VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Title:

Measuring Melanin content on a person's skin using image processing and Artificial Neural Network

Faculty:

Dr. Sridhar Ranganadhan

Team:

Ramsundar Karthisan S (21BCE5148)

Kaushal Kanna (21BCE1409)

Om Prakash (21BCE1950)

Abstract:

Melanin, the primary determinant of skin color in humans, plays a crucial role in protecting the skin against harmful ultraviolet radiation. However, abnormal melanin levels can lead to various skin conditions and diseases, including melanoma. Currently, the measurement of melanin content in a person's skin is typically performed through invasive procedures or specialized equipment, which can be costly, time-consuming, and uncomfortable for the patient.

With the advancement of image processing techniques and artificial neural networks, there is potential for a non-invasive, cost-effective, and efficient method for measuring melanin content. However, the application of these technologies for this specific purpose has not been thoroughly explored.

The aim of this research is to develop and validate a novel method for measuring melanin content in a person's skin using image processing and artificial neural networks. This could provide a valuable tool for dermatologists and potentially lead to earlier detection and treatment of skin conditions related to abnormal melanin levels.

References:

1. **Deep Neural Networks Based Recognition of Plant Diseases by Leaf Image Classification by Srdjan Sladojevic, Marko Arsenovic, Andras Anderla, Dubravko Culibrk, and Darko Stefanovic.**

   This research paper introduces a novel approach to plant disease recognition through the development of a deep convolutional neural network (CNN) model focused on leaf image classification. The primary objective is to provide a quick and efficient system implementation for practical use. The model, trained using a unique methodology, demonstrates the capability to recognize 13 different types of plant diseases in addition to healthy leaves while effectively distinguishing plant leaves from their surroundings. As per the authors' knowledge, this paper marks the first proposal of such a method for plant disease recognition.

The methodology begins with a comprehensive description of essential steps involved in implementing the disease recognition model. The process starts with the collection of images to create a database, which is then assessed by agricultural experts to ensure the authenticity and relevance of the data. Caffe, a deep learning framework developed by the Berkeley Vision and Learning Center, is chosen for conducting the deep CNN training.

The results obtained from the experiments conducted on the developed model showcase promising precision levels. The precision ranges between 91% and 98% for individual class tests, with an impressive average precision of 96.3%. This high precision is indicative of the model's accuracy in correctly identifying various plant diseases and distinguishing them from healthy leaves. The use of deep CNNs in this context is pivotal, as it leverages the advanced capabilities of neural networks in learning complex hierarchical features from images, contributing to the model's robust performance.

The paper's significance lies in its contribution to the field of automated plant disease detection, providing not only a novel approach but also detailed insights into the implementation steps, from image collection to model training. The achieved precision levels underscore the effectiveness of the proposed methodology, making it a valuable addition to the existing literature on plant disease recognition using deep learning techniques. The authors' use of Caffe further highlights the versatility of deep learning frameworks in facilitating efficient model training for real-world applications. Overall, this research paper presents a comprehensive and pioneering contribution to the automated plant disease detection domain.

1. **A Five Convolutional Layer Deep Convolutional Neural Network for Plant Leaf Disease Detection by J. Arun Pandian, K.Kanchanadevi, Dhilip Kumar.**

This research proposes a novel Deep Convolutional Neural Network (DCNN) model for image-based plant leaf disease identification, leveraging advanced techniques such as data augmentation and hyperparameter optimization. The DCNN model is trained on an extensive dataset comprising over 240,000 images, encompassing various healthy and diseased plant leaves along with diverse backgrounds. To enhance the dataset, five image augmentation techniques are employed, including Generative

Adversarial Network, Neural Style Transfer, Principal Component Analysis, Color Augmentation, and Position Augmentation.

A crucial aspect of the research involves the optimization of the DCNN model's hyperparameters using the random search technique. The study emphasizes the significance of choosing an appropriate number of layers and filters in the development of a DCNN, shedding light on the impact of architecture on model performance. The experimental outcomes underscore the importance of both data augmentation and hyperparameter optimization techniques in achieving superior results.

Performance evaluation of the proposed DCNN is conducted using various metrics such as classification accuracy, precision, recall, and F1-Score. The experimental results reveal an impressive average classification accuracy of 98.41% on the test dataset, demonstrating the efficacy of the proposed model in accurately identifying plant leaf diseases. Furthermore, a comparative analysis indicates that the overall performance of the proposed DCNN model surpasses that of advanced transfer learning and traditional machine learning techniques.

The findings of this research highlight the potential of the proposed DCNN model as a valuable tool for plant leaf disease identification. The combination of data augmentation and hyperparameter optimization contributes to the model's robustness and superior performance. The achieved high classification accuracy underscores the effectiveness of the DCNN architecture in handling complex image-based classification tasks. Overall, this research provides insights into the importance of architecture design, data augmentation, and hyperparameter optimization in the development of effective deep learning models for plant leaf disease identification. The proposed DCNN model stands out as a promising solution in this domain, offering advancements over existing methodologies.

2. **Deep Learning for Plant Diseases: Detection and Saliency Map Visualisation by Mohammed Brahimi, Marko Arsenovic, Sohaib Laraba, Srdjan Sladojevic, Kamel Boukhalfa & Abdelouhab Moussaoui.**

This chapter addresses the need for improved performance in plant disease detection systems by harnessing the success of deep learning in computer vision. The researchers emphasize a departure from conventional approaches relying on outdated architectures like AlexNet and GoogleNet, urging a shift towards recent and advanced

deep architectures. Furthermore, they highlight a critical issue in existing studies - the lack of transparency in deep classifiers, often treated as black boxes. To address this limitation, the researchers explore deep learning visualisation methods, aiming to enhance the interpretability and understanding of the classification mechanisms.

The chapter conducts an extensive evaluation of multiple state-of-the-art Convolutional Neural Network (CNN) architectures using three distinct learning strategies. This evaluation is performed on a public dataset dedicated to plant diseases classification. The results showcase the superiority of the tested architectures, surpassing state-of-the-art performance with an impressive accuracy of 99.76%. This signifies a substantial improvement over the traditional reliance on older CNN architectures, demonstrating the efficacy of adopting modern deep learning frameworks for enhanced disease classification.

A notable contribution of the chapter lies in its proposal and utilization of saliency maps as a visualisation method. This innovative approach aims to provide a clearer understanding and interpretation of the CNN classification mechanism, thereby increasing the transparency of deep learning models. By offering insight into the symptoms of plant diseases, the saliency maps contribute to a more informed and interpretable decision-making process. This visualisation method bridges the gap in understanding between the model's predictions and the features that lead to those predictions.

In conclusion, this chapter not only highlights the limitations of prior studies but also provides a robust solution by adopting recent and advanced CNN architectures. The achieved accuracy of 99.76% on plant disease classification is a testament to the effectiveness of this approach. The introduction of saliency maps as a visualisation method is a groundbreaking step toward improving transparency in deep learning models. This comprehensive research contributes significantly to the advancement of plant disease detection systems, setting a new standard for accuracy and interpretability in the field.

3. **Deep learning models for plant disease detection and diagnosis By Konstantinos P. Ferentinos.**

This paper explores the application of Convolutional Neural Network (CNN) models for plant disease detection and diagnosis, employing deep learning methodologies. The research focuses on analyzing simple leaf images of both healthy and diseased

plants, utilizing a vast open database containing 87,848 images. This dataset encompasses 25 different plant species across 58 distinct classes of [plant, disease] combinations, including images of healthy plants. The objective is to develop robust CNN models capable of accurately identifying plant diseases based on leaf images.

Multiple CNN architectures are trained on this comprehensive dataset, and the best-performing model achieves an impressive success rate of 99.53% in correctly identifying the corresponding [plant, disease] combination or identifying a healthy plant. This exceptionally high success rate positions the developed model as a valuable advisory and early warning tool for plant disease detection. The success of the model suggests its potential for expansion into an integrated plant disease identification system suitable for real cultivation conditions.

The paper begins with an introduction to convolutional neural network models, highlighting their basis in artificial neural networks that mimic the principles of brain function. Emphasis is placed on the trainable nature of neural networks through supervised learning, where models are trained using existing data with specific input-output matchings. CNNs, as an evolution of neural networks, are introduced as a specialized architecture particularly effective in image-related tasks, such as plant disease detection.

The subsequent sections likely delve into the training process, detailing the dataset and the various architectures experimented with during the research.

The practical implications of the research are highlighted, with the high success rate of 99.53% positioning the CNN model as a practical advisory tool for farmers, offering timely insights into plant health. The potential expansion of the model into a real-world, integrated plant disease identification system underscores its significance in contributing to precision agriculture.

In summary, this paper presents a robust application of CNN models in the domain of plant disease detection and diagnosis. The use of a substantial dataset and the achievement of a remarkable success rate make the proposed model a promising tool for agricultural practices, demonstrating the potential for real-world implementation in cultivation conditions.

4. **Plant disease identification from individual lesions and spots using deep learning by Garcia Arnal Barbedo.**

This paper addresses the challenge of automatic identification of plant diseases using deep learning, a standard technique for image classification. The primary hurdle faced is the limited availability of image databases that comprehensively represent the diverse conditions and symptom characteristics encountered in real-world agricultural settings. While data augmentation techniques partially mitigate this limitation, they fall short in reproducing the full practical diversity. The paper proposes an innovative approach by focusing on individual lesions and spots rather than considering the entire leaf, thereby increasing data variability without the need for additional images. This approach also enables the identification of multiple diseases affecting the same leaf.

Despite the promise of this approach, a manual step involving suitable symptom segmentation is required, preventing full automation. The paper acknowledges this limitation and explores the potential of the proposed method in enhancing accuracy despite the need for manual intervention. The obtained accuracies using this approach surpass those achieved with the original images, with an average improvement of 12%. Importantly, even when considering up to 10 diseases, no crop exhibits accuracies below 75%. This implies that the proposed method effectively identifies plant diseases across a range of scenarios.

The unique contribution of this paper lies in its departure from the conventional approach of analyzing entire leaves. By focusing on individual lesions and spots, the method increases data variability, providing the deep learning model with a more nuanced understanding of plant diseases. The paper suggests that, with a sufficient amount of data, deep learning techniques prove effective for plant disease detection and recognition.

However, the manual intervention required for symptom segmentation raises practical challenges in terms of full automation. Despite this limitation, the obtained results underscore the efficacy of the proposed approach in enhancing accuracy, showcasing its potential as a valuable tool in precision agriculture. The increased accuracy, even when dealing with multiple diseases on the same leaf, positions the approach as a promising solution for real-world applications.

In conclusion, this paper pioneers a unique perspective in the application of deep learning for plant disease identification, focusing on individual lesions and spots. The results demonstrate a substantial improvement in accuracy, emphasizing the potential

of this approach in overcoming challenges associated with limited image databases. While manual intervention remains a drawback, the overall effectiveness of the proposed method signals a step forward in leveraging deep learning for practical and accurate plant disease detection.

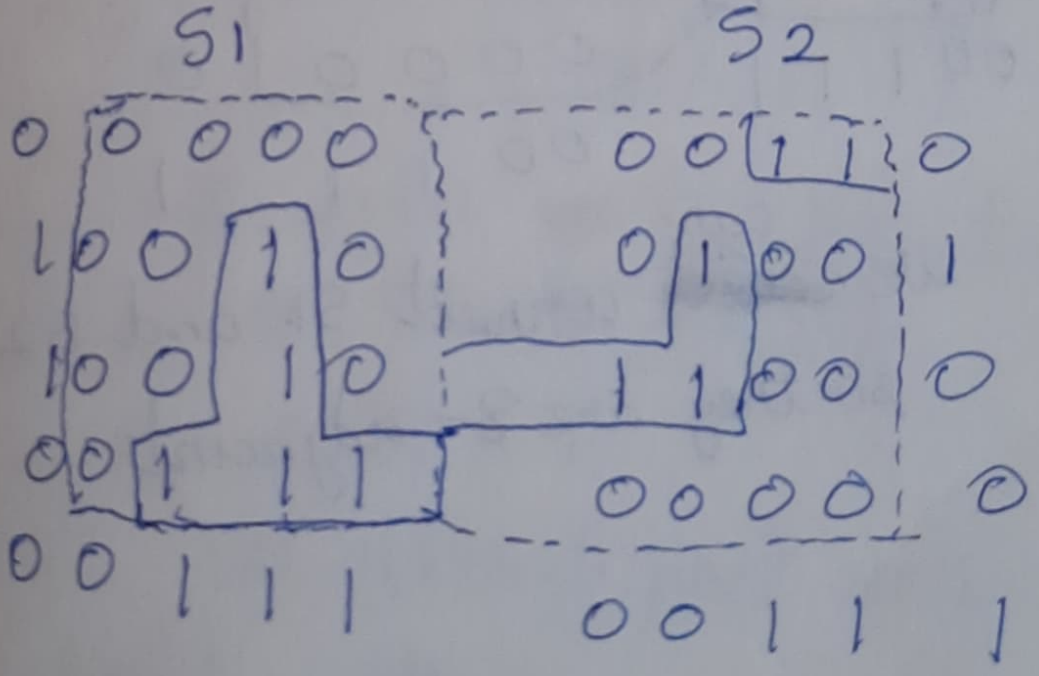
Digital Image Processing

DA - I

2.14 Given that $V = \{1\}$

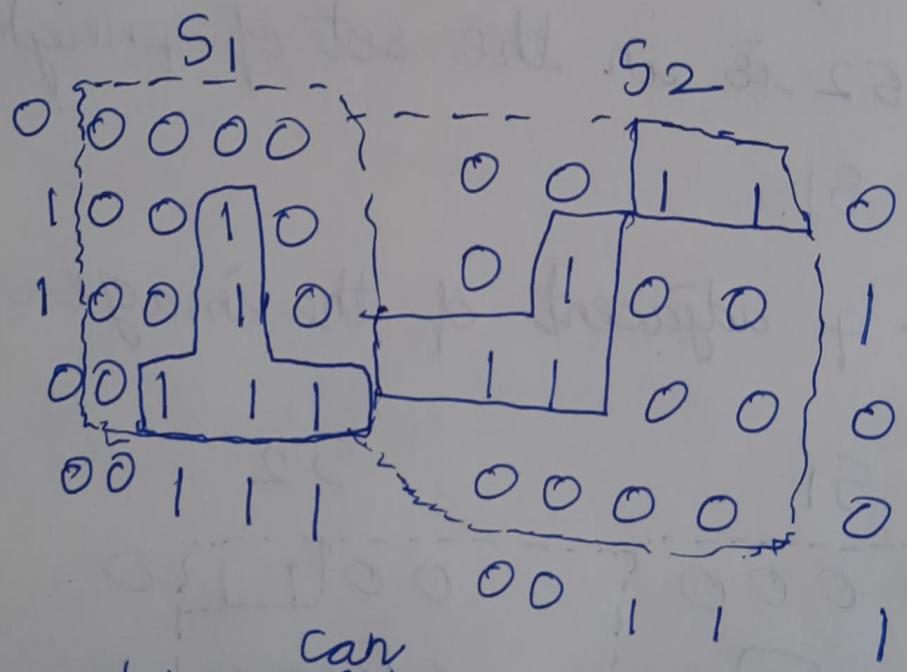
a) Two subset S1 and S2 images with values from V are 4 - adjacent if S2 is in the set of 4 neighbors of S1.

The 4 adjacent of the image is:



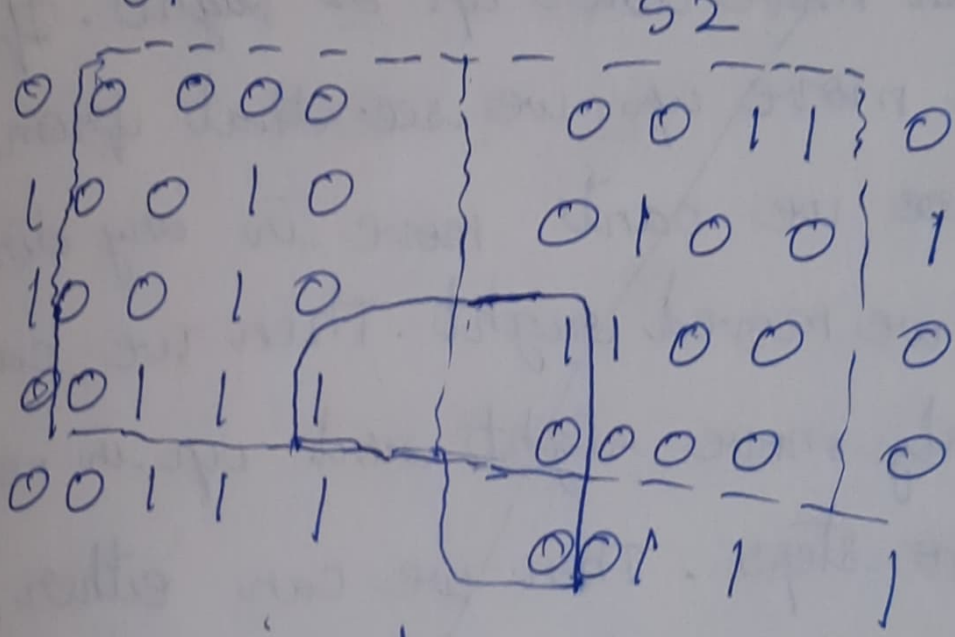
Two subsets S1 and S2 cannot be connected using 4-adjacency so they are not 4 - adjacent.

b) Two subsets S1 and S2 images with values from V are 8-adjacent if S2 is in the set of 8 neighbors of S1.



We can connect S1 and S2 so they are 8-adjacent.

c) Two subsets S1 and S2 images with values from V are m-adjacent if S2 is in the set of intersection with S1. The intersection point should not be equal to V.



The points 1 intersect at 0. check 0 with V.

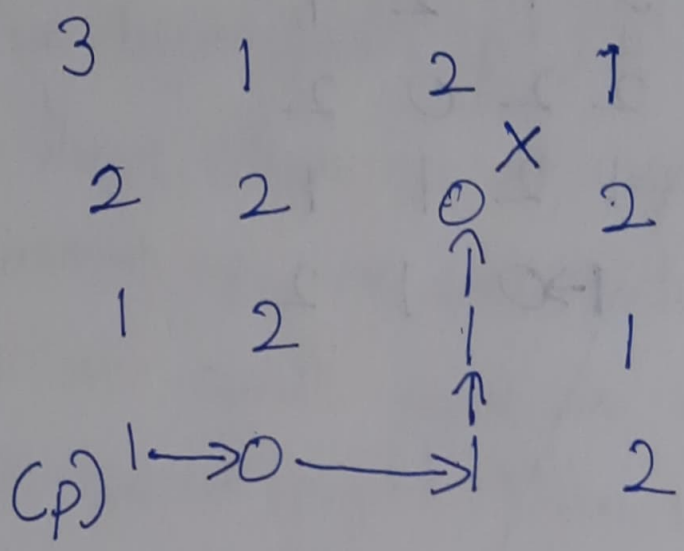
$0 \neq 1$

The intersect point doesn't match with V.

It is m-adjacent.

2.18) Given: $V = \{0, 1\}$

a) 4-path:

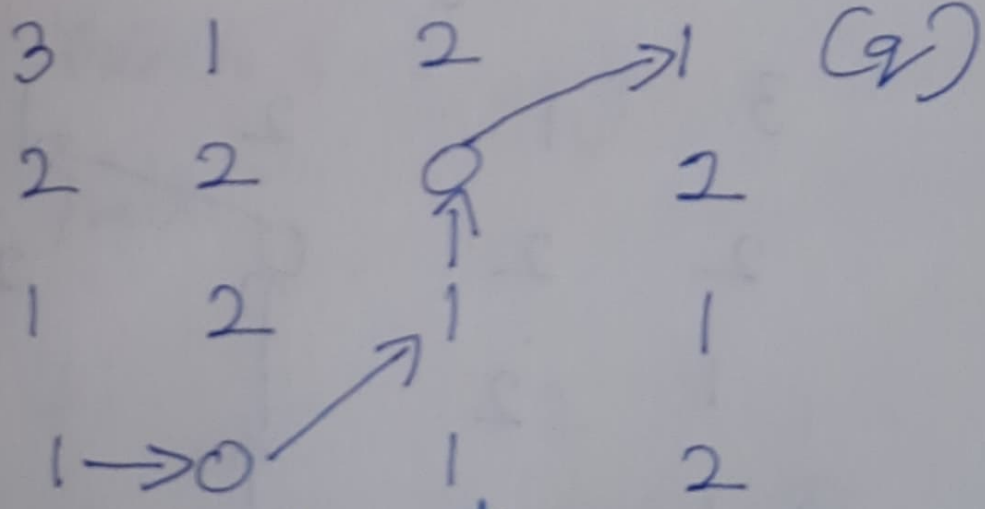


Let us begin from 'p' i.e '1' we can move either up or right. If we move up we see that from there we can't move in any direction. So we moved right. Then we can only move right and up in next two steps. Then we can either move up or right. If we move right it will be deadend for us.

So we move up. From there also we see that no path will lead us to 'q'. Hence no 4-path from p to q exist.

8-path:

Given $V = \{0, 1\}$



8-path = 4

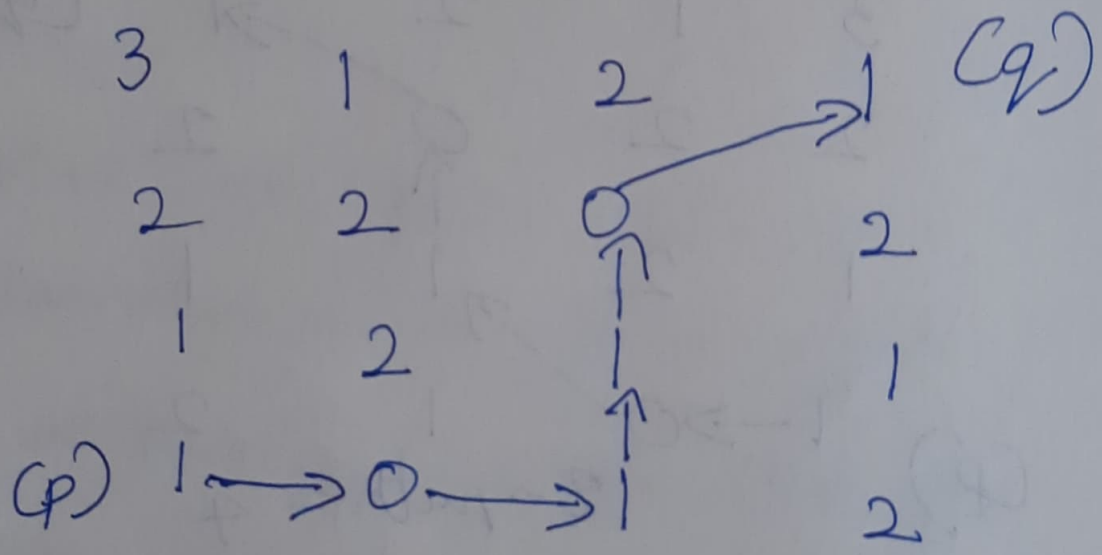
Similiar to first case we can move up or right. If we move up it will be dead end. So we moved right to '0'.

From there we can diagonally move to '1'. Then we have two

choice move to '1' or '0'. But we selected '0' to find shortest path. From '0' we can move diagonally to '1'.

m-path:

Given $V = \{0, 1\}$



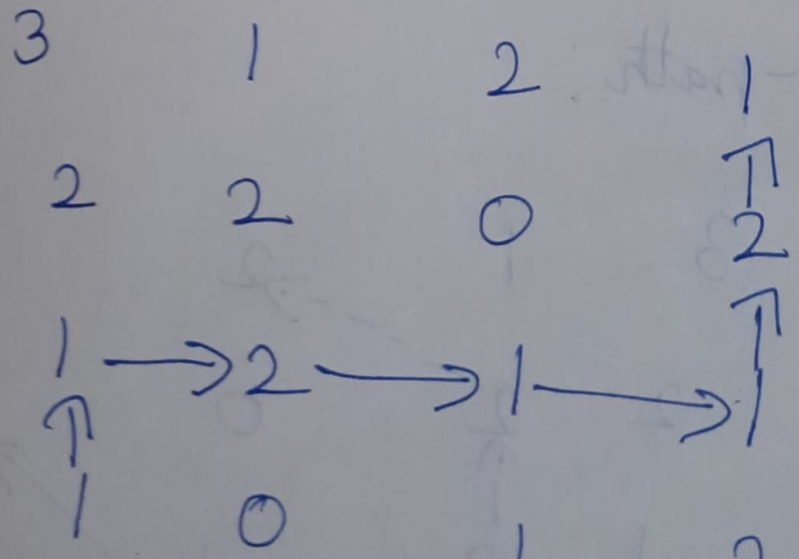
m-path = 5

Here we could move either up or right. But going up would have resulted in end of path. So we moved right to '0'.

From here we can move to right '1' but not to diagonal '1' as it is not m-adjacent to '0'. So we moved right to '1' then to '0' and then diagonally to '1' to find m-path.

b) Given $V = \{1, 2\}$

4-path:

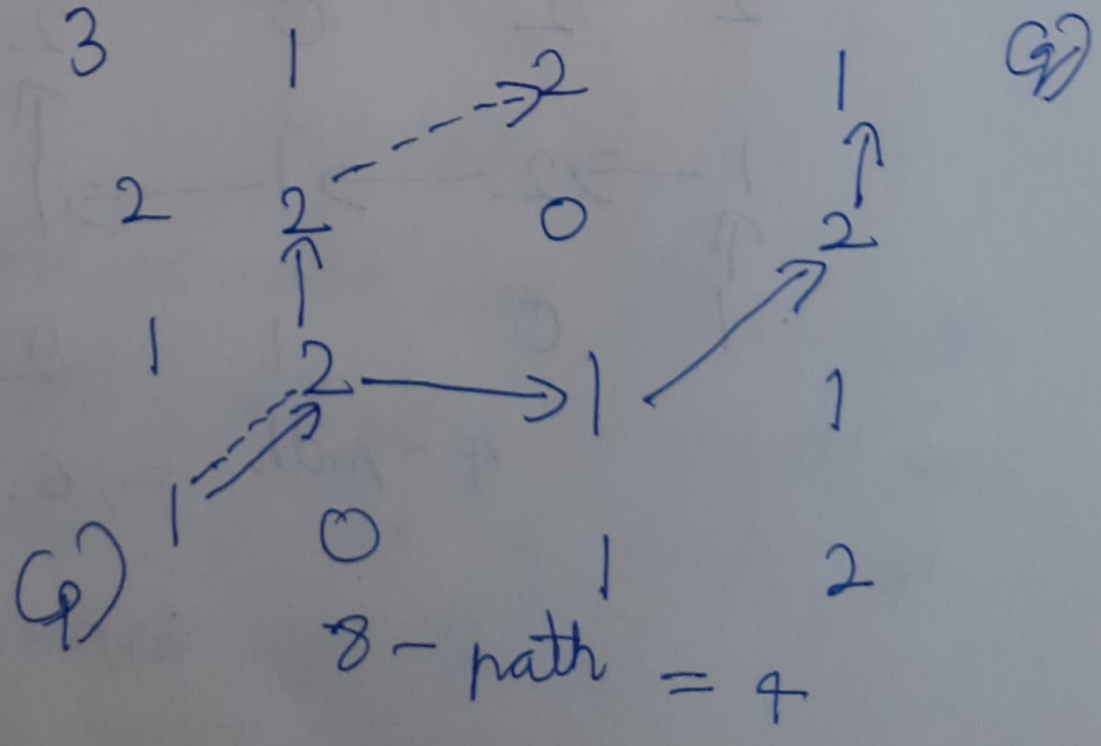


4-path = 6.

We can move up to '1' then we can move up or right to '2' then we can go to '1' then again move right to '1' then upto '2' and then upto '1' i.e q.

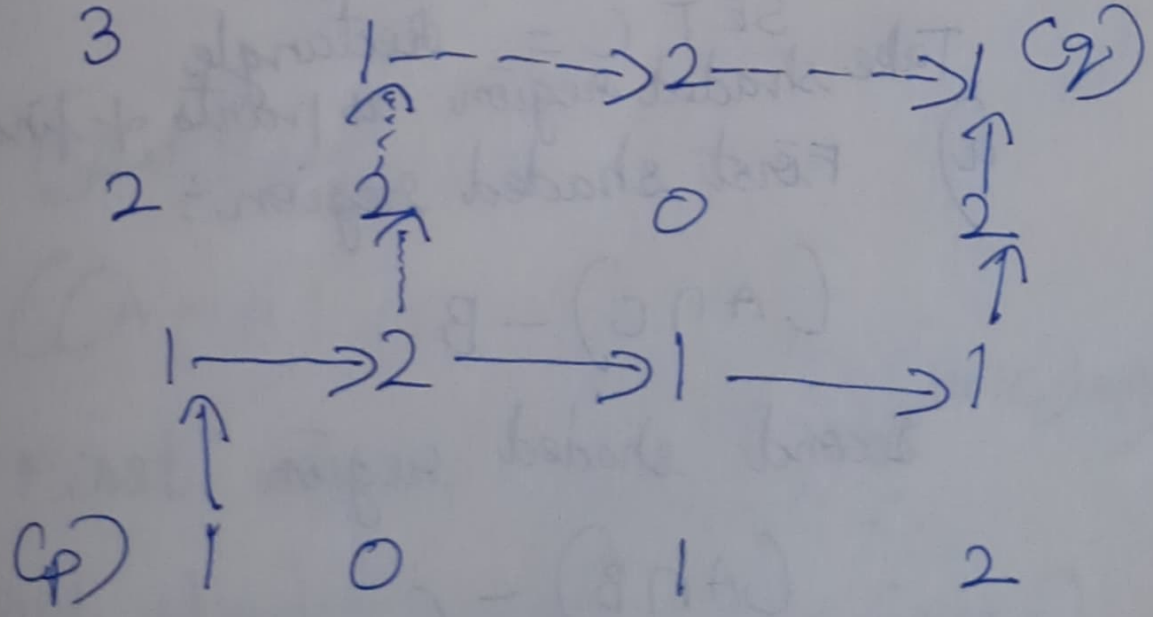
There are various paths possible in given pixels but the smallest is of size: 6.

8-path:



Here we could have gone upto '1' or diagonally to '2'. So we moved diagonally. Then from there we moved right to '1' and from that '1' we moved diagonally upward to '2' and '1' respectively. This path is of length 4 and there are several more paths of size 4 in this or shown using dashed lines.

m-path:



m-path = 6

Here we can move upto '1' and from there we can only go right to '2' then we have two paths are shown using dashed line and one using arrows to reach 'q'.

The length of this path is 6.

2.32)

Given:

SET A = Circle

SET B = Triangle

SET C = Rectangle

Take shaded region as parts & finally add it.

a) First shaded region:

$$(A \cap C) - B$$

second shaded region:

$$(A \cap B) - C$$

Adding both regions

$((A \cap C') - B) \cup ((A \cap B) - C)$

$(A \cap C - B) \cup (A \cap B - C)$

b) First shaded region :

$A - (B \cup C)$

Second shaded region :

$(B \cap C) - A$

Add both regions

$((A - (B \cup C)) \cup ((B \cap C) - A)$

~~$(A - B \cup C)$~~

$((A - B) \cap (A - C)) \cup ((B \cap C) - A)$

c) First shaded region : C

Second shaded region : $B - (A \cup C)$

Add both regions

$C \cup (B - (A \cup C))$

d) We divide shaded region into three parts

First shaded region :-

$A - (B \cup C)$ [from part b]

Second shaded region :-

$C - B$

Third shaded region :-

$B - (A \cup C)$ [from part c]

Add all three regions

$\Rightarrow ((A - (B \cup C)) \cup (C - B) \cup ((B - (A \cup C))$

$\Rightarrow ((A - B) \cap (A - C)) \cup (C - B) \cup ((B - A) \cap (B - C))$

2.36

To obtain single, composite transformation functions we need to use the matrix representation of linear transformations and multiply them in order of operations. The matrix representation of a scaling by a factor of S_x in the x direction and S_y in y direction is:

$$S = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix}$$

Matrix representation of a translation by (t_x, t_y) is

$$T = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

Matrix representation of rotation by an angle θ about origin is

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

Matrix representation of vertical shear by a factor of k is

$$V = \begin{bmatrix} 1 & k \\ 0 & 1 \end{bmatrix}$$

a) To perform scaling & translation, we need to multiply scaling matrix and translation matrix.

$$S.T = \begin{bmatrix} Sx & 0 \\ 0 & Sy \end{bmatrix} \cdot \left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} tx \\ ty \end{bmatrix} \right)$$

$$\begin{bmatrix} Sx & 0 \\ 0 & Sy \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} Sx & 0 \\ 0 & Sy \end{bmatrix} \begin{bmatrix} tx \\ ty \end{bmatrix}$$

$$\begin{bmatrix} Sx\,x \\ Sy\,y \end{bmatrix} + \begin{bmatrix} Sx\,tx \\ Sy\,ty \end{bmatrix}$$

2.36

b) To perform scaling, translation and rotation we need to multiply scaling matrix, translation matrix and rotation matrix.

The single composite transformation function is:

$$S.T.R = \begin{bmatrix} sx & 0 \\ 0 & sy \end{bmatrix} \cdot \left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} tx \\ ty \end{bmatrix} \right) \cdot \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

$$= \left(\begin{bmatrix} sx\,x \\ sy\,y \end{bmatrix} + \begin{bmatrix} sx\,tx \\ sy\,ty \end{bmatrix} \right) \cdot \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

$$\begin{bmatrix} sx\,x\cos\theta - sy\,y\sin\theta + sx\,tx\cos\theta - sy\,ty\sin\theta \\ sx\,x\sin\theta + sy\,y\cos\theta + sx\,tx\sin\theta + sy\,ty\cos\theta \end{bmatrix}$$

(c) To perform vertical shear, scaling, translation and rotation we need to multiply the vertical shear matrix, scaling matrix, translation matrix and rotation matrix.

The single composite transformation function is

$$V.S.T.R = \begin{bmatrix} 1 & k \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} Sx & 0 \\ 0 & Sy \end{bmatrix} \cdot \left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} tx \\ ty \end{bmatrix} \right)$$

$$\cdot \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

$$= \begin{bmatrix} Sx & kSy \\ 0 & Sy \end{bmatrix} \cdot \left(\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} tx \\ ty \end{bmatrix} \right) \cdot$$

$$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

$$\left(\begin{bmatrix} s_x x + k s_y y \\ s_y y \end{bmatrix} + \begin{bmatrix} s_x t_x + k s_y t_y \\ s_y t_y \end{bmatrix} \right) \cdot \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

$$\begin{bmatrix} (s_x x + k s_y y)\cos\theta - s_y y \sin\theta + (s_x t_x + k s_y t_y)\cos\theta - s_y t_y \sin\theta \\ (s_x x + k s_y y)\sin\theta + s_y y \cos\theta + (s_x t_x + k s_y t_y)\sin\theta + s_y t_y \cos\theta \end{bmatrix}$$

d) Yes the order of multiplication of individual matrices to produce a single transformation does make a difference because matrix

multiplication is not cumulative in general i.e, $AB \neq BA$ for most matrices A and B.

Let $S = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$ and $T = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ be the scaling & translation matrices

$$S.T = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \cdot \left(\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right)$$

$$= \begin{bmatrix} 2x+2 \\ 2y+2 \end{bmatrix}$$

which means scaling image by a factor of 2 and then translating it by (1,1).

However $T.S = \left(\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right) \cdot \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$

$= \begin{bmatrix} 2x+2 \\ 2y+2 \end{bmatrix}$ translating image by (1, 1) and then scaling it by a factor of 2. These two operations are not equivalent as they produce different results.

2.37)

a) Inverse scaling transformation

$$A_S = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix}$$

Inverse scaling matrix A_S^{-1} is matrix undergoes scaling operation

a) Scaling involves multiplying by scaling factors a_{11} and a_{22} the inverse factors are $\frac{1}{a_{11}}$ and $\frac{1}{a_{22}}$.

$$A_S^{-1} = \begin{bmatrix} \frac{1}{a_{11}} & \frac{1}{a_{12}} & -\frac{a_{13}}{a_{11}} \\ \frac{1}{a_{21}} & \frac{1}{a_{22}} & -\frac{a_{23}}{a_{22}} \\ 0 & 0 & 1 \end{bmatrix}$$

b) Inverse translation transformation we need to negate translation values.

Translation matrix $A_t = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$

Inverse translation : simply negate translation values.

$$A_t^{-1} = \begin{bmatrix} 1 & 0 & -t_x \\ 0 & 1 & -t_y \\ 0 & 0 & 1 \end{bmatrix}$$

c) To find inverse vertical and horizontal shearing transformations we need to use inverse of shearing matrices.

$$A_{sh} = \begin{bmatrix} 1 & a & 0 \\ b & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

For horizontal shearing the inverse shearing factor is $-a$ and for the vertical shearing the inverse shearing factor is $-b$.

$$A_{shn} = \begin{bmatrix} 1 & -a & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$A_{shn}^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ -b & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

d) To find inverse rotation transformation we need to use inverse of rotation matrix.

$$A_{rot} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$A_{rot}^{-1} = \begin{bmatrix} \cos(-\theta) & -\sin(-\theta) & 0 \\ \sin(-\theta) & \cos(-\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

e) To show composite inverse translation / rotation transformation we multiply inverse matrices of translation and rotation. Let's denote the translation matrix as T and rotation matrix as R. Then composite inverse transformation matrix T^{-1} R^{-1} is

$$T^{-1}R^{-1} = \begin{bmatrix} 1 & 0 & -t_x \\ 0 & 1 & -t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

3.4 b) A bit plane is a way to visualize the contribution of each bit to the overall intensity of the image. For a 4-bit image, there are 4 bit planes each representing one of the four bits.

Given 4 bit image:

1	0	1	8	6
2	2	2	1	1
3	1	15	14	12
4	3	6	9	10

binary representation of these pixels:

1	0000	0001	1000	0110
2	0010	0010	0001	0001
3	0001	1111	1110	1100
4	0011	0110	1001	1010

Now we'll extract bit planes one by one:

Bit Plane 0 (least significant Bit):

1	0000	0000	1000	0110
2	0010	0010	0000	0000
3	0000	1111	1110	1100
4	0011	0100	1000	1000

Bit Plane 1:

1	0000	0000	0000	0000
2	0000	0000	0000	0000
3	0000	0000	0000	0100
4	0000	0000	0000	0000

Bit Plane 2:

1	0000	0000	0000	0010
2	0000	0000	0000	0000
3	0000	0000	0000	0000
4	0000	0000	0000	0010

Bit Plane 3 (Most Significant Bit):

1	0000	0000	0000	0100
2	0000	0000	0000	0000
3	0000	0000	0000	0000
4	0000	0000	0000	0000

These are the four bit planes of given 4-bit image.

3.11) Given PDF:

$$Pr(r) = \frac{2r}{(L-1)^2} \quad \text{for } 0 \le r \le L-1$$

$$Pr(r) = 0 \quad \text{otherwise.}$$

a) Transformation function for histogram equalization

First calculate the cumulative distribution function (CDF)

$$F(r) = \int_0^r Pr(r')\, dr'$$

$$\int_0^r \frac{2r'}{(L-1)^2}\, dr'$$

$$\frac{r'^2}{(L-1)^2}\Bigg|_0^r$$

$$\frac{r^2}{(L-1)^2}$$

Normalize the CDF to span the full intensity range $[0,1]$:

$$G_1(r) = \frac{F(r) - F(0)}{1 - F(0)}$$

$$= \frac{\frac{r^2}{(L-1)^2} - 0}{1 - 0}$$

$$= \frac{r^2}{(L-1)^2}.$$

So transformation function for histogram equalization is $s = G_1(r) = \frac{r^2}{(L-1)^2}$.

b) To Find transformation function for desired intensity PDF:

Given PDF:

$$P_z(z) = \frac{3z^2}{(L-1)^3} \text{ for } 0 \leq z \leq L-1$$

$$P_z(z) = 0 \text{ otherwise.}$$

Transformation function will be the inverse of CDF corresponding to PDF.

calculate cumulative distribution function CDF

$$H(z) = \int_0^z P_z(z')\, dz'$$

$$\int_0^z \frac{3z'^2}{(L-1)^3}\, dz'$$

$$\frac{z'^3}{(L-1)^3}\Bigg|_0^z$$

$$\frac{z^3}{(L-1)^3}$$

Normalise the CDF:

$$J(z) = \frac{H(z) - H(0)}{1 - H(0)}$$

$$\frac{\frac{z^3}{(1-1)^3} - 0}{1 - 0}$$

$$= \frac{z^3}{(1-1)^3}$$

Transformation function of desired intensity PDF is $r = \frac{z^3}{(1-1)^3} = J(z)$

c) Express transformation function from (b) directly in terms of r

$$r = J(z) = \frac{z^3}{(1-1)^3}$$

Express z in terms of x.

$$z^3 = x(1-i)^3$$

$$z = \sqrt[3]{x(1-i)^3}$$

Transformation function from b directly expressed in terms of x

$$z = 3\sqrt{x(1-i)^3}$$

3:18) a)

$$w = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

$$f = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Given that the kernel is centred at point (2, 3) of the image. The ellipse encircles the area where the kernel overlaps with the image. So the ellipse will cover the area where the kernel elements w interact with corresponding elements of image f.

The values of w & f within this area:

$$w = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

$$f = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

b) Compute the convolution w*f at point (2,3) of f. We perform element-wise multiplication b/w kernel and corresponding region of image followed by summation to get result.

$$w * f = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} * \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

$$(1\times0) + (2\times1) + (1\times0) + (2\times0) + (4\times1) + (2\times0) + (1\times0) + (2\times1) + (1\times0)$$

$$= 0+2+0+0+4+0+0+2+0$$

$$= 8$$

The final convolution ~~pair~~ result at point (2, 3) is 8.

c) correlation of w * f. follow the same procedure as in (b) without flipping the kernel:

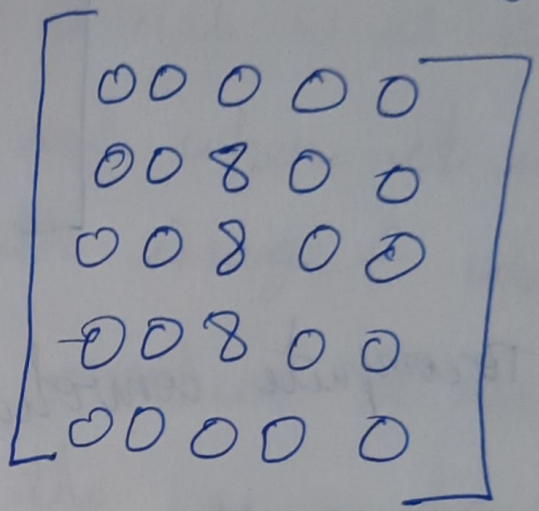
$$w * f = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} * \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

$$= (1\times0) + (2\times1) + (1\times0) + (2\times0) + (4\times1) + (2\times0) + (1\times0) + (2\times1) + (1\times0)$$

$$= 0 + 2 + 0 + 0 + 4 + 0 + 0 + 2 + 0$$

$$= 8$$

correlation result b/w kernel & image

correlation result = 

3.21

a) Compute convolution of given kernel w and image f. Center the kernel at each point of image and perform element wise multiplication followed by summation.

kernel $w = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$

image f =

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

To compute convolution, we slide kernel w over the image f and perform element-wise multiplication and summation at each position

convolution (w, f) =

$$\begin{bmatrix} 10 & 20 & 20 & 20 & 10 \\ 20 & 40 & 40 & 40 & 20 \\ 20 & 40 & 40 & 40 & 20 \\ 20 & 40 & 40 & 40 & 20 \\ 10 & 20 & 20 & 20 & 10 \end{bmatrix}$$

b) Yes the result of convolution has bias. The kernel w is a symmetric matrix and when it is convolving with image f which consists of ones produce a matrix which decrease the value towards edges is a bias introduced by kernel.

3.12) First obtain histogram equalization transformation :

$$s = T(r) = \int_0^r p_r(w)\,dw$$

$$= \int_0^r (-2w+2)\,dw$$

$$= -r^2 + 2r$$

Next,

$$v = G(z) = \int_0^z p_z(w)\,dw$$

$$= \int_0^z 2w\,dw$$

$$= z^2$$

$$z = G^{-1}(v) = \pm\sqrt{v}$$

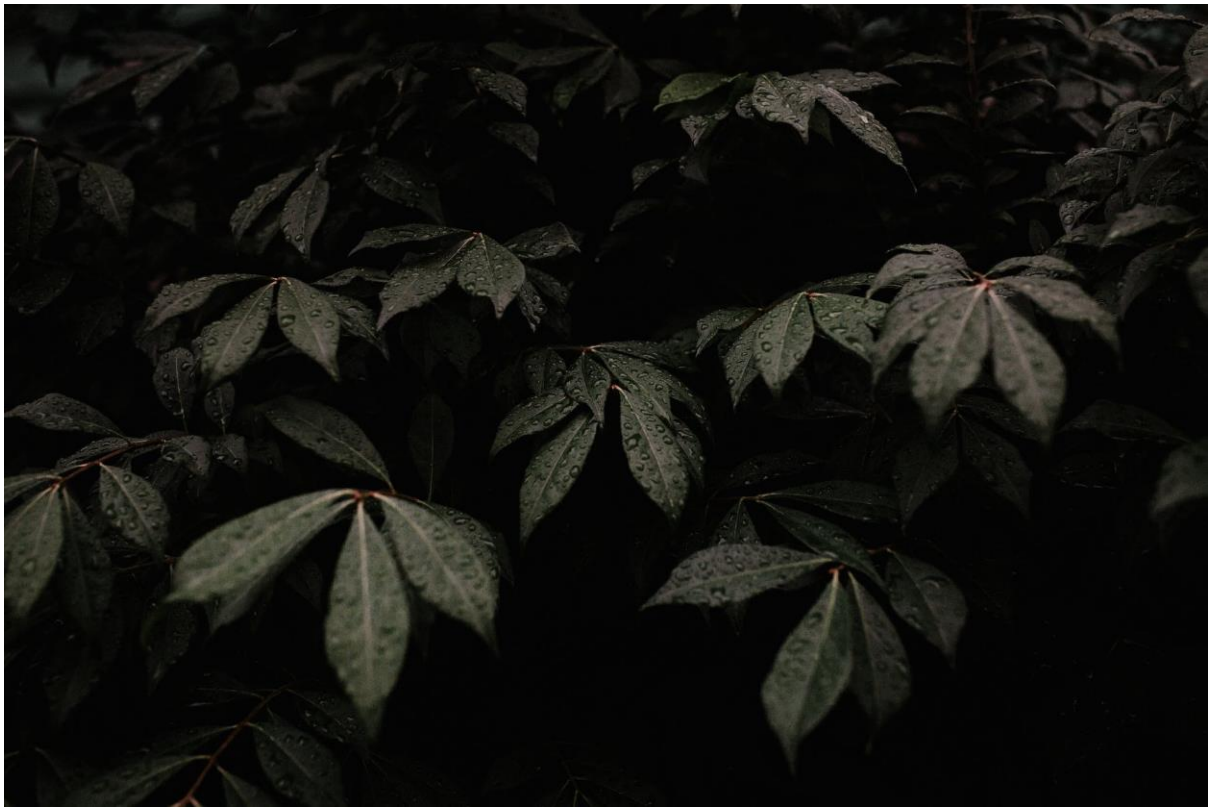
But only positive gray levels are allowed so $z = \sqrt{v}$. Then we replace v with s which in turn is $-x^2 + 2x$

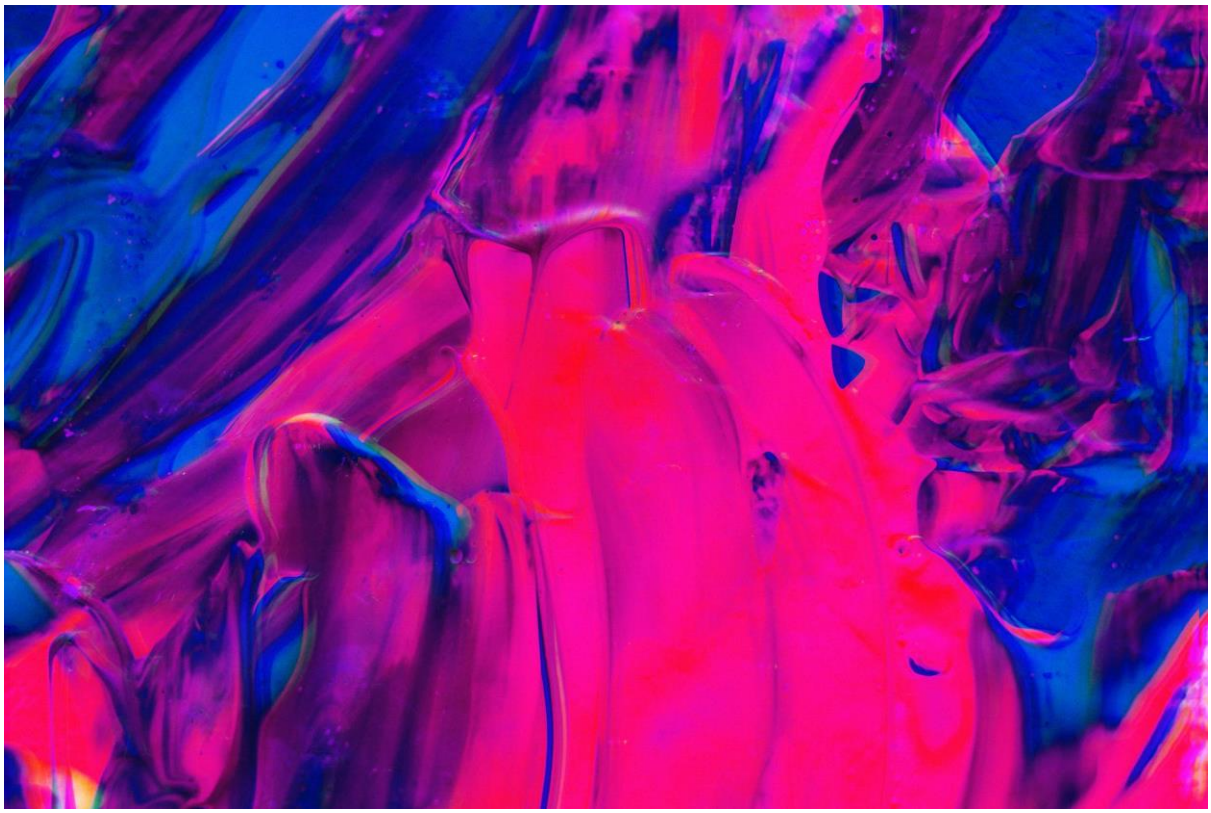
and we have

$$z = \sqrt{-x^2 + 2x}$$

Digital Image Processing

Original Images:





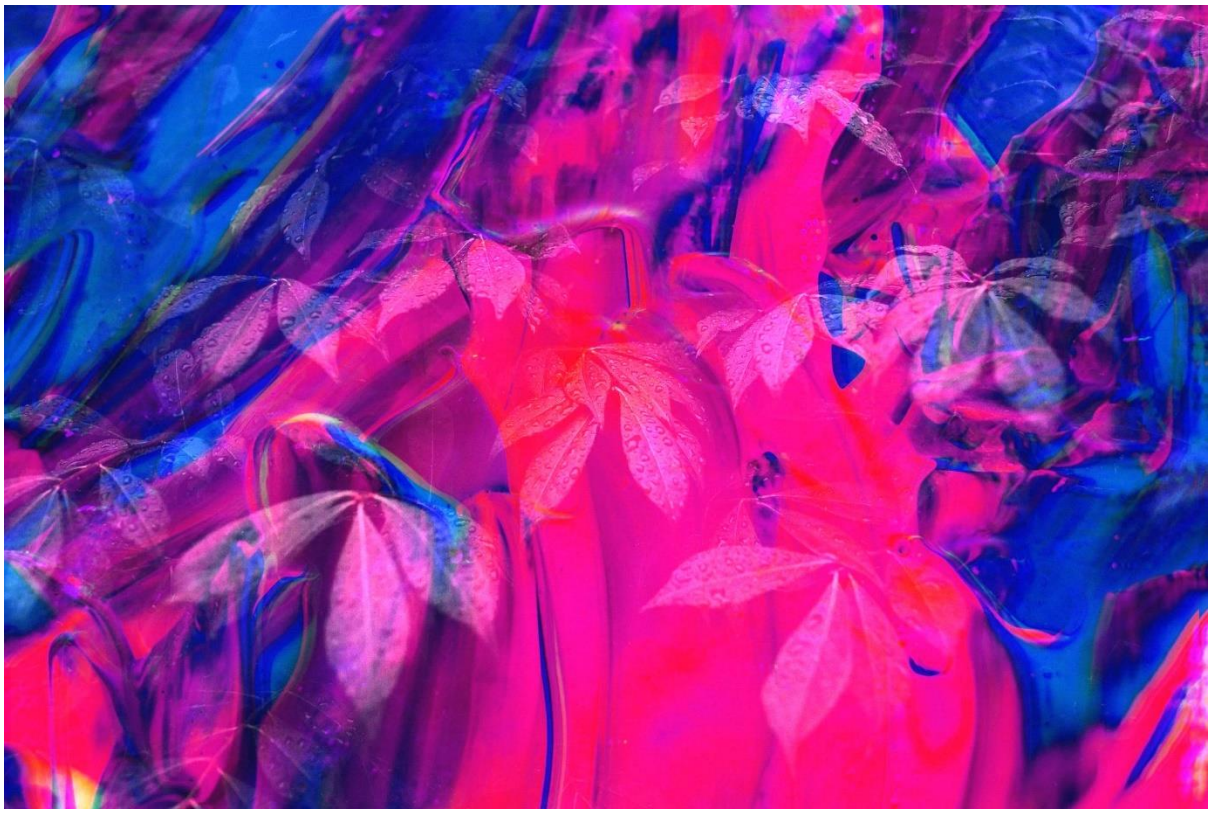
1) Arithmetic Operations on Images:

Addition:

Code:

```
import cv2

image1 = cv2.imread('DIP/image1.jpg')
image2 = cv2.imread('DIP/image2.jpg')

added_image = cv2.add(image1, image2)

cv2.imwrite('added_image.jpg', added_image)

cv2.imshow('Added Image', added_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Screenshots:

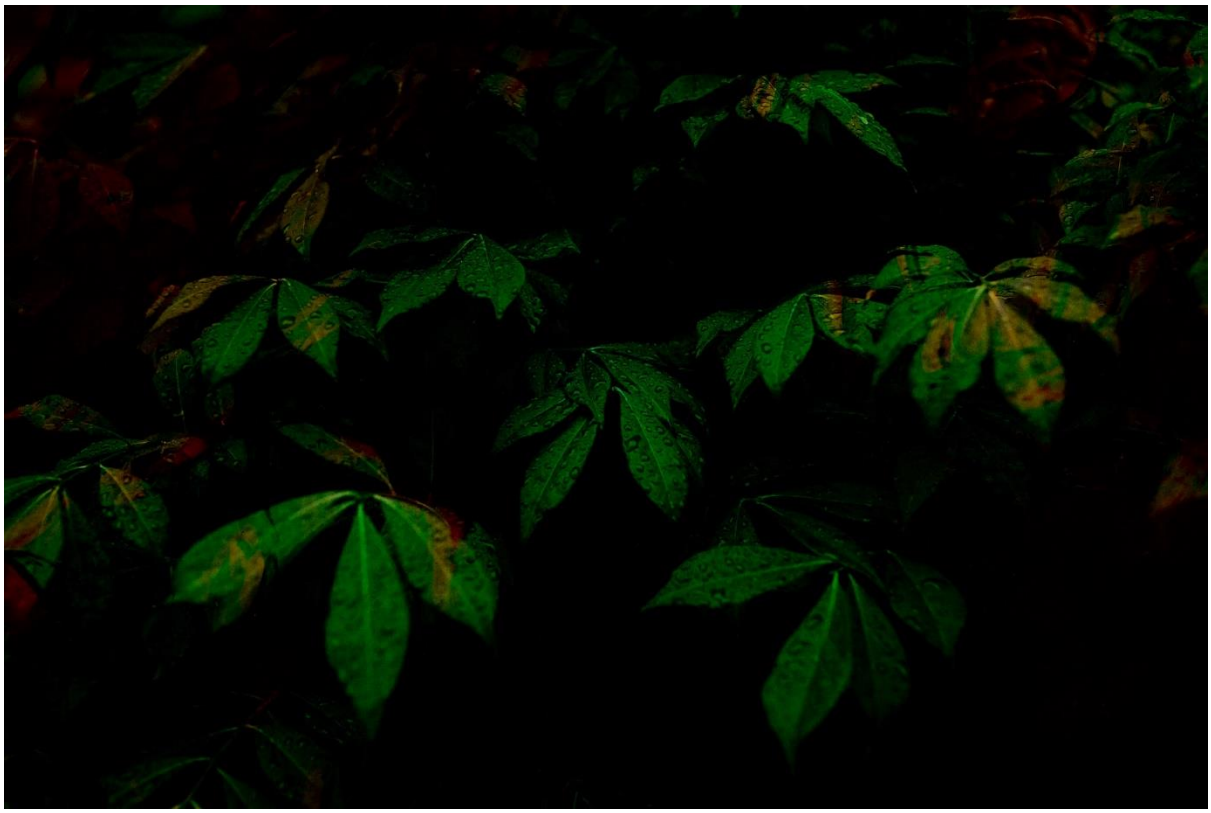


Subtraction:

Code:

```
import cv2

image1 = cv2.imread('DIP/image1.jpg')
image2 = cv2.imread('DIP/image2.jpg')

subtracted_image = cv2.subtract(image1, image2)

cv2.imwrite('subtracted_image.jpg', subtracted_image)

cv2.imshow('Subtracted Image', subtracted_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Screenshots:



Multiplication:

Code:

```
import cv2

image1 = cv2.imread('DIP/image1.jpg')
image2 = cv2.imread('DIP/image2.jpg')

multiplied_image = cv2.multiply(image1, image2)

cv2.imwrite('multiplied_image.jpg', multiplied_image)

cv2.imshow('Multiplied image', multiplied_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Screenshots:

Division:

Code:

```
import cv2

image1 = cv2.imread('DIP/image1.jpg')
image2 = cv2.imread('DIP/image2.jpg')

divided_image = cv2.divide(image1, image2)

cv2.imwrite('divided_image.jpg', divided_image)

cv2.imshow('Divided Image', divided_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Screenshots:

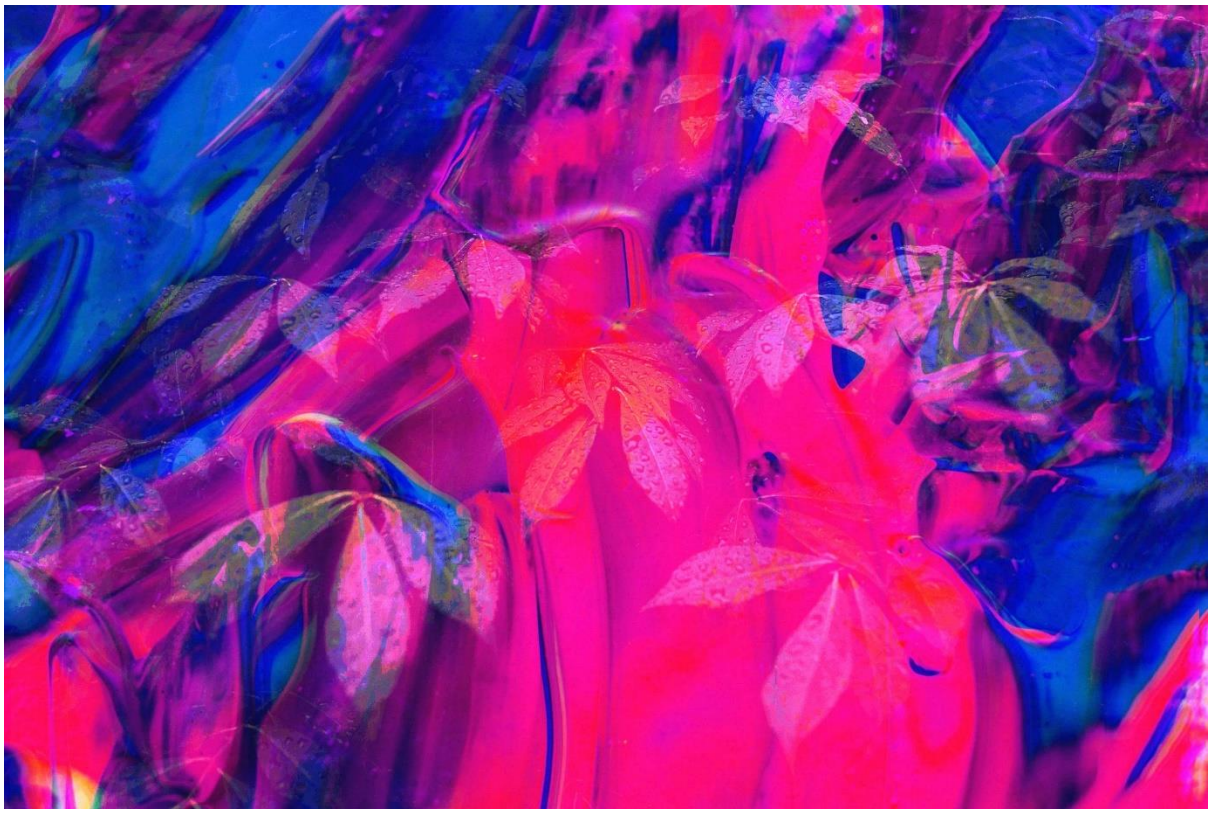
2) Set operations on images:

Union:

Code:

```
import cv2

image1 = cv2.imread('DIP/image1.jpg')
image2 = cv2.imread('DIP/image2.jpg')

union_image = cv2.bitwise_or(image1, image2)

cv2.imwrite('union_image.jpg', union_image)

cv2.imshow('Union Image', union_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Screenshots:



Intersection:

Code:

```
import cv2

image1 = cv2.imread('DIP/image1.jpg')
image2 = cv2.imread('DIP/image2.jpg')

intersection_image = cv2.bitwise_and(image1, image2)

cv2.imwrite('intersection_image.jpg', intersection_image)

cv2.imshow('Intersection Image', intersection_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

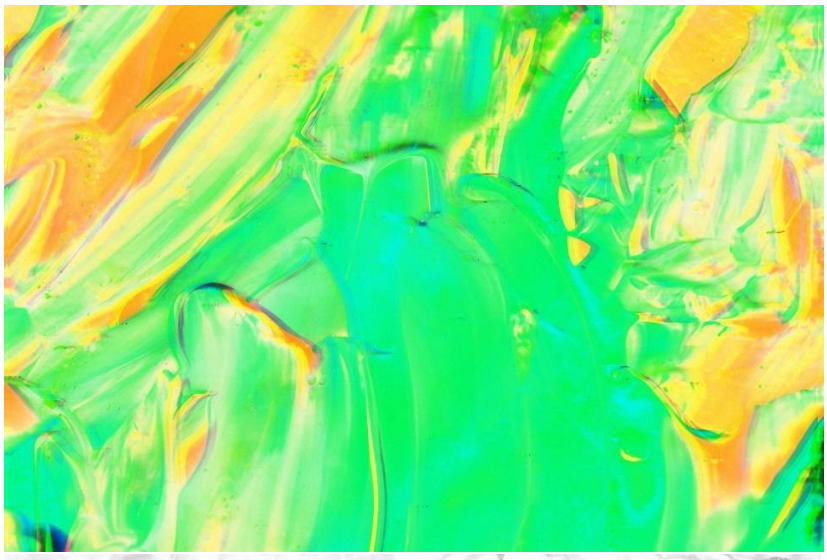
Screenshots:

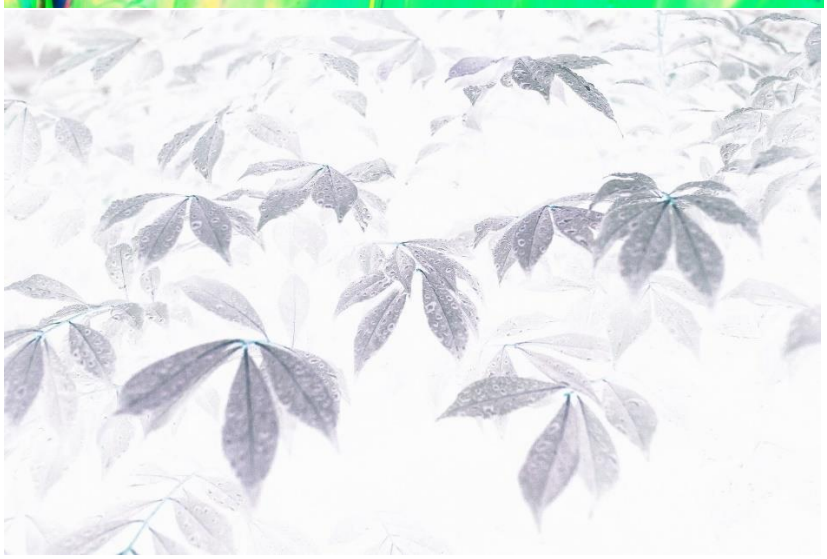
Negation:

Code:

```
import cv2

image = cv2.imread('DIP/image.jpg')

negated_image = cv2.bitwise_not(image)

cv2.imwrite('negated_image.jpg', negated_image)

cv2.imshow('Negated Image', negated_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Screenshots:



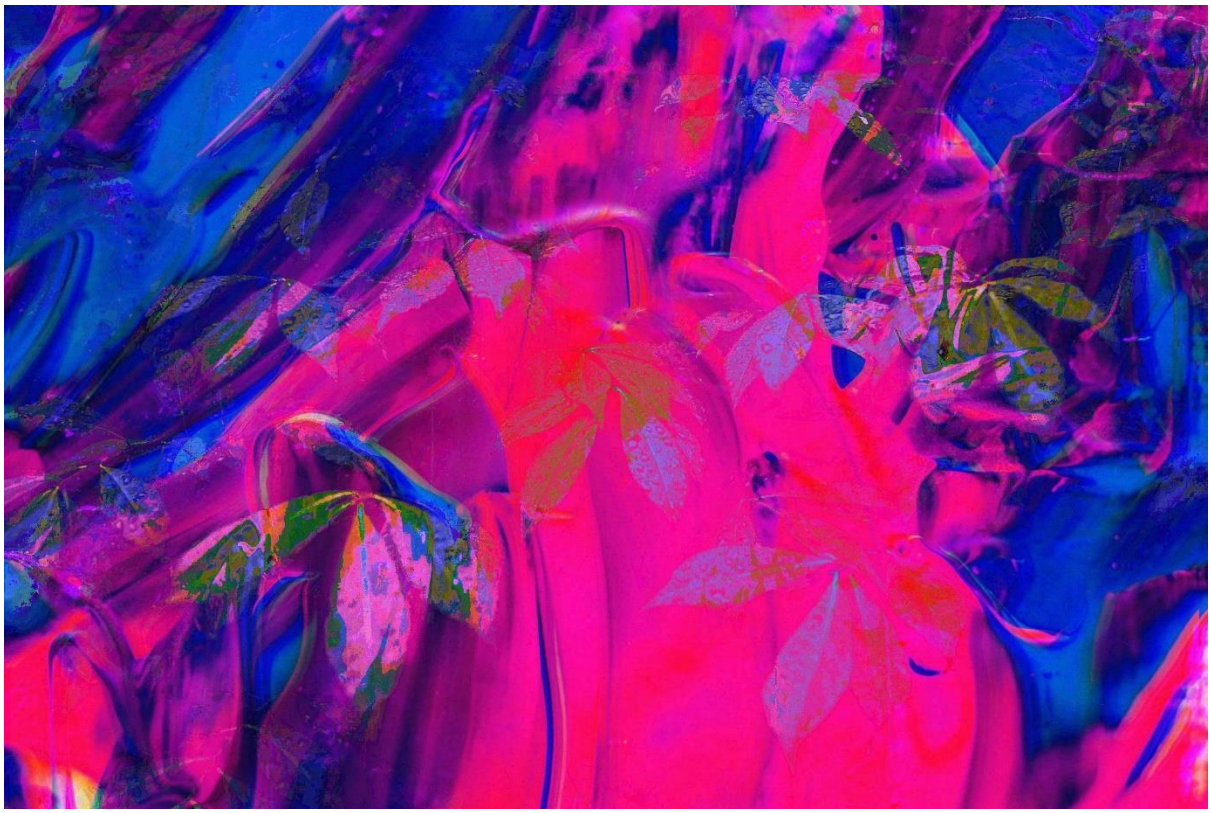


XOR:

Code:

```
import cv2

image1 = cv2.imread('DIP/image1.jpg')
image2 = cv2.imread('DIP/image2.jpg')

xor_image = cv2.bitwise_xor(image1, image2)

cv2.imwrite('xor_image.jpg', xor_image)

cv2.imshow('XOR Image', xor_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Screenshots:



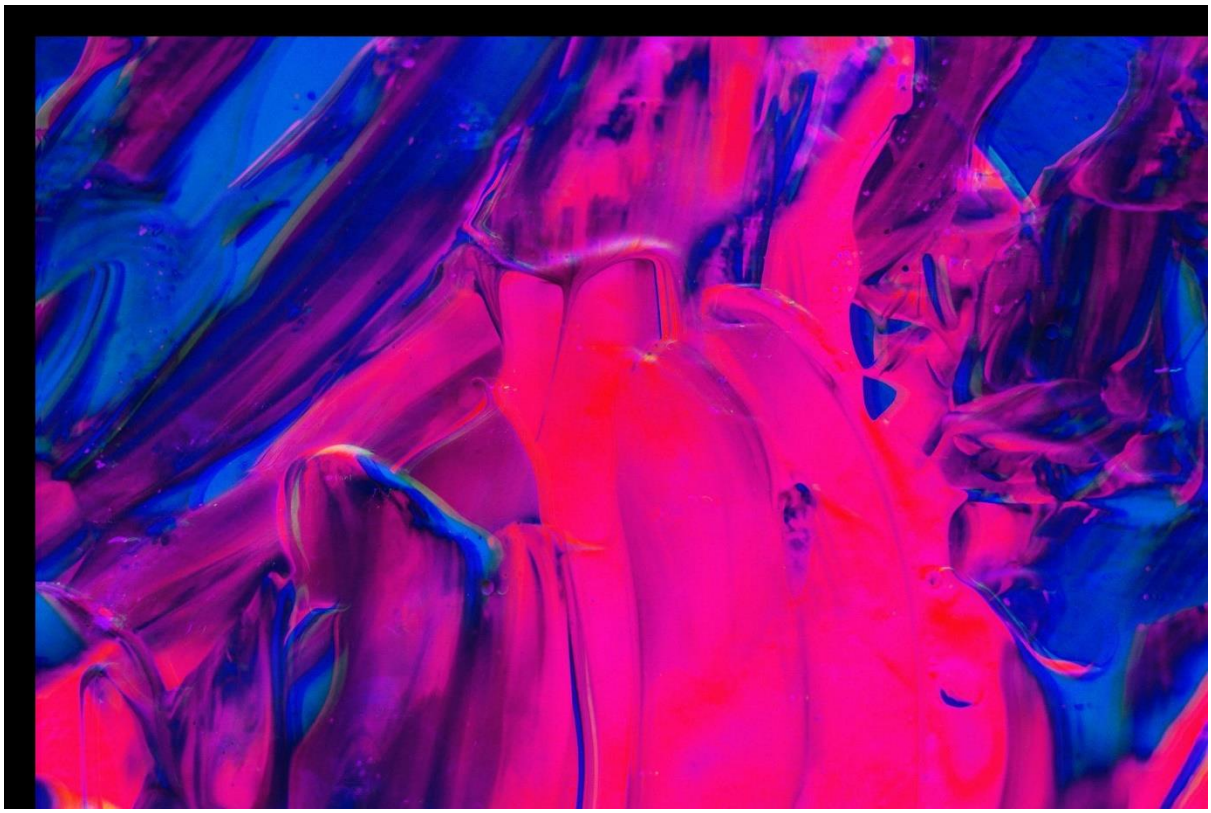
3) Transformation Operations:

Translation:

Code:

```
import cv2
import numpy as np

image = cv2.imread('DIP/image2.jpg')

tx, ty = 50, 50
M = np.float32([[1, 0, tx], [0, 1, ty]])

translated_image = cv2.warpAffine(image, M, (image.shape[1], image.shape[0]))

cv2.imwrite('translated_image.jpg', translated_image)

cv2.imshow('Translated Image', translated_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Screenshots:



Rotation:

Code:

```
import cv2
import numpy as np

image = cv2.imread('DIP/image2.jpg')

(h, w) = image.shape[:2]

center = (w / 2, h / 2)

angle = 45
scale = 1.0
M = cv2.getRotationMatrix2D(center, angle, scale)

rotated_image = cv2.warpAffine(image, M, (w, h))

cv2.imwrite('rotated_image.jpg', rotated_image)

cv2.imshow('Rotated Image', rotated_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Screenshots:

Shear:

Code:

```
import cv2
import numpy as np

image = cv2.imread('DIP/image2.jpg')

shear = 0.3
M = np.float32([[1, shear, 0], [0, 1, 0]])

rows, cols = image.shape[:2]
sheared_image = cv2.warpAffine(image, M, (cols, rows))

cv2.imwrite('sheared_image.jpg', sheared_image)

cv2.imshow('Sheared Image', sheared_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Screenshots:

4) Histogram Equilization:

Code:

```
import cv2

image = cv2.imread('DIP/image1.jpg', cv2.IMREAD_GRAYSCALE)

equalized_image = cv2.equalizeHist(image)

cv2.imwrite('equalized_image.jpg', equalized_image)

cv2.imshow('Equalized Image', equalized_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Screenshots: