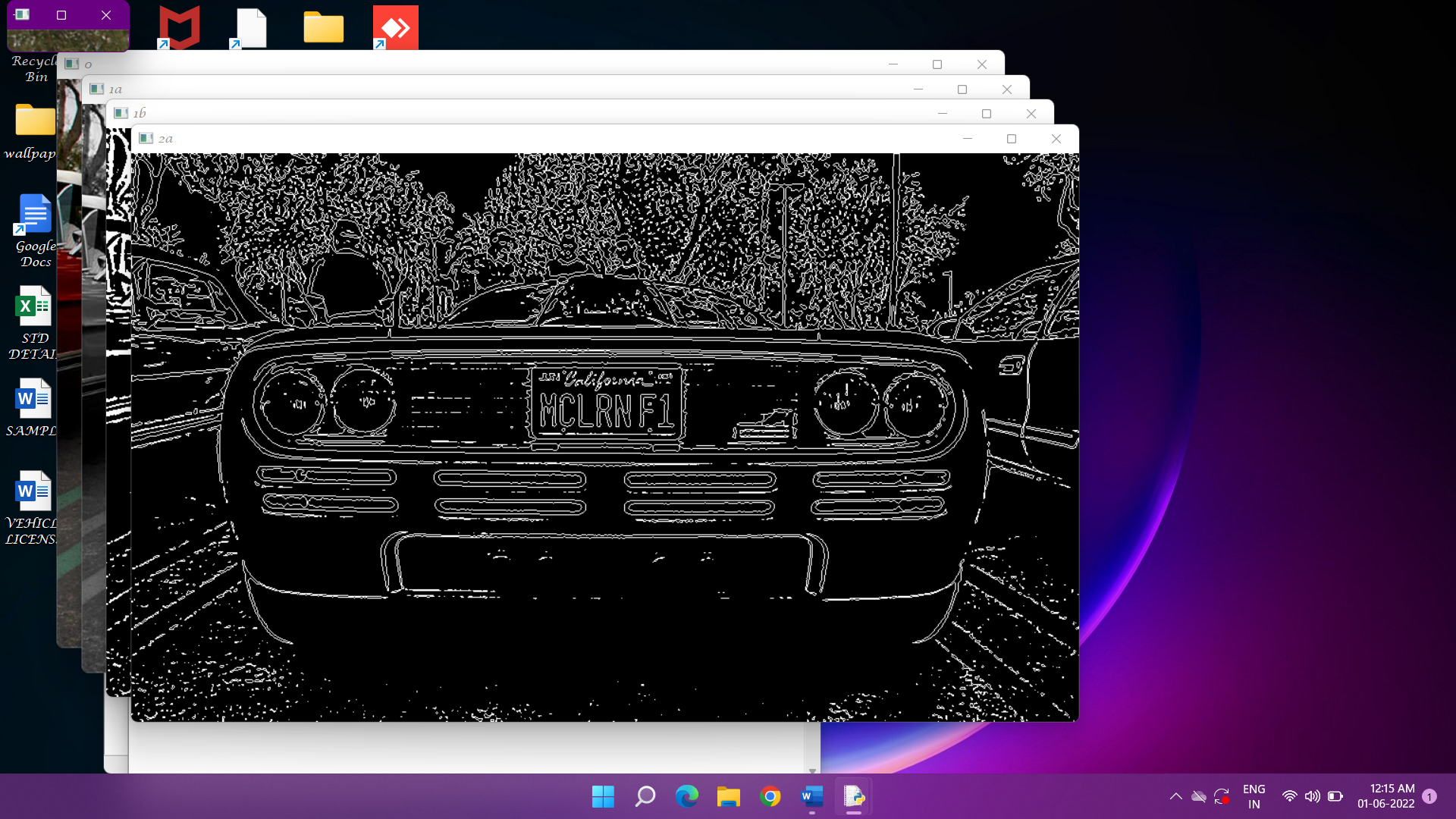
**3.**

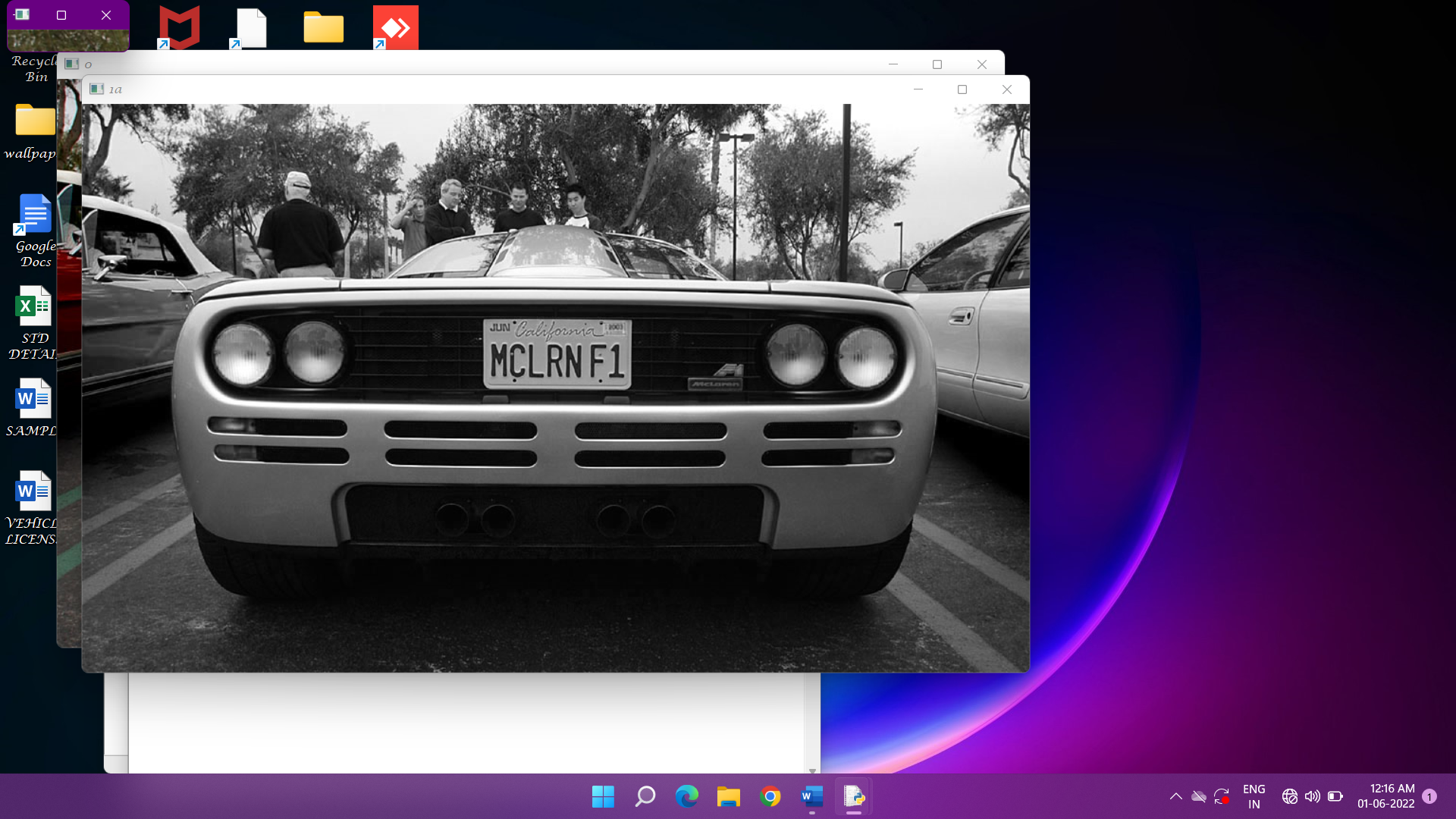
****

**4.**

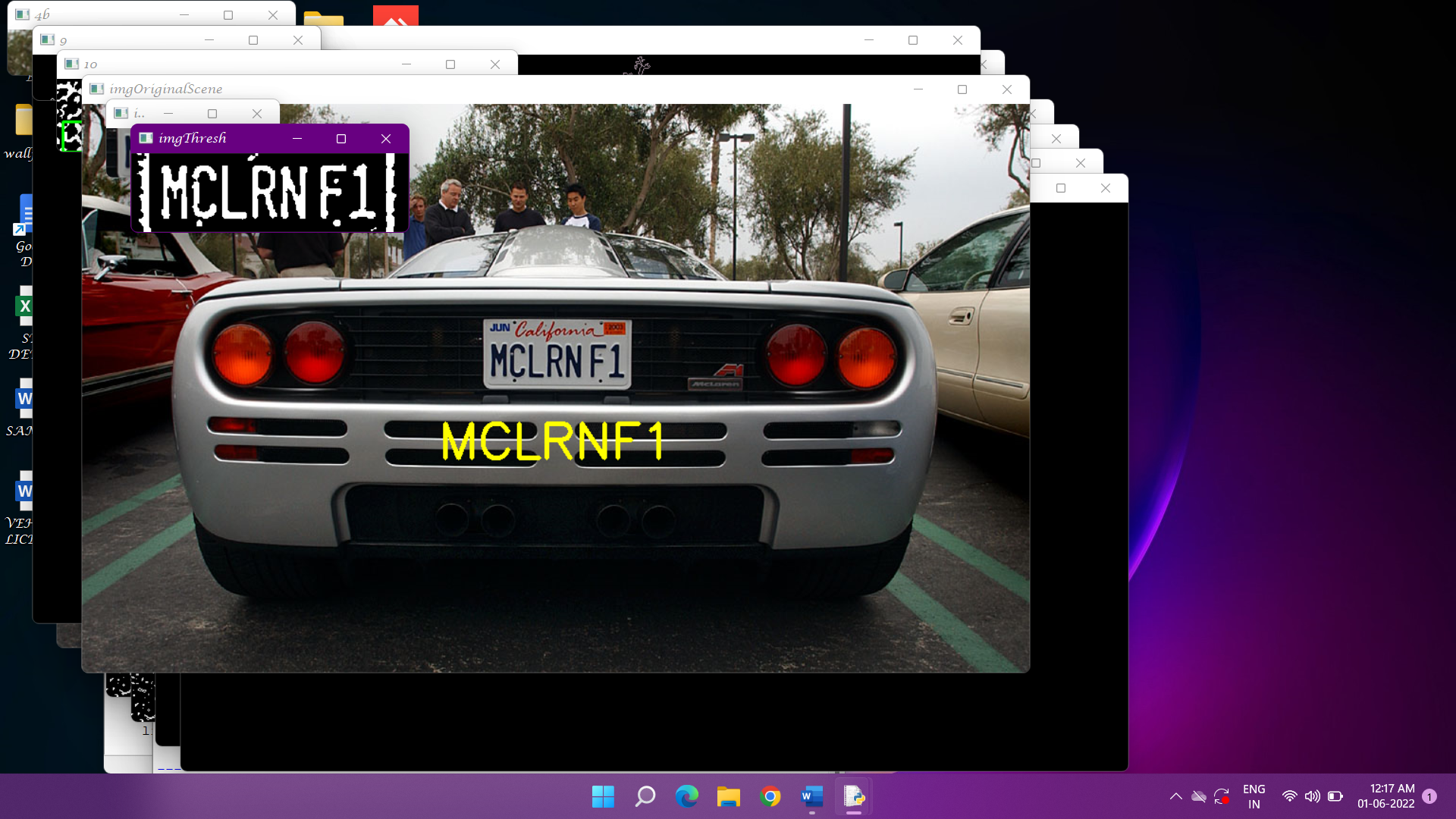
****

**5.**

****

**6.**

**7.**

****

**16. FUTUTRE ENHANCEMENT**

According to the license plate detection feature extraction process is important during this detection we can detect the thief when he will take the vehicle.

Here we are using morphological filters for the addition of color to detect the features. Maximum we are using on roads and car parking areas.

Or the future purpose we can use deep neural network algorithm. Using this process, we can detect more vehicles at a time.

These can be use at traffic, parking Ares, airport etc.. For the future purpose we can avoid robberies.

**17. CONCLUSION**

Four algorithms of image preprocessing, license plate location, license plate segmentation and character recognition are introduced in this paper.

License plate location is the basis of image preprocessing.

The location of license plate has a direct impact on the accuracy of character segmentation.

In our project vehicle plate detection is obtained by using character recognition region of interest.

**18. BIBLIOGRAPHY**

**REFERENCES:**

[1] A. H. Ashtari, “License plate localization and recognition in color images

based on new feature of Iranian license plate and support vector machine,”

M.S. thesis, Dept. Eng., Islamic Azad Univ., Dezful, Iran, 2009.

[2] C. N. E. Anagnostopoulos, I. E. Anagnostopoulos, I. D. Psoroulas,

V. Loumos, and E. Kayafas, “License plate recognition from still images

and video sequences: A survey,” *IEEE Trans. Intell. Transp. Syst.*, vol. 9,

no. 3, pp. 377–391, Sep. 2008.

[3] W. Jia, H. Zhang, X. He, and M. Piccardi, “Mean shift for accurate license

plate localization,” in *Proc. IEEE Intell. Transp. Syst.*, 2005, pp. 566–571.

[4] S. L. Chang, L. S. Chen, Y. C. Chung, and S.W. Chen, “Automatic license

plate recognition,” *IEEE Trans. Intell. Transp. Syst.*, vol. 5, no. 1, pp. 42–

53, Mar. 2004.

[5] X. Shi,W. Zhao, and Y. Shen, “Automatic license plate recognition system

based on color image processing,” in *Proc. ICCSA*, 2005, pp. 307–314.

[6] V. Abolghasemi and A. Ahmadyfard, “An edge-based color-aided method

for license plate detection,” *Image Vis. Comput.*, vol. 27, no. 8, pp. 1134–

1142, Jul. 2009.

[7] R. W. Rodieck, *The First Steps in Seeing*. Sunderland, MA, USA:

Sinauer Associates, 1998.

[8] E. R. Lee, P. K. Kim, and H. J. Kim, “Automatic recognition of a car

license plate using color image processing,” in *Proc. IEEE ICIP*, 1994,

pp. 301–305.

[9] C.-C. Lin and W.-H. Huang, “Locating license plate based on edge features

of intensity and saturation subimages,” in *Proc. 2nd ICICIC*, 2007, p. 227.

[10] F. Wang, L. Man, B. Wang, Y. Xiao, W. Pan, and X. Lu, “Fuzzy-based

algorithm for color recognition of license plates,” *Pattern Recognit. Lett.*,

vol. 29, no. 7, pp. 1007–1020, May 2008.

[11] Y. Yanamura, M. Goto, D. Nishiyama, M. Soga, H. Nakatani, and H. Saji,

“Extraction and tracking of the license plate using Hough transform

and voted block matching,” in *Proc. IEEE Intell. Veh. Symp.*, 2003,

pp. 243–246.

[12] T. D. Duan, T. L. H. Du, T. V. Phuoc, and N. V. Hoang, “Building an

automatic vehicle license plate recognition system,” in *Proc. Int. Conf.*

*Comput. Sci. RIVF*, 2005, pp. 59–63.

[13] N. Zimic, J. Ficzko, M. Mraz, and J. Virant, “The fuzzy logic approach to

the car number plate locating problem,” in *Proc. IIS*, 1997, pp. 227–230.

[14] S. M. Youssef and S. B. AbdelRahman, “RETRACTED: A smart access

control using an efficient license plate location and recognition approach,”

*Exp. Syst. Appl.*, vol. 34, no. 1, pp. 256–265, Jan. 2008.

[15] H. Kwanicka and B. Wawrzyniak, “License plate localization and recognition

in camera pictures,” in *Proc. 3rd Symp. Methods Artif. Intell.*, 2002,

pp. 243–246.

[16] H. Bai, J. Zhu, and C. Liu, “A fast license plate extraction method

on complex background,” in *Proc. IEEE Intel. Transp. Syst.*, 2003,

pp. 985–987.

[17] D. Zheng, Y. Zhao, and J. Wang, “An efficient method of license

plate location,” *Pattern Recognit. Lett.*, vol. 26, no. 15, pp. 2431–2438,

Nov. 2005.