



Assignment of bachelor's thesis

Title:	Automatic recognition of playing cards
Student:	Matěj Putík
Supervisor:	Ing. Jan Glaser
Study program:	Informatics
Branch / specialization:	Artificial Intelligence 2021
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2023/2024

Instructions

The recognition of playing cards from a video is a challenging task that has gained significant attention in recent years. This thesis aims to provide an in-depth analysis of the current state-of-the-art methods for the automatic recognition of playing cards in video streams.

The focus will be on developing and evaluating an algorithm that can effectively and efficiently recognize the playing cards in real-time video. The main objectives of the thesis are to explore the use of computer vision techniques, machine learning algorithms, and deep learning models to accurately detect and recognize playing cards in the video. The results of this research will have important implications for a wide range of applications, including gaming, security and surveillance, and others.

- 1) survey the topic and existing solutions
- 2) create the camera apparatus for scanning the cards
- 3) based on the previous survey, design a solution of automatic playing card recognition using computer vision, and implement it. Therefore, the solution shall be able to locate a playing card in the image or the video feed and recognize the card's color (red, black), suit (hearts, spades, clubs, diamonds) and the rank of the card (A, 2, 3 ... K).
- 4) in case of using a machine learning model as a part of the solution create a training dataset
- 5) implement the solution in the python programming language, and use the jupyter notebook environment
- 6) design and perform experiments when conditions are not ideal for the recognition of



playing cards, such as poor lighting, occlusions, or camera angles.

Choose an appropriate metric for evaluating these experiments.

7) discuss the results of these experiments

8) publish the solution along with the created dataset

References:

- ZHENG, Chunhui; GREEN, Richard. Playing card recognition using rotational invariant template matching. In: Proc. of Image and Vision Computing New Zealand. 2007. p. 276-281.
- PIMENTEL, Joao; BERNARDINO, Alexandre. A Comparison of Methods for Detection and Recognition of Playing Cards. Instituto de Sistemas e Robotica, Instituto Superior Tecnico.
- HU, Xuewen, et al. Poker card recognition with computer vision methods. In: 2021 IEEE International Conference on Electronic Technology, Communication and Information (ICETCI). IEEE, 2021. p. 11-15.
- MARTINS, Paulo; REIS, Luís Paulo; TEÓFILO, Luís. Poker vision: Playing cards and chips identification based on image processing. In: Pattern Recognition and Image Analysis: 5th Iberian Conference, IbPRIA 2011, Las Palmas de Gran Canaria, Spain, June 8-10, 2011. Proceedings 5. Springer Berlin Heidelberg, 2011. p. 436-443.

Bachelor's thesis

AUTOMATIC RECOGNITION OF PLAYING CARDS

Matěj Putík

Faculty of Information Technology
Department of Applied Mathematics
Supervisor: Ing. Jan Glaser
May 11, 2023

Czech Technical University in Prague

Faculty of Information Technology

© 2023 Matěj Putík. All rights reserved.

This thesis is school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).

Citation of this thesis: Putík Matěj. *Automatic recognition of playing cards*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2023.

Contents

Acknowledgments	viii
Declaration	ix
Abstract	x
Introduction	1
Objectives	3
1 Survey	5
1.1 Playing Card Recognition system	5
1.2 Current Solutions	6
2 Theoretical background	11
2.1 Color Spaces	11
2.2 Image Formats	15
2.3 Image Filtering	15
2.3.1 Gaussian Filter	16
2.3.2 Bilateral Filter	17
2.4 Image Processing	18
2.4.1 Gamma Correction	18
2.4.2 Contour and Edge Detection	19
2.4.3 Canny Edge Detection	20
2.4.4 Contour Shape Analysis	21
2.4.5 Recognizing Basic Shapes	23
2.4.6 Thresholding	24
2.4.7 Morphological Operations	25
2.5 Image Similarity	27
2.6 Image Augmentation	29
2.7 Clustering Algorithms	31
2.8 Convolutional Neural Networks	34
2.8.1 Convolutional Layers with Strides and Padding	34
2.8.2 Activation Layers	35
2.8.3 Pooling Layers	36
2.8.4 Fully Connected Layers	36
2.8.5 Training CNNs	37
2.8.6 Regularization Techniques	37
2.8.7 Transfer Learning	37
2.8.8 Popular CNN Architectures	38
2.9 Measuring Model Performance	40
2.9.1 Accuracy	40
2.9.2 Precision and Recall	40
2.9.3 F1 Score	41

2.9.4	Confusion Matrix	41
2.9.5	Mean Average Precision and Average Precision	41
3	Analysis	43
3.1	Physical Deck of Cards	43
3.1.1	Cards Design and Dimensions	43
3.2	Camera Apparatus	44
3.3	Chosen Camera and Software	45
3.4	Working Environment and Libraries	47
4	Realization	49
4.1	Computer Vision Approach	49
4.1.1	Data	49
4.1.2	Card Extraction	50
4.1.3	Classification Preparations	55
4.1.4	Rank and Suit Identification	61
4.1.5	Color Identification	63
4.1.6	Performace Assessment	64
4.2	Convolution Neural Network Approach	66
4.2.1	Sole Cards Dataset	66
4.2.2	Background Dataset	68
4.2.3	Synthetic Dataset Generation	69
4.2.4	Model Training and Validation	72
4.2.5	Model Testing	79
5	Discussion	81
5.1	Convolutional Neural Network Approach	81
5.1.1	Overview	81
5.1.2	Interpretations	81
5.1.3	Implications	81
5.1.4	Limitations	82
5.1.5	Recommendations	82
5.2	Computer Vision Approach	82
5.2.1	Overview	82
5.2.2	Interpretations	82
5.2.3	Implications	83
5.2.4	Limitations	83
5.2.5	Recommendations	83
6	Summary	85

List of Figures

1.1	The whole process of card recognition [14].	6
1.2	The algorithm of the poker card image recognition [20].	7
1.3	The flow chart of the poker card extracting [20].	7
1.4	(a) Who’s Counting? system (b) schematic of the software [24].	8
1.5	(a) Example of image differencing (b) Binary edge images showing edge pixels (c) Binary edge image of a nine and a four of hearts shows valid suit blocks (d) image and disparity map of chip stacks 1, 5, and 10 chips high [24].	9
1.6	The whole process of finding and classifying small images of cards [29].	10
2.1	Conversion chart of different color spaces [37].	13
2.2	Image in different color spaces [38].	14
2.3	Gaussian filter results with different sigma parameter [45].	16
2.4	The bilateral filter converts any input image (a) to a smoothed version (b). It removes most texture, noise, and fine details but preserves large sharp edges without blurring [44].	17
2.5	Pixel value change with different gamma values [54].	18
2.6	Comparison of multiple edge detection filters: Top left: (Original). Top middle: (Sobel). Top right: (Sobel RGB). Bottom left: (Prewitt). Bottom middle: (Laplacian). Bottom right: (Canny) [56].	19
2.7	Canny edge detection process [57].	20
2.8	Visualization of the Ramer-Douglas-Peucker algorithm [60].	22
2.9	Otsu’s method for thresholding the image [66].	24
2.10	Two-dimensional gray scale image dilation, erosion, close, and open on a printed circuit board image [68].	26
2.11	Different results from clustering algorithms performed on the same data.	33
2.12	Neural network with many convolutional layers [86].	34
2.13	Convolution operation [86].	35
2.14	ReLU activation function [86].	35
2.15	Max pooling [86].	36
2.16	Fully connected layer [86].	36
2.17	Performance of YOLOv8 against older versions [98].	39
2.18	YOLOv8 is an anchor-free model. This means it predicts directly the center of an object instead of the offset from a known anchor box [99].	40
3.1	Exact measurements with a caliper of the chosen playing card.	44
3.2	Manufactured camera apparatus.	45
46figure.caption.38		
4.1	Captured image of the card.	49
4.2	Grayscale image of the card.	50
4.3	Gamma corrected image of the card.	50
4.4	Applied bilateral filter.	51
4.5	Canny edge detection.	51
4.6	Found contours.	52

4.7	Approximated contours with four points.	52
4.8	Filtered contours based on rectangularity and aspect ratio.	53
4.9	Largest contour displayed.	53
4.10	Extracted card.	54
4.11	Clipped card with transparent background.	54
4.12	Process of creating the reference templates.	55
4.13	Created rank templates.	56
4.14	Created suit templates.	56
4.15	Clustering results 1.	59
4.16	Clustering results 2.	60
4.17	Process of extracting the rank and suit.	61
4.18	Process of classification of the rank and suit.	62
4.19	Process of color extraction.	63
4.20	Three randomly generated augmented cards from the original image.	68
4.21	Representation of three pickle files contents.	68
4.22	Different backgrounds from testing, training, and validation.	69
4.23	Top-down and side view with (20,20,20) angles of the placed cards into the background with overlapping and 20% scaling.	70
4.24	Display of the contours and annotating areas.	70
4.25	Final structure of the created dataset.	71
4.26	Results of different metrics accumulated during the first training phase where the x-axis is epoch number, and the y-axis is the corresponding score.	74
4.27	Example of validation data in the first training phase on the best weights.	75
4.28	Example of predicting validation data in the first training phase on the best weights with a corresponding probability of classified class.	75
4.29	Results of different metrics accumulated during the second training phase where the x-axis is epoch number, and the y-axis is the corresponding score.	76
4.30	Example of validation data in the second phase on the best weights.	77
4.31	Example of predicting validation data in the second training phase on the best weights with a corresponding probability of classified class.	77

List of Tables

2.1	Overview of common image formats.	15
2.2	Comparison of various YOLO versions and their performance [97].	39
3.1	Filters used in the IP webcam app with corresponding images of the white texture of the card.	46
3.2	List of used libraries, their descriptions, and URLs.	47
4.1	Colors examples from ISCC/NBS and their different color space values	57
4.2	Defined classification colors and their different color space values	57
4.3	Summary of similarity methods for rank and suit classification	62
4.4	Color identification results.	65
4.5	Rank identification results.	65
4.6	Suit identification results.	65

List of code listings

4.1 Advanced Pixel Augmentation Function 67

I would like to express my gratitude to my advisor, Ing. Jan Glaser, for the essential guidance and mentorship received during the research process. Furthermore, I am deeply grateful for the unyielding love, support, and belief of my family, particularly my parents, during my academic journey. Their continual motivation has been a foundation of strength and inspiration.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis. I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Czech Technical University in Prague has the right to conclude a licence agreement on the utilization of this thesis as a school work pursuant of Section 60 (1) of the Act.

In Prague on May 11, 2023

.....

Abstract

This thesis presents a playing card recognition system that uses two approaches. The first approach is based on computer vision techniques to classify playing cards in a controlled environment. The second approach involves creating a comprehensive dataset with augmentation and training a deep-learning model. The first approach uses image processing techniques to detect the rank, suit, and color of a single playing card in the image with multiple methods. The best method achieves an accuracy of 100% for rank and suit identification and 78.16% for color identification on a dataset of 728 playing cards. The second approach involves creating a comprehensive dataset of playing cards with augmentation techniques. Initially, the dataset contains 30,000 images of playing cards labeled with their rank and suit, and these images are subjected to slight augmentations. The YOLOv8 model is pre-trained on this initial dataset in the first phase of the training process. In the second phase, the model is fine-tuned on a larger dataset of 80,000 images, which includes multiple cards in each image and more closely resembles real-world conditions, such as different points of view and varying lighting conditions. The model is also validated during the fine-tuning process to ensure optimal performance. Once the model is trained, it is tested on a set of 30,000 images that maintain the diversity and complexity found in real-world environments. The YOLOv8 model achieves an accuracy of 99.98% on this test set, demonstrating its effectiveness in recognizing playing cards in a wide range of environments.

Keywords playing card recognition, computer vision, data augmentation, image processing, canny edge detection, template matching, convolutional neural network, YOLOv8 deep learning model

Abstrakt

Tato práce popisuje systém rozpoznávání hracích karet, který využívá dvou přístupů. První přístup využívá počítačového vidění pro klasifikaci hracích karet v kontrolovaném prostředí, zatímco druhý přístup zahrnuje vytvoření komplexní datové sady s různými vizuálními úpravami a trénink modelu hlubokého učení. První přístup využívá technik zpracování obrazu ke rozpoznání líce karty a jeho barvy (červená-černá) na jednom obrázku s použitím několika metod. Nejlepší metoda dosahuje přesnosti 100% pro identifikaci líce karty a 78,16% pro identifikaci barvy v souboru dat obsahujícím 728 hracích karet. Druhý přístup spočívá v tvorbě komplexní sady dat hracích karet s použitím vizuálních úprav a technik. Soubor dat obsahuje původně 30 000 obrázků hracích karet, které jsou označeny svým lícem a barvou a mírně upraveny. Model YOLOv8 je nejprve natrénován na této počáteční sadě dat a poté je vyladěn na větší sadě obsahující 80 000 obrázků, které se více podobají reálným podmínkám, jako jsou různé úhly pohledu a světelné podmínky. Během trénování je model validován, aby byla zajištěna optimální výkonnost. Jakmile je model natrénován, je testován na sadě 30 000 snímků, které zachovávají rozmanitost a složitost prostředí v reálném světě. Model YOLOv8 dosahuje na této testovací sadě přesnosti 99,98%, což dokazuje jeho účinnost při rozpoznávání hracích karet v různých prostředích.

Klíčová slova rozpoznávání hracích karet, počítačové vidění, augmentace dat, zpracování obrazu, Cannyho hranový detektor, porovnávání šablon, konvoluční neuronová síť, model hlubokého učení YOLOv8

Introduction

Card games have long been a component of human culture and enjoyment, extending back to ancient civilizations such as China, Persia, and Egypt. They have evolved through time to include a broad range of rules and forms, reflecting players' different cultural origins and tastes worldwide. In recent years, technological improvements have opened up new avenues for improving the gaming experience, making it more immersive, engaging, and accessible to a wider variety of players, including those with visual impairments.

Rapid advancements in domains such as computer vision, artificial intelligence, and machine learning have paved the way for creating novel card game applications. The development of an efficient and accurate real-time playing card identification system is one such application that has the potential to transform the way card games are played, studied, and enjoyed by players of all skill levels and talents. This system can find numerous applications in various contexts, including real-time analytics for card game enthusiasts like poker and blackjack. It helps them study gameplay, identify weaknesses, and analyze strategies to improve performance.

Another application is digital assistance for visually impaired players, enabling them to participate in card games, facilitate social interaction with other players, and receive audio feedback on the game's current state. Additionally, card game simulators can benefit from integrating a real-time playing card recognition system, providing a seamless and engaging gaming experience and closely replicating a traditional tabletop environment.

In conclusion, the main reasons behind the motivation for creating this thesis are that developing an efficient and accurate real-time playing card recognition system can transform how card games are played and analyzed. Furthermore, it makes these games more accessible and enjoyable for a diverse range of players, regardless of their abilities or background.

Objectives

The main objective of this thesis is to develop a robust, fast, and accurate real-time recognition system for playing cards, identified by their rank, suit, and color, using two different approaches. The objectives can be further divided into the following subsections:

- Introduction to playing card recognition system (Chapter 1): Introduce the recognition system components, their functions, and potential use cases.
- Prior solutions (Chapter 1): Research and analyze existing playing card recognition systems available in the market, discussing their strengths and weaknesses.
- Establishing a theoretical foundation (Chapter 2): Review and present the underlying theoretical concepts, algorithms, and methods that will be utilized in developing the recognition system.
- Analysis (Chapter 3): Design a specialized camera setup tailored to the needs of the playing card recognition system, including selecting the appropriate camera, lighting, and mounting equipment to ensure optimal image capture and processing.
- Designing and testing a robust recognition algorithm (Chapter 4): Develop a computer vision algorithm capable of efficiently extracting and accurately recognizing playing cards based on their rank, suit, and color in controlled conditions, employing multiple methods for card classification and testing them on the obtained images.
- Developing a training dataset (Chapter 4): Use the computer vision algorithm to collect, extract, and correctly classify playing cards, creating a diverse and representative dataset of playing card images with varying sizes, conditions, and orientations for training and validating the second recognition algorithm based on a convolutional neural network model.
- Training, validation, and testing of the recognition model (Chapter 4): Train the recognition algorithm using the training dataset, fine-tune its parameters for optimal performance, and validate the model with a separate dataset to ensure accuracy and generalizability. Perform comprehensive testing of the recognition system in various environments and conditions, simulate real-world use cases, and evaluate its performance using appropriate metrics to ensure it meets expectations.
- Discussion (Chapter 5): Discuss both approaches, and provide an overview, interpretation, implications, limitations, and recommendations for each approach to gain a comprehensive understanding of their potential impact and effectiveness.

Chapter 1

Survey

This chapter provides a brief overview of the existing solutions and techniques utilized in playing card recognition systems. It explores the strengths and weaknesses of different approaches and highlights the key innovations and challenges faced by researchers in the domain.

1.1 Playing Card Recognition system

A playing card recognition system is designed to identify and classify playing cards from images. There are two common approaches to building such a system: traditional computer vision techniques and deep learning models based on convolutional neural networks (CNNs) [1, 2]. The whole process can be divided into several steps:

- **Image Acquisition:** The first step is to acquire the image containing the playing cards. This could be done using a camera or any other image-capturing device [3].
- **Preprocessing:** The acquired image is preprocessed to enhance the features of interest and suppress the noise. This step may include resizing, grayscale conversion, histogram equalization, and noise removal [4, 5].
- **Card Detection:** Detecting the cards in the image is usually the next step. This could be done using techniques like contour detection, edge detection, or more advanced object detection models [6, 7].
- **Card Segmentation (Computer Vision Approach):** Once the cards are detected, the regions of interest (which typically are the card corners, as they contain the suit and rank information) need to be segmented. This can be done using image segmentation techniques [4, 8].
- **Feature Extraction and Classification (Computer Vision Approach):** From the segmented regions, features are extracted to help identify the card's rank and suit. This could include color histograms, texture features, or other hand-crafted features. The final step is to classify the card based on the extracted features [9, 10].
- **Card Classification (Deep Learning Approach):** Alternatively, a deep learning model based on convolutional neural networks (CNNs) can be trained on a dataset of labeled playing card images to classify the cards directly [1, 2, 11, 12, 13]. In this approach, the model automatically learns the features and classification without the need for explicit feature extraction and segmentation.

1.2 Current Solutions

This section reviews various methods and surveys on playing card recognition. This literature review aims to identify the most effective approaches, algorithms, and techniques for developing a robust playing card recognition system.

The paper by [14] focuses on recognizing playing cards using their faces as identification targets, converting captured color images into grayscale and then binary images. The contour shape matching technique employs the Hu invariant moment, unaffected by card rotation [15], to compare card suits and ranks with pre-obtained templates.

The study investigates the region of interest in playing cards, extracting it using the zero-order moment of binary images [4]. Card suits and ranks are identified through image recognition algorithms such as graying, Gaussian filtering, binarization, and contour shape matching [16]. Hu invariant moments are used for recognizing large objects with simple textures in images [17]. Contour shape matching (using Hu moment invariants) is the primary identification method for the distinct shapes of card suits and ranks [18]. The outline of a card is compared to all shape templates, with the best match having the smallest value [19].

The study's results show that the recognition process is relatively fast but prone to errors. It can be significantly affected by environmental factors and the shooting angle of the playing card.

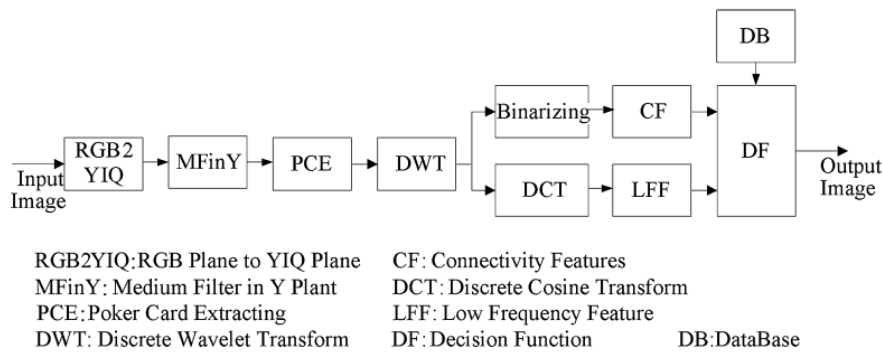


■ **Figure 1.1** The whole process of card recognition [14].

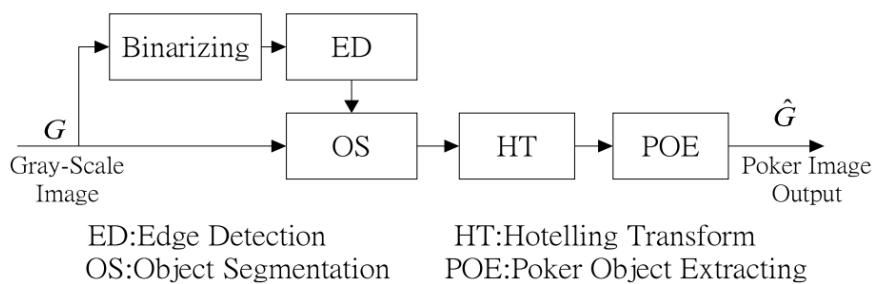
In [20], a recent study, a novel poker recognition scheme was developed that utilizes artificial intelligence to identify poker cards under various conditions effectively. The scheme incorporates three main strategies:

- Employing the Hotelling transform [21] for accurate object positioning.
- Utilizing weighted compacted energy (WCE) as a primary feature derived from the image's DWT [22] and DCT [23].
- Implementing four orientation connectivity run-length values (FOCRLV) to differentiate between card images.

This research contributes to the field by introducing FOCRLV as a unique feature to enhance image recognition and using an image's compact energy band as another distinguishing feature. The proposed scheme was rigorously tested under multiple conditions, including 40% noise and intensity level changes of $\pm 40\%$. The results demonstrated that the scheme could accurately identify ranks and suits of poker cards 100% of the time, showcasing its effectiveness and potential applicability in AI-powered poker systems.



■ **Figure 1.2** The algorithm of the poker card image recognition [20].



■ **Figure 1.3** The flow chart of the poker card extracting [20].

A notable study by [24] addresses the increasing issues of card counting and dealer errors in Blackjack. The researchers developed an innovative computer vision system for monitoring casino games using an overhead stereo camera¹. This system offers a cost-effective and user-friendly alternative to existing solutions, which often require specialized hardware and are financially inaccessible for all but the largest casinos.

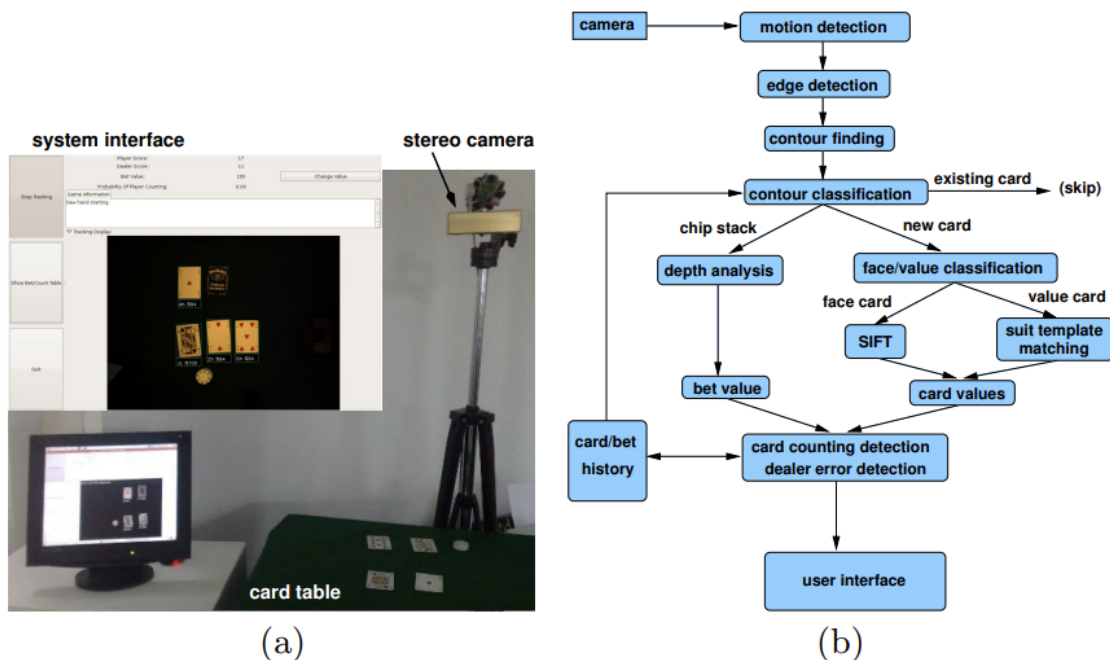
The proposed system utilizes several techniques to ensure accurate card recognition and real-time player bet tracking:

- Contour analysis [18] for detecting and analyzing the shapes and boundaries of objects within images.
- Template matching [25] for comparing and matching card images with predefined templates, aiding in card identification.
- The SIFT algorithm [26] to detect and describe local features in images, further enhancing card recognition accuracy.

The system can identify potential card counters by analyzing the correlation between player bets and card count [27]. Additionally, the system employs stereo imaging to measure chip stack heights, allowing for bet size monitoring [28].

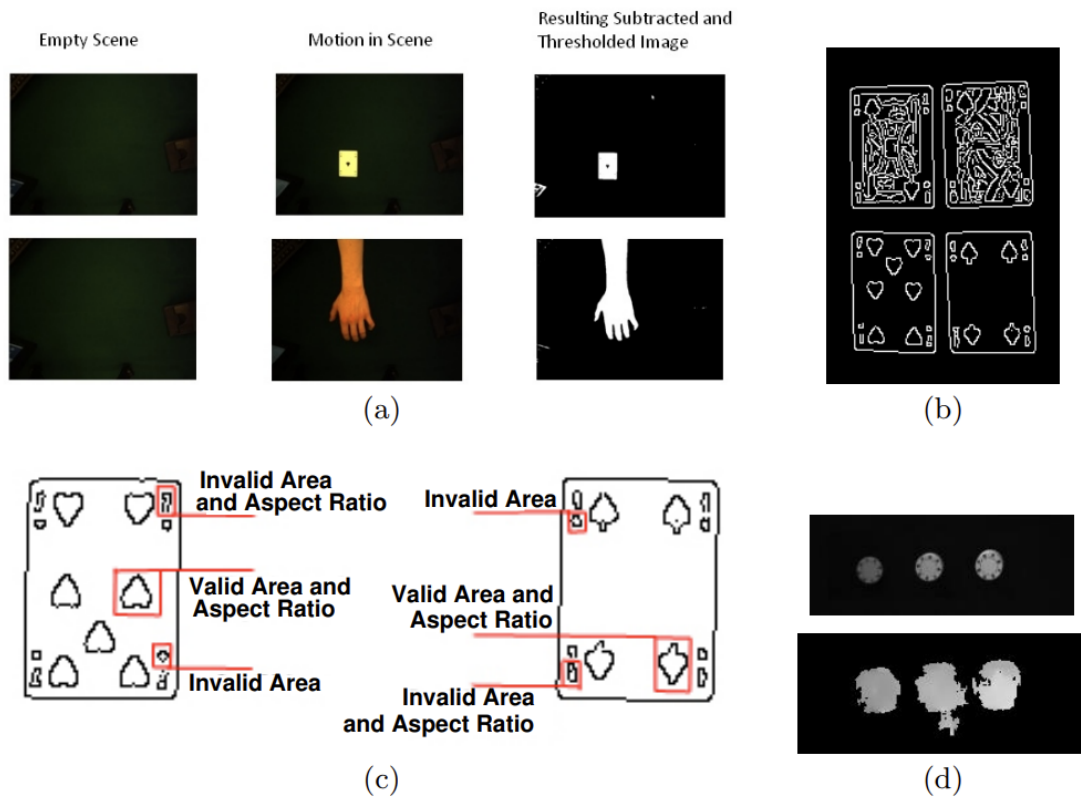
Impressively, the system achieves over 99% accuracy in card recognition and has successfully detected card counters and dealer errors among a diverse user group, including professional dealers and novice players.

This study contributes to the broader understanding of computer vision applications in the gaming industry. It emphasizes the potential for more efficient and cost-effective solutions to address card counting and dealer error concerns, leveraging advanced image processing techniques to improve game monitoring and security.



■ **Figure 1.4** (a) Who's Counting? system (b) schematic of the software [24].

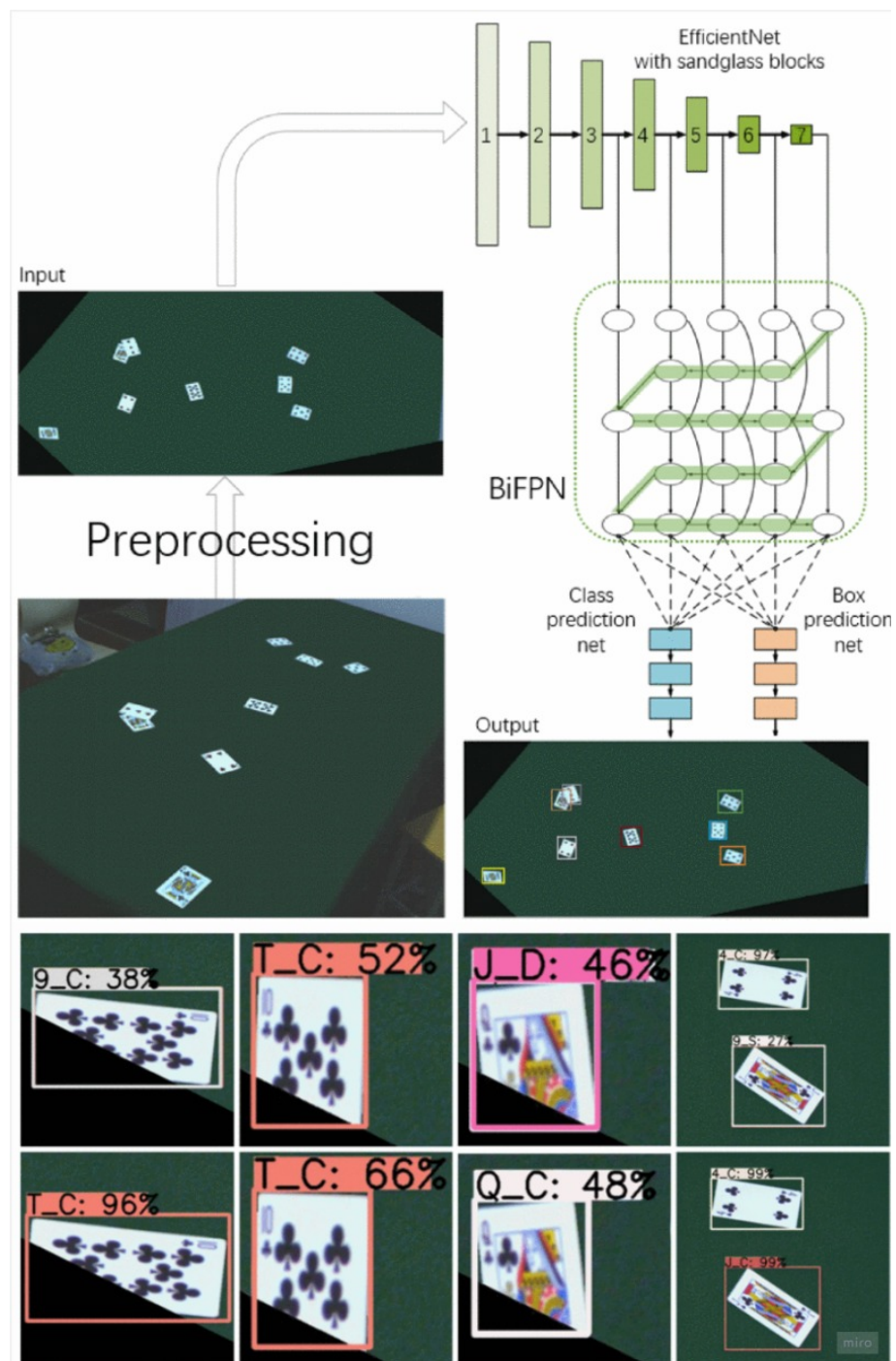
¹<https://www.fir.com/support-center/iis/machine-vision/application-note/stereo-vision-introduction-and-applications/>



■ **Figure 1.5** (a) Example of image differencing (b) Binary edge images showing edge pixels (c) Binary edge image of a nine and a four of hearts shows valid suit blocks (d) image and disparity map of chip stacks 1, 5, and 10 chips high [24].

In a recent study by [29], the researchers present a neural network to detect playing cards in real poker scenes through a camera, where the playing card area represents only 0.7% of the shot table area. In the acquired images, the suits of cards are fuzzy and difficult to identify, even to the naked eye [30]. Because of the relatively few pixels corresponding to the cards, traditional image processing and pattern recognition methods struggle to detect them [14]. Therefore, the authors employ deep learning methods, which have shown to be easy to use, faster, and more accurate in a broad range of computer vision applications over the years [2, 11].

Inspired by the sandglass block, the researchers improved the current state-of-the-art neural network architecture for object detection, EfficientDet [31], to retain more features. Experiments have been conducted to evaluate the performance of the improved EfficientDet model, showing that it achieved considerable performance improvement compared with other deep learning models [32, 7, 33].



■ **Figure 1.6** The whole process of finding and classifying small images of cards [29].

This study contributes to developing more accurate and efficient card detection systems in real-world poker scenes using deep learning techniques. By enhancing the EfficientDet architecture, the authors demonstrate the potential of neural networks in detecting small objects with fuzzy and difficult-to-identify features.

Theoretical background

2.1 Color Spaces

Color Spaces refer to the mathematical models that describe how colors can be represented as tuples of numbers, typically as three or four values or color components [34]. Some of the commonly used color spaces in image processing include:

- **Grayscale:** This color space represents an image in different shades of gray. The colors in a grayscale image range from black to white, with various shades of gray in between. Grayscale is often used in methods where color does not provide essential information or when converting an image to grayscale can simplify the analysis [4].
- **RGB (Red, Green, Blue):** This is one of the most common color spaces. In RGB, each color appears as a combination of red, green, and blue. The color space is based on the human perception of colors as light mixtures [4].
- **CMYK (Cyan, Magenta, Yellow, Black):** Used mainly in printing, the CMYK color space is a subtractive color model based on the color absorption properties of paints and inks. The primary colors are cyan, magenta, and yellow, and a fourth color, black, is often added to produce deeper blacks [34].
- **HSV (Hue, Saturation, Value):** The HSV color space separates the chromatic information (hue, saturation) from the lighting information (value), making it useful for many applications in computer vision, such as object tracking and image segmentation [5].
- **YCbCr:** The YCbCr color space is used mainly in video systems. It separates the brightness information (Y) from the chroma or color information (Cb and Cr). This is useful because the human visual system perceives brightness information more powerfully than color information [35].
- **Lab:** The Lab color space, also known as CIELAB, includes all perceivable colors, which makes it device-independent – a given color is the same regardless of the device that represents it [34].

Color Space Conversions:

- **RGB to Grayscale:** The luminance model is commonly used to convert RGB values to grayscale. It takes into account the fact that human eyes are more sensitive to certain colors than others. The conversion is given by:

$$Y = 0.299R + 0.587G + 0.114B \quad (2.1)$$

where Y is the grayscale value, and R , G , and B are the red, green, and blue values, respectively [4].

- **RGB to CMYK:** The conversion from RGB to CMYK involves two steps. First, RGB is converted to CMY (Cyan, Magenta, Yellow):

$$C = 1 - R \quad (2.2)$$

$$M = 1 - G \quad (2.3)$$

$$Y = 1 - B \quad (2.4)$$

Then, the black component (K) is calculated, and the color values are adjusted accordingly:

$$K = \min(C, M, Y) \quad (2.5)$$

$$C' = (C - K)/(1 - K) \quad (2.6)$$

$$M' = (M - K)/(1 - K) \quad (2.7)$$

$$Y' = (Y - K)/(1 - K) \quad (2.8)$$

where C' , M' , and Y' are the adjusted CMY values [34].

- **RGB to HSV:** The conversion from RGB to HSV can be computed using the following equations:

$$H = \begin{cases} \frac{60(G-B)}{M-m} & \text{if } M = R \text{ and } G \geq B \\ \frac{60(G-B)}{M-m} + 360 & \text{if } M = R \text{ and } G < B \\ \frac{60(B-R)}{M-m} + 120 & \text{if } M = G \\ \frac{60(R-G)}{M-m} + 240 & \text{if } M = B \end{cases} \quad (2.9)$$

$$S = \begin{cases} 0 & \text{if } M = 0 \\ \frac{M-m}{M} & \text{otherwise} \end{cases} \quad (2.10)$$

$$V = M \quad (2.11)$$

Where H , S , and V are the hue, saturation, and value components, respectively; M and m are the maximum and minimum values of the input RGB components, respectively.

- **RGB to Lab:** The conversion from RGB to Lab involves several steps. First, the RGB values are transformed to the XYZ color space using a linear transformation matrix:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.4124 & 0.3576 & 0.1805 \\ 0.2126 & 0.7152 & 0.0722 \\ 0.0193 & 0.1192 & 0.9505 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad (2.12)$$

Then, the XYZ values are normalized with respect to the white point of the color space and the non-linear transformation is applied:

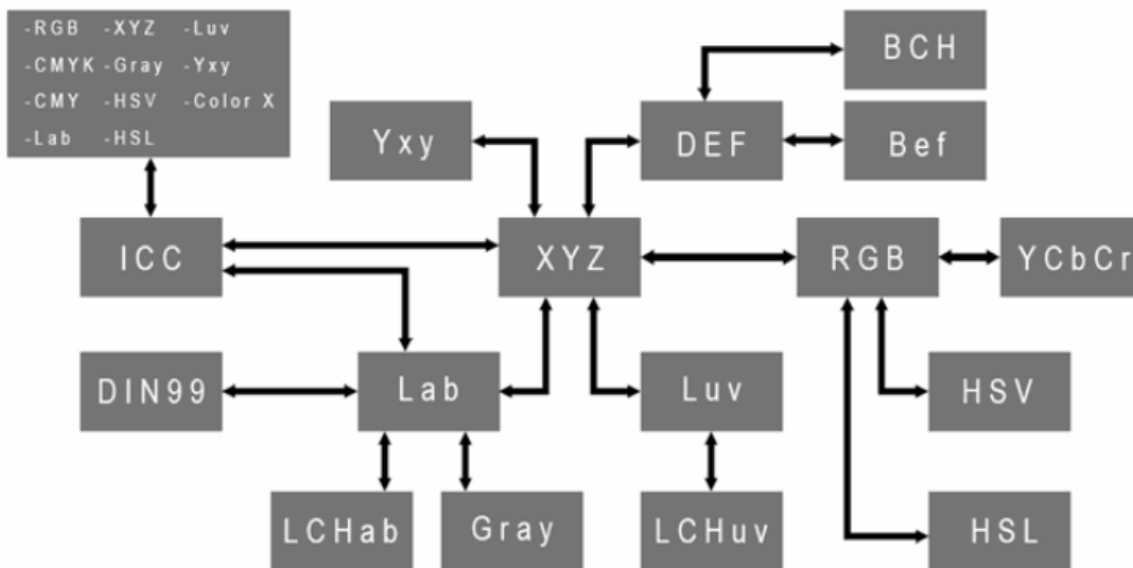
$$L^* = 116f(Y/Y_n) - 16 \quad (2.13)$$

$$a^* = 500(f(X/X_n) - f(Y/Y_n)) \quad (2.14)$$

$$b^* = 200(f(Y/Y_n) - f(Z/Z_n)) \quad (2.15)$$

where $f(t) = \begin{cases} t^{1/3} & \text{if } t > (\frac{6}{29})^3 \\ \frac{1}{3}(\frac{29}{6})^2t + \frac{4}{29} & \text{otherwise} \end{cases}$, and $X_n, Y_n,$ and Z_n are the reference white point values for the color space [34].

Choosing the appropriate color space for a specific task can significantly impact the performance of image processing algorithms [36].



■ **Figure 2.1** Conversion chart of different color spaces [37].



■ Figure 2.2 Image in different color spaces [38].

2.2 Image Formats

Image formats refer to how image data is stored and represented in the digital form [4]. These formats are usually distinguished by their file extensions and can be broadly categorized into lossless and lossy formats [39]. Each format has its characteristics and is suited for specific use cases.

Format	Type	Description
BMP	Lossless	Bitmap format, simple and uncompressed
PNG	Lossless	Portable Network Graphics, supports transparency
GIF	Lossless	Graphics Interchange Format, supports animation
JPEG	Lossy	Joint Photographic Experts Group, suitable for photographs
TIFF	Lossless	Tagged Image File Format, high-quality images
WebP	Both	Developed by Google, supports both lossless and lossy compression

■ **Table 2.1** Overview of common image formats.

Lossless formats, such as BMP, PNG, GIF, and TIFF, preserve all the original image data without any loss of information. These formats are suitable for tasks requiring high-quality images, such as scientific and medical imaging [4].

Lossy formats, such as JPEG, involve a degree of information loss due to compression. These formats are helpful when a trade-off between image quality and file size is acceptable, such as in web applications and digital photography [40].

Some formats, like WebP, support lossless and lossy compression methods, providing a balance between image quality and file size [41].

$$\text{Compression Ratio} = \frac{\text{Uncompressed Image Size}}{\text{Compressed Image Size}} \quad (2.16)$$

The choice of image format depends on the specific requirements of the application and the trade-offs between image quality, compression, and file size [39].

2.3 Image Filtering

Image filtering is a fundamental process in image processing and computer vision used to modify or enhance an image by emphasizing certain features or removing others. This is usually accomplished by convolving the image with a filter or a mask, which can be designed to have specific properties [4].

Different types of image filters are used for different purposes. For example:

- **Low-pass filters:** These filters allow low-frequency content while attenuating high-frequency content. They are often used for noise reduction and image blurring. An example of a low-pass filter is the Gaussian filter [42].
- **High-pass filters:** These filters allow high-frequency content while attenuating low-frequency content. They are often used for edge detection or image sharpening. The Laplacian filter is an example of a high-pass filter [4].
- **Median filters:** These non-linear filters replace each pixel's value with the median value of its neighborhood. They are particularly effective at removing 'salt and pepper' noise while preserving edges [43].
- **Bilateral filters:** This edge-preserving and noise-reducing smoothing filter combines the domain filter and the range filter to perform smoothing while preserving edges [44].

Mathematically, the convolution operation of an image $f(x, y)$ with a filter $h(x, y)$ is defined as:

$$g(x, y) = \iint f(x - x', y - y')h(x', y') dx' dy' \quad (2.17)$$

where $g(x, y)$ is the filtered image.

2.3.1 Gaussian Filter

A Gaussian filter, also known as a Gaussian blur or Gaussian smoothing, is an image processing technique commonly used to reduce noise and detail in an image [4]. The filter works by convolving the image with a Gaussian function, which can be considered a weighted average where the weights decrease as the distance from the center pixel increases.

The Gaussian filter is characterized by its bell-shaped curve, where the values in the middle are higher and decrease symmetrically towards the ends. The shape of this curve is defined by the standard deviation (σ), which controls the amount of blurring. Larger values of σ produce a more comprehensive (and therefore more blurry) kernel.

The 2D Gaussian function is defined as:

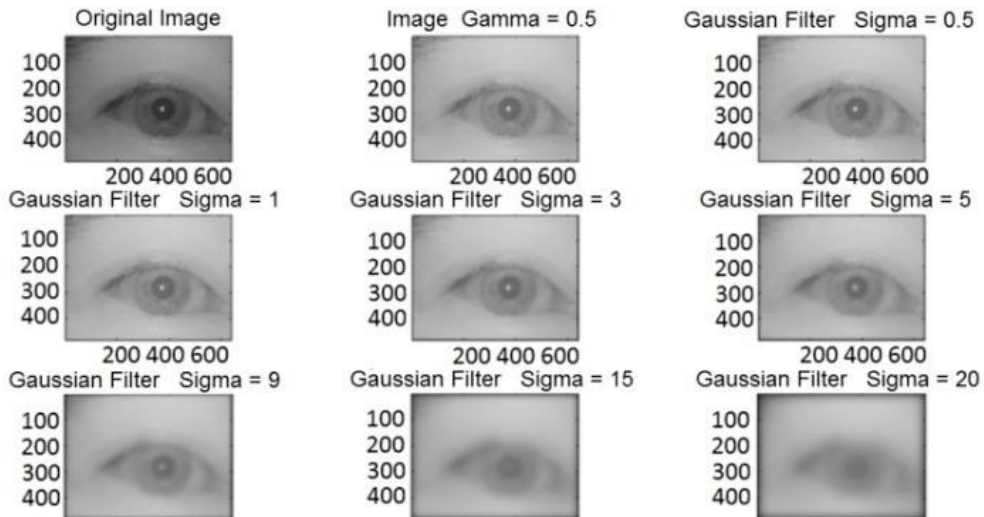
$$G(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right) \quad (2.18)$$

Where:

- x, y are the distances from the origin in the horizontal and vertical directions, respectively.
- σ is the standard deviation of the Gaussian distribution.

Gaussian filters are handy because they are isotropic, meaning they have the same effect in all directions. This is a desirable property in many image processing tasks as it ensures that no artificial directionality is introduced into the processed image.

Gaussian filters are used in various applications, such as edge detection [6], image denoising [44], and feature extraction [26], among others.



■ **Figure 2.3** Gaussian filter results with different sigma parameter [45].

2.3.2 Bilateral Filter

A bilateral filter is a non-linear, edge-preserving, and noise-reducing smoothing filter for images. Unlike the Gaussian filter, which is solely a function of Euclidean distance, the bilateral filter also considers the intensity difference, ensuring that it preserves sharp edges while still reducing noise [44].

The bilateral filter is defined by two Gaussian distributions: the range Gaussian (dependent on the intensity difference) and the spatial Gaussian (dependent on the Euclidean distance). Combining these two allows the filter to consider pixels for smoothing that are not only nearby but also have similar intensity values.

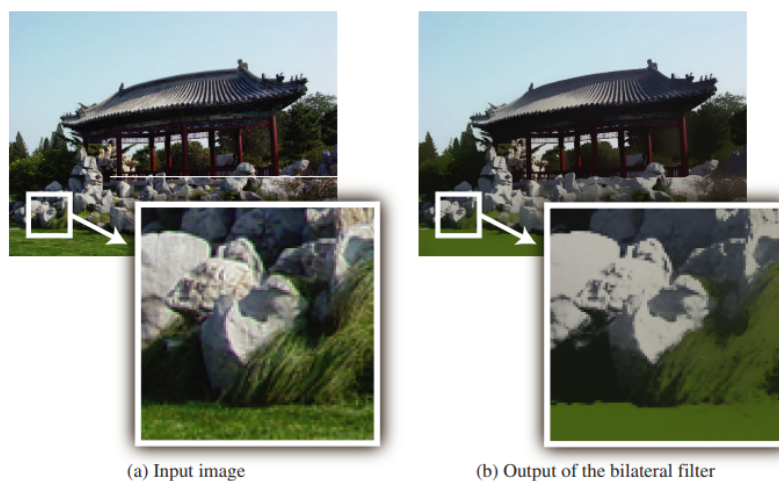
The bilateral filter can be formally defined as:

$$BF[I](x) = \frac{1}{W_p} \sum_{x_i \in \Omega} G_{\sigma_s}(\|x - x_i\|) G_{\sigma_r}(\|I(x) - I(x_i)\|) I(x_i) \quad (2.19)$$

Where:

- x is the coordinate of the center pixel,
- x_i is the coordinate of the neighboring pixels,
- Ω is the spatial neighborhood of the center pixel,
- G_{σ_s} is the spatial Gaussian that considers the spatial closeness of pixels,
- G_{σ_r} is the range Gaussian that considers the similarity of pixel intensities,
- $I(x)$ and $I(x_i)$ are the center pixel and neighboring pixel intensities, respectively,
- W_p is the normalization term.

The bilateral filter has been used in a variety of image processing applications, including HDR tone mapping [46], texture editing [47], and detail enhancement [48].



■ **Figure 2.4** The bilateral filter converts any input image (a) to a smoothed version (b). It removes most texture, noise, and fine details but preserves large sharp edges without blurring [44].

2.4 Image Processing

Image processing is a method to convert an image into digital form and perform operations on it to get an enhanced image or to extract some useful information from it [4]. It is a type of signal processing in which the input is an image, like a photo or video frame. The output of image processing may be either an image or a set of characteristics or parameters related to the image.

There are two types of image processing, Analog, and Digital Image Processing. Digital image processing focuses on two major tasks: improving visual information for human interpretation and processing of image data for storage, transmission, and representation for autonomous machine perception [4, 3].

The general phases of image processing include importation of the image with an optical scanner or by digital photography, manipulation of the image as required (which may consist of image enhancement, image restoration, and feature extraction), and output (which can be altered image or report that is based on image analysis) [4, 49].

Various techniques are used in image processing, including convolution, Fourier transform, edge detection, segmentation, morphological operations, and others. These techniques are used in various applications, including medical imaging, facial recognition, autonomous vehicles, and many more [49].

2.4.1 Gamma Correction

Gamma correction, or often gamma, is a nonlinear operation used to encode and decode luminance or tristimulus values in video or still image systems [50]. Gamma correction increases the perceptual impression of contrast in a low-intensity range, which is generally beneficial because human eyes are more sensitive to changes in dark tones than in bright tones [51].

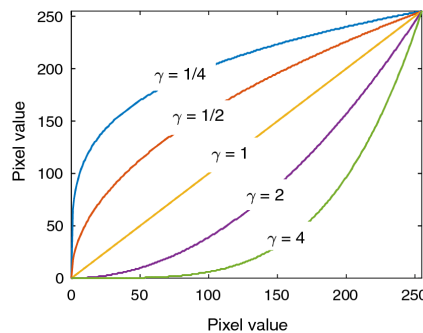
A gamma correction function can be formulated as follows:

$$I_{\text{out}} = A \cdot I_{\text{in}}^{\gamma} \quad (2.20)$$

Where:

- I_{in} is the input intensity,
- I_{out} is the output intensity,
- A is a constant, and
- γ is the gamma value.

In practice, the value of gamma is often set to 2.2 for displays following the sRGB standard [52]. Gamma correction ensures the correct brightness and color balance when images are displayed on different devices [53]. As the value of γ increases above 1.0, the image becomes darker, and as the value of γ decreases below 1.0, the image becomes lighter.



■ **Figure 2.5** Pixel value change with different gamma values [54].

2.4.2 Contour and Edge Detection

Contour detection, a fundamental task in image processing and computer vision, is identifying the boundaries or outlines of objects within an image [4]. Contours provide critical visual cues about objects' shape and sometimes their depth and three-dimensional structure.

The most common approach to contour detection involves edge detection followed by edge linking. Edge detection operators, such as the Sobel, Prewitt, or Canny operators, are used to identify points in an image where the intensity changes sharply [6]. Then, edge-linking methods connect these points into meaningful object boundaries.

Convolutional operators, also known as convolutional kernels, perform convolution operations in edge detection. For example, the Sobel operator uses two 3x3 kernels:

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}, \quad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

The Prewitt operator also uses two 3x3 kernels:

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}, \quad G_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

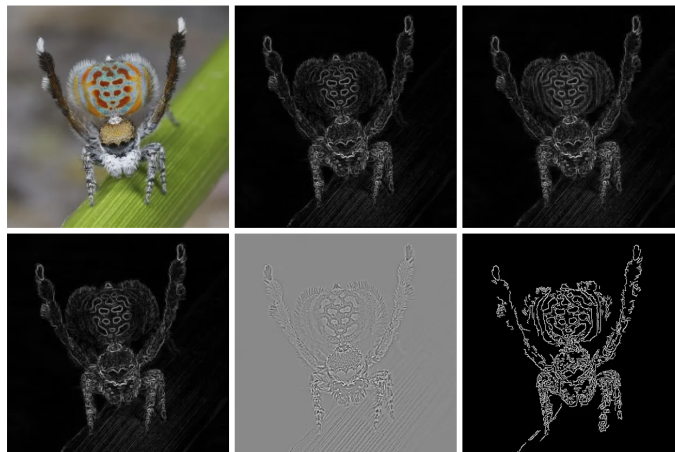
The Laplacian operator uses a single 3x3 kernel:

$$L = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

These operators are convolved with the image to detect edges.

An essential aspect of contour detection is dealing with noise since noise can create spurious edges and affect the accuracy of contour detection. Techniques like Gaussian smoothing are often used before edge detection to reduce noise [6].

More recently, more sophisticated methods for contour detection have been developed. These methods are often based on machine learning algorithms trained to detect contours, often by using convolutional neural networks (CNNs) [55].



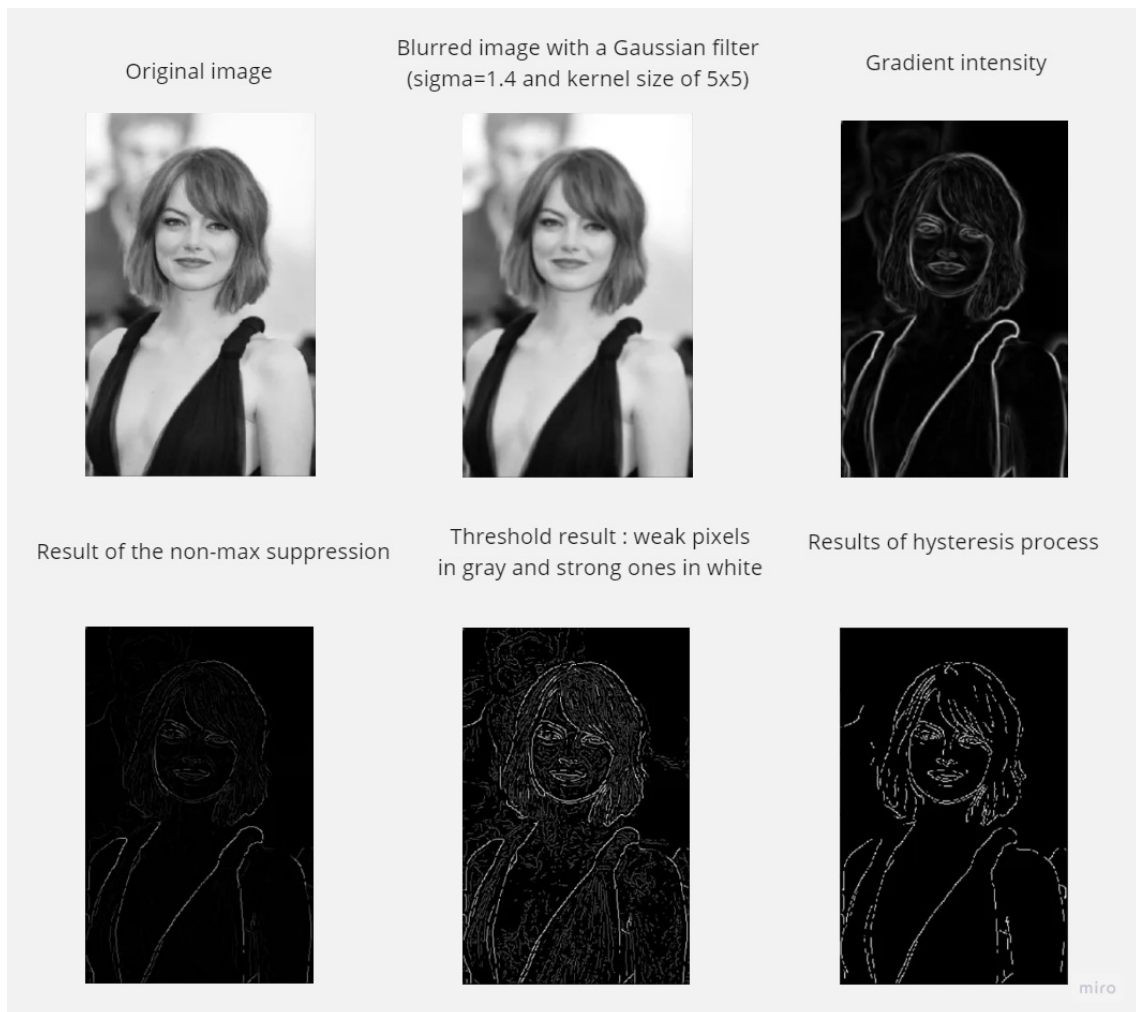
■ **Figure 2.6** Comparison of multiple edge detection filters: Top left: (Original). Top middle: (Sobel). Top right: (Sobel RGB). Bottom left: (Prewitt). Bottom middle: (Laplacian). Bottom right: (Canny) [56].

2.4.3 Canny Edge Detection

The Canny edge detection algorithm, developed by John Canny in 1986, is a multi-stage process used for detecting a wide range of image edges. It is considered one of the standard edge-detection algorithms in the field of computer vision [6].

The Canny edge detection algorithm involves the following steps:

1. **Noise Reduction:** As edge detection is susceptible to noise in an image, the first step is to remove the noise in the image with a 5x5 Gaussian filter.
2. **Gradient Calculation:** The edges should be marked where the gradients of the image have large magnitudes. So, the Gradient calculation of the image is obtained.
3. **Non-maximum Suppression:** Ideally, the final image should have thin edges. Thus, we must perform non-maximum suppression to thin the edges.
4. **Double Thresholding:** Potential edges are determined by thresholding.
5. **Edge Tracking by Hysteresis:** Final edges are determined by suppressing all edges that are not connected to a very specific (strong) edge.



■ **Figure 2.7** Canny edge detection process [57].

2.4.4 Contour Shape Analysis

In computer vision and image processing, determining the shape of contours in an image is often crucial. Shape analysis of contours can be performed using different techniques, such as contour approximation, aspect ratio calculation, and area measurement. These methods help identify, track, and segment objects based on their shapes [4, 5].

- **Contour Approximation:** Contour approximation is a method that represents a contour using a reduced number of points. The Ramer-Douglas-Peucker algorithm is a popular contour approximation algorithm that recursively removes unnecessary points for describing the contour's general shape [58, 59]. This process reduces the complexity of the contour while preserving its essential features. Given a contour with n points, $C = \{P_1, P_2, \dots, P_n\}$, the algorithm aims to find a reduced set of points, $C' = \{P'_1, P'_2, \dots, P'_m\}$, where $m < n$, that approximates the original contour within a specified tolerance, ϵ .
- **Aspect Ratio:** The aspect ratio of a contour is the ratio between the width and height of its bounding rectangle. This measurement can provide useful information about the shape and orientation of an object. For instance, an aspect ratio close to 1 indicates a square or circular shape, whereas a high or low aspect ratio suggests a more elongated shape [4]. The aspect ratio, AR , can be calculated as:

$$AR = \frac{w}{h} \quad (2.21)$$

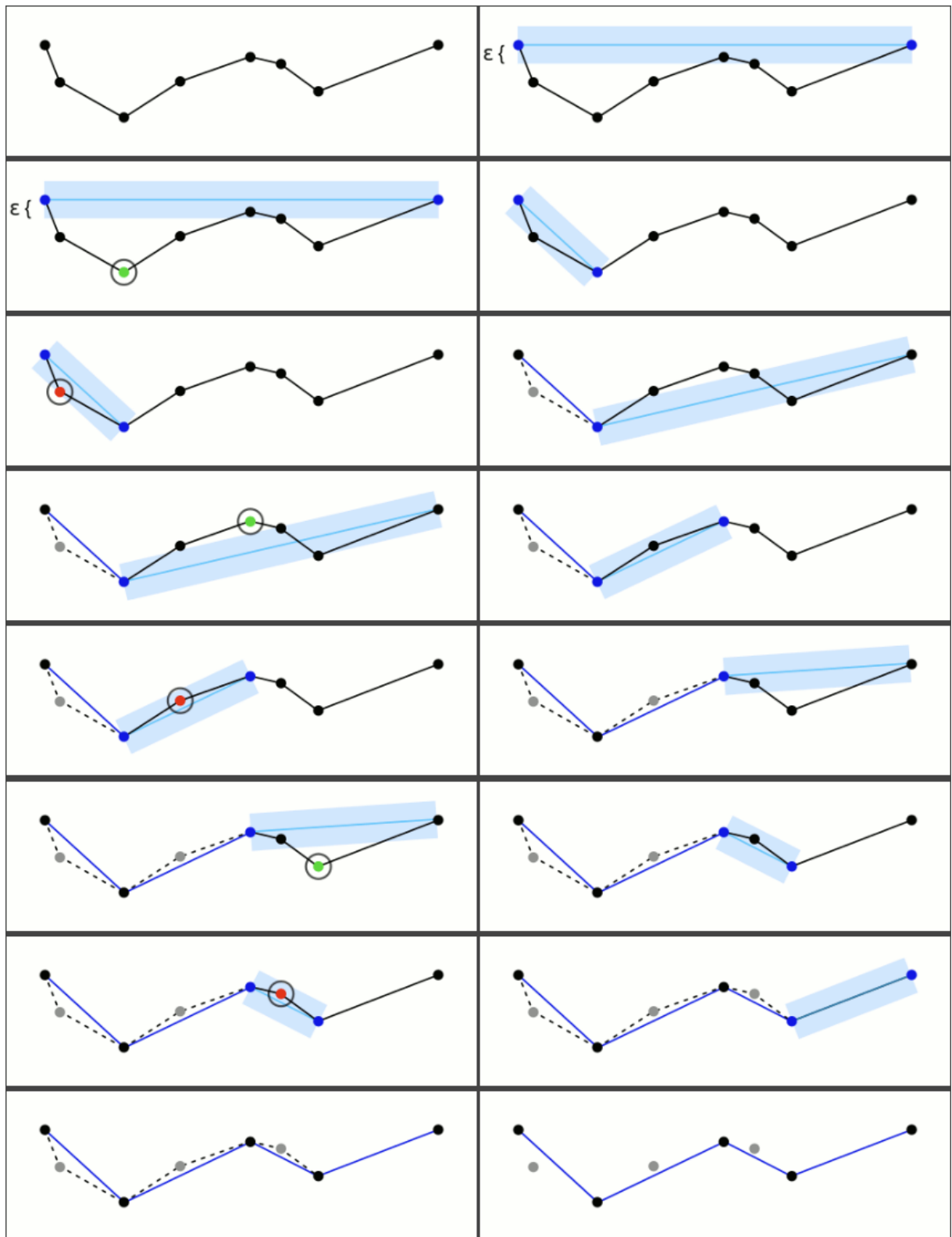
where w is the width, and h is the height of the bounding rectangle.

- **Area Measurement:** Area measurement involves calculating the total number of pixels inside a contour. This can be useful for filtering and distinguishing between objects of different sizes. In addition, the area of a contour can be normalized by dividing it by the area of the contour's bounding rectangle, providing a scale-invariant measure of the contour's complexity [5]. The normalized area, NA , can be calculated as:

$$NA = \frac{A}{w \times h} \quad (2.22)$$

where A is the area of the contour.

These techniques can be combined to analyze and classify contours based on their shapes, enabling various computer vision applications such as object recognition, tracking, and segmentation.



■ **Figure 2.8** Visualization of the Ramer-Douglas-Peucker algorithm [60].

2.4.5 Recognizing Basic Shapes

Using contour approximation and shape properties, we can recognize basic shapes such as rectangles, circles, and triangles. The process involves analyzing the number of vertices and other attributes of the approximated contour to determine the shape of the object [4, 5].

- **Rectangle:** A rectangle can be recognized by the number of vertices in its approximated contour. The object is likely a rectangle if the contour has four vertices and the angles between consecutive edges are close to 90 degrees. To calculate the angle between two edges, we can use the dot product formula:

$$\cos \theta = \frac{\vec{u} \cdot \vec{v}}{\|\vec{u}\| \times \|\vec{v}\|} \quad (2.23)$$

where \vec{u} and \vec{v} are the consecutive edge vectors, and θ is the angle between them.

- **Circle:** To recognize a circle, we can use the aspect ratio and the normalized contour area. If the aspect ratio is close to 1 and the normalized area is close to $\pi/4$, the object is likely a circle. The normalized area of a circle, NA_{circle} , can be calculated as:

$$NA_{\text{circle}} = \frac{\pi r^2}{(2r) \times (2r)} = \frac{\pi}{4} \quad (2.24)$$

where r is the radius of the circle.

- **Triangle:** A triangle can be recognized by the number of vertices in its approximated contour. If the contour has three vertices, the object is likely a triangle.

2.4.5.1 Hough Transform

The Hough Transform is a popular technique for detecting shapes and lines in an image [61]. It is advantageous when the image contains incomplete or noisy shape information. The main idea behind the Hough Transform is to represent image features in a parameter space that makes it easier to detect the presence of specific shapes.

For line detection, the Hough Transform maps each point in the image space to a curve in the parameter space, defined by the line equation:

$$\rho = x \cos \theta + y \sin \theta \quad (2.25)$$

Where ρ is the perpendicular distance from the origin to the line, and θ is the angle between the ρ line and the x-axis. Points belonging to the same line will generate curves that intersect at a common point in the parameter space. The more points that belong to the same line, the higher the accumulation at the intersection point.

For circle detection, the Hough Transform maps each point in the image space to a sphere in the parameter space, defined by the circle equation:

$$(x - a)^2 + (y - b)^2 = r^2 \quad (2.26)$$

where (a, b) is the center of the circle, and r is the radius. Points belonging to the same circle will generate spheres that intersect at a common point in the parameter space, indicating the presence of a circle with a specific center and radius.

2.4.6 Thresholding

Thresholding is a simple yet effective method for image segmentation, often used to separate an object in an image from the background [49]. The idea of thresholding is to replace each pixel in an image with a black pixel if the image intensity I is less than some fixed constant T (that is, $I < T$) or a white pixel if the image intensity is more significant than that constant.

The simplest form of thresholding is global thresholding, which is defined by the equation:

$$g(x, y) = \begin{cases} 1 & \text{if } f(x, y) > T, \\ 0 & \text{otherwise,} \end{cases} \quad (2.27)$$

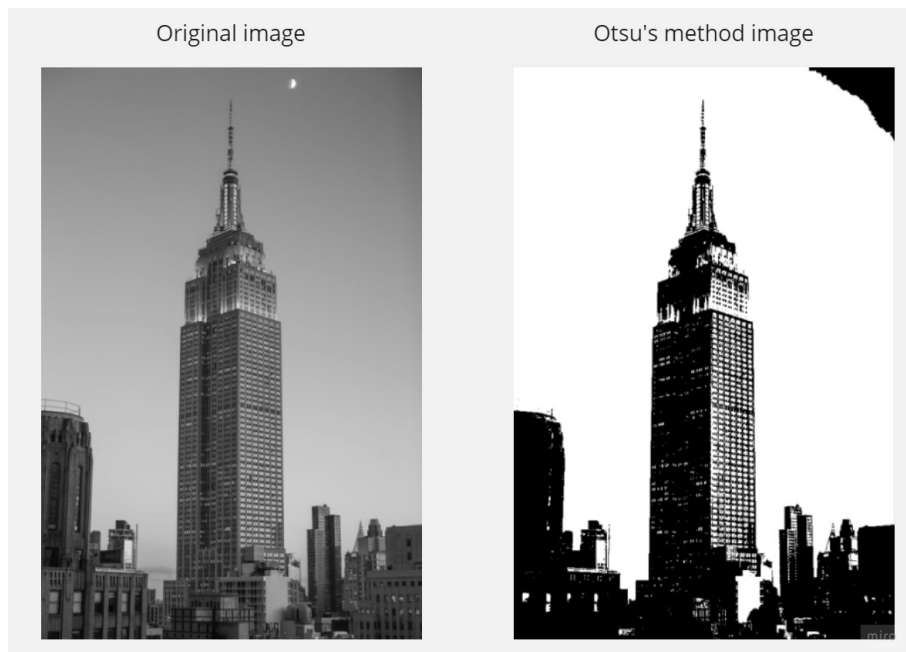
Where:

- $f(x, y)$ is the input image,
- $g(x, y)$ is the output (thresholded) image,
- T is the threshold value.

In many cases, the image's background and foreground intensities may not be distinct, making it challenging to select a suitable threshold. In such cases, an adaptive thresholding method might be used, where each pixel's threshold value is determined based on a statistical measure of its surrounding region [62].

Otsu's method is a popular adaptive thresholding method that assumes the image contains two classes of pixels (e.g., foreground and background), then calculates the optimum threshold separating the two classes so that their combined spread (intra-class variance) is minimal, or equivalently (because the total variance is constant) so that their inter-class variance is maximal [63].

Thresholding has wide applications in various image processing domains, including medical imaging [64], object tracking [65], and computer vision.



■ **Figure 2.9** Otsu's method for thresholding the image [66].

2.4.7 Morphological Operations

Morphological operations are a set of image processing techniques used to process and analyze the geometric structures within an image, typically binary or grayscale images [67]. The two fundamental morphological operations are dilation and erosion, which can be combined to form more complex operations such as opening, closing, and morphological gradient [4].

Dilation expands the boundaries of the objects in an image and is defined as:

$$(A \oplus B)(x, y) = \max_{(s,t) \in B} A(x - s, y - t) \quad (2.28)$$

Erosion shrinks the objects in an image and is defined as:

$$(A \ominus B)(x, y) = \min_{(s,t) \in B} A(x + s, y + t) \quad (2.29)$$

Where:

- A is the input image,
- B is the structuring element,
- (x, y) are the coordinates of the image,
- (s, t) are the coordinates of the structuring element.

Opening is an operation that smooths the contours, breaks narrow isthmuses, and eliminates thin protrusions. It is defined as the erosion of A by B , followed by dilation:

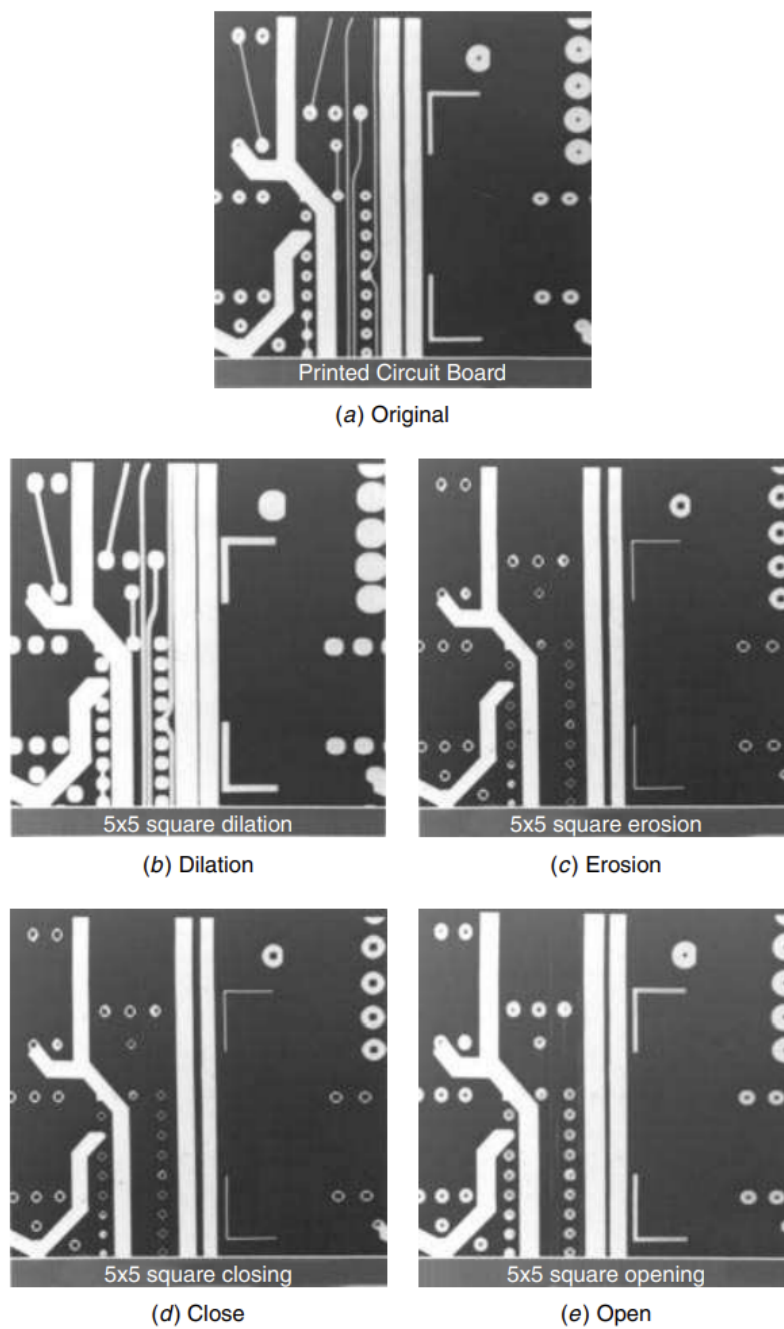
$$A \circ B = (A \ominus B) \oplus B \quad (2.30)$$

Closing is an operation that fuses narrow breaks and long, thin gulfs, eliminates small holes, and fills gaps. It is defined as the dilation of A by B , followed by erosion:

$$A \bullet B = (A \oplus B) \ominus B \quad (2.31)$$

Morphological Gradient is an operation that computes the difference between the dilation and erosion of an image, highlighting the regions of rapid intensity change:

$$\nabla_M A = (A \oplus B) - (A \ominus B) \quad (2.32)$$



■ **Figure 2.10** Two-dimensional gray scale image dilation, erosion, close, and open on a printed circuit board image [68].

2.5 Image Similarity

Image similarity is a fundamental concept in image processing and computer vision, referring to the quantification of how much two images resemble each other [69]. Depending on the specific application, many measures can be used to compute image similarity. Here are some commonly used methods:

- **Mean Squared Error (MSE):** This computes the average of the squared differences between corresponding pixels in two images. The MSE is given by:

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i, j) - K(i, j)]^2 \quad (2.33)$$

where $I(i, j)$ and $K(i, j)$ are the pixel intensities at location (i, j) in the two images, and m and n are the dimensions of the images [4].

- **Squared Difference:** This is a simple measure that calculates the total square difference between two images. It is given by:

$$SD = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i, j) - K(i, j)]^2 \quad (2.34)$$

where $I(i, j)$ and $K(i, j)$ are the pixel intensities at location (i, j) in the two images, and m and n are the dimensions of the images [4].

- **Correlation Coefficient:** The correlation coefficient measures the linear correlation between two datasets. In the context of image similarity, it can be used to measure how closely related two sets of pixel intensities are. It is given by:

$$r = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}} \quad (2.35)$$

where x_i and y_i are the pixel intensities in the two images, \bar{x} and \bar{y} are the mean pixel intensities, and n is the number of pixels [4].

- **Pearson Correlation Coefficient:** This is a case of the correlation coefficient when the data is normalized to have zero mean and unit variance. It is given by the same equation as above, with the understanding that the pixel intensities have been normalized [4].

- **Hu Moments:** Hu Moments are a set of seven numbers calculated from the image moments and used for image comparison. They are invariant to image transformations such as scaling, rotation, and translation. The Hu moments are given by:

$$\begin{aligned}
\phi_1 &= \eta_{20} + \eta_{02}, \\
\phi_2 &= (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2, \\
\phi_3 &= (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2, \\
\phi_4 &= (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2, \\
\phi_5 &= (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] \\
&\quad + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2], \\
\phi_6 &= (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] \\
&\quad + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03}), \\
\phi_7 &= (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] \\
&\quad - (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2],
\end{aligned} \tag{2.36}$$

where η_{pq} are the normalized central moments of the image [70].

- **Structural Similarity Index (SSIM):** SSIM compares local patterns of pixel intensities normalized for luminance and contrast. SSIM is designed to improve traditional methods like MSE by considering changes in structural information, luminance, and contrast [69].
- **Features from Convolutional Neural Networks (CNNs):** In the context of deep learning, features extracted from pre-trained CNNs are often used to compute image similarity. These features capture high-level semantic information about the images, allowing for more robust and flexible similarity comparisons [71].

Image similarity has a wide range of applications, including image retrieval [72], image registration [73], and object recognition [74]. Other applications involve image stitching [75], change detection [76], and remote sensing [77].

2.6 Image Augmentation

Image Augmentation is a strategy used to significantly increase the diversity of images available for training models without collecting new images. This provides a way to add variability and reduce overfitting in the model, making it possible to improve the performance of deep learning models [78, 79].

There are numerous techniques of image augmentation used in practice. Some of the most common ones include:

■ Affine Transformations:

- **Identity:** The identity matrix is a square matrix with ones on the diagonal and zeros elsewhere. It represents no transformation when applied to an image or a point. The 3x3 identity matrix is as follows:

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (2.37)$$

- **Rotation:** The image is rotated by angles α , β , and γ around the x , y , and z axes, respectively. First, the rotation matrices for each axis are calculated:

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha \\ 0 & \sin \alpha & \cos \alpha \end{bmatrix}, \quad R_y = \begin{bmatrix} \cos \beta & 0 & \sin \beta \\ 0 & 1 & 0 \\ -\sin \beta & 0 & \cos \beta \end{bmatrix}, \quad R_z = \begin{bmatrix} \cos \gamma & -\sin \gamma & 0 \\ \sin \gamma & \cos \gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (2.38)$$

The combined rotation matrix is obtained by multiplying the individual rotation matrices:

$$R = R_z \cdot R_y \cdot R_x. \quad (2.39)$$

- **Translation:** The image is translated (moved) by t_x horizontally and t_y vertically. The translation matrix is constructed to center the image around the origin and move it to the new position in the output space:

$$T = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \quad (2.40)$$

- **Scaling:** The image is scaled by factors s_x and s_y . The scaling matrix is applied to scale the image by these factors:

$$S = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (2.41)$$

- **Shear:** The image is distorted such that the lines parallel to the x -axis or y -axis are mapped to lines that remain parallel but are tilted by shearing angles α , β , and γ along the x , y , and z axes, respectively. The shearing operation can be represented with separate matrices for each axis:

$$SH_x = \begin{bmatrix} 1 & \alpha & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad SH_y = \begin{bmatrix} 1 & 0 & 0 \\ \beta & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad SH_z = \begin{bmatrix} 1 & 0 & \gamma \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (2.42)$$

The combined shear matrix is obtained by multiplying the individual shear matrices:

$$SH = SH_z \cdot SH_y \cdot SH_x. \quad (2.43)$$

- **Noise Injection:** Random noise is added to the image. The noise is often Gaussian distributed and is added to the pixel values of the image:

$$I' = I + \mathcal{N}(0, \sigma), \quad (2.44)$$

where I is the original image, I' is the image after noise injection, $\mathcal{N}(0, \sigma)$ represents Gaussian noise with mean 0 and standard deviation σ .

- **Blurring/Sharpening:** Blurring is achieved by convolving the image with a low-pass filter, whereas sharpening is achieved with a high-pass filter.
- **Contrast adjustment:** Contrast of an image can be altered by changing the range of intensity values. The new pixel value I' is calculated from the original pixel value I as follows:

$$I' = \alpha \cdot (I - \bar{I}) + \bar{I}, \quad (2.45)$$

where $\alpha > 1$ increases contrast and $\alpha < 1$ decreases contrast.

- **Brightness adjustment:** Brightness of an image can be increased or decreased by adding or subtracting a constant value to all pixels:

$$I' = I + \beta, \quad (2.46)$$

where $\beta > 0$ increases brightness and $\beta < 0$ decreases brightness.

- **Saturation adjustment:** Saturation refers to the intensity of color in the image. Adjusting the saturation involves converting the image to the HSV color space and modifying the S channel.
- **Elastic Deformation:** Elastic deformation involves simulating a non-linear deformation of the image.
- **MixUp:** MixUp is an augmentation strategy where two images and their corresponding labels are linearly interpolated with a random mix ratio.

Image augmentation is a powerful technique to improve the performance of deep learning models, especially when dealing with limited data.

2.7 Clustering Algorithms

Clustering is an unsupervised machine-learning technique that groups similar data points based on their features. Various clustering algorithms have been developed, each with its unique approach to creating clusters. In this section, we will discuss some popular clustering algorithms and the concept of metrics used in these algorithms.

- **Metric:** A metric is a function that measures the distance between data points in the feature space. Euclidean distance is a widely used metric in clustering algorithms, as it corresponds to the ordinary distance between points in Euclidean space. The Euclidean distance between two points x and y in d -dimensional space is defined as:

$$d(x, y) = \sqrt{\sum_{i=1}^d (x_i - y_i)^2} \quad (2.47)$$

- **K-means++:** K-means++ is an extension of the k-means clustering algorithm that improves the initialization of cluster centers [80]. The algorithm aims to minimize the within-cluster sum of squares, which can be defined as:

$$\sum_{i=1}^k \sum_{x \in C_i} \|x - \mu_i\|^2 \quad (2.48)$$

where k is the number of clusters, C_i is the set of data points in cluster i , and μ_i is the mean of the data points in cluster i . The k-means++ initialization procedure consists of the following steps:

1. Choose the first cluster center uniformly randomly from the data points.
 2. For each data point x , compute the distance $D(x)^2$ between x and the nearest cluster center that has already been chosen.
 3. Choose the next cluster center from the data points with probability proportional to $D(x)^2$.
 4. Repeat steps 2-3 until k cluster centers have been selected.
- **Mean Shift:** Mean shift is a nonparametric, iterative clustering algorithm that finds dense regions of data points by shifting points towards the mean of their local neighborhood [81]. The algorithm is robust to noise and can identify clusters with different shapes and sizes. The mean shift update formula is given by:

$$x_i \leftarrow x_i + m(x_i) \quad (2.49)$$

where $m(x_i)$ is the mean shift vector, computed as the weighted mean of the data points in the local neighborhood of x_i .

- **Spectral Clustering:** Spectral clustering algorithms use eigenvectors of the Laplacian matrix derived from a similarity graph to cluster data points [82]. The main steps involve constructing the similarity graph, computing the Laplacian matrix, and applying k-means clustering to the first k eigenvectors.
- **Agglomerative Clustering:** Agglomerative clustering is a hierarchical clustering algorithm that successively merges the closest pairs of clusters until a single cluster or a specified number of clusters is obtained [83]. The algorithm can produce a dendrogram representing the hierarchical structure of the data. Agglomerative clustering can be performed using different linkage criteria, such as single, complete, and average. The distance between two clusters, C_i and C_j , can be defined as:

$$d(C_i, C_j) = \min_{x \in C_i, y \in C_j} d(x, y) \quad (2.50)$$

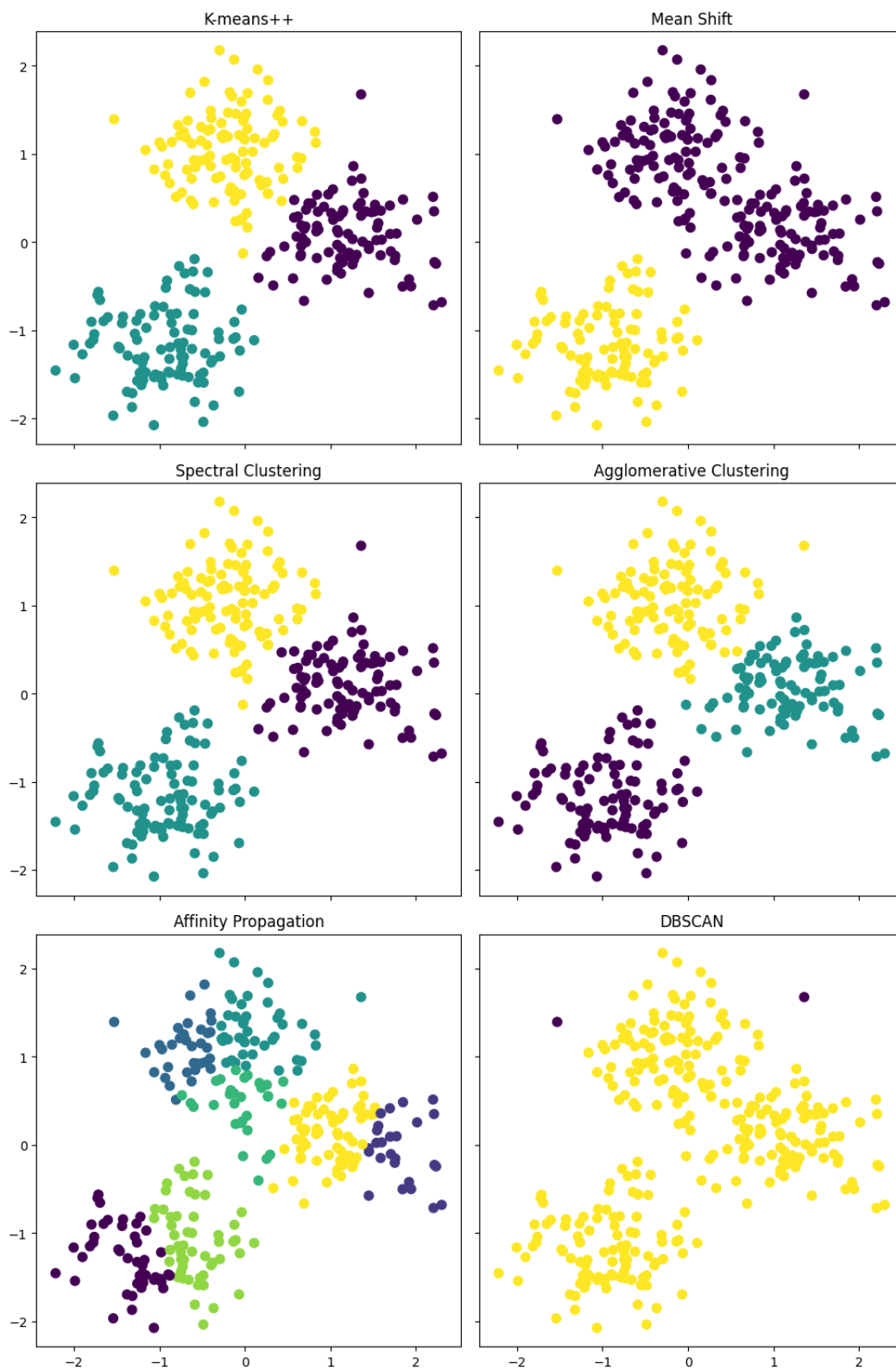
where $d(x, y)$ is the distance between data points x and y .

- **Affinity Propagation:** Affinity propagation is an iterative clustering algorithm that treats all data points as potential cluster centers, or exemplars, and iteratively refines the cluster membership [84]. The algorithm updates responsibility and availability messages between data points based on their similarity and does not require specifying the number of clusters beforehand.
- **OPTICS:** OPTICS (Ordering Points To Identify the Clustering Structure) is a density-based clustering algorithm similar to DBSCAN, but it addresses some of the limitations of DBSCAN, such as the need to choose a global density threshold [85]. OPTICS creates an augmented cluster ordering of the dataset that represents its density-based clustering structure, allowing for easier identification of clusters with varying densities.

The algorithm calculates a reachability distance for each data point and orders the points according to their reachability distances. The reachability distance of a point p from another point o is defined as the smaller value between the core distance of point o and the Euclidean distance between o and p :

$$\text{reachability} - \text{distance}(o, p) = \max(\text{core} - \text{distance}(o), \|o - p\|), \quad (2.51)$$

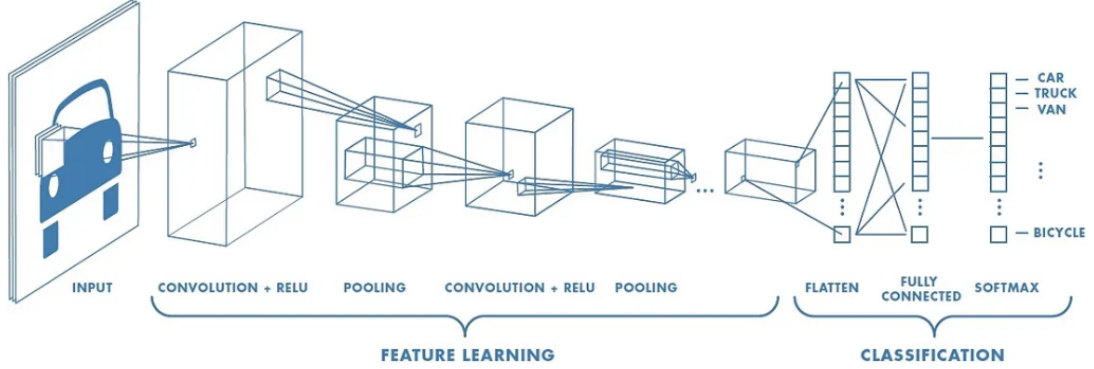
where $\text{core} - \text{distance}(o)$ is the smallest distance such that o is a core point, and $\|o - p\|$ is the Euclidean distance between o and p . The OPTICS algorithm processes the data points in the order specified by the reachability distances, forming clusters as it goes along.



■ **Figure 2.11** Different results from clustering algorithms performed on the same data.

2.8 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a class of deep learning models that have shown exceptional performance in various computer vision tasks, such as image classification, object detection, and segmentation [2, 11]. CNNs consist of multiple layers, including convolutional layers, activation layers, pooling layers, and fully connected layers, to learn hierarchical feature representations from input images.



■ **Figure 2.12** Neural network with many convolutional layers [86].

2.8.1 Convolutional Layers with Strides and Padding

The convolutional layer is the core building block of a CNN. It applies a set of learnable filters to the input image, allowing the network to learn local features from the input data. Each filter is convolved across the input image, computing the dot product between the filter's weights and the input data at each position. The convolution output is a feature map representing the presence of a specific feature at different locations in the input image [1]. To control the output size and the amount of information processed, strides and padding can be used.

A stride is a parameter that determines the step size to move the filter during the convolution process. A larger stride results in a smaller output size, reducing the computational complexity. Padding, on the other hand, is the process of adding extra pixels around the input image, which can help maintain the spatial dimensions of the output feature map and improve the performance of the CNN.

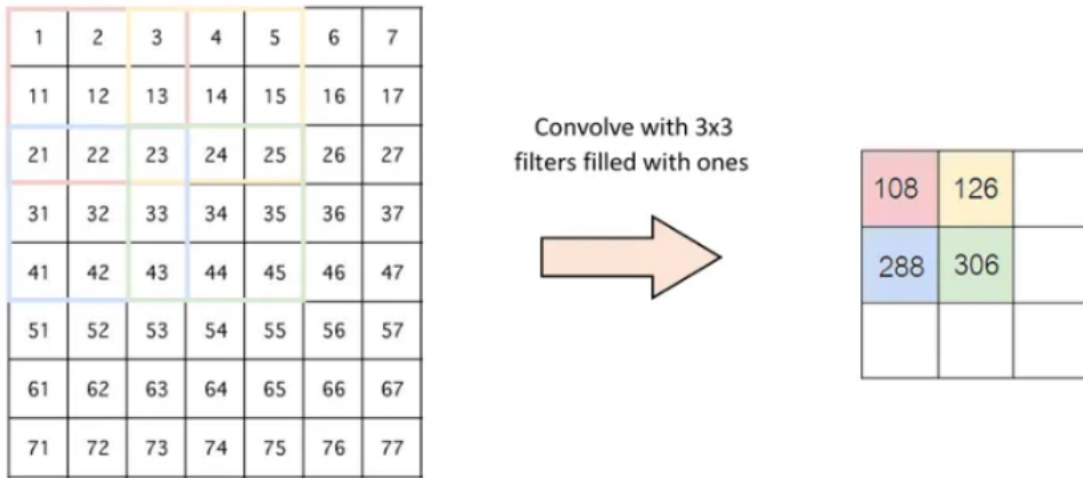
Mathematically, the convolution operation with strides and padding can be defined as:

$$(F * I)(x, y) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} F(i, j) \cdot I_p(x \cdot s_x - i, y \cdot s_y - j) \quad (2.52)$$

where F is the filter, I_p is the padded input image, $*$ denotes the convolution operation, (s_x, s_y) are the strides in the x and y directions, respectively. The padded input image, I_p , can be defined as:

$$I_p(x, y) = \begin{cases} I(x - p_x, y - p_y) & \text{if } 0 \leq x - p_x < w_I \text{ and } 0 \leq y - p_y < h_I \\ 0 & \text{otherwise} \end{cases} \quad (2.53)$$

where (p_x, p_y) are the padding sizes in the x and y directions, respectively, and (w_I, h_I) are the width and height of the input image.

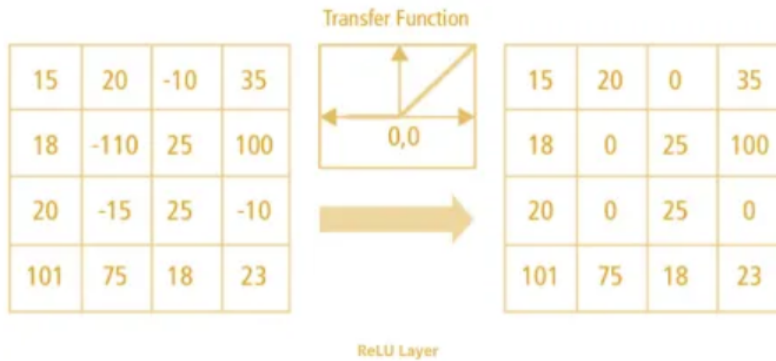


■ **Figure 2.13** Convolution operation [86].

2.8.2 Activation Layers

Activation layers introduce nonlinearity into the network, allowing the model to learn complex, non-linear relationships between the input data and the output. Commonly used activation functions include the Rectified Linear Unit (ReLU), the sigmoid function, and the hyperbolic tangent (tanh) function [1]. The ReLU activation function is defined as:

$$\text{ReLU}(x) = \max(0, x) \tag{2.54}$$



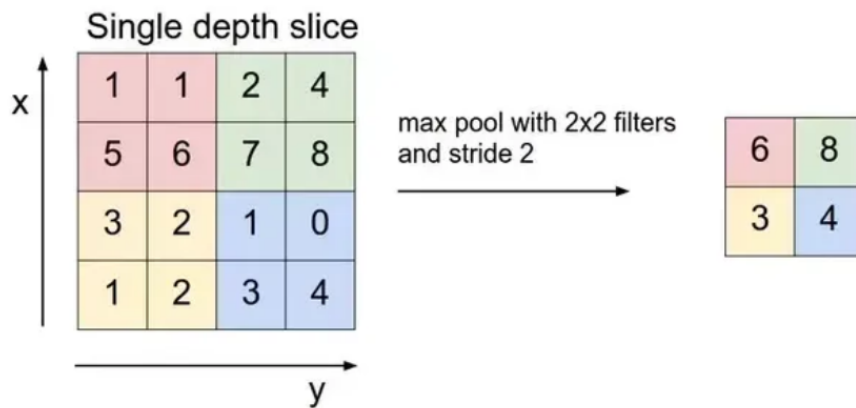
■ **Figure 2.14** ReLU activation function [86].

2.8.3 Pooling Layers

Pooling layers reduce the spatial dimensions of the feature maps, making the network more computationally efficient and invariant to small translations in the input data. The most common pooling operation is max-pooling, which selects the maximum value from a local neighborhood of the input data [1]. The max-pooling operation can be defined as:

$$\text{MaxPooling}(x, y) = \max_{i \in \{0, \dots, k-1\}} \max_{j \in \{0, \dots, k-1\}} x_{x+i, y+j} \quad (2.55)$$

where k is the size of the pooling kernel.

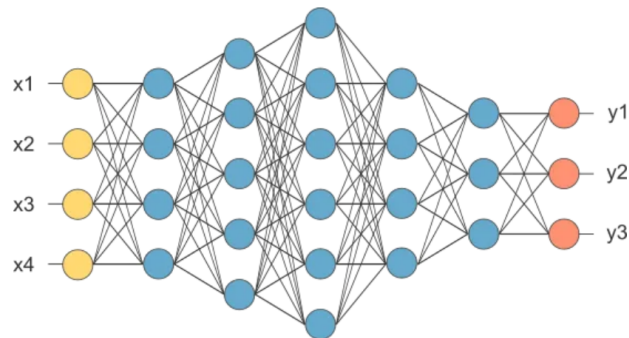


■ **Figure 2.15** Max pooling [86].

2.8.4 Fully Connected Layers

Fully connected layers, also known as dense layers, combine the local features learned by the previous layers and produce the final output of the network. Each neuron in a fully connected layer receives input from all the neurons in the previous layer, allowing the network to learn global relationships between the input data and the output [2]. In a typical CNN architecture, the last fully connected layer is followed by a softmax activation function, which produces the probabilities for the output classes.

When combined in a deep architecture, these layers allow CNNs to learn hierarchical feature representations from input images, resulting in high performance on various computer vision tasks.



■ **Figure 2.16** Fully connected layer [86].

2.8.5 Training CNNs

Training a CNN involves optimizing the weights of the filters and the neurons in the fully connected layers to minimize a loss function, which measures the difference between the predicted output and the ground truth labels. The optimization process is usually performed using stochastic gradient descent (SGD) or one of its variants, such as Adam [87].

During training, the gradients of the loss function concerning the network parameters are computed using the backpropagation algorithm [88]. The gradients are then used to update the network weights in a direction that minimizes the loss function. The learning rate, a hyperparameter that controls the step size of the weight updates, is crucial in determining the convergence and performance of the training process.

2.8.6 Regularization Techniques

Various regularization techniques are employed during training to prevent overfitting and improve the generalization of CNNs. Some standard regularization techniques include dropout, weight decay, and data augmentation [89, 1, 78].

- **Dropout:** Dropout is a regularization technique that randomly drops neurons from the network during training with a certain probability, preventing the network from relying too heavily on any single neuron. This leads to a more robust model that can generalize better to unseen data [89].
- **Weight Decay:** Weight decay, also known as L2 regularization, adds a penalty term to the loss function proportional to the squared Euclidean norm of the weights. This encourages the network to learn smaller weights, leading to a simpler and more stable model [1].
- **Data Augmentation:** Data augmentation involves applying random transformations to the input images, such as rotation, scaling, and flipping, to create new training samples. This increases the diversity of the training data, helping the network learn more robust and invariant features [78].

By employing these regularization techniques, CNNs can be trained to achieve high performance on various computer vision tasks while maintaining good generalization to unseen data.

2.8.7 Transfer Learning

Transfer learning is a technique that leverages the knowledge acquired by a pre-trained model on a large dataset to improve the performance of a new model on a related but smaller dataset [90, 91]. In the context of CNNs, this is often done by using the learned weights of a pre-trained model as the initial weights for a new model, which is then fine-tuned on the target task using the smaller dataset.

The underlying assumption of transfer learning is that the features learned by the pre-trained model on the large dataset are general and can help solve the new task. This is particularly effective for CNNs, as the early layers of the network learn low-level features such as edges and textures, which are common across many visual tasks [90].

To fine-tune the pre-trained model, the following steps are typically performed:

1. Replace the last layer(s) of the pre-trained model with new layers specific to the target task.
2. Freeze the weights of the early layers in the network, preventing them from being updated during training.
3. Train the new model on the target task for a few epochs, updating only the weights of the newly added layers.

4. Optionally, unfreeze some or all of the early layers and continue training the entire network with a lower learning rate.

Transfer learning allows for faster convergence and improved performance compared to training a model from scratch, especially when the target dataset is small or the task is closely related to the one the pre-trained model was initially trained on.

2.8.8 Popular CNN Architectures

Over the years, numerous CNN architectures have been proposed for various computer vision tasks, such as image classification, object detection, and semantic segmentation. These architectures have significantly advanced the state of the art in the field. In this section, we discuss some popular CNN architectures and their contributions to image classification and object detection.

2.8.8.1 Image Classification

For image classification tasks, several influential CNN architectures have emerged, including:

- **LeNet** [92]: An early convolutional neural network that demonstrated the effectiveness of CNNs for handwritten digit recognition.
- **AlexNet** [11]: A deeper and wider CNN that won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012, significantly outperforming traditional computer vision methods.
- **VGG** [93]: Known for its simplicity and uniform architecture, VGG demonstrated the benefits of increasing network depth for improved performance.
- **ResNet** [12]: Introduced the concept of residual connections, enabling the training of very deep networks and achieving state-of-the-art performance on various benchmarks.
- **EfficientNet** [94]: A family of CNNs that leverages a systematic approach to model scaling, achieving state-of-the-art accuracy while maintaining computational efficiency.

2.8.8.2 Object Detection

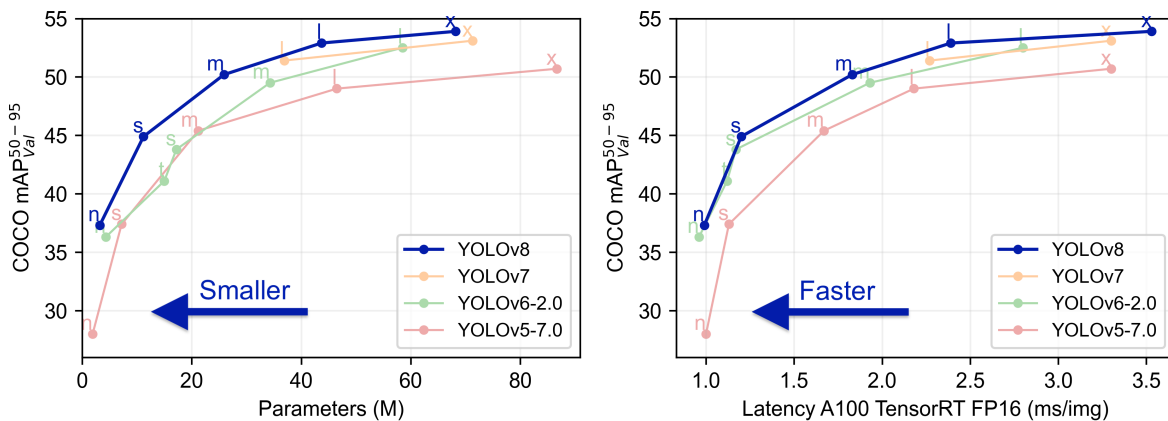
In the realm of object detection, the following architectures have been particularly influential:

- **R-CNN** [95]: A pioneering approach that combines region proposals with CNNs for object detection, providing a significant boost in performance compared to previous methods.
- **Fast R-CNN** [96]: An improvement over R-CNN that significantly speeds up the detection process by sharing computation across region proposals.
- **YOLO (You Only Look Once)** [7]: A real-time object detection system that uses a single CNN to simultaneously predict multiple bounding boxes and class probabilities for those boxes, fundamentally differing from region proposal-based methods like R-CNN. YOLO divides the input image into a grid and predicts bounding boxes and class probabilities for each grid cell. The final object detections are obtained by thresholding and non-maximum suppression of the predicted boxes.

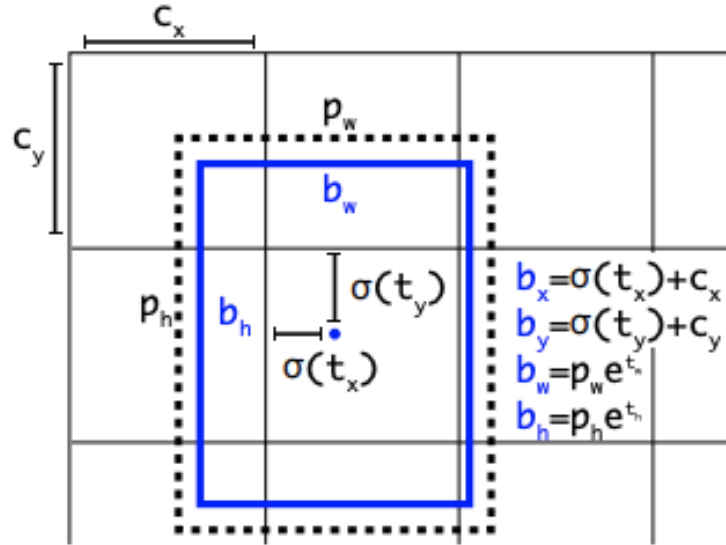
YOLO, in particular, has been improved through several versions. These improvements have enhanced accuracy and efficiency, making YOLO one of the most popular choices for real-time object detection tasks.

Version	Date	Anchor	Framework	Backbone
YOLO AP (%)				
YOLO	2015	No	Darknet	Darknet24
63.4				
YOLOv2	2016	Yes	Darknet	Darknet24
63.4				
YOLOv3	2018	Yes	Darknet	Darknet53
36.2				
YOLOv4	2020	Yes	Darknet	CSPDarknet53
43.5				
YOLOv5	2020	Yes	Pytorch	Modified CSP v7
55.8				
PP-YOLO	2020	Yes	PaddlePaddle	ResNet50-vd
45.9				
Scaled-YOLOv4	2021	Yes	Pytorch	CSPDarknet
56.0				
PP-YOLOv2	2021	Yes	PaddlePaddle	ResNet101-vd
50.3				
YOLOv5	2021	Yes	Pytorch	CSPDarknet
55.4				
YOLOX	2021	No	Pytorch	Modified CSP v5
51.2				
PP-YOLOE	2022	No	PaddlePaddle	CSPRepResNet
54.7				
YOLOv6	2022	No	Pytorch	EfficientRep
52.5				
YOLOv7	2022	No	Pytorch	RepConvN
56.8				
DAMO-YOLO	2022	No	Pytorch	MAE-NAS
50.0				
YOLOv8	2023	No	Pytorch	YOLO v8
53.9				

■ **Table 2.2** Comparison of various YOLO versions and their performance [97].



■ **Figure 2.17** Performance of YOLOv8 against older versions [98].



■ **Figure 2.18** YOLOv8 is an anchor-free model. This means it predicts directly the center of an object instead of the offset from a known anchor box [99].

2.9 Measuring Model Performance

Evaluating the performance of a machine learning model is crucial for understanding its effectiveness and comparing it with other models. Several metrics can be used to measure the performance of classification models, such as accuracy, precision, recall, F1 score, confusion matrix, mAP, and AP [100, 101]. In this section, we will discuss these metrics and their corresponding equations.

2.9.1 Accuracy

Accuracy is the most straightforward metric for classification tasks. It measures the proportion of correct predictions made by the model out of the total number of predictions. The accuracy is defined as:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} = \frac{TP + TN}{TP + TN + FP + FN} \quad (2.56)$$

where TP, TN, FP, and FN are true positives, true negatives, false positives, and false negatives, respectively.

2.9.2 Precision and Recall

Precision and recall are two important metrics that focus on the model's performance concerning positive class predictions. Precision, also known as a positive predictive value, measures the proportion of true positive predictions out of all positive predictions. Recall, also known as sensitivity or true positive rate, measures the proportion of true positive predictions out of all actual positive instances.

$$\text{Precision} = \frac{TP}{TP + FP} \quad (2.57)$$

$$\text{Recall} = \frac{TP}{TP + FN} \quad (2.58)$$

2.9.3 F1 Score

The F1 score is the harmonic mean of precision and recall. It is a commonly used metric for binary classification problems, as it balances the trade-off between precision and recall. The F1 score is defined as:

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2TP}{2TP + FP + FN} \quad (2.59)$$

2.9.4 Confusion Matrix

A confusion matrix is a table that summarizes the performance of a classification model by presenting the actual class values and predicting class values. The matrix is organized as follows:

- True Positives (TP): The number of positive instances correctly predicted as positive.
- True Negatives (TN): The number of negative instances correctly predicted as negative.
- False Positives (FP): The number of negative instances incorrectly predicted as positive.
- False Negatives (FN): The number of positive instances incorrectly predicted as negative.

For multiclass classification, the confusion matrix is extended to include multiple rows and columns representing each class. The diagonal elements of the matrix represent the correct predictions for each class, while the off-diagonal elements represent the misclassifications.

2.9.5 Mean Average Precision and Average Precision

Mean Average Precision (mAP) is a widely used metric for measuring the performance of object detection and segmentation models, especially in multiclass classification tasks. mAP calculates the average precision (AP) for each class and takes the mean of all class AP values. AP is the area under the Precision-Recall curve, which is a plot of precision and recall at different classification thresholds.

The mAP is calculated as follows:

$$\text{mAP} = \frac{1}{N} \sum_{i=1}^N \text{AP}_i \quad (2.60)$$

where N is the number of classes and AP_i is the average precision for class i . The AP for each class is calculated by integrating the Precision-Recall curve or by using the following approximation:

$$\text{AP} = \sum_{k=1}^n (R_k - R_{k-1}) P_k \quad (2.61)$$

where n is the number of recall values, R_k is the recall at rank k , R_{k-1} is the recall at rank $k - 1$, and P_k is the precision at rank k .

Chapter 3

Analysis

This chapter discusses the crucial aspects of choosing the right playing cards, camera apparatus, and development environment.

3.1 Physical Deck of Cards

To begin with, we will require an actual deck of cards. For this purpose, we will use the Rider Back design with the Bicycle Playing Cards¹. Bicycle playing cards are a well-known brand extensively used in various card games, such as poker, bridge, and blackjack [102]. These cards are manufactured by the United States Playing Card Company (USPCC), the world's largest playing card producer [103, 104]. For our dataset creation, we will use these cards as they are a popular choice among players and can provide us with a diverse range of card images [105].

3.1.1 Cards Design and Dimensions

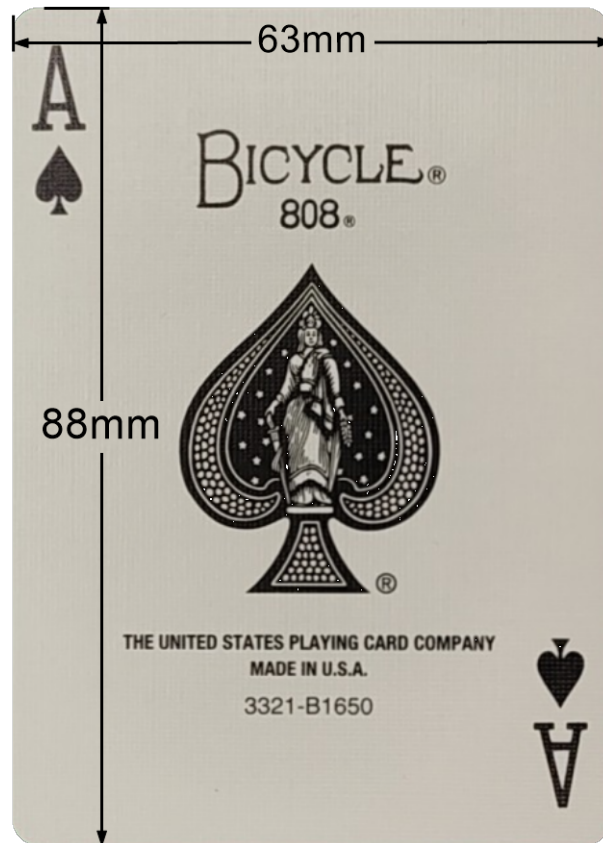
Bicycle cards adhere to the French deck pattern, containing 52 cards (13 in each of two red and two black suits) and excluding the two jokers to train the dataset [102]. The Bicycle trademark appears on the Ace of Spades, and the current decks contain two information/instruction cards. The most popular design for Bicycle playing cards is the Rider Back design, available in various formats.

These cards come in standard indexes:

- poker-size (3.5 by 2.5 inches [8.9 cm × 6.4 cm])
- bridge-size (3.5 by 2.25 inches [8.9 cm × 5.7 cm])
- pinochle decks, as well as "Jumbo Index" poker decks
- low vision cards designed for the visually impaired

Manufacturers also produce other cards with varying backs, sizes, colors, and custom designs for magic tricks, novelty purposes, and collectors' items [104]. For our purposes, we will be utilizing the standard poker-size deck.

¹<https://bicyclecards.com/>



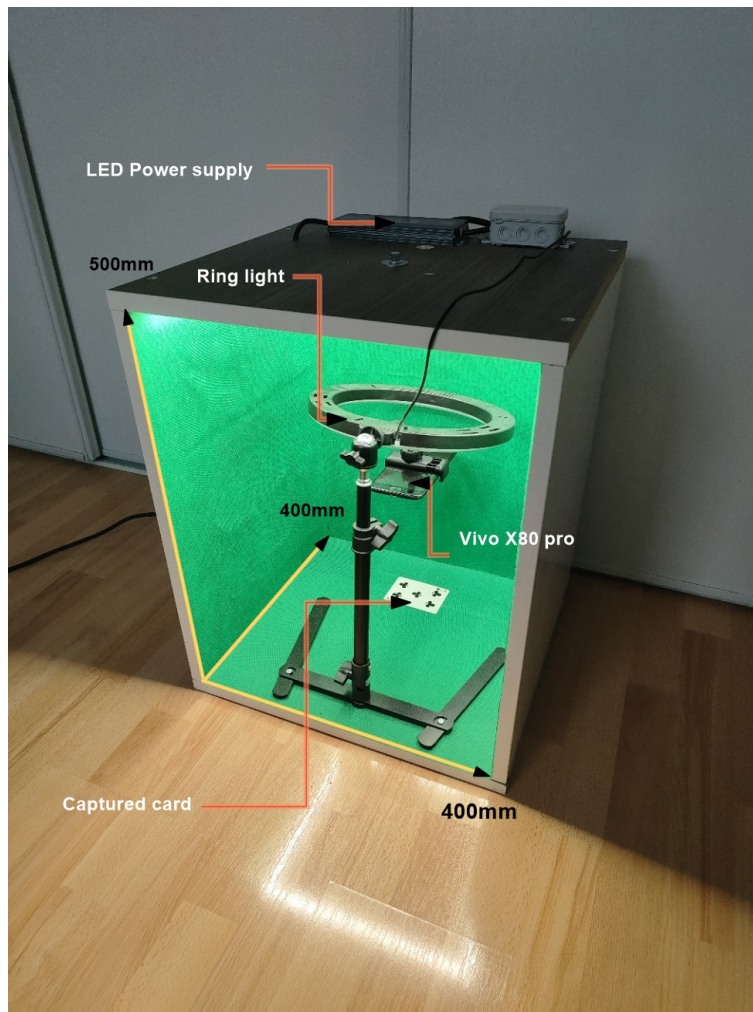
■ **Figure 3.1** Exact measurements with a caliper of the chosen playing card.

3.2 Camera Apparatus

Having acquired the physical deck of cards, the subsequent step entails developing a functional capturing device to take images of the cards. This crucial process component requires constructing a custom box with a mobile phone holder explicitly designed for image capture [5]. In addition, the box will be covered with the green screen fabric, a choice to facilitate easy background removal and post-processing of the images [106].

Furthermore, the box is equipped with LED lighting inside, ensuring the cards receive uniform illumination, which is critical for accurate feature extraction and analysis [107]. Finally, a vital apparatus component will be a mobile phone tripod with a ring light. This ring light will help capture clear, sharp images of the cards, significantly improving the overall quality of the captured images [69].

By creating an optimized environment for capturing card images, we can guarantee that the computer vision algorithms will have the best possible input data to work with [3]. This attention to detail in the capturing process will increase the accuracy of the playing card recognition system and ensure its effectiveness and reliability across various card designs and lighting conditions.



■ Figure 3.2 Manufactured camera apparatus.

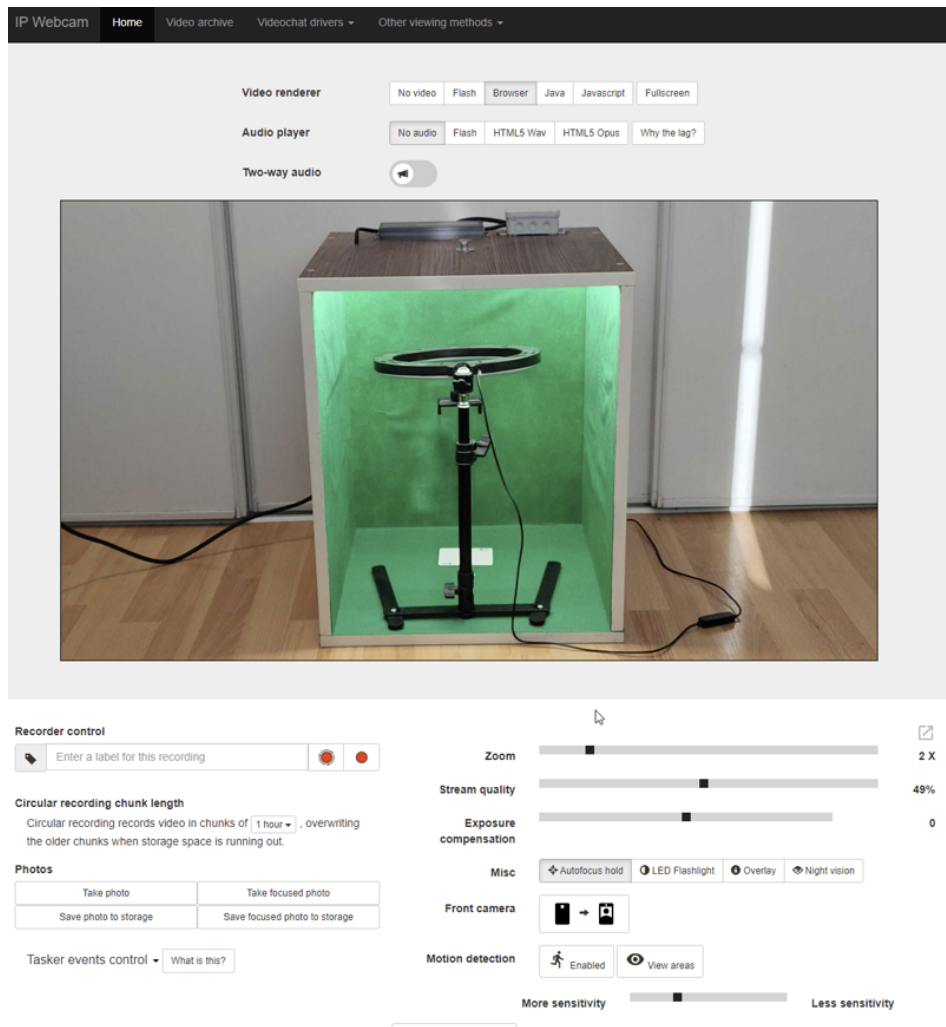
3.3 Chosen Camera and Software

We captured playing cards using a mobile phone camera after finding other webcams and IP cameras unsuitable for our needs due to limited settings and image quality [5]. We opted for a mobile phone with the IP webcam app ², which provided high-quality images and more flexibility in adjusting camera settings [108].

Industrial-type cameras were deemed unnecessary for our project, as playing cards are small and easily captured using a mobile phone camera [3]. Industrial cameras can be costly and require specialized equipment, making them impractical for our purposes.

Using the IP webcam app, we captured high-resolution images of playing cards with minimal noise and distortion [4]. The mobile phone was positioned above the cards, and the app was configured for low-latency streaming and high-quality resolution [69]. This approach was simple, cost-effective, and met our image quality requirements [34].

²https://play.google.com/store/apps/details?id=com.pas.webcam&hl=en_US



■ Figure 3.3 IP webcam interface³.

Filter	Image	Filter	Image
-30% exposure		-20% exposure	
-10% exposure		0% exposure	
+10% exposure		+20% exposure	
+30% exposure		Incandescent lighting	
Fluorescent lighting		Warm fluorescent lighting	
Daylight		Cloudy daylight	
Twilight		Shade	

■ Table 3.1 Filters used in the IP webcam app with corresponding images of the white texture of the card.

³<http://192.168.31.250:8080> (local address based on server settings)

3.4 Working Environment and Libraries

We created our playing card recognition dataset using the Jupyter Notebook⁴ environment and a Python⁵ 3.9 kernel with Anaconda⁶ interpreter.

Library	Description	URL
OpenCV	Computer vision library for image processing and object recognition.	https://opencv.org
NumPy	Numerical computing library for array and matrix manipulation.	https://numpy.org
Matplotlib	Data visualization library for creating graphs and charts.	https://matplotlib.org
tqdm	Fast progress bar library for loops and iterable objects.	https://github.com/tqdm/tqdm
PIL	Image processing library for opening, manipulating, and saving images.	https://pillow.readthedocs.io/en/stable/
pickle	Module for serializing and deserializing Python objects.	https://docs.python.org/3/library/pickle.html
sklearn	Machine learning library for data mining and analysis.	https://scikit-learn.org
concurrent.futures	Module for asynchronous execution of callables using thread or process pools.	https://docs.python.org/3/library/concurrent.futures.html
threading	Module for creating and managing threads for concurrent task execution.	https://docs.python.org/3/library/threading.html
requests	Library for making HTTP requests and working with RESTful APIs.	https://docs.python-requests.org
BeautifulSoup	Library for parsing HTML/XML documents and extracting data from web pages.	https://www.crummy.com/software/BeautifulSoup/
skimage	Image processing library for image manipulation and analysis.	https://scikit-image.org
Ultralytics YOLO	Real-time object detection and classification library (YOLO system).	https://github.com/ultralytics/ultralytics

■ **Table 3.2** List of used libraries, their descriptions, and URLs.

⁴<https://jupyter.org/>

⁵<https://www.python.org/>

⁶<https://www.anaconda.com/>

Realization

This chapter outlines our process for developing a robust playing card detection and recognition system with two different approaches.

4.1 Computer Vision Approach

4.1.1 Data

A total of 728 images (14 sets of 52 cards) of playing cards have been successfully captured in a controlled environment with a highly contrasted background and various filters. The playing cards were captured in both Full HD (1920 x 1080) and 2K (2560 x 1440) resolutions, ensuring a high level of detail and clarity in each image.

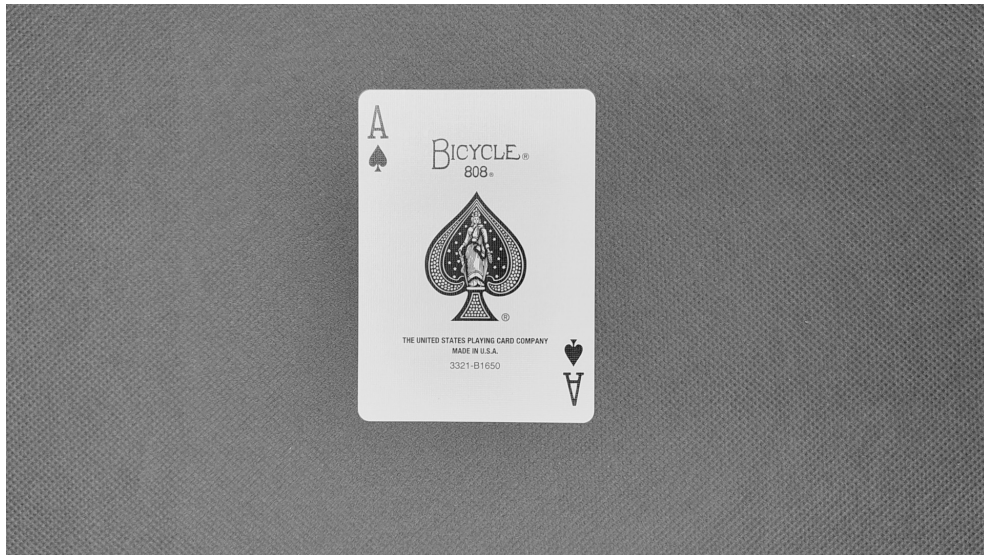


■ Figure 4.1 Captured image of the card.

4.1.2 Card Extraction

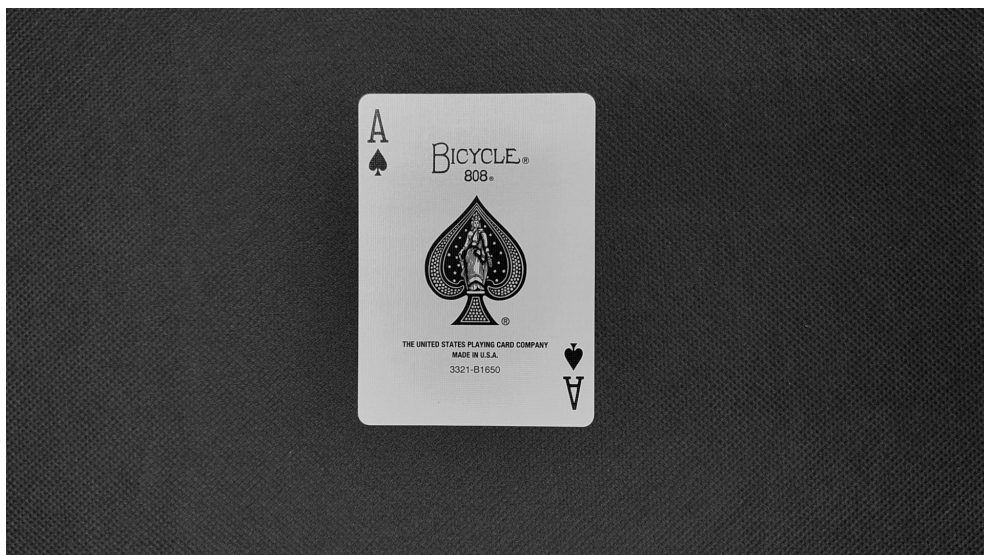
The card extraction process involves several steps:

1. **Preprocessing:** First, the input image is loaded from a file (in *.jpg format) and then converted to grayscale from RGB space, simplifying the image for more efficient processing.



■ **Figure 4.2** Grayscale image of the card.

2. **Gamma correction:** Additionally, uniform gamma correction is applied to the image to achieve consistent lighting. This is the most crucial step in the preprocessing stage since it allows to extraction of the cards in too-light or too-dark images. For these purposes universal gamma correction method was created:



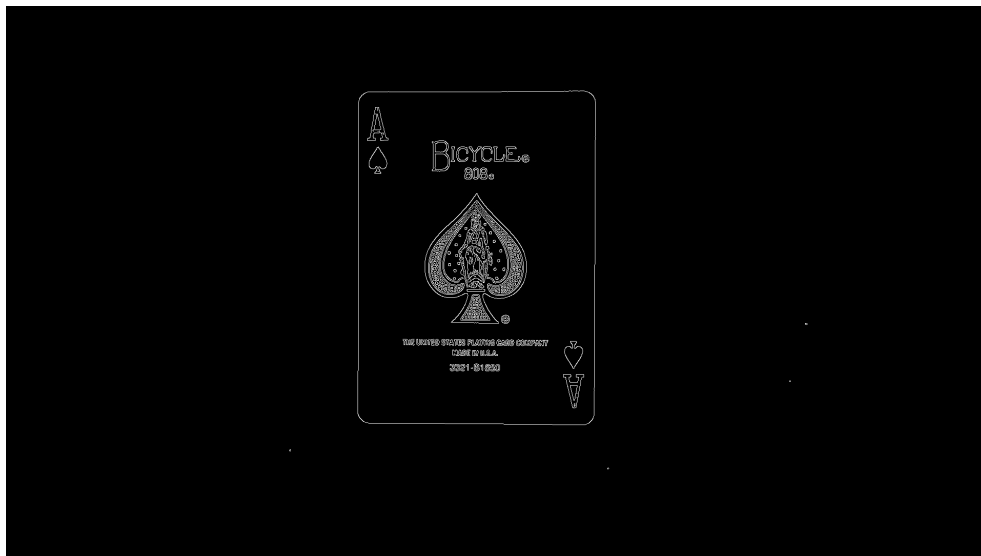
■ **Figure 4.3** Gamma corrected image of the card.

- 3. Edge smoothing:** The next step is to apply the bilateral filter to the grayscale image. This step helps to smooth the edges, preserving the overall structure and reducing noise.



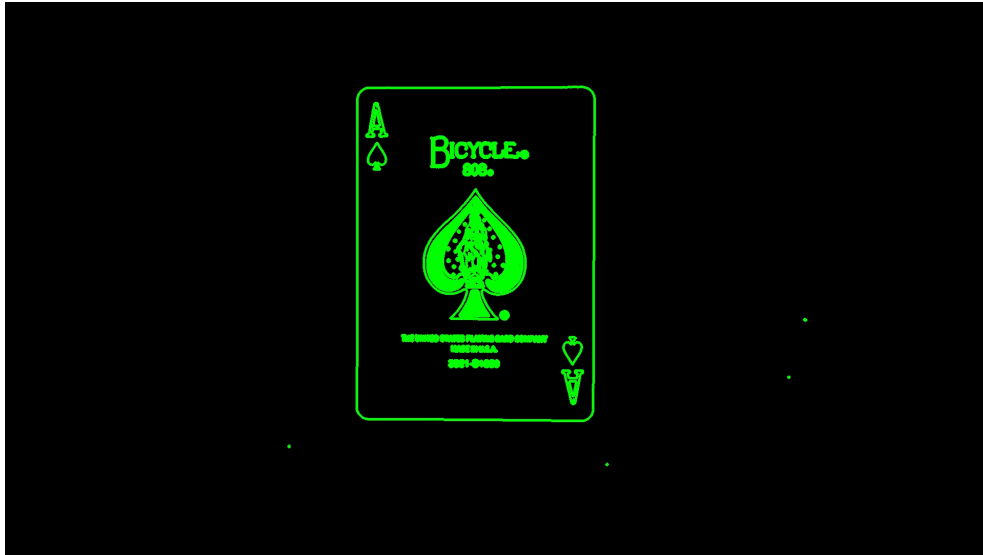
■ **Figure 4.4** Applied bilateral filter.

- 4. Edge detection:** The Canny edge detection algorithm is applied to the smoothed image. This step highlights the edges in the image, which are essential for identifying card boundaries.

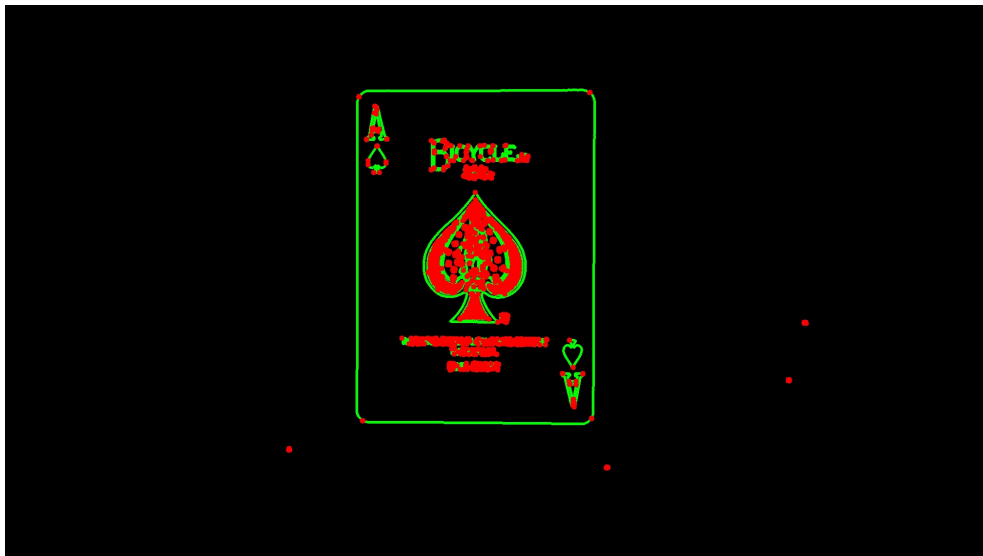


■ **Figure 4.5** Canny edge detection.

5. **Contour approximation:** Contours in the Canny image are detected and approximated with 4 points, which aids in identifying rectangular shapes representing the cards. It is important to note that the points were approximated using the Ramer–Douglas–Peucker algorithm.

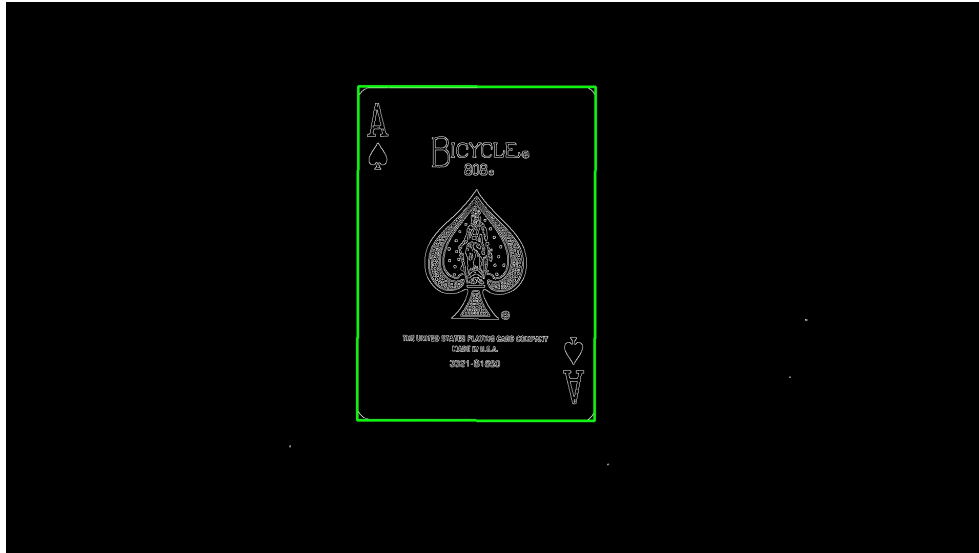


■ Figure 4.6 Found contours.



■ Figure 4.7 Approximated contours with four points.

- 6. Filtering contours:** Contours are filtered based on area, rectangularity, and aspect ratio. This step ensures that only contours representing cards are retained, discarding unrelated shapes.



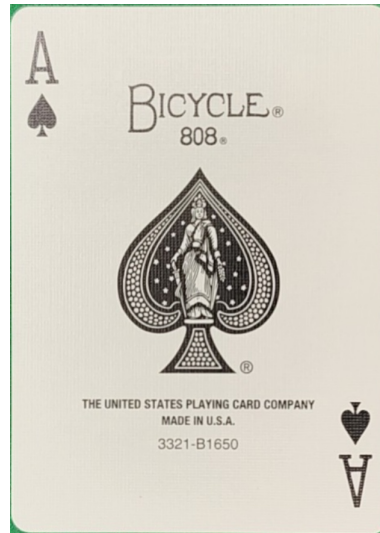
■ **Figure 4.8** Filtered contours based on rectangularity and aspect ratio.

- 7. Highlighting contours:** The largest contour representing a card is highlighted in the input image, providing visual feedback on the card detection process.



■ **Figure 4.9** Largest contour displayed.

8. **Cropping and resizing:** The original image is cropped and resized based on the card box points and the specified card width and height. This step extracts the card from the image, preparing it for further analysis or recognition.



■ **Figure 4.10** Extracted card.

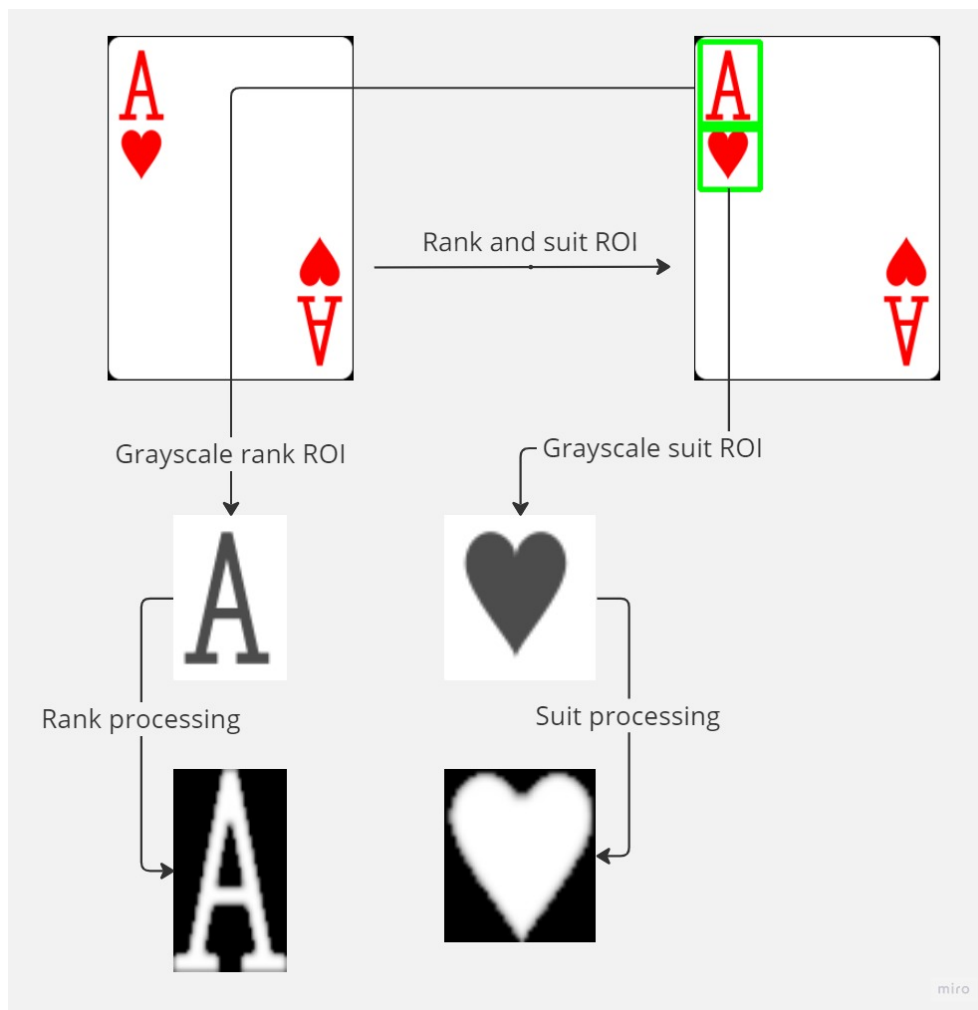
9. **Card clipping:** Finally, the extracted card is clipped with a defined mask for discarding any unwanted edges from the original image such that the extracted card resembles a real-world appearance with rounded corners. Also, the background of the extracted card is made transparent with an alpha layer.



■ **Figure 4.11** Clipped card with transparent background.

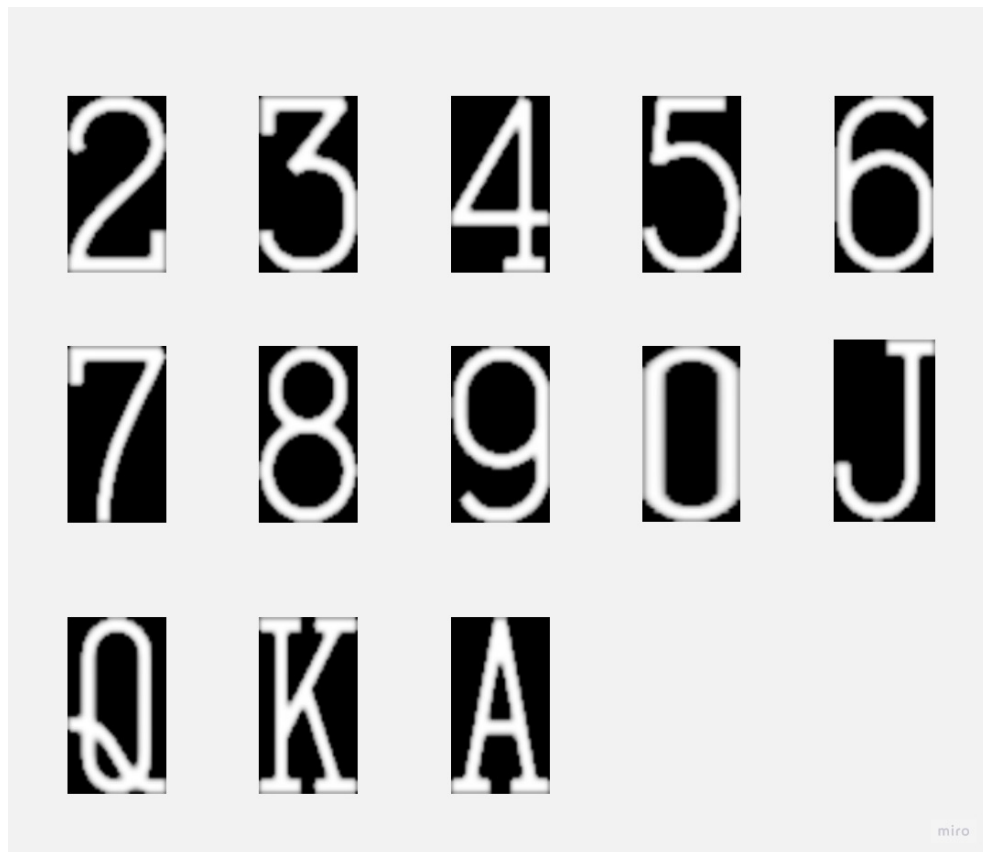
4.1.3 Classification Preparations

A set of reference templates has been created for each card rank and suit to enable rank and suit identification. Typically, the rank and suit are extracted from a set of reference cards, and image processing techniques are used to isolate the rank and suit (ROI) in each reference image. The isolated rank and suit images are then saved as grayscale templates for further use. A customized deck¹, an already-created deck of cards with the required card ranks and suits in SVG format, is utilized for this purpose.

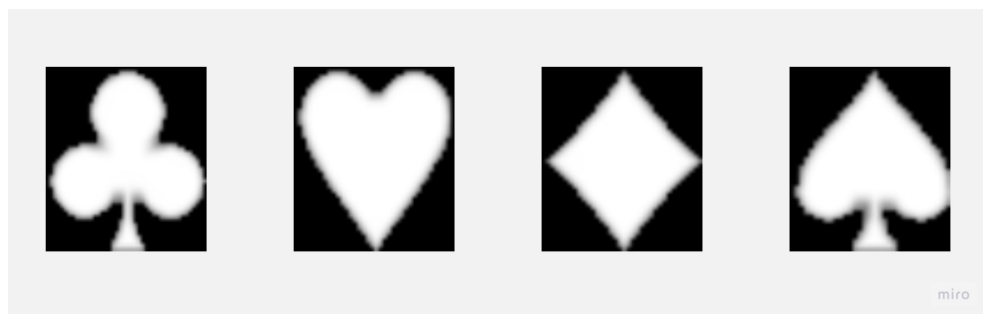


■ **Figure 4.12** Process of creating the reference templates.

¹<https://www.me.uk/cards/makeadeck.cgi>



■ Figure 4.13 Created rank templates.













■ Figure 4.14 Created suit templates.



Another preparation step is to prepare lookup tables (LUT) for color identification. We can divide this process into a couple of procedures:

1. The process began with the ISCC/NBS², a system for color names based on a set of 12 basic color terms and a small set of adjectives. The RGB color values from this system were then converted to the CIE Lab and HSV color spaces. The CIE Lab color space provides a more perceptually uniform representation of colors, which is crucial for accurate distance calculations between colors. The HSV color space, on the other hand, is helpful for color-based feature extraction in image analysis due to its closer alignment with how humans perceive color. The exact conversions were also applied to the defined classification colors.

■ **Table 4.1** Colors examples from ISCC/NBS and their different color space values

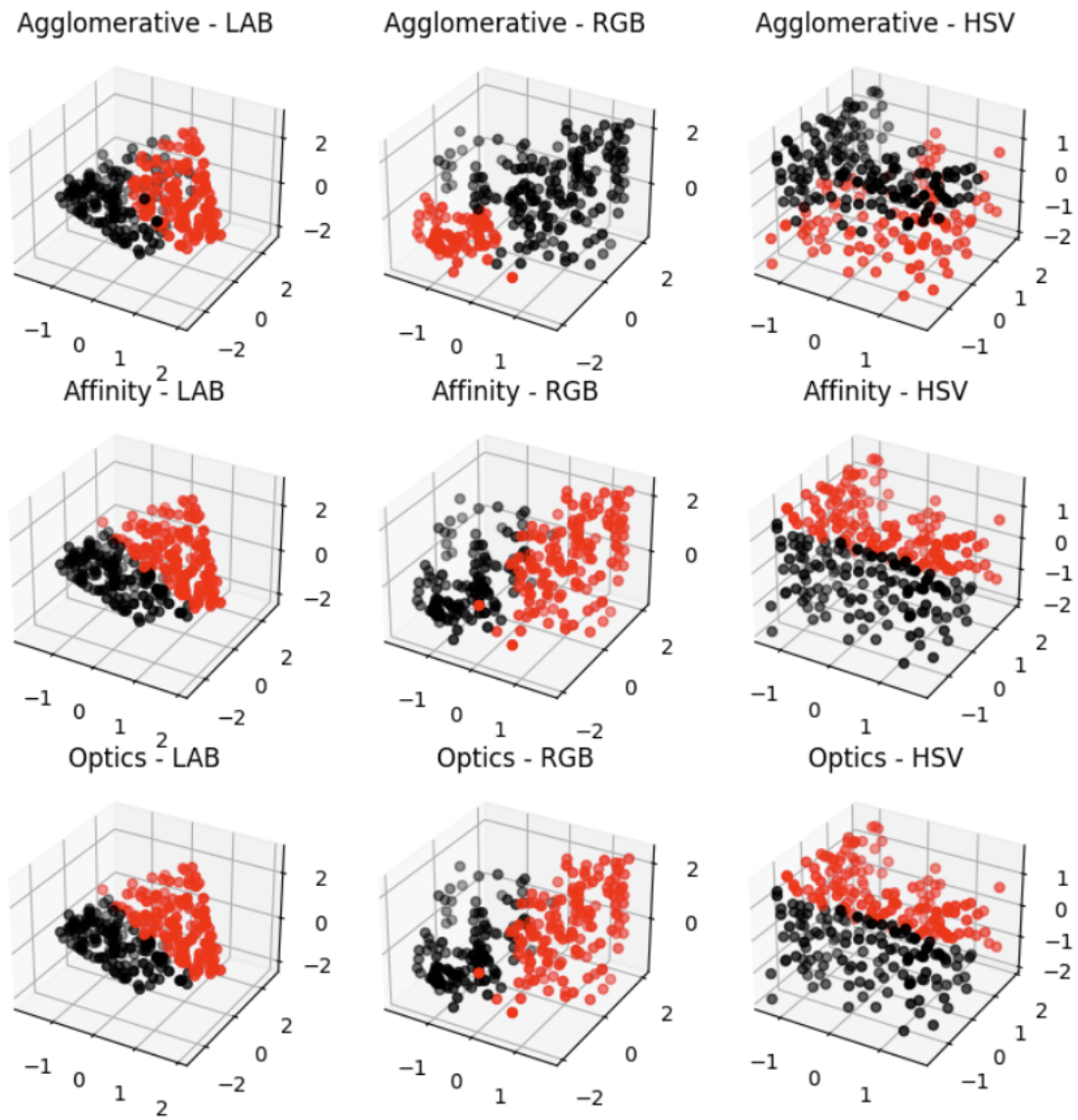
Color Name Color Sample	RGB Value	HSV Value	CIE Lab Value
Dark Gray 	(85, 85, 85)	(0, 0, 33)	(36.15, 0.00, 0.00)
Deep Pink 	(228, 113, 122)	(355, 50, 89)	(61.90, 46.17, 17.15)
Deep Purple 	(96, 47, 107)	(289, 56, 42)	(28.20, 30.25, -25.77)
Vivid Reddish Orange 	(226, 88, 34)	(17, 85, 89)	(56.10, 53.05, 56.98)
Vivid Blue 	(0, 161, 194)	(190, 100, 76)	(59.71, -35.19, -12.17)
Vivid Yellow 	(243, 195, 0)	(48, 100, 95)	(81.33, 7.59, 81.86)
Deep Yellowish Green 	(0, 98, 45)	(148, 100, 38)	(35.82, -35.94, 22.65)
Deep Reddish Orange 	(170, 56, 30)	(11, 82, 67)	(40.99, 46.52, 41.65)
Strong Green 	(0, 121, 89)	(164, 100, 47)	(44.68, -36.50, 9.09)
Vivid Orange 	(243, 132, 0)	(33, 100, 95)	(66.89, 38.91, 73.23)

■ **Table 4.2** Defined classification colors and their different color space values

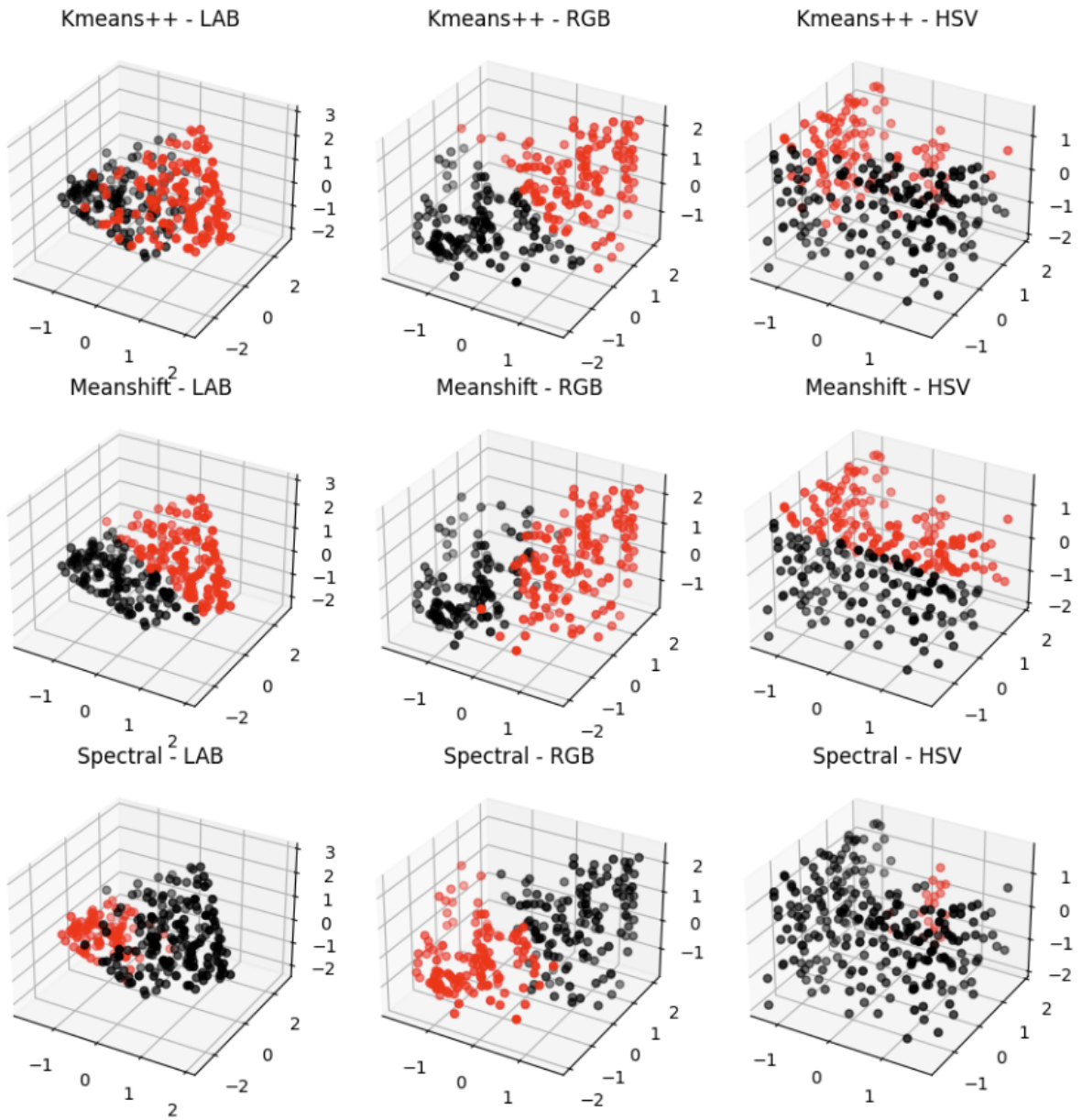
Color Name Color Sample	RGB Value	HSV Value	CIE Lab Value
Red 	(85, 85, 85)	(0, 100, 100)	(54.29, 80.81, 69.89)
Black 	(0, 0, 0)	(0, 0, 0)	(0.00, 0.00, 0.00)

²https://www.w3schools.com/colors/colors_nbs.asp

2. The next idea involves mapping the ISCC/NBS colors to the classification colors in the RGB, HSV, and CIE Lab color spaces. This is achieved by employing multiple clustering and nearest-neighbor methods:
 - a. **KMeans++:** The KMeans++ algorithm partitions the colors into clusters, using the classification colors as initial cluster centroids. This ensures that the ISCC/NBS colors are grouped based on their proximity to the classification colors. The algorithm iteratively updates the centroids and assigns each color to the nearest centroid until convergence.
 - b. **MeanShift:** MeanShift, a non-parametric algorithm, clusters colors based on their density in the color space. It does not require specifying the number of clusters in advance. After applying MeanShift, each color is mapped to the nearest user-defined color.
 - c. **Spectral Clustering:** The Spectral Clustering algorithm employs graph partitioning to cluster colors. Treating each color as a graph node, the algorithm creates edges between nodes based on their similarity. After partitioning the graph, each color is assigned to the nearest user-defined color.
 - d. **Agglomerative Clustering:** Agglomerative Clustering begins by treating each color as a separate cluster and then iteratively merges the closest pair of clusters. This process continues until the desired number of clusters remains. The colors are then assigned to the nearest user-defined color.
 - e. **Affinity Propagation:** Affinity Propagation, a clustering algorithm, sends messages between pairs of colors until convergence. A set of "exemplars," the most representative of other colors, is chosen. It doesn't require specifying the number of clusters in advance. After applying Affinity Propagation, each color is mapped to the nearest user-defined color.
 - f. **OPTICS:** The OPTICS (Ordering Points To Identify the Clustering Structure) algorithm groups the ISCC/NBS colors. As a density-based clustering algorithm, it identifies clusters of varying density and noise points. It creates an ordering of the data points based on their reachability and core distances. After clustering, the closest classification color to each cluster is assigned.
3. The final step in the process involves creating lookup tables (LUTs) for each color space and clustering method. For each color space (RGB, HSV, and CIE Lab) and each clustering method, a separate LUT is created. The LUTs store the color representation of the ISCC/NBS colors in the respective color space and map each color to the nearest user-defined color according to the specific clustering method. This comprehensive set of LUTs allows for accurate and versatile color mappings across multiple color spaces using various clustering techniques.



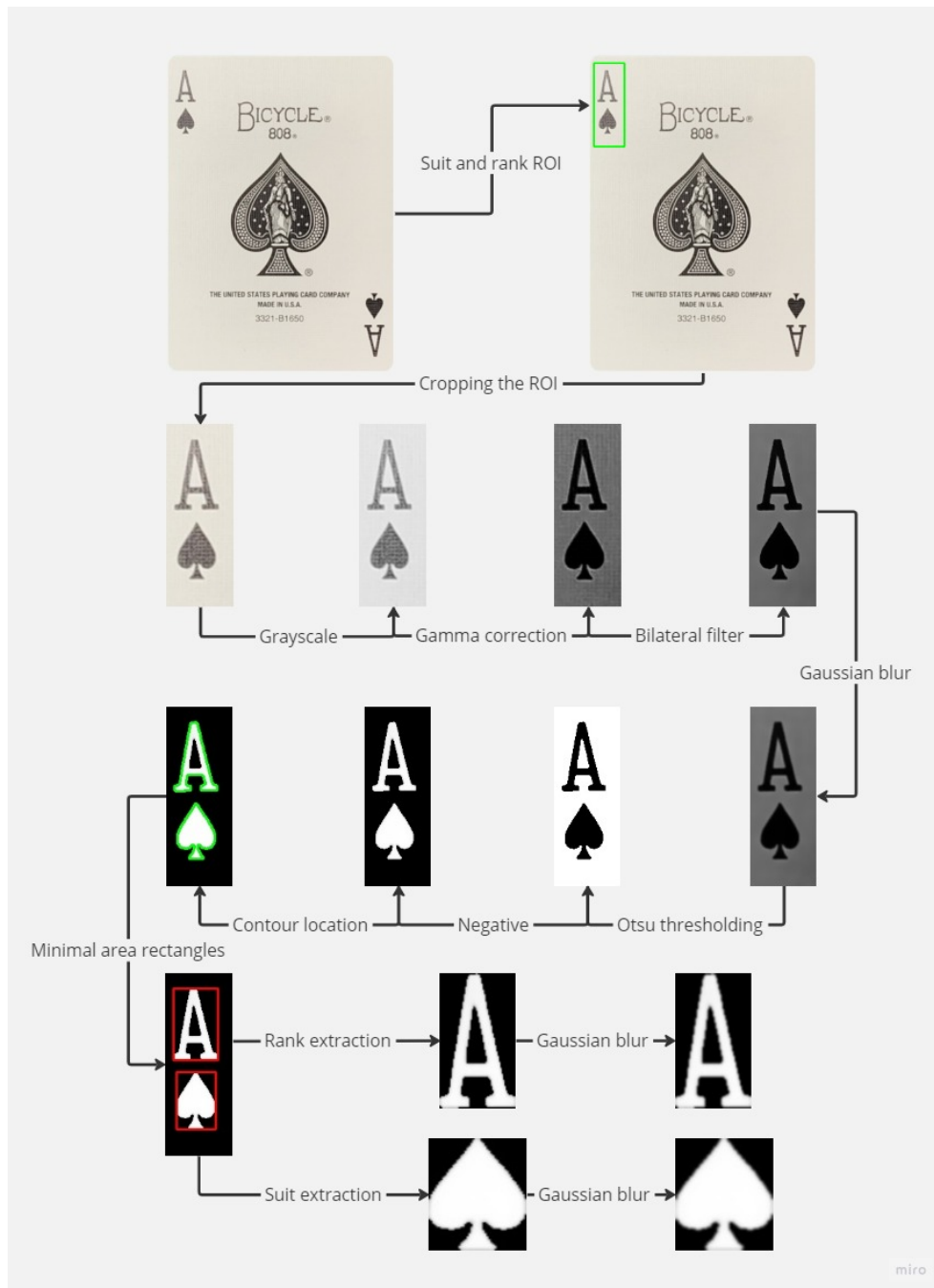
■ Figure 4.15 Clustering results 1.



■ Figure 4.16 Clustering results 2.

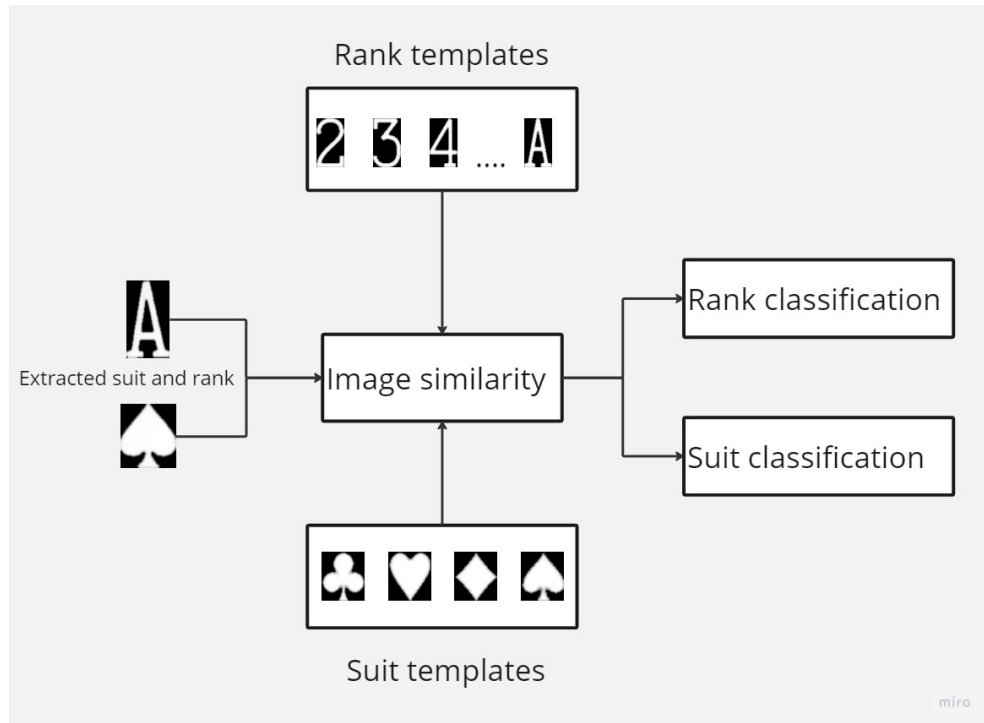
4.1.4 Rank and Suit Identification

The process of identifying the rank and suit of the card is very similar to the previous step, where templates were prepared. First, the region of interest (ROI) containing the rank and suit symbols is extracted from the card image and converted to grayscale. Then, the contours of the rank and suit symbols are detected with prior preprocessing and resized to fixed dimensions (the exact dimensions as the templates).



■ Figure 4.17 Process of extracting the rank and suit.

The next step is to find the best match for the rank and suit symbols in the ROI based on their similarity to the reference images. We have exploited five different methods for measuring the similarity between two images.



■ **Figure 4.18** Process of classification of the rank and suit.

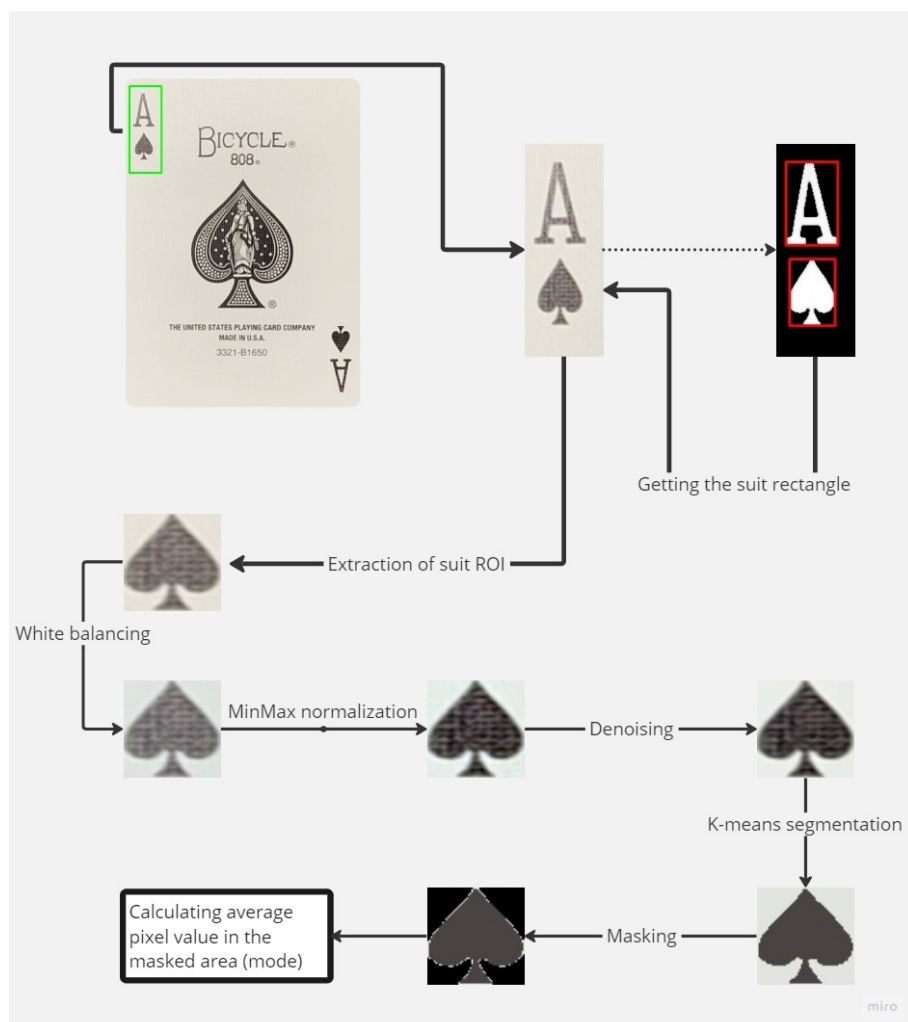
Method	Function	Best result
Sum of Squared Differences	<code>cv2.matchTemplate(..., cv2.TM_SQDIFF_NORMED)</code>	<code>min(...)</code>
Cross-Correlation	<code>cv2.matchTemplate(..., cv2.TM_CCORR_NORMED)</code>	<code>max(...)</code>
Pearson Correlation Coefficient	<code>cv2.matchTemplate(..., cv2.TM_CCOEFF_NORMED)</code>	<code>min(...)</code>
Hu Moments (I1)	<code>cv2.matchShapes(..., cv2.CONTOURS_MATCH_I1, 0)</code>	<code>min(...)</code>
Hu Moments (I2)	<code>cv2.matchShapes(..., cv2.CONTOURS_MATCH_I2, 0)</code>	<code>min(...)</code>
Hu Moments (I3)	<code>cv2.matchShapes(..., cv2.CONTOURS_MATCH_I3, 0)</code>	<code>min(...)</code>
Structural Similarity Index Measure	<code>structural_similarity(..., full=True, multichannel=True)</code>	<code>max(...)</code>

■ **Table 4.3** Summary of similarity methods for rank and suit classification

4.1.5 Color Identification

To accurately classify the card's color, it is necessary to determine which card element will decide the color. Usually, the playing card's color in the chosen deck is determined by the color of the suit and rank. However, since rank is represented by a letter, there would be a small area of pixels to obtain the color information. For this reason, the suit of the playing card is used as the main area to detect the color.

Information obtained from the suit classification is utilized to extract the suit from the captured card. The suit's location is determined using the suit contour and the corresponding minimal area rectangle. The area is then extracted and resized to 60 x 60 pixels. Due to inadequate initial conditions, multiple preprocessing steps are implemented to ensure the best color representation. Image segmentation with K-Means clustering with two centroids is used to segment the picture into a two-colored image, with the first color representing the background and the other representing the suit color. The average pixel value is calculated using mode, taking into account that the pixels of the contour also consist of background pixels. This solution proves more effective than using a morphological operation such as erosion.



■ Figure 4.19 Process of color extraction.

To classify the color, prepared color mappings are relied upon. The averaged pixel value of the suit contour is used as input, and the color in the mappings closest to this value is selected according to the Euclidean distance, which is then classified as red or black. This approach allows for accurate color classification that considers perceptual differences between colors in different color spaces (RGB, HSV, and CIE Lab) and is robust to variations in color representation.

4.1.6 Performance Assessment

In this performance assessment, various methods were applied for color, rank, and suit identification in playing cards. Based on the results presented in Tables 4.4, 4.5, and 4.6, it is evident that different methods and color spaces perform better for specific identification tasks.

Three color spaces were investigated for color identification: LAB, RGB, and HSV. In each color space, six different clustering methods were used: k-means++, mean shift, spectral clustering, agglomerative clustering, affinity propagation, and OPTICS.

For color identification, the HSV color space with the k-means++ method achieved the highest accuracy at 78.16%. However, the performance differences between LAB, RGB, and HSV color spaces were relatively small. This suggests that selecting the appropriate color space and clustering method is crucial for optimal performance in color identification tasks.

In the case of rank and suit identification, template matching methods such as SSD, CC, and PCC demonstrated exceptional performance, achieving 100.0% accuracy. This indicates that these methods are highly reliable and effective for identifying the rank and suit of playing cards. SSIM also showed a strong performance with 99.86% accuracy for rank identification and 100.0% accuracy for suit identification, making it a suitable alternative to the template matching methods.

On the other hand, Hu moments (I1, I2, and I3) showed significantly lower accuracy rates for both rank and suit identification tasks. This suggests that these methods may not be well-suited for playing card identification, at least in their current configuration.

In conclusion, the optimal method for identifying the color, rank, and suit of playing cards largely depends on the specific task and the chosen color space. Template matching methods, particularly SSD, CC, and PCC, appear to be the most reliable and accurate methods for rank and suit identification. For color identification, the choice of color space and clustering algorithm plays a significant role, with the HSV color space and k-means++ method providing the best results in this study.

■ **Table 4.4** Color identification results.

Colorspace	Method	Accuracy (%)
lab	kmeans++	75.00
	meanshift	73.21
	spectral	25.00
	agglomerative	75.00
	affinity	73.21
	optics	73.21
rgb	kmeans++	75.00
	meanshift	75.00
	spectral	25.00
	agglomerative	25.00
	affinity	75.00
	optics	75.00
hsv	kmeans++	78.16
	meanshift	75.00
	spectral	25.00
	agglomerative	25.00
	affinity	75.00
	optics	75.00

■ **Table 4.5** Rank identification results.

Method	Identification	Accuracy (%)
SSD	Rank	100.0
CC	Rank	100.0
PCC	Rank	100.0
HuMoments_I1	Rank	32.97
HuMoments_I2	Rank	33.93
HuMoments_I3	Rank	56.18
SSIM	Rank	99.86

■ **Table 4.6** Suit identification results.

Method	Identification	Accuracy (%)
SSD	Suit	100.0
CC	Suit	100.0
PCC	Suit	100.0
HuMoments_I1	Suit	52.20
HuMoments_I2	Suit	25.27
HuMoments_I3	Suit	50.27
SSIM	Suit	100.0

4.2 Convolution Neural Network Approach

This approach was based on a synthetic playing card dataset, which had been automatically created to represent various conditions with different backgrounds, number of playing cards, angles, card sizes, and other augmentations. The dataset was generated using a custom script that combined two smaller datasets: one containing individual playing cards and another containing various backgrounds. The individual playing cards dataset was created from scratch, while the background dataset was derived from an existing source.

Creating this robust and diverse dataset allowed for better control over the distribution of cards and the ability to generate different variations to suit specific needs. This method provided a more cost-effective alternative to using real-world datasets, which could be expensive and time-consuming to acquire and process.

With this synthetic playing card dataset in hand, a YOLOv8 model was trained for object detection. This machine learning model was designed explicitly for accurately identifying and classifying objects, such as playing cards in this case.

4.2.1 Sole Cards Dataset

A computer vision approach was initially used to extract and classify playing cards from 728 captured images. However, color classification was left out because the approach did not correctly classify the color in all cases. Instead, Spades and Clubs were identified as black, while Diamonds and Hearts were identified as red.

The next step involved annotating specific areas of each card image containing the suit and rank symbols. This process included converting the card images to grayscale, preprocessing, thresholding, extracting contours from the top and bottom regions of the card image, calculating convex hulls for the extracted contours, and adjusting the coordinates of these convex hulls to the original image scale.

A pivotal aspect of this approach, playing card augmentation, was implemented to generate a diverse and robust set of playing card images. An advanced pixel augmentation function was created that involved applying various pixel transformations to the images of playing cards, such as adding random noise, random brightness, random contrast, random saturation, random Gaussian blurring, and random sharpening. This function was used to generate additional augmented cards from the original image.

■ **Code listing 4.1** Advanced Pixel Augmentation Function

```
def advanced_pixel_augmentation(image: Image.Image,
                                noise_stddev=0.1,
                                brightness_range=(0.5, 1.5),
                                contrast_range=(0.5, 1.5),
                                saturation_range=(0.8, 1.2),
                                gaussian_blur_chance=0.5,
                                sharpen_chance=0.1):

    # Convert PIL Image to OpenCV format (BGR format)
    image = cv2.cvtColor(np.array(image), cv2.COLOR_RGB2BGR)

    # Add random noise
    noise = np.random.normal(0, noise_stddev, image.shape)
    image = np.clip(image + noise * 255, 0, 255).astype(np.uint8)

    # Random brightness
    image = cv2.convertScaleAbs(image,
                                alpha=random.uniform(*brightness_range))

    # Random contrast
    img_yuv = cv2.cvtColor(image, cv2.COLOR_BGR2YUV)
    img_yuv[:, :, 0] = cv2.convertScaleAbs(img_yuv[:, :, 0],
                                            alpha=random.uniform(*contrast_range))
    image = cv2.cvtColor(img_yuv, cv2.COLOR_YUV2BGR)

    # Random saturation
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    hsv[:, :, 1] = hsv[:, :, 1] * random.uniform(*saturation_range)
    image = cv2.cvtColor(hsv, cv2.COLOR_HSV2BGR)

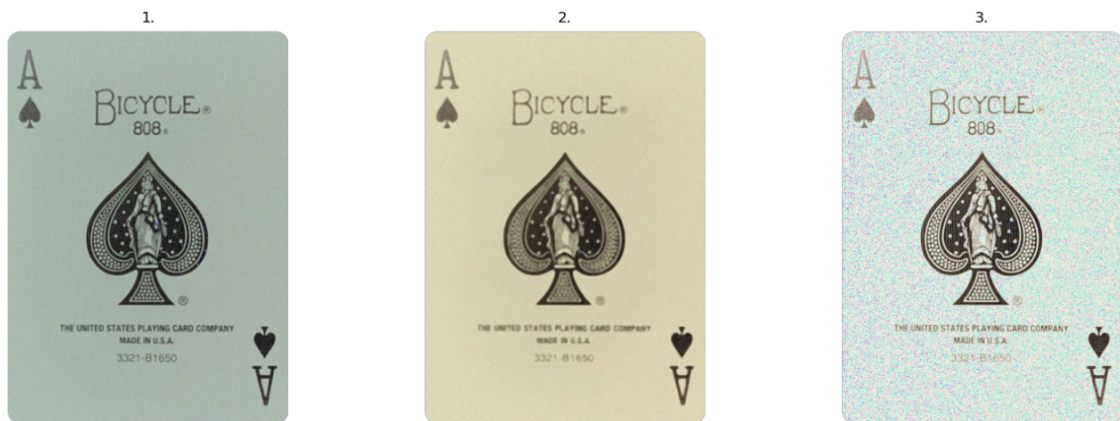
    # Random Gaussian blurring
    if random.random() < gaussian_blur_chance:
        image = cv2.GaussianBlur(image, (5, 5), 0)

    # Random sharpening
    if random.random() < sharpen_chance:
        kernel = np.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]])
        image = cv2.filter2D(image, -1, kernel)

    # Clip the values to be in the correct range
    image = np.clip(image, 0, 255).astype(np.uint8)

    # Convert OpenCV image (in BGR format) to PIL format (in RGB format)
    image = Image.fromarray(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))

    return np.array(image)
```



■ **Figure 4.20** Three randomly generated augmented cards from the original image.

Following the application of all the aforementioned steps, a total of 10000 pickle files were generated, each containing an image in numpy format, annotation areas (represented as contours), rank, suit, and color information. The dataset was then randomly split into train, validation, and test sets, with a split percentage of 60%, 20%, and 20%, respectively.



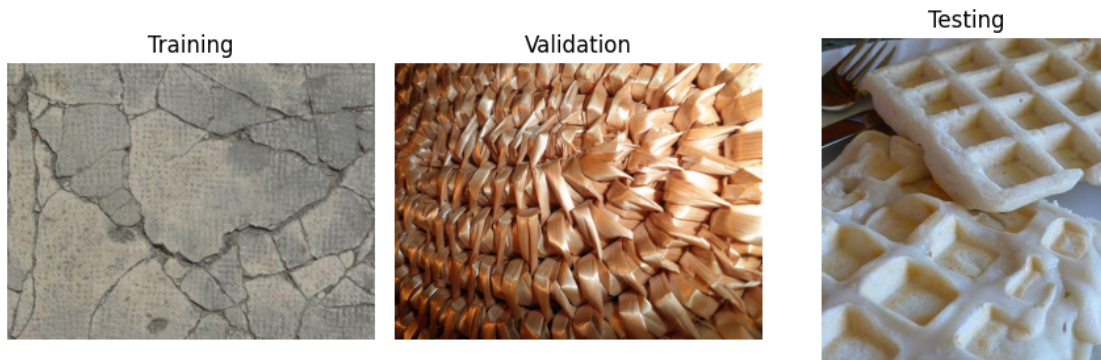
■ **Figure 4.21** Representation of three pickle files contents.

4.2.2 Background Dataset

Another essential step in our approach was to gather a background dataset. To accomplish this, we utilized the Describable Textures Dataset (DTD) [109], a collection of textural images annotated with human-centric attributes inspired by the perceptual properties of textures.

The DTD contains 5640 images organized into 47 categories, with 120 images for each category. The background dataset was split into three parts using a custom script, with 60% of the images allocated to training, 20% to validation, and the remaining 20% to testing. Each split was done with equal probability across all categories of images. Lastly, all background images were randomly reshuffled at the end.

Overall, the DTD provided a valuable background dataset for our approach and allowed us to incorporate a diverse range of textures and patterns into our synthetic dataset generation process.



■ **Figure 4.22** Different backgrounds from testing, training, and validation.

4.2.3 Synthetic Dataset Generation

To generate the dataset, three cycles were employed. In the first cycle, the number of cards to place on the background was selected sequentially, ranging from a minimum of 1 card to a maximum of n cards. In the second cycle, the percentage value representing how much background the cards would cover was chosen sequentially, starting from a lower limit and incrementing up to an upper limit. In the third cycle, a point of view (POV) was randomly selected for each background image, determining the angle at which the cards were viewed. For a portion of the images, a top-down perspective (0, 0, 0) POV was chosen, while for the remaining portion, a random angle within a specified range in the x , y , and z axes was selected. This approach created a more diverse dataset with varying angles of card placements.

For each combination of the number of cards, percentage value, and POV, a background image was randomly selected from the available training, validation, and testing backgrounds. If the chosen number of cards exceeded the number of available card images, the set of available card images was reset, and the selection process started from the beginning.

All card images used in the dataset were chosen randomly from the generated set of card images. Similarly, all backgrounds used in the dataset were randomly selected from the available training, validation, and testing backgrounds. This ensured that the dataset represented the complete set of card images and backgrounds and that the model could generalize to new card images and backgrounds.

4.2.3.1 Card Superimposition on Background

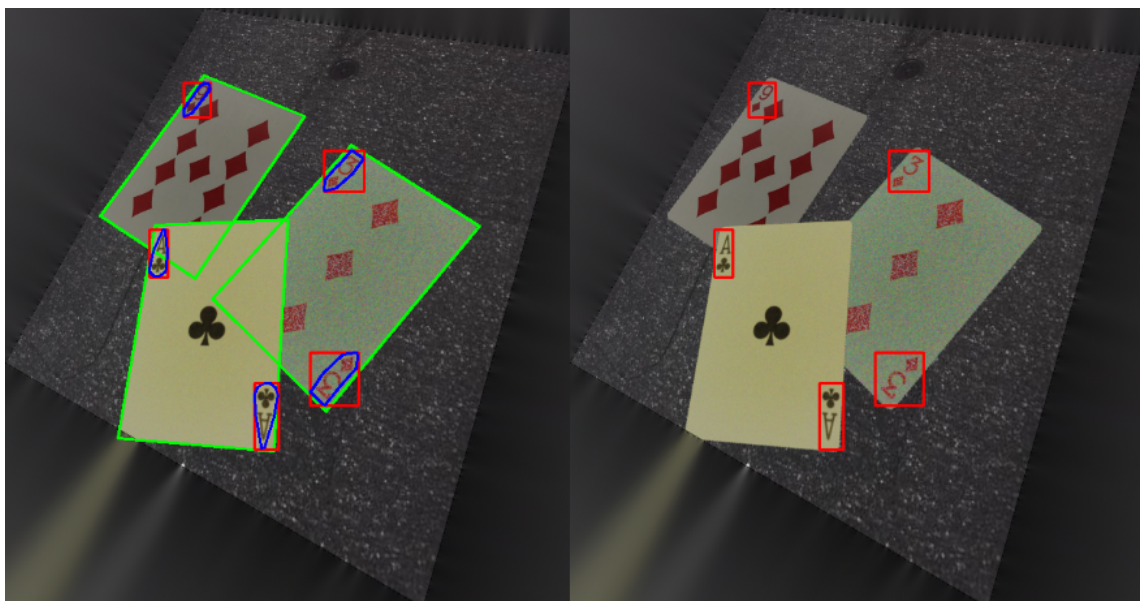
Once the background and the cards were obtained, the cards were scaled to occupy the desired percentage of the background. The cards were then randomly rotated and resized in the z -axis to add variation to the dataset, with the annotating areas recalculated accordingly.

Subsequently, the cards were placed randomly (x , y coordinates) to ensure that both annotating areas were always within the background. If another card was placed on top of an already-placed card, the overlap percentage of the placed card and the area beneath it was calculated. If there was more than 20% overlap, the overlapping area was erased to avoid inconsistency.



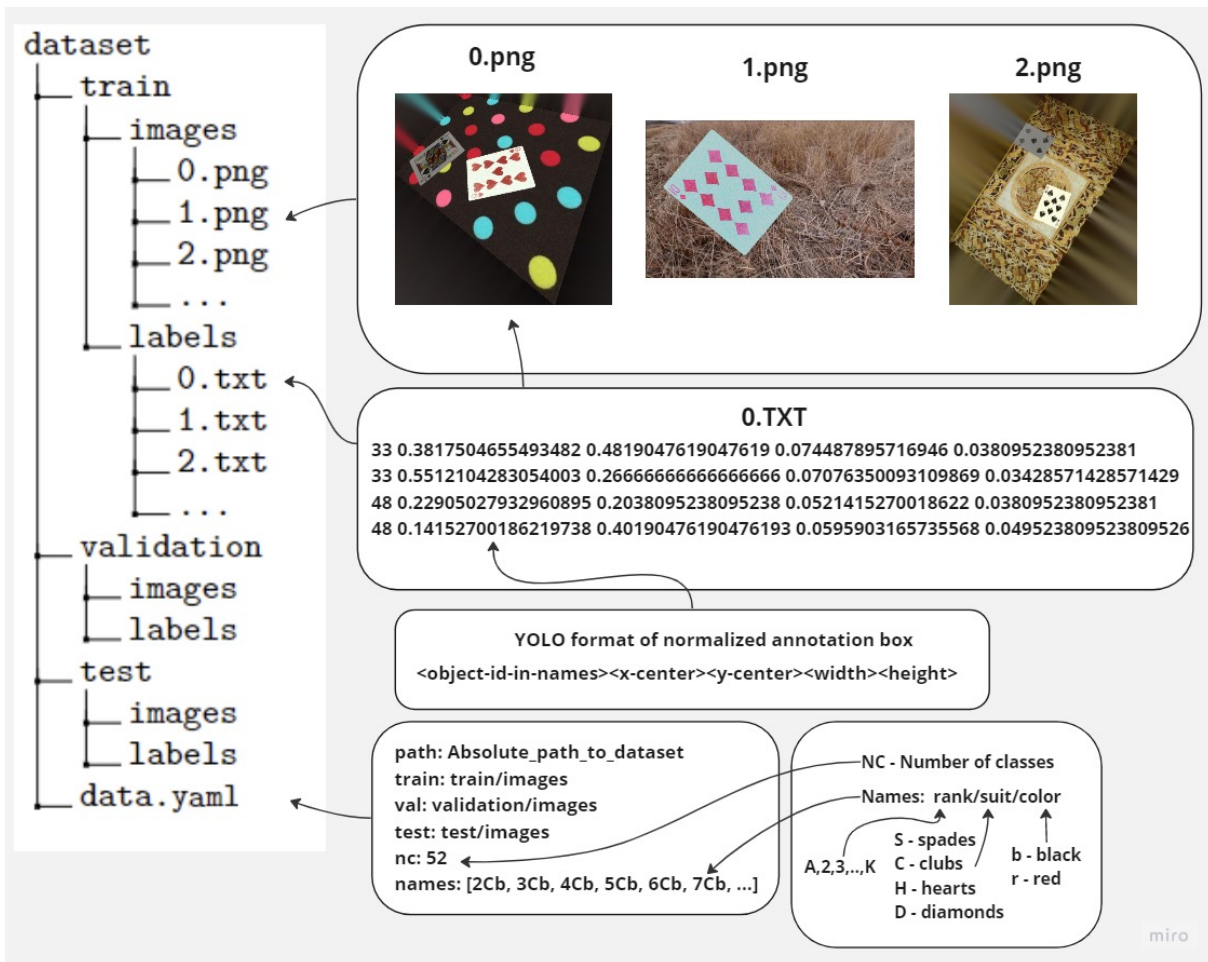
■ **Figure 4.23** Top-down and side view with (20,20,20) angles of the placed cards into the background with overlapping and 20% scaling.

After placing all the cards and recalculating the annotating areas, the entire image was transformed to appear as if it was viewed from the selected POV. However, this transformation often created black bars or triangles around the image. To address this issue, an inpainting method was employed to change these black pixels to pixels resembling the background.



■ **Figure 4.24** Display of the contours and annotating areas.

Finally, corresponding labels in COCO format were accompanied by the generated images, which were subsequently converted to YOLO format. The YOLO-format labels included object class information (rank, suit, and color of the card) and the bounding box coordinates relative to the image size. The images and labels were then saved in the form of a text file, where each line represented an object in the image and contained the following information: class (class number) and YOLO-format coordinates of the bounding box. It's worth noting that a naming convention was adopted that labeled the image name and text file the same because that format was required for the YOLOv8 model.



■ Figure 4.25 Final structure of the created dataset.

4.2.4 Model Training and Validation

This section explains the training and validation process of the YOLOv8 model for the specific use case of detecting multiple classes of playing cards in images. As the dataset had already been prepared, the pre-trained models provided by the YOLOv8 developers were leveraged. These models had been trained on the COCO dataset, a large-scale object detection, segmentation, and captioning dataset containing over 200,000 labeled images with 80 object categories.

The YOLOv8 model is available in different versions, supporting various tasks. The supported tasks and their corresponding pre-trained weights are listed below:

- **YOLOv8 (Detection):**
 - yolov8(n, s, m, l, x).pt
- **YOLOv8-seg (Instance Segmentation):**
 - yolov8(n, s, m, l, x)-seg.pt
- **YOLOv8-pose (Pose/Keypoints):**
 - yolov8(n, s, m, l, x)-pose.pt
- **YOLOv8-cls (Classification):**
 - yolov8(n, s, m, l, x)-cls.pt

Different sizes of pre-trained weights were designed to offer a trade-off between accuracy and performance:

- **n - Nano:** Fast but not very accurate.
- **s - Small:** Faster with a slight compromise in accuracy.
- **m - Medium:** Balanced performance and accuracy.
- **l - Large:** Slower but more accurate.
- **x - Extra Large:** Slowest but offering the highest accuracy.

The YOLOv8 model designed for detection tasks was chosen to detect multiple classes of playing cards in images. The medium-sized neural network, YOLOv8m, was selected for its balance between performance and accuracy, using its corresponding pre-trained weights (yolov8m.pt).

4.2.4.1 Fine-tuning the YOLOv8 Model

The YOLOv8m model was fine-tuned using the custom playing cards dataset. The fine-tuning process consisted of the following steps:

1. The pre-trained yolov8m.pt weights were downloaded.
2. Configuration files for the custom dataset were set up, including the number of classes, class names, and paths to the training and validation data.
3. The desired hyperparameters for the training were set, such as learning rate, batch size, and the number of epochs.
4. The fine-tuning process was initialized using the YOLOv8m pre-trained weights and the custom dataset.

5. The training progress was monitored, and the model's performance on the validation dataset was validated throughout the training.
6. The resulting model weights were saved upon completion of the training for use in the detection phase.

By leveraging the pre-trained YOLOv8m model, the training time and computational resources required to achieve good results were significantly reduced, as the model had already learned many general features from the COCO dataset. The custom-trained YOLOv8m model was thus capable of detecting multiple classes of playing cards in images with a reasonable balance between performance and accuracy.

4.2.4.2 Pretraining with Initial Dataset

In the initial pretraining phase, the following steps were performed:

1. The training dataset comprising 30,000 images (larger and higher number of cards) with multiple augmentations provided by YOLOv8 (such as changes in hue, rotations, flipping, mixup, etc.), and the validation dataset containing 10,000 images was utilized. It is important to note that during this phase, the model was not trained to recognize the color of the cards, as color augmentations were applied.
2. The YOLOv8m model was pretrained for 150 epochs using the following parameters:
 - **Batch size (16):** Determines the number of images processed simultaneously during training, which affects memory usage and training stability.
 - **Image size (640):** The input image size for the model. Higher values improve accuracy but increase computational requirements.
 - **Optimizer (SGD):** The optimization algorithm is used to update the model's weights. Stochastic Gradient Descent (SGD) is a popular choice for training deep learning models.
 - **Learning rate (0.01):** Controls the step size during weight updates. A smaller value results in slower convergence but may yield better accuracy.
 - **Weight decay (0.0005):** A regularization technique to prevent overfitting by adding a penalty term to the loss function, which encourages smaller weights in the model.
3. The model's performance on the initial validation dataset was assessed.

4.2.4.3 Fine-tuning with Augmented Dataset

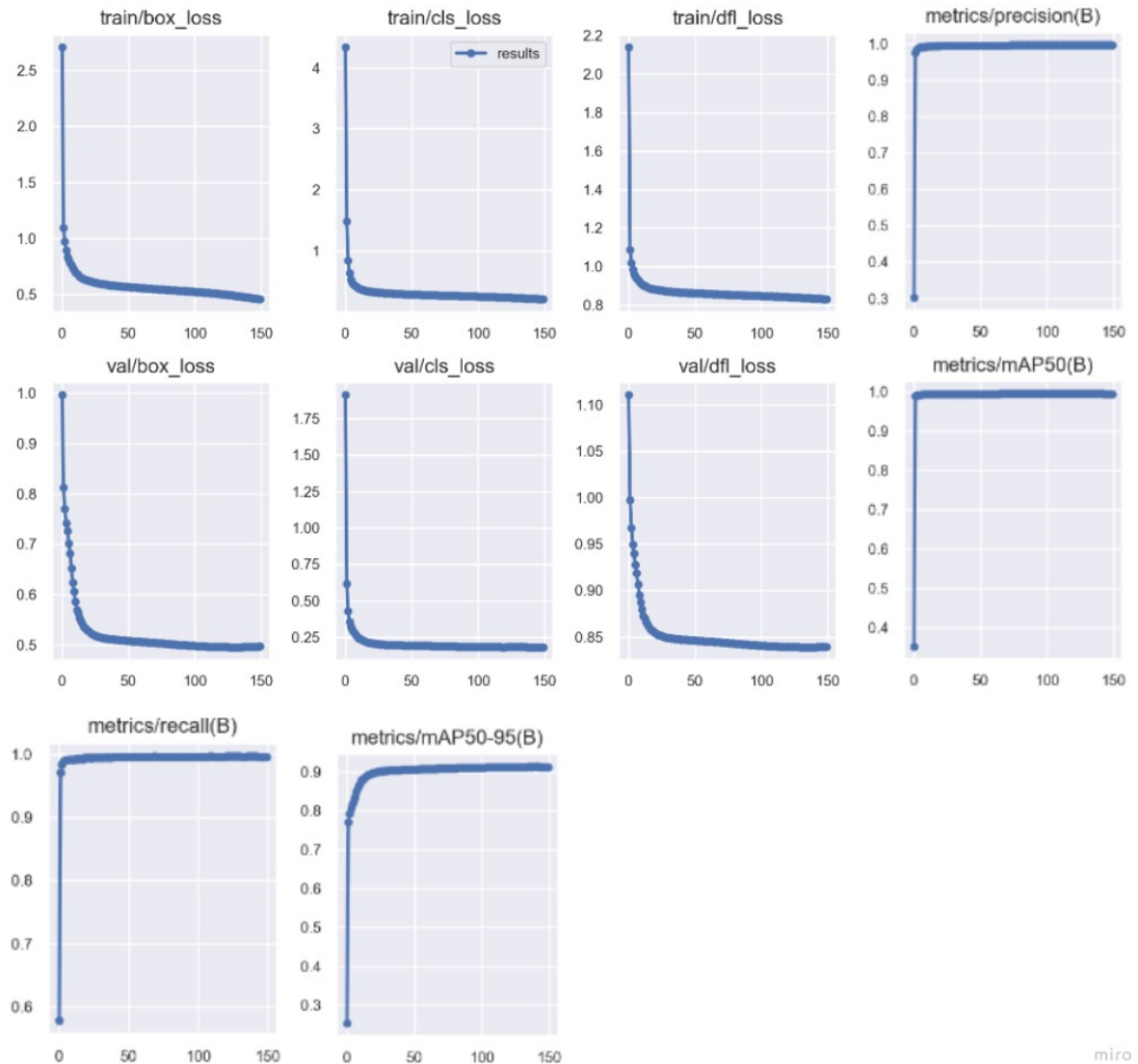
After the initial pretraining, the model was further fine-tuned on a new augmented dataset. This stage involved the following steps:

1. The training dataset comprising 80,000 images (smaller cards and larger angles) with no YOLOv8 augmentations and a new validation dataset containing 30,000 images were utilized.
2. The YOLOv8m model was retrained for 50 epochs using the best weights obtained from the initial pretraining phase.

The crucial parameters used for the second training phase were the same except for the augmentations, such as changes in hue, rotations, flipping, mix-up, etc.

This two-stage fine-tuning process ensured that the custom-trained YOLOv8m model could detect multiple classes of playing cards in images with various card sizes and orientations. By first pretraining the model on a dataset with larger cards and standard augmentations, the model learned to recognize the basic features of playing cards. The subsequent fine-tuning with a dataset containing greater augmentations, including smaller cards and larger angles, enhanced the model's robustness to variations in playing card size and orientation.

In summary, the YOLOv8m model was trained in two phases using the described parameters and dataset configurations. Each training phase was completed in approximately 20 hours on an NVIDIA RTX 3080 GPU.



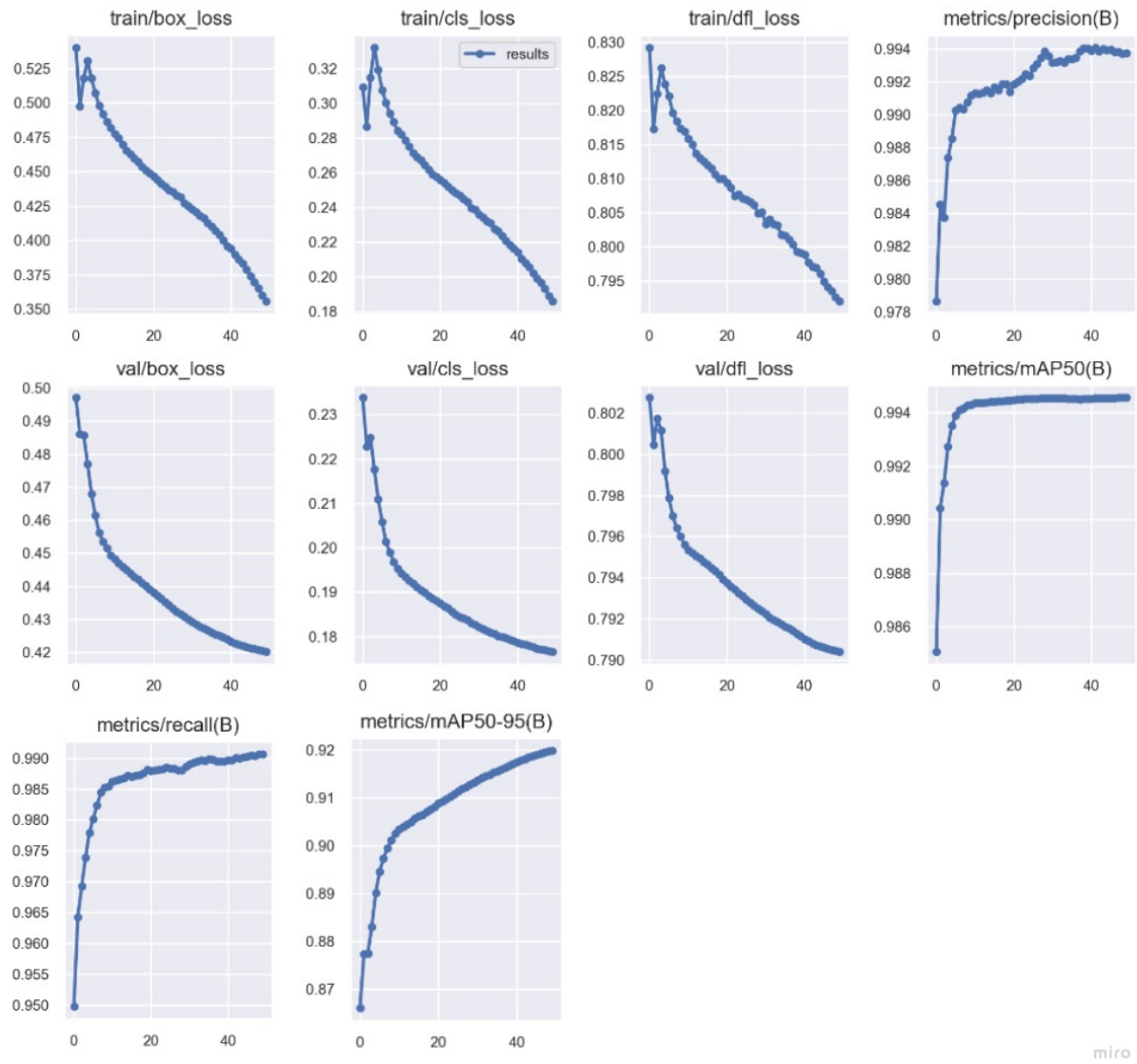
■ **Figure 4.26** Results of different metrics accumulated during the first training phase where the x-axis is epoch number, and the y-axis is the corresponding score.



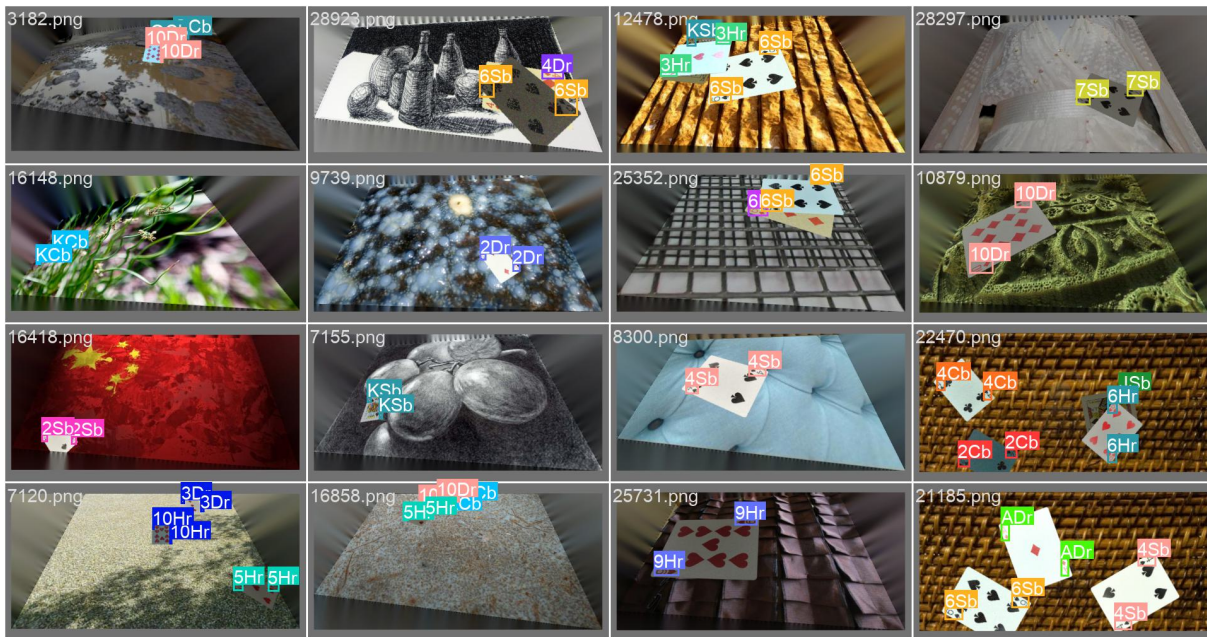
■ Figure 4.27 Example of validation data in the first training phase on the best weights.



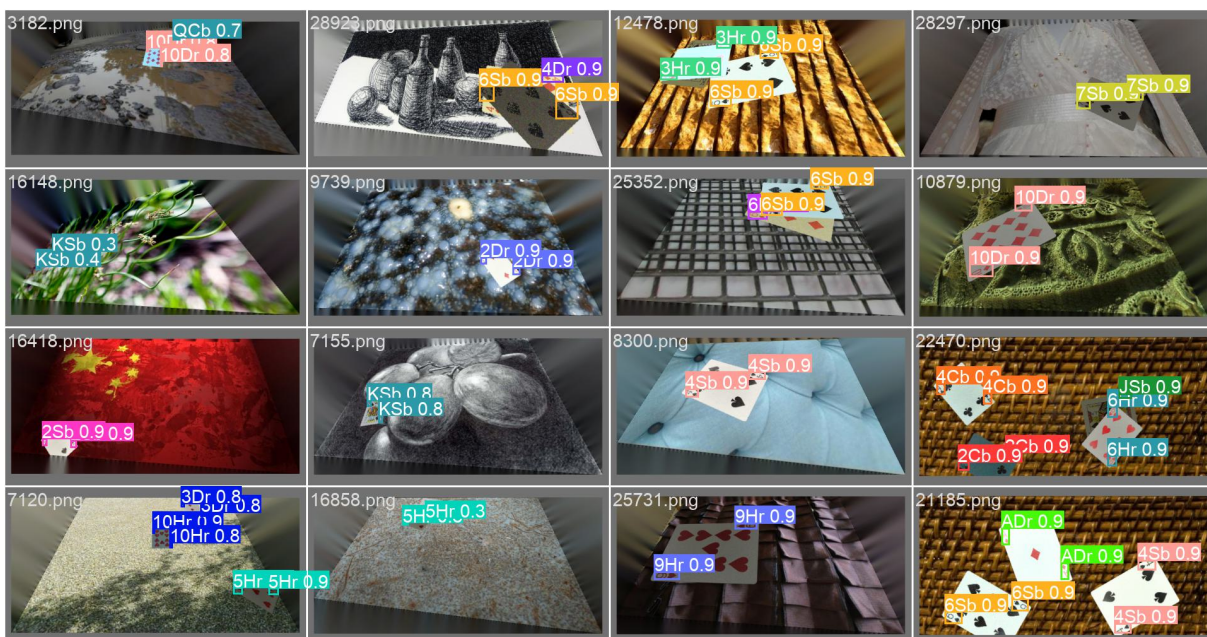
■ Figure 4.28 Example of predicting validation data in the first training phase on the best weights with a corresponding probability of classified class.



■ **Figure 4.29** Results of different metrics accumulated during the second training phase where the x-axis is epoch number, and the y-axis is the corresponding score.



■ **Figure 4.30** Example of validation data in the second phase on the best weights.



■ **Figure 4.31** Example of predicting validation data in the second training phase on the best weights with a corresponding probability of classified class.

4.2.4.4 Analysis of the First Training Phase at Epoch 150

At the end of the first training phase at epoch 150, the following observations can be made:

1. The training losses - box loss (0.45555), class loss (0.20735), and deep feature loss (0.83111) - show that the model has successfully learned to identify the bounding boxes, classes and in-depth features of the playing cards in the initial dataset.
2. The high precision (0.99615) and recall (0.99573) values for the training dataset (B) demonstrate the model's ability to accurately detect and correctly classify the playing cards in the dataset, even though it was not trained to recognize card colors due to color augmentations.
3. The high mAP50 (0.99467) and mAP50-95 (0.91282) values for the training dataset (B) indicate that the model performs well in terms of both localization and classification of the playing cards across different levels of Intersection over Union (IoU) thresholds.
4. The validation losses - box loss (0.49685), class loss (0.18212), and deep feature loss (0.83945) - suggest that the model generalizes well on the initial validation dataset, with similar performance to the training dataset.
5. The learning rate values for the three-parameter groups (pg0, pg1, pg2) are all at 0.000232, indicating consistent weight updates across all model layers.

4.2.4.5 Analysis of the Second Training Phase at Epoch 50

From the data provided for the second training phase at the 50th epoch, the following observations can be made:

1. The training losses - box loss (0.35566), class loss (0.18586), and deep feature loss (0.79199) - indicate that the model has further refined its ability to identify the bounding boxes, classes, and in-depth features of the playing cards in the augmented training dataset.
2. The high precision (0.99373) and recall (0.99065) values for the training dataset (B) demonstrate that the model continues to accurately detect and correctly classify the playing cards, even with the additional augmentations and variations in the dataset.
3. The high mAP50 (0.99455) and mAP50-95 (0.91977) values for the training dataset (B) indicate that the model performs well in terms of both localization and classification of the playing cards across different levels of Intersection over Union (IoU) thresholds.
4. The validation losses - box loss (0.42031), class loss (0.17662), and deep feature loss (0.79041) - suggest that the model generalizes well on the new validation dataset, with similar performance to the training dataset.
5. The learning rate values for the three-parameter groups (pg0, pg1, pg2) are all at 0.000496, which means that the model's weights are being updated consistently across all layers.

In summary, the YOLOv8m model underwent two training phases with different dataset configurations. The first phase focused on training the model to recognize the basic features of playing cards without considering card colors. The second phase further refined the model's ability to detect and classify multiple classes of playing cards in images with increased augmentations and variations. The results from both training phases demonstrate high performance and good generalization on both the training and validation datasets, indicating that the model is robust and capable of handling variations in playing card size, orientation, and color.

4.2.5 Model Testing

To evaluate the performance of the custom-trained YOLOv8m model, it is tested on a previously unseen dataset containing 30,000 images. The results from the final validation on this dataset are as follows:

- **Precision:** 0.9914
- **Recall:** 0.9839
- **mAP50:** 0.9939
- **mAP50-95:** 0.9202
- **Fitness:** 0.9275

The model demonstrates a high level of accuracy in detecting and classifying multiple classes of playing cards, as indicated by the high values for precision, recall, and mAP. The balanced performance and accuracy of the model are evident in the consistent results across these metrics.

Additionally, the speed metrics for the model are as follows:

- **Preprocessing time (0.1891 seconds):** The time taken to preprocess the input images before feeding them to the model. Preprocessing typically includes resizing, normalization, and other necessary image transformations.
- **Inference time (7.8510 seconds):** The time taken by the model to perform predictions on the input images. Inference time is an important factor in determining the real-time applicability of a model.
- **Loss computation time (0.0006 seconds):** The time taken to compute the loss values during model evaluation. This metric reflects the model's efficiency in calculating the differences between predictions and ground truth values.
- **Postprocessing time (0.7788 seconds):** The time taken to process the model's output, which may include tasks such as non-maximum suppression, confidence thresholding, and converting the output to a more interpretable format.

The speed metrics indicate that the model processes images efficiently, with reasonable preprocessing, inference, and postprocessing times. The overall performance of the custom-trained YOLOv8m model demonstrates its suitability for real-world applications requiring accurate and robust playing card detection and classification.

Discussion

In this chapter, we will discuss both designed approaches for playing card recognition, their implications, and their limitations.

5.1 Convolutional Neural Network Approach

5.1.1 Overview

The custom-trained YOLOv8m model was developed to detect and classify multiple classes of playing cards in images. The model was pretrained and fine-tuned in a two-stage process using two different datasets, with training phases taking approximately 20 hours each on an NVIDIA RTX 3080 GPU. The final model demonstrated high accuracy and robust performance in detecting and classifying playing cards, as evidenced by the high precision, recall, and mAP values during testing on a dataset of 30,000 unseen images. The speed metrics also indicate efficient image processing, making the model suitable for real-world applications.

5.1.2 Interpretations

The results indicate that the two-stage fine-tuning process effectively taught the model to recognize various card sizes and orientations and the color of the playing cards. The model's high precision and recall values suggest that it can accurately identify and localize playing cards in images. The mAP values further confirm the model's consistent performance across a range of evaluation criteria.

5.1.3 Implications

The custom-trained YOLOv8m model can be a valuable tool for applications that require playing card detection and classification, such as card game analysis, computer vision-assisted card games, and security systems in casinos. The model's efficient processing times make it suitable for real-time applications, while its high accuracy ensures reliable and robust performance.

5.1.4 Limitations

Although the model demonstrates high accuracy and robust performance, there are some limitations to consider:

- The model was trained and tested on a limited set of playing card images, which may not cover all possible variations in card design, lighting conditions, and background clutter.
- The model may not perform as well in scenarios with significantly different card sizes, orientations, or image resolutions than those in the training and testing datasets.
- The model's performance may be affected by choice of hyperparameters and the specific YOLOv8 architecture used, which may not be optimal for all use cases.

5.1.5 Recommendations

Based on the limitations and the achieved results, the following recommendations are suggested for future research:

- Expand the training and testing datasets to include a wider variety of playing card designs, lighting conditions, and background clutter to improve the model's generalizability.
- Explore alternative model architectures or hyperparameter configurations further to optimize the model's performance for specific use cases.
- Investigate the potential of incorporating additional features or modalities, such as depth information or card texture, to improve the model's robustness in challenging conditions.
- Evaluate the model's performance further in real-world scenarios to better understand its applicability and limitations.

5.2 Computer Vision Approach

5.2.1 Overview

The computer vision-based approach for playing card detection and classification involved capturing 728 high-resolution images (14 sets of 52 cards) in a controlled environment, taking images from a top-down view, and performing a series of image processing steps for card extraction, classification preparations, rank and suit identification, and color identification. This approach was initially used to extract cards for the CNN approach. The process leveraged various techniques and algorithms, including gamma correction, edge detection, contour approximation, clustering, and template matching.

5.2.2 Interpretations

The computer vision approach demonstrates that it is possible to detect and classify playing cards in images by employing traditional image processing and computer vision techniques. The process requires careful calibration of preprocessing steps and extracting relevant card features, such as rank, suit, and color. The variety of algorithms and techniques used for various tasks in this approach highlights the complexity and challenges of playing card detection and classification in real-world scenarios.

5.2.3 Implications

The computer vision approach offers an alternative to deep learning methods for detecting and classifying playing cards in images. This method can be advantageous in situations with limited computational resources or where it is necessary to have a more in-depth understanding of the underlying algorithms and techniques used in the detection process. However, the complexity of the approach and the need for manual calibration of various parameters may limit its applicability in some scenarios.

5.2.4 Limitations

The computer vision approach has several limitations:

- The method relies on a controlled environment with high-contrast backgrounds and consistent lighting conditions, which may not represent real-world scenarios.
- The approach requires manual calibration of various parameters, such as the gamma correction and edge detection thresholds, which can be time-consuming and error-prone.
- The method's performance may be sensitive to changes in card design, lighting conditions, and image resolution, which could limit its generalizability.
- The process involves numerous steps and algorithms, which can be computationally expensive and may not be suitable for real-time applications.

5.2.5 Recommendations

To address the limitations and improve the performance of the computer vision approach, the following recommendations are suggested for future research:

- Investigate ways to make the approach more robust to variations in lighting conditions, background clutter, and card design by incorporating additional preprocessing steps or adaptive algorithms.
- Explore methods for automatically calibrating the various parameters used in the approach, such as machine learning techniques or optimization algorithms.
- Evaluate the method's performance on a larger and more diverse dataset of playing card images, including images captured in uncontrolled environments and with various card designs.
- Investigate ways to reduce the computational complexity of the approach, such as by using more efficient algorithms or parallel processing techniques.
- Incorporate the results from the 728 images, which show the accuracies of different methods for color identification, rank identification, and suit identification, to refine the approach and further improve the overall performance.

Chapter 6

Summary

In this thesis, a playing card recognition system is presented that leverages two distinct approaches to identify playing cards in various environments. The first approach focuses on computer vision techniques to classify cards in a controlled setting, while the second approach emphasizes the creation of a comprehensive dataset with augmentation and the training of a deep learning model.

The first approach utilizes a combination of image processing techniques to detect the rank, suit, and color of a single playing card in an image. This approach involves a series of steps that include image preprocessing, edge detection, contour extraction, and feature extraction. In the preprocessing stage, the image is converted to grayscale, and noise is reduced using Gaussian blurring. This step prepares the image for the subsequent edge detection process. Edge detection is performed using the Canny edge detection algorithm, which helps identify the boundaries of the playing card in the image. The detected edges are then used to extract contours that represent the card's outline. The extracted contour information is utilized to segment the playing card from the image, allowing for further processing and analysis. Once the playing card has been segmented, the next step involves feature extraction, where the card's rank, suit, and color are determined. To achieve this, template matching and pattern recognition techniques are employed. A set of predefined templates for each rank and suit is used to compare against the segmented card. The highest correlation between the templates and the card image indicates the correct rank and suit. Color identification is performed by analyzing the segmented card's color distribution. The dominant color is determined, and based on predefined color mappings, the playing card's color is identified as either red or black. The first approach tests multiple methods for each step of the process, and the best method achieves an accuracy of 100.00% for rank and suit identification and 78.16% for color identification on a dataset of 728 playing cards. This approach demonstrates the potential of computer vision techniques for playing card recognition in controlled environments, although it may face challenges in more complex real-world conditions.

The second approach is centered around the creation of a comprehensive dataset containing 30,000 images of playing cards, labeled with their respective rank and suit. The images undergo slight augmentations, such as rotation, scaling, and flipping, to diversify the dataset and improve model robustness. This dataset is used to pre-train a YOLOv8 model in the first phase of a two-step training process. In the second phase, an additional 80,000 images, more closely resembling real-world conditions, are utilized to fine-tune the model. These images were created to represent various angles, distances, and lighting conditions, ensuring the model's adaptability to a wide range of environments. The model is validated during the training process to monitor performance and prevent overfitting. After completing the two-phase training process, the YOLOv8 model is evaluated on a test set of 30,000 images. The model demonstrates exceptional performance, achieving an accuracy of 99.98%.

In summary, the proposed playing card recognition system offers a highly accurate and efficient solution for detecting playing cards in diverse environments. The system has potential applications across multiple domains, such as gaming, security, and education. Future work will focus on improving the system's performance and precision, addressing the limitations of the computer vision approach, and exploring methods for automatic calibration of various parameters.

Bibliography

1. GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. *Deep learning*. MIT press, 2016. Available also from: <http://www.deeplearningbook.org>.
2. LECUN, Yann; BENGIO, Y.; HINTON, Geoffrey. Deep Learning. *Nature*. 2015, vol. 521, pp. 436–44. Available from DOI: <https://doi.org/10.1038/nature14539>.
3. SZELISKI, R. *Computer Vision: Algorithms and Applications*. Springer, 2023. Texts in Computer Science Ser.
4. GONZALEZ, R.; WOODS, R. *Digital Image Processing*. 2017.
5. BRADSKI, G.; KAEHLER, A. *Learning OpenCV: Computer Vision with the OpenCV Library*. 2008.
6. CANNY, John. A Computational Approach To Edge Detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*. 1986, vol. PAMI-8, pp. 679–698. ISBN 9780080515816. Available from DOI: <https://doi.org/10.1109/TPAMI.1986.4767851>.
7. REDMON, Joseph; DIVVALA, Santosh; GIRSHICK, Ross; FARHADI, Ali. You Only Look Once: Unified, Real-Time Object Detection. 2016, pp. 779–788. Available from DOI: <https://doi.org/10.1109/CVPR.2016.91>.
8. RONNEBERGER, Olaf; FISCHER, Philipp; BROX, Thomas. U-Net: Convolutional Networks for Biomedical Image Segmentation. In: 2015, vol. 9351, pp. 234–241. ISBN 978-3-319-24573-7. Available from DOI: https://doi.org/10.1007/978-3-319-24574-4_28.
9. BISHOP, C. M. *Pattern Recognition and Machine Learning*. 2006.
10. DUDA, R. O.; HART, P. E.; STORK, D. G. *Pattern Classification*. Wiley-Interscience, 2012.
11. KRIZHEVSKY, Alex; SUTSKEVER, Ilya; HINTON, Geoffrey. ImageNet Classification with Deep Convolutional Neural Networks. In: 2012, vol. 25. Available from DOI: <https://doi.org/10.1145/3065386>.
12. HE, Kaiming; ZHANG, Xiangyu; REN, Shaoqing; SUN, Jian. Deep Residual Learning for Image Recognition. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778. Available from DOI: <https://doi.org/10.1109/CVPR.2016.90>.
13. DAI, Jifeng; QI, Haozhi; XIONG, Yuwen; LI, Yi; ZHANG, Guodong; HU, Han; WEI, Yichen. Deformable Convolutional Networks. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 764–773. Available from DOI: <https://doi.org/10.1109/ICCV.2017.89>.

14. HU, Xuewen; YU, Tao; WAN, Kai; YUAN, Jie. Poker card recognition with computer vision methods. In: *2021 IEEE International Conference on Electronic Technology, Communication and Information (ICETCI)*. 2021, pp. 11–15. Available from DOI: <https://doi.org/10.1109/ICETCI53161.2021.9563607>.
15. HU, Ming-Kuei. Visual pattern recognition by moment invariants. *IRE Transactions on Information Theory*. 1962, vol. 8, no. 2, pp. 179–187. Available from DOI: <https://doi.org/10.1109/TIT.1962.1057692>.
16. DAVIES, E. R. *Computer and Machine Vision: Theory, Algorithms, Practicalities*. Academic Press, 2012.
17. FLUSSER, Jan; SUK, Tomáš; ZITOVÁ, Barbara. Moments and Moment Invariants in Pattern Recognition. 2009. Available from DOI: <https://doi.org/10.1002/9780470684757>.
18. ROSIN, Paul L. Ellipse fitting by accumulating five-point fits. *Pattern Recognition Letters*. 1993, vol. 14, no. 8, pp. 661–669. ISSN 0167-8655. Available from DOI: [https://doi.org/10.1016/0167-8655\(93\)90052-F](https://doi.org/10.1016/0167-8655(93)90052-F).
19. BELONGIE, S.; MALIK, J.; PUZICHA, J. Shape matching and object recognition using shape contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2002, vol. 24, no. 4, pp. 509–522. Available from DOI: [10.1109/34.993558](https://doi.org/10.1109/34.993558).
20. CHEN, Wen-Yuan; CHUNG, Chin-Ho. Robust poker image recognition scheme in playing card machine using Hotelling transform, DCT and run-length techniques. *Digital Signal Processing*. 2010, vol. 20, no. 3, pp. 769–779. ISSN 1051-2004. Available from DOI: <https://doi.org/10.1016/j.dsp.2009.09.008>.
21. SANCHEZ-MARIN, Francisco J. Automatic recognition of biological shapes using the Hotelling transform. *Computers in Biology and Medicine*. 2001, vol. 31, no. 2, pp. 85–99. ISSN 0010-4825. Available from DOI: [https://doi.org/10.1016/S0010-4825\(00\)00027-5](https://doi.org/10.1016/S0010-4825(00)00027-5).
22. MALLAT, S.G. A theory for multiresolution signal decomposition: the wavelet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 1989, vol. 11, no. 7, pp. 674–693. Available from DOI: [10.1109/34.192463](https://doi.org/10.1109/34.192463).
23. AHMED, N.; NATARAJAN, T.; RAO, K.R. Discrete Cosine Transform. *IEEE Transactions on Computers*. 1974, vol. C-23, no. 1, pp. 90–93. Available from DOI: [10.1109/T-C.1974.223784](https://doi.org/10.1109/T-C.1974.223784).
24. ZUTIS, Kristis; HOEY, Jesse. Who’s Counting? Real-Time Blackjack Monitoring for Card Counting Detection. In: 2009, pp. 354–363. ISBN 978-3-642-04666-7. Available from DOI: https://doi.org/10.1007/978-3-642-04667-4_36.
25. BRUNELLI, R. Template Matching Techniques in Computer Vision. 2009. Available from DOI: <https://doi.org/10.1002/9780470744055>.
26. LOWE, David. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*. 2004, vol. 60, pp. 91–. Available from DOI: <https://doi.org/10.1023/B:VISI.0000029664.99615.94>.
27. ZIMRAN, Ari; KLIS, Anna A.; FUSTER, Alejandra; RIVELLI, Christopher. The Game of Blackjack and Analysis of Counting Cards. In: 2009.
28. SCHARSTEIN, D.; SZELISKI, R.; ZABIH, R. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. In: *Proceedings IEEE Workshop on Stereo and Multi-Baseline Vision (SMBV 2001)*. 2001, pp. 131–140. Available from DOI: <https://doi.org/10.1109/SMBV.2001.988771>.

29. CHEN, Qianmin; RIGALL, Eric; WANG, Xianglong; FAN, Hao; DONG, Junyu. Poker Watcher: Playing Card Detection Based on EfficientDet and Sandglass Block. In: *2020 11th International Conference on Awareness Science and Technology (iCAST)*. 2020, pp. 1–6. Available from DOI: [10.1109/iCAST51195.2020.9319468](https://doi.org/10.1109/iCAST51195.2020.9319468).
30. YU, Xiaodong; KUAN, Ta Wen; ZHANG, Yuhan; YAN, Taijun. YOLO v5 for SDSB Distant Tiny Object Detection. In: *2022 10th International Conference on Orange Technology (ICOT)*. 2022, pp. 1–4. Available from DOI: [10.1109/ICOT56925.2022.10008164](https://doi.org/10.1109/ICOT56925.2022.10008164).
31. TAN, Mingxing; PANG, Ruoming; LE, Quoc. EfficientDet: Scalable and Efficient Object Detection. In: 2020, pp. 10778–10787. Available from DOI: [10.1109/CVPR42600.2020.01079](https://doi.org/10.1109/CVPR42600.2020.01079).
32. REN, S.; HE, K.; GIRSHICK, R.; SUN, J. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2017, vol. 39, no. 6, pp. 1137–1149.
33. HE, Kaiming; GKIOXARI, Georgia; DOLLÁR, Piotr; GIRSHICK, Ross B. Mask R-CNN. *CoRR*. 2017. Available also from: <http://arxiv.org/abs/1703.06870>.
34. SHARMA, Gaurav. *Digital Color Imaging Handbook*. CRC Press, 2003. Available from DOI: <https://doi.org/10.1201/9781420041484>.
35. POYNTON, Charles. *Frequently-asked questions about color*. 1996. Available also from: http://www.poynton.com/notes/colour_and_gamma/ColorFAQ.html.
36. CHENG, H.D.; JIANG, X.H.; SUN, Y.; WANG, Jingli. Color image segmentation: advances and prospects. In: 2001, vol. 34, pp. 2259–2281. No. 12. ISSN 0031-3203. Available from DOI: [https://doi.org/10.1016/S0031-3203\(00\)00149-7](https://doi.org/10.1016/S0031-3203(00)00149-7).
37. BILDSTEIN, J. *Colorspaces and Conversions*. 2013. Available also from: <https://www.codeproject.com/Articles/613798/Colorspaces-and-Conversions>.
38. CHAUHAN, A. *OpenCV: Different Color Spaces in Image Processing with Python*. 2022. Available also from: <https://pub.towardsai.net/opencv-different-color-spaces-in-image-processing-with-python-17bbed3592ad>.
39. SAYOOD, K. *Introduction to Data Compression*. Morgan Kaufmann, 2017. The Morgan Kaufmann Series in Multimedia Information and Systems Ser.
40. WALLACE, Gregory K. The JPEG still picture compression standard. *IEEE Transactions on Consumer Electronics*. 1992, vol. 38, no. 1, pp. xviii–xxxiv. Available also from: <https://web.stanford.edu/class/ee398a/handouts/papers/Wallace%5C%20-%5C%/20JPEG%5C%20-%5C%201992.pdf>.
41. GROUP, Joint Photographic Experts. *JPEG XR Image Coding System*. 2011. Available also from: <https://jpeg.org/jpegxr/index.html>.
42. BURT, P.; ADELSON, E. The Laplacian Pyramid as a Compact Image Code. In: 1983, vol. 31, pp. 532–540. No. 4. Available from DOI: <https://doi.org/10.1109/TCOM.1983.1095851>.
43. GEORGE, Ginu; OOMMEN, Rinoy Mathew; SHELLY, Shani; PHILIPOSE, Stephanie Sara; VARGHESE, Ann Mary. A Survey on Various Median Filtering Techniques For Removal of Impulse Noise From Digital Image. In: *2018 Conference on Emerging Devices and Smart Systems (ICEDSS)*. 2018, pp. 235–238. Available from DOI: <https://doi.org/10.1109/ICEDSS.2018.8544273>.
44. TOMASI, C.; MANDUCHI, R. Bilateral filtering for gray and color images. In: *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*. 1998, pp. 839–846. Available from DOI: <https://doi.org/10.1109/ICCV.1998.710815>.

45. JIMÉNEZ L., Fabián; PARDO-BEAINY, Camilo; MENDEZ, Oscar. Biometric iris recognition using Hough Transform. In: 2013, pp. 1–6. ISBN 978-1-4799-1121-9. Available from DOI: [10.1109/STSIWA.2013.6644905](https://doi.org/10.1109/STSIWA.2013.6644905).
46. DURAND, Frédo; DORSEY, Julie. Fast Bilateral Filtering for the Display of High - dynamic - range Images. *ACM Trans. Graph.* 2002, vol. 21, pp. 257–266. Available from DOI: <https://doi.org/10.1145/566654.566574>.
47. WEISS, Ben. Fast median and bilateral filtering. *ACM Trans. Graph.* 2006, vol. 25, pp. 519–526. Available from DOI: <https://doi.org/10.1145/1141911.1141918>.
48. KORNPORST, Pierre; TUMBLIN, Jack; DURAND, Frédo. Bilateral Filtering: Theory and Applications. *Foundations and Trends in Computer Graphics and Vision.* 2009, vol. 4, pp. 1–74. Available from DOI: <https://doi.org/10.1561/06000000020>.
49. SONKA, Milan; HLAVAC, Vaclav; BOYLE, Roger. *Image processing, analysis and machine vision (3. ed.)*. 2008. ISBN 978-0-495-24438-7.
50. LARSON, Gregory. The LogLuv Encoding for Full Gamut, High Dynamic Range Images. *JGT.* 2003, vol. 3.
51. RAHMAN, Shanto; RAHMAN, Md. Mostafijur; ABDULLAH-AL-WADUD, M.; AL-QUADERI, Golam Dastagir; SHOYAIB, Mohammad. An adaptive gamma correction for image enhancement. *EURASIP Journal on Image and Video Processing.* 2016, vol. 35. Available from DOI: <https://doi.org/10.1186/s13640-016-0138-1>.
52. STOKES, Mike; ANDERSON, Matthew; CHANDRASEKAR, Srinivas; MOTTA, Ricardo. *A Standard Default Color Space for the Internet - sRGB*. 1996. Available also from: <http://www.w3.org/Graphics/Color/sRGB>.
53. VELUCHAMY, Magudeeswaran; SUBRAMANI, Bharath. Image contrast and color enhancement using adaptive gamma correction and histogram equalization. *Optik.* 2019, vol. 183, pp. 329–337. ISSN 0030-4026. Available from DOI: <https://doi.org/10.1016/j.ijleo.2019.02.054>.
54. PALANISAMY, Gopinath; PONNUSAMY, Palanisamy; GOPI, Varun. An improved luminosity and contrast enhancement framework for feature preservation in color fundus images. *Signal, Image and Video Processing.* 2019, vol. 13. Available from DOI: <https://doi.org/10.1007/s11760-018-1401-y>.
55. SHEN, Wei; WANG, Xinggang; WANG, Yan; BAI, Xiang; ZHANG, Zhijiang. DeepContour: A deep convolutional feature learned by positive-sharing loss for contour detection. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 3982–3991. Available from DOI: <https://doi.org/10.1109/CVPR.2015.7299024>.
56. TSANKASHVILI, N. *Comparing Edge Detection Methods*. 2020. Available also from: <https://medium.com/@nikatsanka/comparing-edge-detection-methods-638a2919476e>.
57. SAHIR, S. *Canny Edge Detection Step by Step in Python — Computer Vision*. 2019. Available also from: <https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123>.
58. RAMER, Urs. An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing.* 1972, vol. 1, no. 3, pp. 244–256. ISSN 0146-664X. Available from DOI: [https://doi.org/10.1016/S0146-664X\(72\)80017-0](https://doi.org/10.1016/S0146-664X(72)80017-0).
59. POIKER, T.; DOUGLAS, D.H. Reflection Essay: Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or its Caricature. *Classics in Cartography: Reflections on Influential Articles from Cartographica.* 2011, pp. 29–36. Available from DOI: <https://doi.org/10.1002/9780470669488.ch3>.

60. MOHNEESH, S. *Applications of Ramer-Douglas-Peucker Algorithm in Machine Learning That You Might Not Have Heard...* 2021. Available also from: <https://medium.com/mllearning-ai/applications-of-ramer-douglas-peucker-algorithm-in-machine-learning-that-you-might-not-have-heard-63b0c4f15a43>.
61. DUDA, R. O.; HART, P. E. Use of the Hough transformation to detect lines and curves in pictures. *Communications of the ACM*. 1972, vol. 15, no. 1, pp. 11–15. Available from DOI: <https://doi.org/10.1145/361237.361242>.
62. SAHOO, Prasanna; SOLTANI, Sasan; WONG, Andrew. A Survey of Thresholding Techniques. *Computer Vision, Graphics, and Image Processing*. 1988, vol. 41, pp. 233–260. Available from DOI: [https://doi.org/10.1016/0734-189X\(88\)90022-9](https://doi.org/10.1016/0734-189X(88)90022-9).
63. OTSU, Nobuyuki. A Threshold Selection Method from Gray-Level Histograms. *IEEE Transactions on Systems, Man, and Cybernetics*. 1979, vol. 9, no. 1, pp. 62–66. Available from DOI: <https://doi.org/10.1109/TSMC.1979.4310076>.
64. MARTIN, Stéphane; TROCCAZ, Jocelyne; DAANEN, Vincent. Automated segmentation of the prostate in 3D MR images using a probabilistic atlas and a spatially constrained deformable model. *Medical Physics*. 2010, vol. 37, no. 4, pp. 1579–1590. Available from DOI: <https://doi.org/10.1118/1.3315367>.
65. RUBLEE, Ethan; RABAUD, Vincent; KONOLIGE, Kurt; BRADSKI, Gary. ORB: An efficient alternative to SIFT or SURF. In: *2011 International Conference on Computer Vision*. 2011, pp. 2564–2571. Available from DOI: <https://doi.org/10.1109/ICCV.2011.6126544>.
66. PAIALUNGA, P. *Hands on Otsu Thresholding Algorithm for Image Background Segmentation, using Python*. 2023. Available also from: <https://towardsdatascience.com/hands-on-otsu-thresholding-algorithm-for-image-background-segmentation-using-python-9fa0575ac3d2>.
67. HARALICK, Robert M.; STERNBERG, Stanley R.; ZHUANG, Xinhua. Image Analysis Using Mathematical Morphology. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 1987, vol. PAMI-9, no. 4, pp. 532–550. Available from DOI: <https://doi.org/10.1109/TPAMI.1987.4767941>.
68. PRATT, William K. *Digital Image Processing: PIKS Inside*. 3rd ed. Wiley-Interscience, 2001.
69. WANG, Zhou; BOVIK, A.C.; SHEIKH, H.R.; SIMONCELLI, E.P. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*. 2004, vol. 13, no. 4, pp. 600–612. Available from DOI: <https://doi.org/10.1109/TIP.2003.819861>.
70. HU, Ming-Kuei. Visual pattern recognition by moment invariants. *IRE Transactions on Information Theory*. 1962, vol. 8, no. 2, pp. 179–187. Available from DOI: <https://doi.org/10.1109/TIT.1962.1057692>.
71. RAZAVIAN, Ali; AZIZPOUR, Hossein; SULLIVAN, Josephine; CARLSSON, Stefan. CNN Features Off-the-Shelf: An Astounding Baseline for Recognition. In: 2014, vol. 1403. Available from DOI: <https://doi.org/10.1109/CVPRW.2014.131>.
72. SIVIC; ZISSERMAN. Video Google: a text retrieval approach to object matching in videos. In: *Proceedings Ninth IEEE International Conference on Computer Vision*. 2003, 1470–1477 vol.2. Available from DOI: <https://doi.org/10.1109/ICCV.2003.1238663>.
73. ZITOVÁ, Barbara; FLUSSER, Jan. Image registration methods: a survey. *Image and Vision Computing*. 2003, vol. 21, no. 11, pp. 977–1000. ISSN 0262-8856. Available from DOI: [https://doi.org/10.1016/S0262-8856\(03\)00137-9](https://doi.org/10.1016/S0262-8856(03)00137-9).

74. LOWE, D.G. Object recognition from local scale-invariant features. In: *Proceedings of the Seventh IEEE International Conference on Computer Vision*. 1999, vol. 2, 1150–1157 vol.2. Available from DOI: <https://doi.org/10.1109/ICCV.1999.790410>.
75. BROWN, Matthew; LOWE, David G. Automatic Panoramic Image Stitching using Invariant Features. *International Journal of Computer Vision*. 2006, vol. 74, no. 1, pp. 59–73. Available from DOI: <https://doi.org/10.1007/s11263-006-0002-3>.
76. MISHRA, Shivangi; SHRIVASTAVA, Priyanka; DHURVEY, Priyanka. Change Detection Techniques in Remote Sensing: A Review. *International Journal of Wireless and Mobile Communication for Industrial Systems*. 2017, vol. 4, pp. 1–8. Available from DOI: <https://doi.org/10.21742/ijwmcis.2017.4.1.01>.
77. MELGANI, F.; BRUZZONE, L. Classification of hyperspectral remote sensing images with support vector machines. *IEEE Transactions on Geoscience and Remote Sensing*. 2004, vol. 42, no. 8, pp. 1778–1790. Available from DOI: <https://doi.org/10.1109/TGRS.2004.831865>.
78. PEREZ, Luis; WANG, Jason. *The Effectiveness of Data Augmentation in Image Classification using Deep Learning*. 2017. Available from arXiv: 1712.04621 [cs.CV].
79. SHORTEN, Connor; KHOSHGOFTAAR, Taghi M. A survey on Image Data Augmentation for Deep Learning. *Journal of Big Data*. 2019, vol. 6, no. 1. Available from DOI: <https://doi.org/10.1186/s40537-019-0197-0>.
80. ARTHUR, David; VASSILVITSKII, Sergei. K-Means++: The Advantages of Careful Seeding. In: 2007, vol. 8, pp. 1027–1035. Available from DOI: <https://doi.org/10.1145/1283383.1283494>.
81. COMANICIU, D.; MEER, P. Mean shift: a robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2002, vol. 24, no. 5, pp. 603–619. Available from DOI: <https://doi.org/10.1109/34.1000236>.
82. LUXBURG, Ulrike. A Tutorial on Spectral Clustering. *Statistics and Computing*. 2004, vol. 17, pp. 395–416. Available from DOI: <https://doi.org/10.1007/s11222-007-9033-z>.
83. WARD, Joe H. Hierarchical Grouping to Optimize an Objective Function. *Journal of the American Statistical Association*. 1963, vol. 58, no. 301, pp. 236–244. Available from DOI: <https://doi.org/10.1080/01621459.1963.10500845>.
84. FREY, Brendan; DUECK, Delbert. Clustering by Passing Messages Between Data Points. *Science (New York, N.Y.)* 2007, vol. 315, pp. 972–6. Available from DOI: <https://doi.org/10.1126/science.1136800>.
85. ANKERST, Mihael; BREUNIG, Markus M.; KRIEGEL, Hans-Peter; SANDER, Jörg. OPTICS: Ordering Points to Identify the Clustering Structure. In: *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: Association for Computing Machinery, 1999, pp. 49–60. ISBN 1581130848. Available from DOI: [10.1145/304182.304187](https://doi.org/10.1145/304182.304187).
86. RAGHAV, P. *Understanding of Convolutional Neural Network (CNN) — Deep Learning*. 2019. Available also from: <https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148>.
87. KINGMA, Diederik; BA, Jimmy. Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations*. 2014.
88. RUMELHART, David E; HINTON, Geoffrey E; WILLIAMS, Ronald J. Learning representations by back-propagating errors. *nature*. 1986, vol. 323, no. 6088, pp. 533–536.

89. SRIVASTAVA, Nitish; HINTON, Geoffrey; KRIZHEVSKY, Alex; SUTSKEVER, Ilya; SALAKHUTDINOV, Ruslan. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*. 2014, vol. 15, pp. 1929–1958.
90. YOSINSKI, Jason; CLUNE, Jeff; BENGIO, Y.; LIPSON, Hod. How transferable are features in deep neural networks? *Advances in Neural Information Processing Systems (NIPS)*. 2014, vol. 27.
91. TAN, Cheng; SUN, Fuchun; KONG, Tao; ZHANG, Wenchang; YANG, Chongyang; LIU, Chun. A survey on deep transfer learning. *Artificial Intelligence Review*. 2018, vol. 52, no. 1, pp. 1–34.
92. LECUN, Y.; BOTTOU, L.; BENGIO, Y.; HAFFNER, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*. 1998, vol. 86, no. 11, pp. 2278–2324. Available from DOI: <https://doi.org/10.1109/5.726791>.
93. SIMONYAN, Karen; ZISSERMAN, Andrew. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv 1409.1556*. 2014.
94. TAN, Mingxing; LE, Quoc V. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. *CoRR*. 2019, vol. abs/1905.11946. Available also from: <http://arxiv.org/abs/1905.11946>.
95. GIRSHICK, Ross; DONAHUE, Jeff; DARRELL, Trevor; MALIK, Jitendra. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2013. Available from DOI: <https://doi.org/10.1109/CVPR.2014.81>.
96. GIRSHICK, Ross. Fast R-CNN. In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 1440–1448. Available from DOI: <https://doi.org/10.1109/ICCV.2015.169>.
97. TERVEN, Juan; CORDOVA-ESPARZA, Diana. *A Comprehensive Review of YOLO: From YOLOv1 to YOLOv8 and Beyond*. 2023. Available from arXiv: 2304.00501 [cs.CV].
98. GLENN, Johera; YONGHYE, Kwon. *GitHub - ultralytics/ultralytics: NEW - YOLOv8 in PyTorch > ONNX > CoreML > TFLite*. 2023. Available also from: <https://github.com/ultralytics/ultralytics>.
99. *What is YOLOv8? The Ultimate Guide*. 2023. Available also from: <https://blog.roboflow.com/whats-new-in-yolov8/>.
100. POWERS, David. Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness Correlation. *Mach. Learn. Technol.* 2008, vol. 2.
101. SOKOLOVA, Marina; LAPALME, Guy. A systematic analysis of performance measures for classification tasks. *Information Processing Management*. 2009, vol. 45, pp. 427–437. Available from DOI: 10.1016/j.ipm.2009.03.002.
102. SCARNE, John. *Scarne's encyclopedia of card games*. Harper & Row, 1986.
103. HARGRAVE, Catherine P. *A History of Playing Cards and a Bibliography of Cards and Gaming*. Dover Publications, 1966.
104. *The United States Playing Card Company*. The United States Playing Card Company, [n.d.]. Available also from: <https://www.usplayingcard.com/>.
105. GREEN, Richard D. Playing Card Recognition Using Rotational Invariant Template Matching. In: 2007.
106. SENGUPTA, Soumyadip; JAYARAM, Vivek; CURLESS, Brian; SEITZ, Steven M.; KEMEL-MACHER, Ira. Background Matting: The World Is Your Green Screen. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020. Available from DOI: <https://doi.org/10.1109/cvpr42600.2020.00236>.

107. ILIE, Adrian; WELCH, Greg. Ensuring Color Consistency across Multiple Cameras. In: 2005, vol. 2, pp. 1268–1275. Available from DOI: <https://doi.org/10.1109/ICCV.2005.88>.
108. PELTOKETO, Veli-Tapani. Evaluation of mobile phone camera benchmarking using objective camera speed and image quality metrics. *Journal of Electronic Imaging*. 2014, vol. 23, no. 6, p. 061102. Available from DOI: <https://doi.org/10.1117/1.JEI.23.6.061102>.
109. CIMPOI, Mircea; MAJI, Subhansu; KOKKINOS, Iasonas; MOHAMED, Sammy; VEDALDI, Andrea. *Describing Textures in the Wild*. 2013. Available from arXiv: 1311.3618 [cs.CV].