الجمهورية الجزائرية الديمقراطية الشعبية

People's Democratic Republic of Algeria

وزارة التعليم العالي والبحث العلمي

Ministry of Higher Education and scientific research

المدرسة الوطنية العليا للتكنولوجيات المتقدمة

National Higher School of Advanced Technologies

Department of Electrical Engineering And Industrial Computing

---

Final Year Project to Obtain the Diploma of Engineering

Field:
Electronics

Speciality:
Embedded Systems

Automatic and Early Detection of Defects on Printed Circuits Bords (PCBs)

Realized by
AOUIR Khalil

HARAOUBIA Brahim	Chair
BOUCHAMA Samira	Examiner
ZELLAT Khadidja	Examiner
CHOUAF Seloua	Supervisor

Algiers, the 25/06/2024

Academic year 2023 –2024

Acknowledgments

First and foremost, I am deeply grateful to Allah for granting me the strength complete this work. Alhamdulillah for His guidance and blessings.

I owe a special thanks to my father and mother for their endless love, encouragement, and prayers. Their belief in me has been a constant source of strength. To my siblings, your support and encouragement have been invaluable throughout this journey.

I extend my heartfelt thanks to my supervisor, Ms. Chouaf, whose unwavering support and patience were indispensable throughout the research and writing of this thesis.

I also express my gratitude to the members of my thesis committee: Mr. Haraoubia , Ms. Zellat, and Ms. Bouchama, for their valuable time and thoughtful evaluation of my work.

To my friends, your unwavering support and understanding made this journey not only bearable but enjoyable Thanks.

Table of Contents

List of Acronyms

AI	Artificial Intelligence
BKPN	Best Keypoints Pair Number
CNN	Convolutional Neural Network
FP	False Positives
FN	False Negatives
GPU	Graphics Processing Unit
IOU	Intersection over Union
KPN_min	Minimum Number of Keypoints
MAP	Mean Average Precision
NCC	Normalized Cross-Correlation
PCB	Printed Circuit Board
ROI	Region of Interest
SGD	Stochastic Gradient Descent
SIFT	Scale-Invariant Feature Transform
TP	True Positives
TN	True Negatives
YOLO	You Only Look Once

List of Figures

List of Tables

General Introduction

Printed Circuit Boards (PCBs) are fundamental components in electronic devices, serving as the foundation for connecting various electronic parts to create functional circuits. They are widely used across industries, from everyday gadgets to aerospace and automotive applications, highlighting their crucial role in advancing technology.

In PCB manufacturing, where tiny components are densely packed onto intricately designed boards, thorough inspection is essential to detect any defects that could affect performance or reliability. Visual Inspection Systems designed for PCB manufacturing utilize advanced imaging technologies, including high-resolution cameras and sophisticated lighting setups, to capture detailed images of PCBs with exceptional clarity and precision.

These systems employ advanced image processing techniques, such as pattern recognition, edge detection, and machine learning algorithms, to analyze captured images and identify potential defects. By automating the inspection process, Visual Inspection Systems significantly reduce the time and effort required for quality control tasks while enhancing the accuracy and consistency of defect detection.

This project focuses on designing and implementing a Visual Inspection System specifically tailored for PCB manufacturing. By leveraging image processing and machine learning techniques, our goal is to develop a robust and efficient solution that seamlessly integrates into existing manufacturing processes, ultimately improving the quality and reliability of electronic devices based on PCB technology.

The objective of the work presented in this document is to develop a system capable of automatically detecting defects and classifying each PCB as either clean or defective using deep learning and image processing algorithms. Our approach mainly involves leveraging pre-trained models known for their performance and applying to them transfer learning.

This document is structured as follows: It begins with a chapter that introduces to visual inspection systems for product quality control, highlighting traditional techniques as well as recent methods based on deep learning.

Subsequently, in Chapter 2, we address artificial intelligence theories. and image processing algorithms . We focus on deep learning techniques used at each stage from data preparation to model evaluation. We also describe the transfer learning approach, its relevance, with specific projection on the pre-trained models used in our work.

The third chapter is devoted to the presentation of the solution we proposed for the automatic and early defect detection in PCBs. It's worthy to mention that the automatic detection means that intervention of the human operator isn't mendatory. Howerver, the early defects detection refers to identifying anomalies at early stages in the production process with the aim of catching issues before they propagate further, minimizing the impact and the cost.

Chapter 1

Introduction to Visual Inspection Systems

1.1 Introduction

Defect detection is crucial across various industries, spanning from manufacturing to distribution and logistics. Defects can profoundly impact product quality, consumer safety, production costs, and company reputation. Consequently,VISs are increasingly utilized to identify anomalies at different production stages.

The design of a VIS is influenced by various application factors and user requirements. Typically, these systems leverage both hardware and software resources to capture product images, process them, and pinpoint any anomalies present.

Conventional visual inspection solutions typically consist of a camera or sensors, appropriate lighting, and image processing software. However, with the rapid advancements in artificial intelligence, particularly deep learning, newer VISs offer superior performance and enhanced accuracy in defect detection.

1.2 Image processing

Image processing is a method used to perform operations on an image to enhance the perception of its content or extract useful information. It involves manipulating and analyzing images to improve their quality, highlight important features, and derive meaningful insights. Image processing techniques are used in various applications, including medical imaging, remote sensing, industrial inspection, and computer vision.

In industry, image processing is used to enhance efficiency and accuracy. Here are some notable uses of image processing in different applications :

- Medical imaging : used to enhance and analyze medical images acquired using X-rays, MRIs, CT scans, and ultrasounds to help diagnose diseases and plan treatments. and Real-time image processing aids in minimally invasive surgeries by providing clear and enhanced images of the surgical site[1].

- Autonomous vehicle : Image processing algorithms are critical for object detection, lane detection, and navigation in self-driving cars[2].
- Security systems : Image processing used in security systems to identify individuals based on facial features,and Processing images from surveillance cameras allows to detect and alert about unauthorized access or suspicious activities [3].

1.3 Visual inspection system (VIS)

VIS have transformed quality control across various industrial fields by automating the detection and analysis of defects, thus enhancing product reliability and production efficiency. These systems employ advanced imaging technologies and algorithms to inspect products for surface defects, dimensional accuracy, and proper assembly, ensuring that only high-quality items reach the market.

1.3.1 VIS component analysis

Visual inspection systems are composed of several key components, each playing a crucial role in the inspection process:

- **High-resolution cameras** capture images of the product. The type of camera (e.g., CCD, CMOS) depends on the specific application and required resolution,In addition to cameras, various sensors (e.g., infrared, X-ray) can be used to capture different types of data, such as thermal images or internal structures [4].
- **Proper lighting** is essential to ensure that images are clear and defects are visible. Types of lighting include LED, fluorescent, and laser, with configurations such as backlighting, ring lighting, and directional lighting.
- **High-performance processors** CPUs, GPUs, or FPGAs are required to handle the complex computations involved in image processing and analysis.
- **Controllers** that manage the operation of cameras, lighting, and actuators, ensuring synchronized and efficient inspection processes.
- **User Interfaces** which provide operators with real-time feedback, control options, and visualization of inspection results.

1.3.2 VIS classification

Visual inspection could be classified based on the need of humain intervention into two main categories :

Human visual inspection

Historically, inspection relied heavily on human operators to visually inspect products for defects. While effective, this method is subjective and can be prone to human error. However, human inspectors bring expertise and judgment that can be valuable in certain contexts[5].

Figure 1.1: Human inspection

Machine visual inspection

Machine vision systems encompass a range of technologies that employ cameras, lighting, and image processing algorithms to automate VI tasks. These systems can perform various inspection tasks, including dimensional measurement, surface inspection, defect detection, and barcode reading, across industries such as food and beverage, packaging, and textiles[6].

Figure 1.2: Machine vision systems

- Infrared thermography represents an example of a non-destructive inspection method that uses thermal imaging cameras to detect variations in temperature. This method is commonly used in industries such as building inspection and predictive maintenance to identify defects like heat loss, electrical faults, and mechanical wear [7].

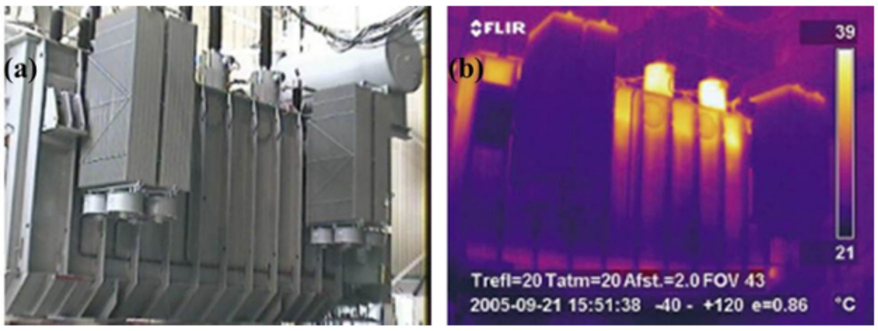

(a)
(b)
FLIR
39
21
Trefl=20 Tatm=20 Afst.=2.0 FOV 43
2005-09-21 15:51:38 -40 - +120 e=0.86
°C

Figure 1.3: Infrared thermography for condition monitoring[7]

- With advancements in artificial intelligence, particularly deep learning, VIS can now leverage neural networks to learn and recognize patterns indicative of defects. Deep learning-based inspection offers superior accuracy and adaptability, making it suitable for complex inspection tasks in industries like aerospace, medical devices, and consumer electronics [8] . the next images illustrates how an AI model detects wheel discs and brake components on an automobile assembly line.

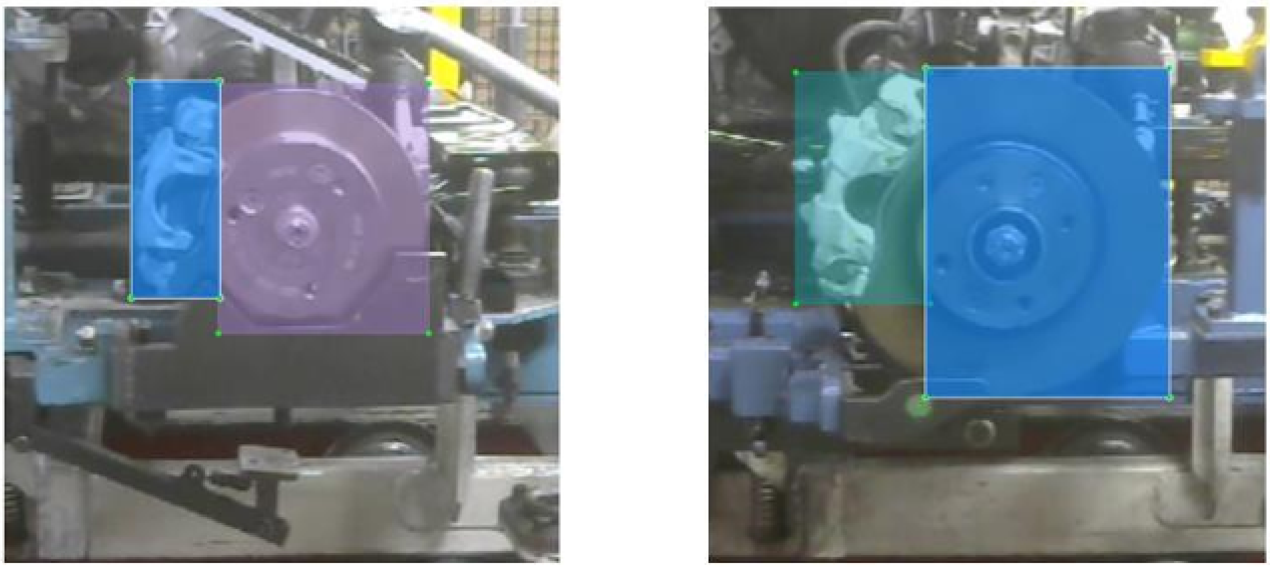


Figure 1.4: Deep learning models for visual inspection on automotive assembling line[8]

These VI methodes are just a few examples of the diverse techniques employed across industries to ensure product quality, safety, and compliance with regulations. Each method has its advantages and applications, contributing to efficient manufacturing processes and customer satisfaction.

1.3.3 VIS operational modes

Another approach suggest broadly categorizing the VI systems into inline and end-of-line visual inspection systems

Inline Visual Inspection Systems

Inline VIS are integrated directly into the production line. Their primary purpose is to perform real-time inspection of products as they are being manufactured. This allows for continuous quality control and minimize the risk of defective products reaching the market.This kind of systems offer a lot of advantages such as [9]:

- Immediate detection and correction of defects.
- Reduces downtime and increases overall production efficiency.
- Early detection of defects minimizes waste and rework costs

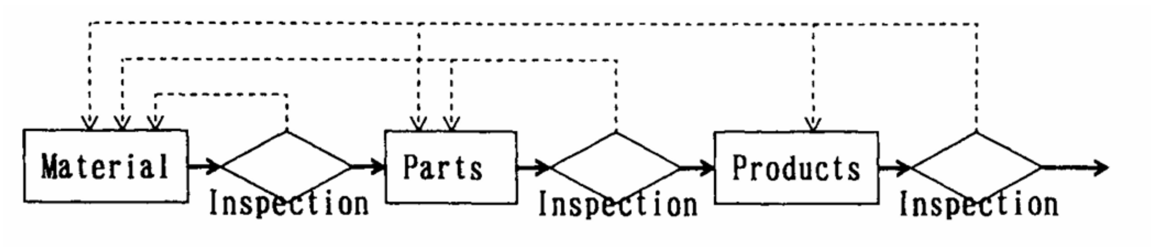


Figure 1.5: Inline visual inspection systems

End-of-Line Visual Inspection Systems

End-of-line VIS are deployed at the final stage of the production process. These systems inspect finished products before they are packaged and shipped. The goal is to ensure that only products meeting quality standards leave the factory. such systems offer interesting advantages that include [10] :

- Ensuring packaging integrity and label accuracy.
- Checking for cosmetic defects in products
- Final inspection of assembled parts and systems .

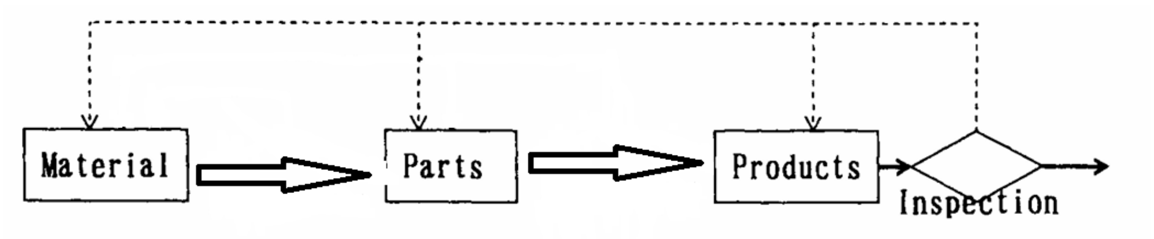


Figure 1.6: End line visual inspection systems

1.4 Printed Circuit Boards (PCBs)

A PCB is a flat board made from a non-conductive material, typically fiberglass, onto which a conductive layer, usually copper, is etched to create pathways for electrical current to travel. These pathways connect various electronic components, such as resistors, transistors, and integrated circuits, allowing the components to communicate and function together as a system.

importance of PCBs can be shown through the characteristics described below :

- PCBs enable the development of compact and sophisticated electronic devices. This miniaturization is critical for creating portable electronics like smartphones, tablets, and wearables, which are integral to modern life.
- The standardized design and manufacturing processes of PCBs allow for the mass production of electronic devices. This standardization reduces production costs and improves the quality and consistency of products.
- Modern PCBs are designed to support advanced functionalities such as high-speed data transfer, effective power management, and signal integrity. These capabilities are essential for the performance of high-frequency and high-power devices like servers, routers, and power.

1.5 Automated VIS of PCBs

An automated VIS for PCBs integrates several components to ensure efficient and accurate inspection. The main components of such a system include process control, data collection, rejection mechanisms, image acquisition, and illumination[5] .As it is illustrated in the following figure.

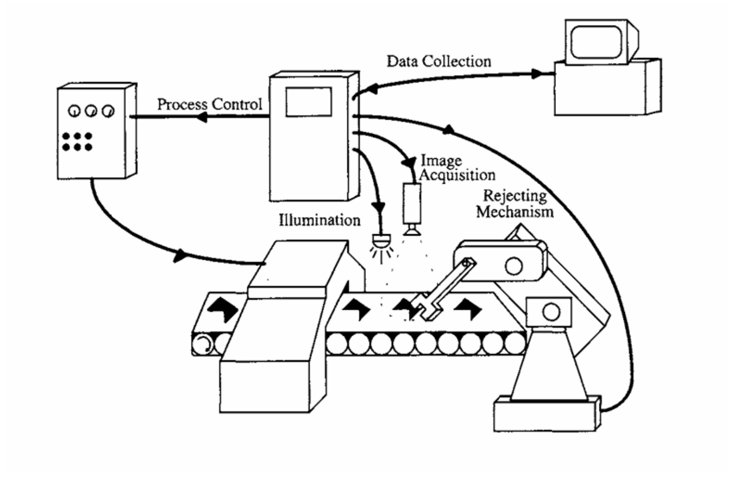


Figure 1.7: Automated visual inspection system for PCBs[5]

1.5.1 Process control

Process control is the central management system that oversees and coordinates all aspects of the inspection process. It involves:

- **Workflow Management:** Scheduling and directing PCBs through various inspection stages.
- **Parameter Settings:** Defining inspection parameters, such as resolution, thresholds for defects, and acceptable tolerance levels.
- **Real-Time Monitoring:** Continuously tracking the system's performance and making adjustments as needed.
- **Feedback Loop:** Utilizing data from inspections to make immediate adjustments to the production process, thereby enhancing overall quality control.

1.5.2 Image acquisition and illumination

Image acquisition is the process of capturing high-quality images of PCBs for analysis. This involves:

- **Cameras:** High-resolution cameras positioned to capture detailed images of PCBs. These may include multiple cameras for different angles and aspects of the PCB.
- **Movement Mechanism:** Conveyor belts or robotic arms to position PCBs accurately under the cameras.
- **Triggering System:** Sensors or software triggers that ensure images are captured at the precise moment for optimal clarity.
- **Lighting Setup:** LED or other appropriate lighting sources positioned to minimize shadows and glare, ensuring consistent illumination across the PCB surface.
- **Adjustable Intensity:** The ability to adjust the intensity and angle of lighting to highlight different features and potential defects.

1.5.3 Data collection

Data collection involves gathering and storing information throughout the inspection process. This includes:

- **Inspection Results:** Logging details of each inspection, including pass/fail status and types of defects detected.

- **Statistical Analysis:** Aggregating data to identify trends, common defects, and areas needing improvement.
- **Traceability:** Maintaining records of each PCB's inspection history for quality assurance and compliance purposes.
- **Reporting:** Generating reports for quality control teams, production managers, and regulatory bodies.

1.5.4 Rejection mechanism

The rejection mechanism is responsible for segregating defective PCBs from those that meet quality standards. It includes:

- **Defect Identification:** Utilizing inspection data to determine if a PCB fails to meet specified criteria.
- **Sorting System:** Automated conveyors and sorting mechanisms that physically separate defective PCBs from the production line.
- **Alerts and Notifications:** Notifying operators or maintenance personnel about repeated defects or system issues requiring attention.

By integrating these components, an automated VIS for PCBs ensures high accuracy, consistency, and efficiency in identifying and addressing defects, ultimately improving the quality and reliability of electronic products [7].

1.6 Deep Learning for PCB inspection

The realm of PCB inspection has undergone a significant transformation with the advent of deep learning (DL). Traditional methods often relied on manual inspection or basic image processing algorithms, which could be time-consuming, prone to human error, and less efficient in identifying subtle defects.

Deep learning has revolutionized PCB inspection by offering superior capabilities compared to traditional methods. Unlike relying on pre-defined defect patterns, DL algorithms excel at automatically extracting intricate features from vast datasets of PCB images. This allows them to identify even the subtlest defects like misaligned components, solder issues, and microcracks. Additionally, deep learning automates the inspection process, significantly improving efficiency. Faster inspection times, increased production throughput, and reduced labor costs are all benefits of this automation. Furthermore, deep learning models are remarkably adaptable and scalable. They can be trained on a wide range of PCB designs and defect types, making them suitable for various scenarios. As production volumes increase, these models can seamlessly handle larger datasets, ensuring consistent performance. Finally, deep learning models continuously improve as new data is fed into the system. This allows them to become more accurate and identify new types of defects, keeping the inspection process at the cutting edge of technology.

1.7 Conclusion

This chapter has provided a comprehensive overview of VISs, particularly their application in various industries for defect detection and quality assurance. By integrating advanced imaging technologies and sophisticated algorithms, these systems have revolutionized quality control processes, ensuring enhanced product reliability and efficiency. Key components of VISs, such as high-resolution cameras, effective lighting setups, and powerful image processing capabilities, play a crucial role in identifying and addressing defects.

The chapter specifically highlighted the importance of printed circuit boards (PCBs) in modern electronic devices and the critical role of automated VISs in maintaining their quality and functionality.

This sets a solid foundation for the subsequent chapters, which will delve deeper into specific methodologies and applications of VISs .

Chapter 2

Theoretical Framework / Image Processing Techniques

2.1 Introduction

In industrial applications, image processing is essential for automating tasks like defect detection, dimensional measurement, and anomaly identification. This technology enhances efficiency and reduces human error in production lines.

Developing a visual inspection system for PCBs relies heavily on advanced image processing techniques. This chapter covers key methodologies:

- Image Registration and Geometrical Transformations
- Feature extraction with Scale-Invariant Feature Transform (SIFT)
- Template Matching using ormalized Cross-Correlation (NCC)
- AI with You Only Look Once (YOLO) Algorithm

By integrating these techniques, our visual inspection system achieves precise and comprehensive analysis, combining traditional image processing with advanced AI models for robust PCB inspection. This chapter provides an overview of these methods, illustrating their synergy in enhancing manufacturing quality control.

2.2 Image registration

Image registration takes two images as input and aims to find a geometrical transformation. Geometric or spatial transformations modify the spatial relationships between pixels in an image. The image can be made larger or smaller, rotated, shifted, or otherwise stretched in a variety of ways. This transformation acts like a mathematical bridge, mapping corresponding points from one image to the other. A successful registration ensures these corresponding points are accurately matched.

2.2.1 Geometrical transformation

In the context of registration, each image is associated with a coordinate system, defining a unique space for that view.

Geometric transformations are like rules that move points from one picture to another. Each point is represented as a column of numbers called a column vector. When we apply a transformation to a point, it changes its position, giving us a new point in the transformed image.

If a point in the first picture matches a point in the second picture, after the transformation, these points should end up very close to each other. If they don't, it means there's a mistake in the transformation, and the points are not aligned properly. This difference between where the points should be and where they end up is what we call an error

in the scope of this application mappings points from the space X one image to the space Y of a second image. The transformation γ applied to a point in X represented by the column vector x produces a transformed point x'.

$$x' = \gamma(x) \tag{2.1}$$

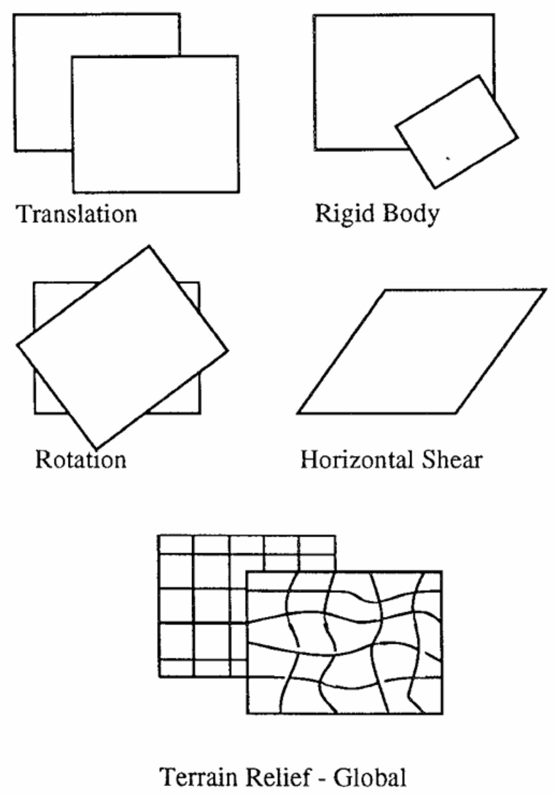


Figure 2.1: Geometrical transformations [12]

Rigid and nonrigid transformation are fundamental concepts in image processing that determine how images are spatially altered and aligned for accurate analysis.

1. **Rigid Transformation**

Rigid transformations, or rigid mappings, are defined as geometrical transformations that preserve all distances. These transformations also preserve the straightness of lines (and the planarity of surfaces) and all nonzero angles between straight lines. Registration problems that are limited to rigid transformations are called rigid registration problems. Rigid transformations are simple to specify, and there are several methods of doing so. In each method, there are two components to the specification, a translation and a rotation. The translation is a three dimensional vector t that may be specified by giving its three coordinates t_x,t_y,t_z relative to a set of x,y,z . Rotations can be parameterized in terms of three angles of rotation θ_x , θ_y , θ_z . If γ is rigid then.

$$x' = Rx + t \tag{2.2}$$

where R is a 3x3 orthogonal matrix,this class of matrices includes both the proper and improper rotations Proper rotations can be parameterized in terms of three angles of rotation, about the respective Cartesian axes,θ_x, θ_y, θ_z the so-called “Euler angles”.[11]

$$R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\theta_x) & -sin(\theta_x) \\ 0 & sin(\theta_x) & cos(\theta_x) \end{bmatrix} \begin{bmatrix} cos(\theta_y) & 0 & sin(\theta_y) \\ 0 & 1 & 0 \\ -sin(\theta_y) & 0 & cos(\theta_y) \end{bmatrix} \begin{bmatrix} cos(\theta_z) & -sin(\theta_z) & 0 \\ sin(\theta_z) & cos(\theta_z) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R = \begin{bmatrix} cos(\theta_y)cos(\theta_z) & -sin(\theta_z)cos(\theta_x) + sin(\theta_x)cos(\theta_z)sin(\theta_y) & sin(\theta_x)sin(\theta_y) + cos(\theta_x)cos(\theta_z)sin(\theta_y) \\ cos(\theta_y)sin(\theta_z) & cos(\theta_z)cos(\theta_x) + sin(\theta_y)sin(\theta_z)sin(\theta_x) & -sin(\theta_x)cos(\theta_z) + sin(\theta_y)sin(\theta_z)cos(\theta_x) \\ -sin(\theta_y) & sin(\theta_x)cos(\theta_y) & cos(\theta_y)cos(\theta_x) \end{bmatrix} \tag{2.3}$$

2. **Nonrigid Transformation**

   Nonrigid transformation refers to a spatial transformation that allows for local deformations or distortions, enabling the alignment of images with varying shapes or structures.

   There are multiple types of non-rigid transformations, including [12] :

   Scaling transformations, affine transformations, projective transformations, and perspective transformations. However, in the scope of this research, we will be focusing on scaling transformations

   Scaling transformation is a type of non-rigid transformation that uniformly resizes the spatial dimensions of an object or image. This transformation can enlarge or shrink the object or image along each axis independently or uniformly.

   scaling transformation can be represented by a diagonal scaling matrix S=diag(s_x,s_y,s_z)with scaling factors S_x , S_y and S_z for the x-axis, y-axis and z-axis respectively.

$$x' = RSx + t \tag{2.4}$$

The two images in Figure 2.2 depicts rigid transformations, specifically translation and rotation, which maintain the jet's shape while altering its position and orientation. In contrast, the second set in Figure 2.3 illustrates nonrigid transformations, focusing on zooming, where the size of the fighter jet image changes while preserving its overall shape

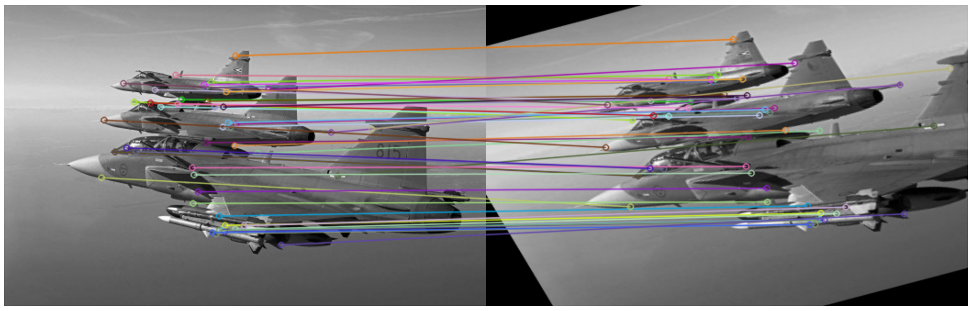


Figure 2.2: Rigid geometrical transformation

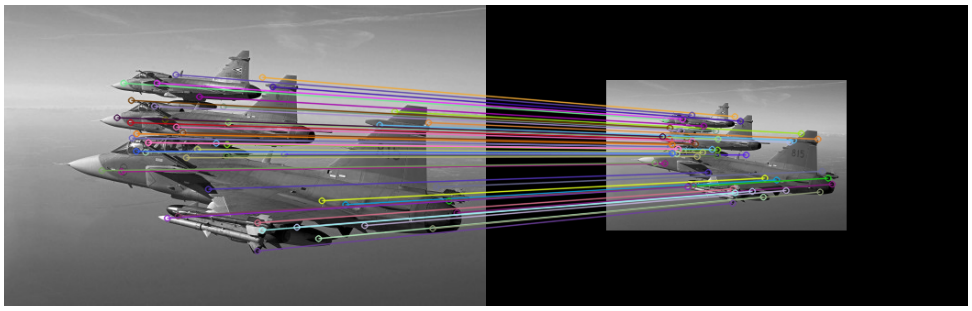


Figure 2.3: Non rigid geometrical transformation

2.3 Features extraction algorithmes

Feature extraction algorithms are used to identify and extract distinctive and informative characteristics from an image. These features represent the essential information within the image that can be used for various tasks like object recognition , image retrieval , image segmentation and image compression.

The principal role of a feature extraction algorithm is to transform raw data into a set of informative features. These features capture the meaningful aspects of the data .

All the features extraction algorithm work with the same principal steps defined for the scope of image registration :

- **Feature detection:**
  The algorithm locates potential zones within the images that could contain informative features.
- **Feature description:**
  Once potential features are detected, the algorithm extracts a descriptor for each

feature. This descriptor is a mathematical representation that captures the characteristics of the feature in a way that is robust to variations like illumination changes or rotations.

- **Feature matching:**
  After features are described in both images, the algorithm attempts to find corresponding features between the images. This involves comparing feature descriptors and identifying the best match (or closest match) in the other image.

2.4 Scale-Invariant Feature Transform (SIFT)

SIFT was developed by David Lowe in 1999 as a method to detect and describe distinctive local features in images. The purpose of SIFT is to identify keypoints in images that are invariant to scale, rotation, and illumination changes. These keypoints serve as robust landmarks, allowing for reliable matching between different instances of objects or scenes in various applications such as image stitching, object recognition, and 3D reconstruction.

2.4.1 SIFT principle

SIFT detects a serie of keypoints from a multi-scale image representation. This multi-scale representation consists of the Gaussian scale-space . Each keypoint is a blob-like structure whose center position (x, y) and characteristic scaleσ are accurately located. SIFT computes the dominant orientation θ over a region surrounding each one of these keypoints. For each keypoint,

The quadruple (x, y, σ, θ) defines the center, size and orientation of a normalized patch where the SIFT descriptor is computed. As a result of this normalization, SIFT keypoint descriptors are in theory invariant to any translation, rotation and scale change. The descriptor encodes the spatial gradient distribution around a keypoint by a vector.

This feature vector is generally used to match keypoints extracted from different images.

2.4.2 Keypoints definition

A keypoint in SIFT refers to a distinctive and stable location within an image that can be reliably detected and matched across different viewpoints and illumination conditions. SIFT relies on a multi-stage process to identify keypoints[13] .

(a) Scale-Space Construction:
The Gaussian scale-space representation consists of a family of increasingly blurred

images. This blurring process mimics the loss of detail that occurs when capturing a scene from varying distances. Scale-space provides SIFT with scale invariance by simulating snapshots of a scene at different scales [14]. The Gaussian function used for blurring is defined as:

$$G(x,y;\sigma) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) \tag{2.5}$$

Properties of the Gaussian Kernel:

- **Bell-shaped Curve:** The function peaks at the center (x = 0, y = 0) and smoothly decreases as we move away, controlled by the parameter σ.
- **Symmetry:** The Gaussian function is symmetric around its center (x = 0, y = 0).
- **Zero-Mean:** The integral over the entire plane is zero, indicating no bias towards positive or negative values.

By convolving the original image with Gaussian kernels at different scales (different σ values), we create a series of blurred images, each representing the original image smoothed to a different degree. This collection of images forms the scale-space representation.

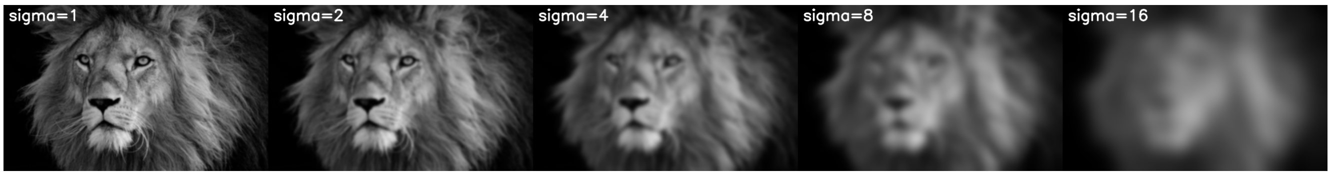


Figure 2.4: Gaussian scale-space filtering: results with sigma values from 1 to 16

The process of constructing scale-space begins with convolving the original image with Gaussian kernels at various scales (σ). This results in multiple blurred versions of the original image, each capturing the image details at different levels of smoothing.

(b) Difference-of-Gaussian (DoG) Function:
The next stage involves identifying potential keypoints within the scale-space. SIFT achieves this by computing the DoG function across neighboring scales.

$$D(x,y,\sigma) = (G(x,y,k\sigma) - G(x,y,\sigma)) \tag{2.6}$$

The DoG function emphasizes regions with significant intensity variations across scales. Edges, corners, and other image features that are stable across scales tend to have high values in the DoG function, making them potential candidates for keypoints.

(c) Keypoint Localization:
SIFT analyzes the DoG function at multiple scales and across neighboring pixels. A potential keypoint is identified if it's a local maximum or minimum in both:
Scale: Compared to its neighbors in the same scale of the DoG image.
Space: Compared to its neighbors in adjacent scales (typically one scale above and below) at the same location in the DoG image.

(d) Keypoint Refinement: Subpixel Localization:
While the DoG function identifies potential keypoint locations, they might not be exact due to image interpolation. SIFT refines the location by fitting a quadratic function around the DoG peak and finding the subpixel position with the maximum value. Orientation Assignment:
To achieve rotation invariance, SIFT assigns a dominant orientation to each keypoint. This is done by analyzing a local image gradient histogram around the keypoint. The direction with the highest gradient magnitude is considered the keypoint's orientation.

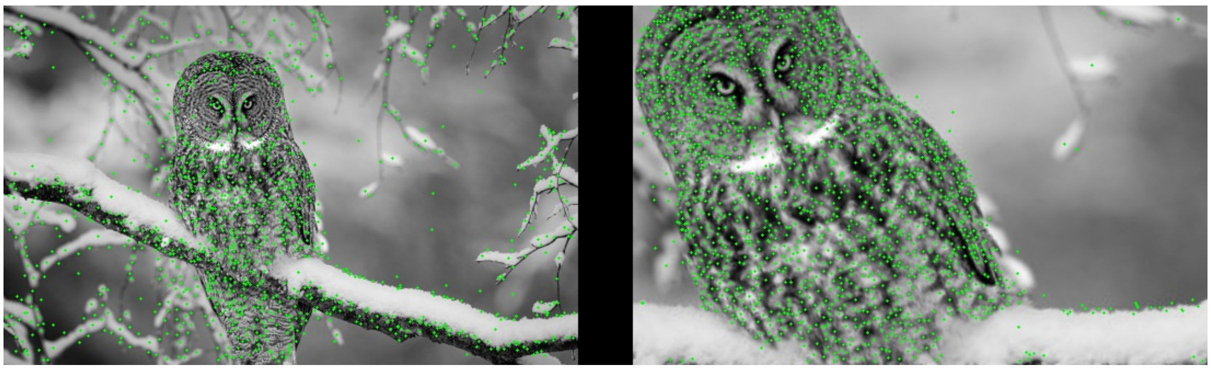


Figure 2.5: Keypoints extraction from the test and reference images

2.4.3 SIFT Keypoint Descriptors

Once SIFT identifies and refines keypoints, it proceeds to create a descriptor for each keypoint. This descriptor is a vector that captures the local characteristics of the image around the keypoint.

SIFT utilizes a technique called a histogram of oriented gradients (HOG) to construct the keypoint descriptor. Gradient Calculation: For each pixel in a small neighborhood around the keypoint, the image gradient magnitude (m) and orientation (θ) are computed. The mathematical formulas for gradient calculation are the same as those mentioned in keypoint refinement (see previous explanation) [14] .

(a) Orientation Binning: The neighborhood is divided into subregions (typically 4x4 grid), and the gradient information (magnitude and orientation) is accumulated into a histogram for each subregion. These histograms capture the distribution of local gradients within each subregion relative to the keypoint's dominant orientation.

(b) Normalization: To achieve some degree of illumination invariance, the histograms are normalized to a unit length.Descriptor Vector: The individual histograms from each subregion are concatenated to form a single high-dimensional descriptor vector for the keypoint. This vector encodes the local gradient information around the keypoint in a way that is partially invariant to illumination changes.Benefits of HOG-based Descriptors.

(c) Informative Representation: The HOG descriptor captures the local spatial distribution of gradients with respect to the keypoint's orientation. This allows for encoding both edge and corner information, making it suitable for describing various image features.Partial Illumination Invariance: By normalizing the histograms, SIFT reduces the effect of varying illumination on the descriptor values, improving matching performance across different lighting conditions.

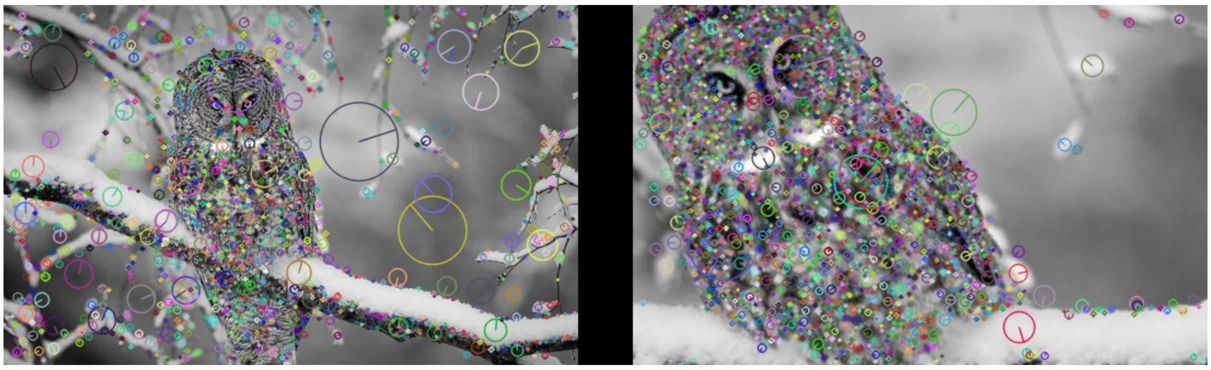


Figure 2.6: Keypoints description in the test and reference images

2.4.4 SIFT Keypoint Matching Process

Once SIFT extracts and describes keypoints in two images (image A and image B), the matching stage attempts to find keypoints in image B that correspond to each keypoint in image A [14].

(a) Descriptor distance calculation: To compare keypoint descriptors, SIFT employs a distance metric. A common choice is the Euclidean distance, which measures the difference in intensity values between corresponding elements of the descriptor vectors.

(b) Nearest Neighbor Search: For each keypoint in image A, SIFT searches for its closest match in image B based on the chosen distance metric. This is typically done using efficient nearest neighbor search algorithms like k-nearest neighbors (k-NN). Ratio Test: To improve matching accuracy and reduce the number of false matches, SIFT employs a ratio test. This test involves finding the second-closest match for the keypoint in image B.

(c) Thresholding: If the distance between the first (closest) and second-closest matches is less than a predefined threshold (typically between 0.6 and 0.8), the first match is considered a good candidate. This thresholding helps eliminate ambiguous matches where a keypoint in image A might have similar descriptors to multiple keypoints in image B.

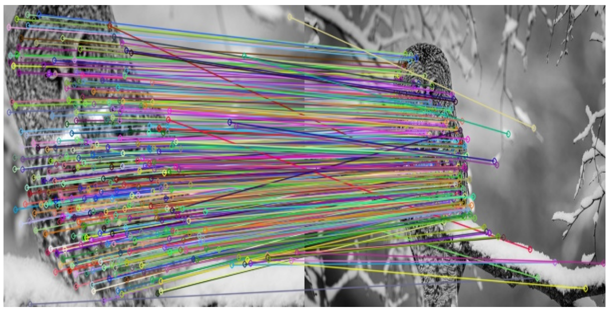


Figure 2.7: Matching the reference and test images

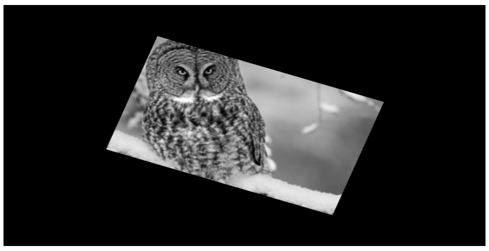


Figure 2.8: The registred test image

2.5 Image comparison using feature extraction algorithms

Once two images are processed using the SIFT algorithm and the good key point matches are obtained, the results can be interpreted in various ways. Generally, a higher number of good matches suggests that the images are more similar. This means that more key points from one image have corresponding key points in the other image, indicating a stronger resemblance between the two [15].

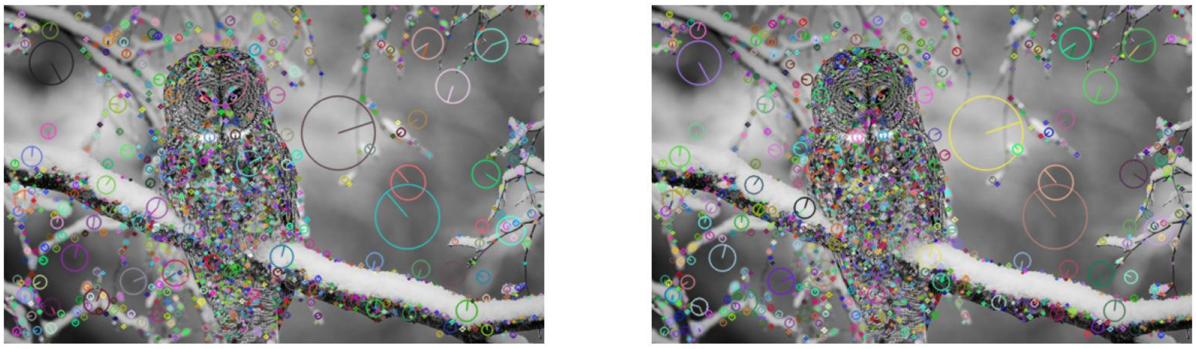


Figure 2.9: SIFT Matching reference Image and test Image (same image)

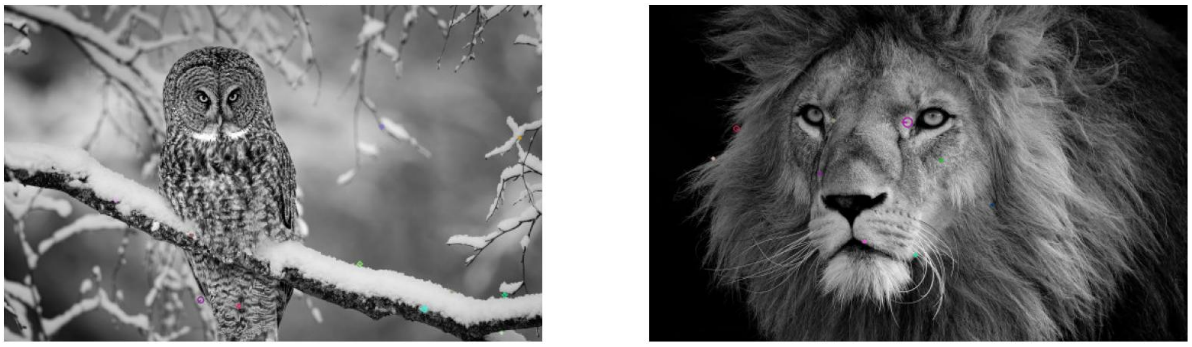


Figure 2.10: SIFT matching reference image and test Image (different images)

2.6 Template Matching

TM is a technique used in image processing to locate specific objects within an image. It works by comparing a smaller image, called a template, to a larger image (search image) to find regions that closely resemble the template.

There are two main applications of TM :

1. Search: This involves finding occurrences of a single template within the search image. Here, the template remains fixed, and the algorithm scans the search image to locate areas with the best match.

2. Classification: This involves classifying a pre-extracted image region (sample) against a set of template prototypes. In this case, the sample is fixed, and the algorithm compares it to each template to determine the most likely match.

The term TM will refer to the detection of the *nxn* subimage g within *NxN* search area s that best matches *nxn* template *f*

2.6.1 Matching filters

Exhaustive search TM algorithms can be conceptualized as applying a matching filter to a search area. This filter, denoted by h(x, y), aims to identify instances of a template f(x, y) within the search area s(x, y). The filter is designed to produce a maximum response at locations where a region of the search area closely resembles the template. The filter response, denoted by z(x, y), is calculated by convolving the search area with the matching filter, represented mathematically as:

$$z(x,y) = s(x,y) * h(x,y). \tag{2.7}$$

2.6.2 Correlation Measures

TM encompasses various methods to assess the similarity between a template image and regions within a search image. While techniques like Sum of Squared Differences (SSD) and template matching with transformations exist, this research focuses on the Normalized Cross-Correlation (NCC) method and a specific feature-based matching approach [16].

2.6.3 Normalized cross-correlation for template matching

NCC is a popular technique in template matching used to assess the similarity between a template (fixed image) and a region within a larger search image

f(x, y) represent the template image with pixel intensity values at coordinates (x, y).

s(x, y) represent the search image with pixel intensity values at coordinates (x, y). The template size is denoted by m x n (m rows and n columns).

The window size in the search image where we compare with the template is also m x n (same size as the template).

The Cross-Correlation Function (CC) between the template (f) and a window in the search image (s) centered at position (x, y) is calculated as:

$$CC(x,y) = \sum_{i=0}^{m-1}\sum_{j=0}^{n-1} f(i,j)\cdot s(i+x,j+y) \tag{2.8}$$

Raw CC values are sensitive to overall image intensity variations. NCC addresses this by normalizing the CC values to a range of -1 to 1.

$$NCC(x,y) = \frac{CC(x,y)}{\sqrt{\sum_{i=0}^{m-1}\sum_{j=0}^{n-1} f(i,j)^2 \cdot \sum_{i=0}^{m-1}\sum_{j=0}^{n-1} s(i+x,j+y)^2}} \tag{2.9}$$

In template matching, we calculate the NCC for the template at different positions (x, y) within the search image. The location with the highest NCC value is considered the most likely match for the template in the search image. This indicates the window in the search image that exhibits the strongest similarity to the template based on their intensity patterns [16] .

Limitations of NCC:

Can be sensitive to noise and illumination changes in the image. May struggle with occlusions (part of the template is hidden) or complex backgrounds.

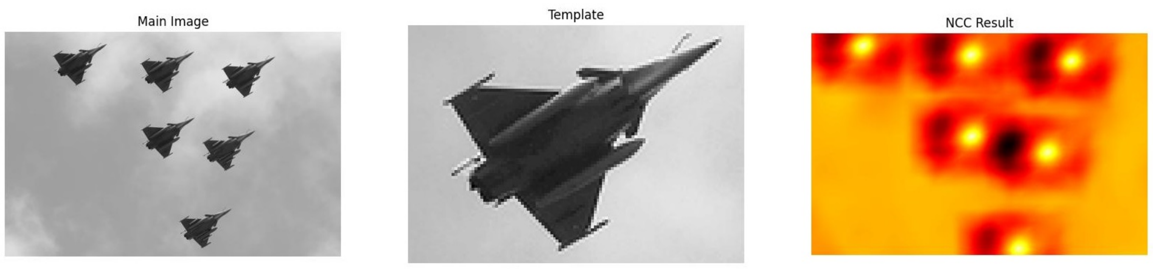


Figure 2.11: NCC result map

brighter regions indicate higher similarity between the template and the corresponding region of the main image. Darker regions represent lower similarity. This visualization helps in identifying the location(s) where the template best matches the main image.

2.6.4 SIFT template matching

SIFT offers robustness to geometric transformations like scaling, rotation, and illumination changes, making it a valuable tool for tasks where the template's appearance might vary in the search image.

1. Feature Extraction in the Template:

   SIFT identifies keypoints within the image that represent distinctive features like corners, edges, and blobs. For each keypoint, SIFT calculates a descriptor..

2. Feature Detection in the Search Image:

   SIFT is then applied to the search image. Similar to the template, SIFT identifies keypoints and calculates descriptors for each keypoint in the search image.

3. Feature Matching:

   Once features are extracted from both the template and the search image, the algorithm performs matching between them.

4. Template Localization and Verification:

   After identifying potential feature matches, the algorithm performs geometric verification to determine the template's location within the search image. This is often achieved by techniques like RANSAC (Random Sample Consensus). RANSAC helps eliminate false matches by ensuring a sufficient number of matching features are clustered within a spatially coherent region.

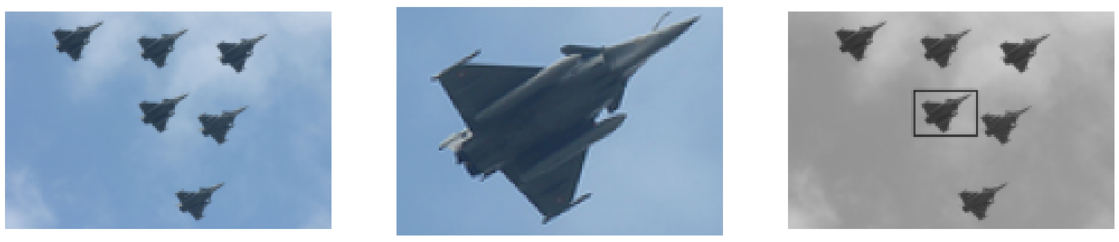


Figure 2.12: SIFT matching results with high match score

2.7 Deep learning models for visual inspection

Deep learning (DL) is a revolutionary subfield of artificial intelligence (AI) that has transformed various domains, including image processing. Inspired by the intricate structure of the human brain, DL employs artificial neural networks (ANNs) to learn from vast amounts of data and perform complex tasks with remarkable accuracy. This chapter delves into the fascinating realm of DL and its applications in image processing.

Artificial Neural Networks Principles and Fundamental Concepts

An artificial neural network (ANN) is a computational model that mimics the brain's information processing. It consists of numerous units (neurons) connected by weights and biases (Figure 1.7). Depending on the activation function, the model can be used for classification tasks, regression tasks, or other functions. In classification, the multilayer Perceptron network represents the basic model (Figure 1.8). It is composed of:

- **Input Layer:** An input layer has one unit for each input feature. These units transmit their values to the next layer without performing any calculations.
- **Hidden Layers:** One or more hidden layers. Each layer contains one or more units (neurons) that perform a calculation. The calculation involves summing the weighted inputs from the previous layer and the bias term, then applying an activation function. The activation function (f) can be linear, sigmoid, hyperbolic tangent (tanh), ReLU, etc.
- **Neuron Output Calculation:** The output of a neuron is calculated according to the following equation:
$$y = f\left(\sum_i (w_i \times \text{inputs}) + b_i\right) \tag{2.10}$$
- **Output Layer:** The output layer has one neuron for each class. These neurons produce the final output of the network by performing the same calculation as the hidden units.
- **Learning Weights and Biases:** The weights and biases of the network are learned by minimizing a loss function that measures the difference between the network's output and the desired output. The loss function can be mean squared error, cross-entropy, hinge loss, etc. The process of training involves adjusting these weights and biases to reduce the loss.

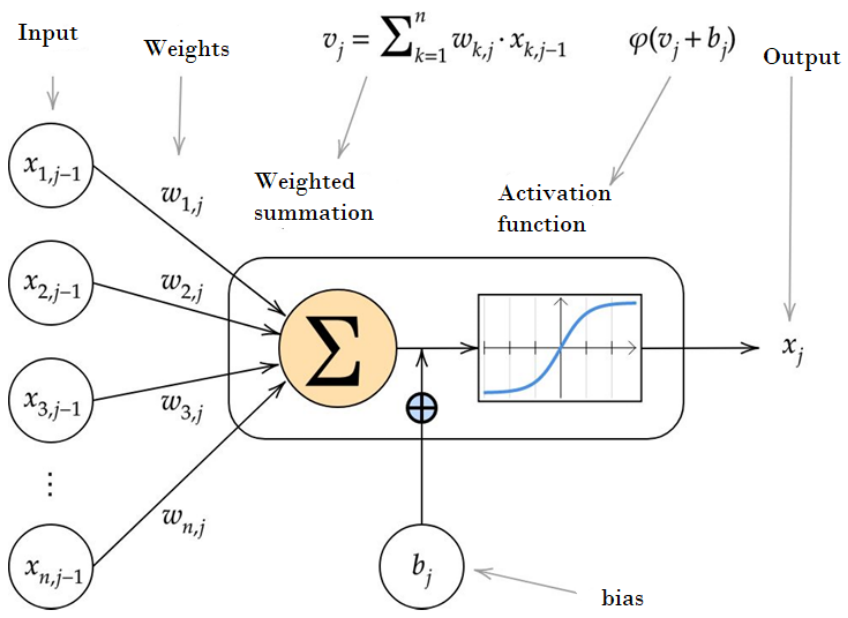


Figure 2.13: Computational Mechanism of an Artificial Neuron

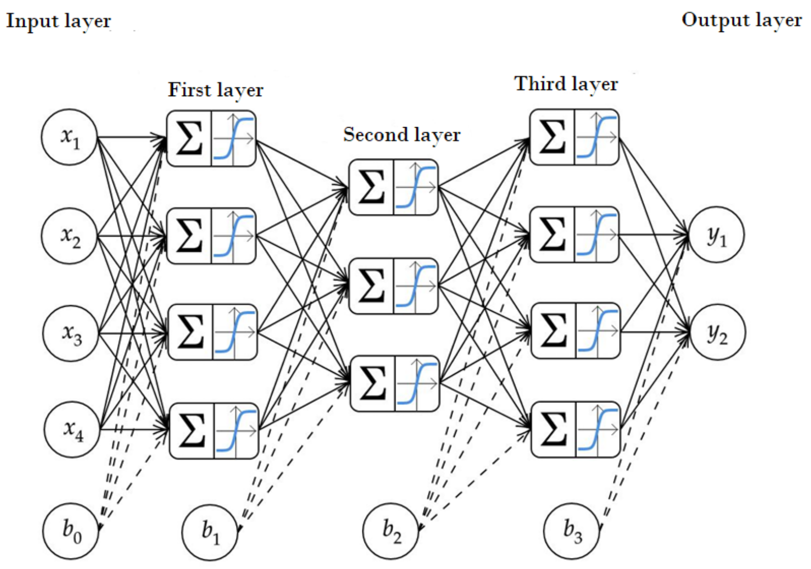


Figure 2.14: Example of a multilayer perceptron with 5 layers

2.8 Training fundamentals

Neural network training fundamentals involve optimizing model parameters through backpropagation and gradient descent to minimize the error between predicted and actual outcomes. This process requires careful tuning of hyperparameters, such as learning rate and batch size, to achieve optimal performance.

Loss function

The loss function in a neural network quantifies the difference between predicted and actual outputs. It guides training by minimizing this difference. Cross-entropy loss is popular for classification tasks, penalizing high-confidence incorrect predictions. Mean squared error (MSE) is common for regression tasks. Other loss functions like mean absolute error (MAE) can also be used. The choice depends on the problem [17].

Back propagation The aim of the backpropagation algorithm is to determine the effect of each weight and bias in the network on the loss function. By doing so, it becomes possible to adjust the weights and biases using an appropriate learning algorithm, such as stochastic gradient descent. The objective of backpropagation is to compute $\frac{\partial L}{\partial W_{jk}^{(l)}}$ where $W_{jk}^{(l)}$ represents the weight associated with the connection between the k-th neuron in the $(l-1)$-th layer and the j-th neuron in the l-th layer of a neural network. These values are obtained by recursively applying the chain rule from the network's output to its input.

Optimization

Optimization is the process of finding the optimal parameters W that minimize the loss function $J(W)$, which measures the error of the network's predictions on the data. Optimization in neural networks is challenging because the loss function is often non-convex, meaning it has many local minima. To find the global minimum, various optimization methods update the parameters based on the gradient of the loss function, which indicates the direction of the steepest descent. Some of these methods include gradient descent, stochastic gradient descent, momentum, and adaptive learning rates (ADAM). These methods aim to improve the speed and accuracy of optimization in neural networks [18].

Stochastic Gradient Descent (SGD) is an optimization algorithm used in machine learning to train models by updating their parameters based on gradients calculated from randomly selected mini-batches of training data. It strikes a balance between computational efficiency and convergence speed, making it suitable for training large-scale and deep learning models. SGD minimizes the loss function by adjusting parameters in the direction opposite to the gradients, allowing it to escape local minima and explore the parameter space effectively. However, it may result in slower convergence and fluctuations. Variants like mini-batch SGD and momentum-based methods have been developed to enhance performance and stability.

Adam is a powerful optimization algorithm designed for deep learning, combining the concepts of Momentum and RMSprop. It dynamically adjusts learning rates based on past gradients and second moments to optimize parameter updates. By integrating momentum and adaptive learning rates, Adam enhances training by ensuring the correct direction and speed of parameter updates. This algorithm is known for its fast convergence and consistently produces excellent results, often outperforming traditional stochastic gradient descent (SGD). Due to its effectiveness, Adam is widely utilized in various deep learning tasks.

2.9 Learning Quality and Regularization methods

Regularization methods are techniques used to prevent overfitting in machine learning models. Overfitting occurs when a model or neural network performs exceptionally well on the training data but struggles to generalize to new, unseen data. This phenomenon arises when the model begins to memorize specific details and noise in the training set rather than learning the underlying patterns that would apply more broadly. Conversely, underfitting occurs when the model fails to capture the complexities of the data, resulting in poor performance on both the training and test sets. To combat overfitting, several regularization methods have been developed. These include weight decay techniques such as L1 and L2 regularization, which penalize large weights to encourage simpler models. Additionally, techniques like batch normalization, dropout, and early stopping are employed to enhance model generalization by improving learning stability and reducing reliance on specific features or patterns in the data

Regularization Methods

- Weight Decay: Weight decay is a regularization technique used to prevent overfitting by adding a penalty term to the loss function that depends on the size of the weights. The larger the weights assigned to each input or connection, the more complex and strong the connections become. Therefore, to address this issue, weight decay increases the penalty term to reduce the size of the weights. Two commonly used weight decay regularization techniques are L1 and L2 regularization [19].

    1. L1: Also known as Lasso regression, the penalty term in L1 regularization is the sum of the absolute values of the weights. This leads to a sparser solution, meaning that some of the weights become zero, helping the model focus only on the important weights or features.

       The L1 regularization term is defined as:

       $$\text{L1 regularization term} = \lambda \sum_{i=1}^{m} |w_i| \tag{2.11}$$

       where λ is the L1 regularization parameter.

       The objective function with L1 regularization becomes:

       $$\min_{W} J(W) = \frac{1}{m} \sum_{i=1}^{n} L(y_i, f(x_i;W)) + \lambda \sum_{i=1}^{m} |w_i| \tag{2.12}$$

       This formulation encourages sparsity in the weights, leading to a simpler and more interpretable model.

    2. L2 Regularization This method is also known as Ridge regression and is one of the most common and widely used weight decay methods. Similar to L1 regularization, it adds a penalty term to the objective function, but in this case, the penalty is the

square root of the sum of squares of each weight. This penalty discourages large or extreme values for individual parameters, but unlike L1 regularization, it does not force any weights to zero. Therefore, L2 regularization promotes a more balanced distribution of weight values, reducing the magnitude of parameters and the risk of overfitting.

The L2 regularization term is defined as:

$$\text{L2 regularization term} = \lambda_2 \sum_{i=1}^{m} |w_i|^2 \tag{2.13}$$

where λ_2 represents the L2 regularization parameter.

The objective function with L2 regularization becomes:

$$\min_{W} J(W) = \frac{1}{m} \sum_{i=1}^{n} L(y_i, f(x_i; W)) + \lambda_2 \sum_{i=1}^{m} |w_i|^2 \tag{2.14}$$

- batch normalization :

  Batch normalization can be considered as a form of regularization that helps the network learn more effectively and reduces the risk of overfitting. In particular, it addresses the issue of internal covariate shift, which alters the distribution of inputs for each layer and affects the network's learning rate. This is achieved by ensuring that each layer has the same scale and distribution by linearly transforming the inputs to have zero mean and unit variance. This technique is particularly important for deep recurrent neural networks[19].

- drop out : Dropout is one of the most effective methods for addressing the issue of overfitting. It involves randomly setting a fraction p p of weights to zero during the training process. This reduces the complexity of the model and prevents it from relying too heavily on specific features or patterns in the data, forcing it to be more robust and generalizable. During inference, all weights are used, but they are multiplied by the dropout percentage p [19]

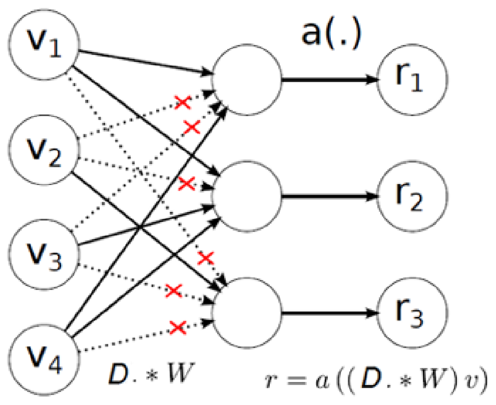


Figure 2.15: Dropout principle in neural networks[19]

- early stopping: Early stopping is a regularization technique commonly used to prevent overfitting in models trained iteratively, such as those using gradient descent and its variants. It involves selecting a model corresponding to an earlier time point than the final epoch based on the reported training and validation errors. Early stopping can be implemented with various criteria, but typically involves monitoring the model's performance on a validation set. The selected model is the one where its performance starts to degrade or its loss increases. Overall, early stopping is widely recommended for training neural networks [20] .

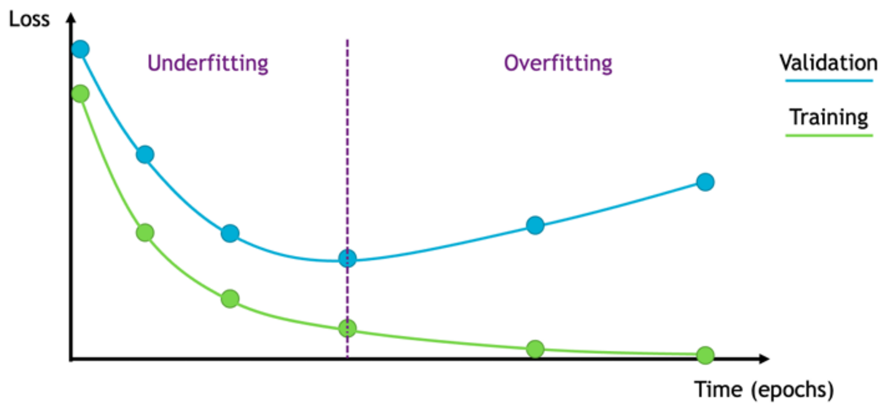


Figure 2.16: Early stopping principle in neural networks[20]

2.10 Learning paradigms in Deep Learning

Deep Learning Approaches There are several methods used in training models in deep learning, including supervised learning, unsupervised learning, semi-supervised learning, reinforcement learning, and transfer learning [21].

- **Supervised Learning:** Supervised learning involves training a network to learn a specific task or function by using both input data and their corresponding outputs. Common problems addressed by supervised learning are regression and classification tasks. Regression tasks predict continuous values, such as size, while classification tasks categorize data into discrete classes, such as classifying products as good or defective. The challenge of supervised learning lies in obtaining labeled data, which can be time-consuming and costly.

- **Unsupervised Learning:** In contrast to supervised learning, unsupervised learning does not use labeled data. Here, the model tries to discover patterns and structures in the input data. Therefore, unsupervised learning can be used for clustering tasks such as the K-means algorithm, anomaly detection like anomaly detection with Generative Adversarial Networks (AnoGAN), and autoencoder algorithms, as well as dimensionality reduction techniques such as Principal Component Analysis (PCA). This approach can be useful when labeled data is unavailable or for exploratory data analysis.

- **Semi-Supervised Learning:** This approach is a mix of supervised and unsupervised learning, where some data have their corresponding labels, but not all. Sometimes, labeling data samples can be time-consuming, so it might be more practical to label only some data and then use a semi-supervised learning algorithm such as self-training.

- **Reinforcement Learning:** Reinforcement learning, or Deep Reinforcement Learning (DRL), involves a dynamic interaction between an agent and its environment. The agent performs actions and receives rewards or penalties from the environment. The data are sequential, so the learning process repeats until the agent finds the optimal policy that maximizes the cumulative reward. This approach is generally useful for problems involving sequential decision-making and interaction with the environment. An example of a DRL algorithm is imitation learning, which was used to build the AlphaGo Zero model by David Silver and colleagues at Google DeepMind in 2016. AlphaGo Zero is a computer program that defeated the world champion in the complex board game Go.

- **Transfer learning**

  Transfer learning is a machine learning technique where a model trained for one task is reused for a related task . For example, a model capable of recognizing cats can be utilized to speed up the learning process for recognizing dogs. Transfer learning is particularly valuable in deep learning, where training models often requires extensive data and computational resources. By leveraging pre-trained models, one can apply the knowledge gained from solving a complex and large-scale problem to a new problem with less data.

  Transfer learning is most effective when the tasks are similar and the features are general enough to be relevant to both tasks. There are different approaches to transfer learning, including:

  Fine-tuning:
  The pre-trained model is used with its weights as initial weights for the new model, and training continues on the new data
  Domain adaptation:
  The model is adjusted to a new dataset of related examples, such as images of pedestrians under different lighting conditions. This is necessary when dealing with a domain shift problem, where the data distribution changes between the training and test sets
  Feature extraction:
  A pre-trained model can be used as a feature extractor. In this case, only new classification layers are retrained. This approach can speed up training when the tasks are different or when limited data is available to avoid overfitting Transfer learning can be applied to both supervised and unsupervised learning models . In recognition tasks like visual inspection of products, it is common to utilize a pre-trained model for classification on large

datasets . Some of the leading models include the VGG model from Oxford, Google's Inception model, and Microsoft's ResNet model. Another popular approach is to use models that have excelled in various detection tasks, such as those in the COCO dataset. Examples of such pre-trained models include Faster R-CNN developed by Microsoft Research, RetinaNet, and YOLO (You Only Look Once) [22].

2.11 Object detection : Algorithms and classification

Object detection algorithms are tools used in computer vision to identify and locate objects within images or videos. They enable machines to recognize multiple objects simultaneously, which is essential for tasks like autonomous driving, surveillance, and image analysis. These algorithms have evolved from methods like R-CNN, which processes regions of interest sequentially, to newer approaches such as YOLO and SSD, which optimize speed and accuracy by processing the entire image in a single pass through a neural network. Each algorithm offers unique advantages suited to different applications, driving advancements in object recognition technology.

Algorithms for object detection can be categorized into two main types: classification-based algorithms and regression-based algorithms.

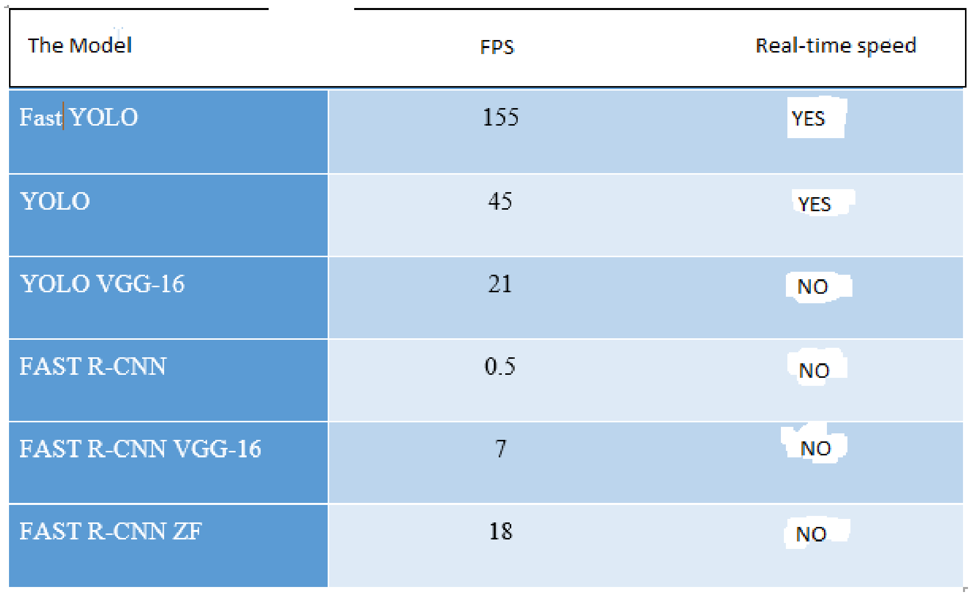
- **Classification-based algorithms**: These algorithms operate in two stages. Firstly, regions of interest (RoI) are identified within the image. Then, these regions undergo classification using a convolutional neural network (CNN). However, this sequential approach can be slow because it requires running prediction algorithms on each selected region individually. Examples of classification-based algorithms include RetinaNet, Region-based CNN (RCNN), Fast-RCNN, Faster R-CNN, and Mask-RCNN, which is considered state-of-the-art among regional-based CNN algorithms.

- **Regression-based algorithms**: In contrast, regression-based algorithms predict classes and bounding boxes for the entire image in a single pass through the model, eliminating the need for region selection. Since object detection is treated as a regression problem, these algorithms do not require complex pipelines. Notable examples of regression-based algorithms include the Single Shot Multibox Detector (SSD) and YOLO algorithms. Due to their ability to simultaneously detect objects and their high-speed nature (although with a tradeoff in accuracy), they are commonly used for real-time object detection. Understanding YOLO algorithms requires defining the objects to be predicted beforehand, with predictions resulting in bounding boxes specifying object locations and the most probable class.

- **YOU ONLY LOOK ONCE (YOLO) ALGORITHM**

  YOLO represents a novel approach to real-time object detection, capable of identifying multiple objects in an image while delineating bounding boxes around them. In the following sections, we will delve deeper into the workings and applications of YOLO.

Table 2.1: Comparison between R-CNN and YOLO



The Model	FPS	Real-time speed
Fast YOLO	155	YES
YOLO	45	YES
YOLO VGG-16	21	NO
FAST R-CNN	0.5	NO
FAST R-CNN VGG-16	7	NO
FAST R-CNN ZF	18	NO

2.12 You Only Look Once (YOLO)

The YOLO algorithm is a pioneering object detection method that has significantly advanced the field of computer vision. Unlike traditional object detection systems that involve multiple stages of processing, YOLO simplifies the task by predicting bounding boxes and class probabilities for objects within an image in a single pass. This innovative approach allows YOLO to achieve real-time performance with high accuracy, making it an essential tool for various applications, from autonomous driving and surveillance systems to industrial inspection and medical imaging. YOLO's efficiency and versatility have set a new standard in object detection technology, enabling rapid and precise identification of multiple objects in complex scenes.

2.12.1 YOLO Architecture

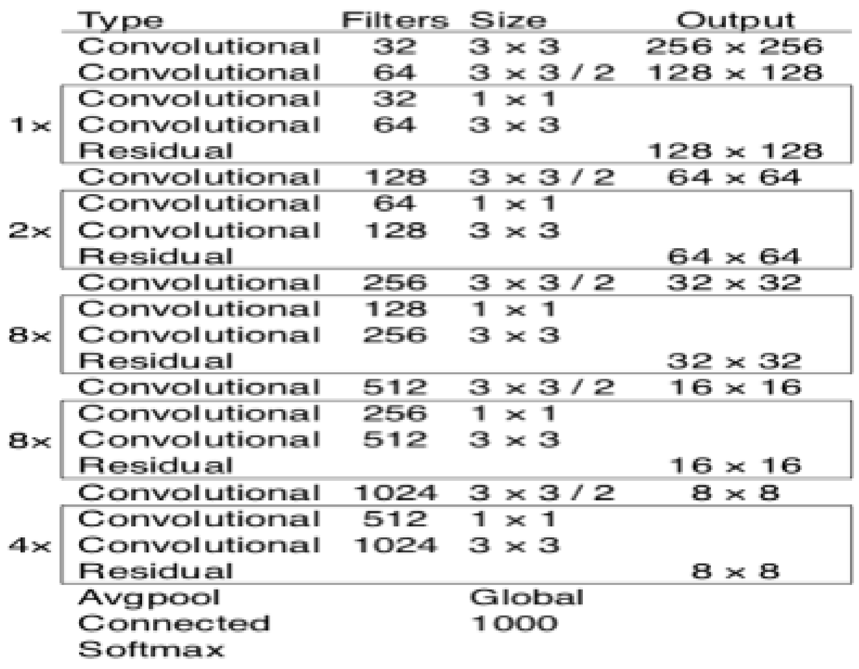
YOLO is built entirely on convolutional layers, distinguishing it as a fully convolutional network (FCN). In YOLO v3, a significant advancement was introduced with Darknet-53, a deep feature extractor consisting of 53 convolutional layers. Each layer is followed by batch normalization and leaky ReLU activation. Unlike traditional networks, YOLO avoids pooling layers to preserve low-level features and utilizes convolutional layers with a stride of 2 for downsampling feature maps.

YOLO maintains scale invariance with respect to input image size, although in practical applications, consistent input sizes are typically preferred due to implementation considerations.

Batch processing poses a challenge in ensuring all images within a batch have identical dimensions (height and width). This uniformity is crucial for leveraging GPU parallelism and optimizing processing speed.

The network achieves downsampling using a stride parameter, where the stride value determines how much the input image size is reduced at each layer. For example, a network with a stride of 32 would reduce an input image of size 416 x 416 to an output size of 13 x 13, where each value indicates the network's stride length relative to the input image size.

Table 2.2: Darknet-53 Model



	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

YOLO is scale-invariant regarding input image size. However, in practical applications, maintaining a consistent input size is often preferable due to implementation challenges.

One such challenge arises when processing images in batches, where all images within a batch must have fixed height and width dimensions. This standardization allows parallel processing by GPUs, significantly enhancing processing speed.

The network downsamples images using a parameter known as the network's stride. For instance, if the network's stride is 32, an input image of size 416 x 416 will produce an output size of 13 x 13. Typically, the stride length of a network layer equals the factor by which the layer's output dimensions are reduced compared to the input image size.

2.12.2 Interpretation of the prediction

The input is a batch of images in the shape (m, 416, 416, 3).

The output is a list of bounding boxes with recognized classes. Each bounding box is represented by 6 numbers (pc, bx, by, bh, bw, c). If we expand c into an 80-dimensional vector, each bounding box is then represented by 85 numbers.

As with all object detectors, features learned by convolutional layers are passed to a classifier/regressor that performs detection prediction (bounding box coordinates, class label, etc.).

In YOLO, prediction is made using a convolutional layer that employs 1x1 convolutions. Thus, the first thing to note is that our output is a feature map. Since we used 1x1 convolutions, the size of the prediction map is exactly the size of the feature map that precedes it. In YOLO v3, the way to interpret this prediction map is that each cell can predict a fixed number of bounding boxes.

For instance, if we have inputs (B x (5 + C)) in the feature map. B represents the number of bounding boxes each cell can predict. According to the paper, each of these bounding boxes B can specialize in detecting a certain type of object. Each bounding box has attributes 5 + C, describing center coordinates, dimensions, objectiveness score, and C class confidences for each bounding box. YOLO v3 predicts 3 bounding boxes for each cell.

We expect each cell in the feature map to predict an object through one of its bounding boxes if the object's center falls within the cell's receptive field.

This relates to how YOLO is trained, where a single bounding box is responsible for detecting a given object. Initially, we determine which cell this bounding box belongs to.

To do this, we divide the input image into a grid of dimensions equal to those of the final feature map. Consider the example below, where the input image is 416 x 416, and the network's stride length is 32. As mentioned earlier, the dimensions of the feature map will be 13 x 13. We then divide the input image into cells of 13 x 13.

Next, the cell (on the input image) containing the center of the ground truth bounding box of an object is chosen to be responsible for predicting the object. In the image, this is the cell marked in red, which contains the center of the ground truth bounding box (marked in yellow).

The red cell is the 7th cell in the 7th row of the grid. after that we assign the 7th cell of the 7th row of the feature map (corresponding cell in the feature map) as the one responsible for detecting the object.

This cell can predict three bounding boxes. One of these will be assigned the object label. To understand this, we need to delve into the concept of anchors. (Note that the cell we are discussing here is a cell in the prediction feature map. We divide the input image into a grid to determine which cell in the prediction feature map is responsible for the prediction).

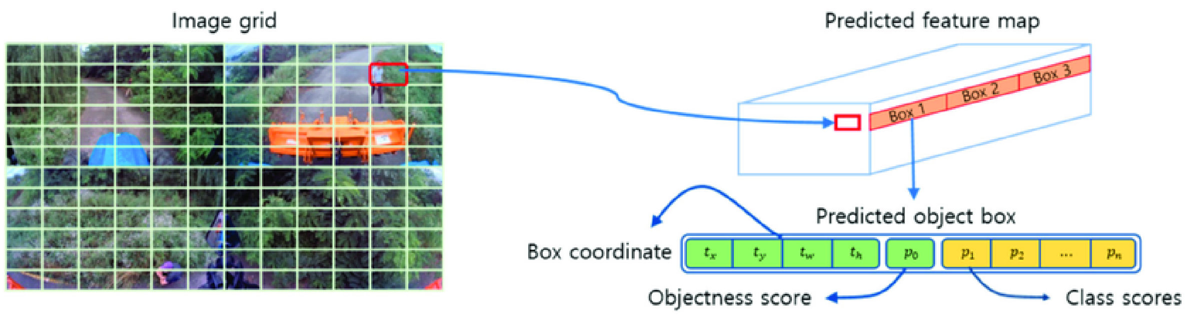


Figure 2.17: The attributes of bounding boxes

2.12.3 Anchor Boxes and Prediction

Predicting the width and height of bounding boxes directly can lead to unstable gradients during training. Instead, modern object detectors often predict log-space transformations or offsets to default predefined bounding boxes called anchors.

These transformations are then applied to the anchor boxes to make predictions. YOLO v3 utilizes three anchors, allowing it to predict three bounding boxes per grid cell.

Anchors are predefined bounding boxes calculated on datasets like COCO. The width and height of the box are predicted as offsets from the centroid of the cluster. The center coordinates of the box relative to the grid cell location are predicted using a sigmoid function.

The following formula describes how the network output is transformed to obtain bounding box predictions:

Image for post:

Here b_x, b_y, b_w, b_h are the center coordinates (x, y), width, and height of our prediction. t_x, t_y, t_w, t_h is what the network produces. c_x and c_y are the coordinates of the top-left corner of the grid. p_w and p_h are the anchor box dimensions.

Center Coordinates:

We use a sigmoid function to predict the center coordinates. Normally, YOLO doesn't predict the absolute coordinates of the bounding box center. Instead, it predicts offsets that are:

Relative to the top-left corner of the grid cell predicting the object. Normalized by the dimensions of the feature map cell, i.e., 1.

Take, for example, our rose image case above. If the predicted center coordinates are (0.4, 0.7), this means the center is at (6.4, 6.7) on the 13 x 13 feature map. (Since the top-left coordinates of the red square are (6, 6)).

But wait, what happens if the predicted *x* and *y* coordinates are greater than one, for example, (1.2, 0.7)? This means the center is at (7.2, 6.7). The center is in the cell just to the right of our red cell, or in the 8th cell of the 7th row. This breaks YOLO's theory because if we assume the red cell is responsible for predicting the rose, the rose's center must lie within the red cell, not the adjacent one. To address this issue, the output is passed through a sigmoid function, which squashes the output into a range of 0 to 1, effectively keeping the center within the predicted grid cell.

2.12.4 Processing Outputs (Filtering with Class Score Threshold)

For an image of size 416 x 416, YOLO predicts ((52 x 52) + (26 x 26) + 13 x 13)) x 3 = 10647 bounding boxes. However, in the case of our image, there is only one object, a rose. So, you might wonder how to reduce detections from 10647 to 1?

Firstly, we filter the boxes based on their confidence score. Generally, boxes with a score below a threshold (e.g., less than 0.5) are ignored. Next, Non-Maximum Suppression (NMS) addresses the issue of multiple detections of the same object in an image. For instance, the 3 neighboring boxes of the red grid might detect one box or adjacent boxes might detect the same object after eliminating multiple detections.

More precisely, we will perform these steps:

- Remove boxes with low scores (i.e., the box is not very confident about detecting a class).
- Select only one box when multiple boxes overlap and detect the same object (Non-Maximum Suppression).

Initially, we apply a thresholding filter. We want to eliminate any box for which the class ”score” is lower than a chosen threshold.

The model gives us a total of 19x19x5x85 numbers, with each box described by 85 numbers. It will be convenient to rearrange the tensor dimensional (19,19,5,85) (or (19,19,425)) in the following variables:

- `box_confidence`: tensor of shape (19×19,5,1) containing p_c (probability confidence that there is an object) for each of the 5 predicted boxes in each of the 19x19 cells.
- `boxes`: tensor of shape (19×19,5,4) containing (b_x, b_y, b_h, b_w) for each of the 5 boxes per cell.
- `box_class_probs`: tensor of shape (19×19,5,80) containing detection probabilities $(c_1, c_2, ..., c_80)$ for each of the 80 classes for each of the 5 boxes per cell.

Even after filtering by class score threshold, we end up with many overlapping boxes. A second filter to select the correct boxes is called NMS. NMS uses the very important function called ”Intersection over Union (IoU)”,

2.12.5 Implementation of the agreement protocol

To define a box, we use its two corners (top-left and bottom-right) coordinates: $(x1, y1, x2, y2)$ instead of the midpoint with height/width.

To calculate the area of a rectangle, multiply its height $(y2 - y1)$ by its width $(x2 - x1)$.

We also need to find the coordinates $(xi1, yi1, xi2, yi2)$ of the intersection of two boxes:

1. $xi1 = \max(x1$ coordinates of the two boxes$)$
2. $yi1 = \max(y1$ coordinates of the two boxes$)$
3. $xi2 = \min(x2$ coordinates of the two boxes$)$
4. $yi2 = \min(y2$ coordinates of the two boxes$)$

Arguments:

- `case1`: first box, defined by coordinates $(x1, y1, x2, y2)$

- `case2`: second box, defined by coordinates $(x1, y1, x2, y2)$

$$xi1 = \max(\texttt{case1}[0], \texttt{case2}[0])$$

$$yi1 = \max(\texttt{case1}[1], \texttt{case2}[1])$$

$$xi2 = \min(\texttt{case1}[2], \texttt{case2}[2])$$

$$yi2 = \min(\texttt{case1}[3], \texttt{case2}[3])$$

$$\text{inter\_area} = (xi2 - xi1) \times (yi2 - yi1)$$

To calculate the Union(A, B):

$$\text{box1\_area} = (\texttt{box1}[2] - \texttt{box1}[0]) \times (\texttt{box1}[3] - \texttt{box1}[1])$$

$$\text{box2\_area} = (\texttt{box2}[2] - \texttt{box2}[0]) \times (\texttt{box2}[3] - \texttt{box2}[1])$$

$$\text{union\_area} = \text{box1\_area} + \text{box2\_area} - \text{inter\_area}$$

IoU is calculated as:

$$IoU = \frac{\text{inter\_area}}{\text{union\_area}}$$

To implement non-maximum suppression, follow these key steps:

- Select the box with the highest score.
- Calculate its overlap with all other boxes, and remove boxes that overlap more than the `iou_threshold`.
- Return to step 1 and repeat until no boxes with a score lower than the currently selected box remain.

These steps ensure removing boxes that significantly overlap with selected boxes, retaining only the ”best” boxes.

2.13 Comparison between versions of YOLO

The latest iteration of YOLO is its third version, developed to enhance object detection accuracy compared to previous versions.

- YOLO V1:

  The first version, introduced in 2015, trained on the ImageNet-1000 dataset. YOLO V1 struggled with identifying small grouped objects and was ineffective at generalizing objects of varying sizes within images, resulting in inaccurate object localization.

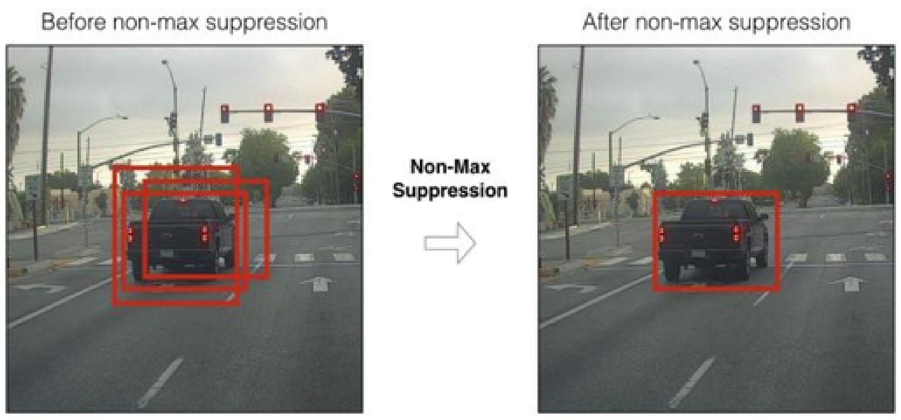


Figure 2.18: Non-maximum suppression filter

- YOLO V2:

  Released in 2016 as YOLO9000, YOLO V2 featured a 19-layer network with an additional 11 layers dedicated to object detection. It aimed to support Faster R-CNN and Single Shot MultiBox Detector (SSD), which achieved better object detection scores.

- YOLO V3:

  YOLO V3 represents a progressive upgrade over YOLO V2. This architecture consists of 53 layers trained on ImageNet and another 53 layers dedicated to object detection, totaling 106 layers. While significantly improving network accuracy, it reduced processing speed from 45 to 30 frames per second.

- YOLO v4 :
  YOLO v4, the latest version released in 2020, incorporates advancements to further enhance object detection performance. It introduces a redesigned backbone network, enhanced feature pyramid, and improved training techniques such as mosaic data augmentation and multi-scale training. YOLOv4 aims to achieve state-of-the-art results in terms of both accuracy and speed in object detection tasks.

YOLOv5 vs YOLOv4:

There is no research paper published for YOLOv5 from which we can drive the advantages and disadvantages of the model. However,some AI practitioners have tested the performance of YOLOv5 on many benchmarks, bellow is the summary of the benchmarks made by roboflow after a discussion with the author of the model.

YOLOv5 Architecture

YOLOv5 was released with five different sizes:

- n for extra small (nano) size model.

- s for small size model.
- m for medium size model.
- l for large size model
- x for extra large size model

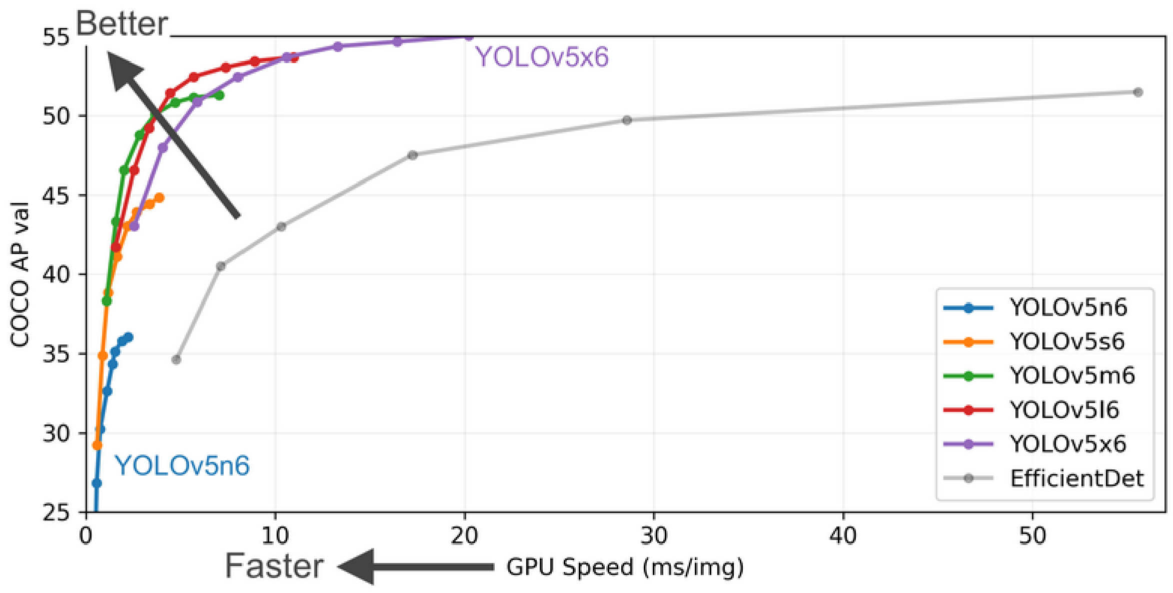


Figure 2.19: Performance of YOLOv5 different sizes models

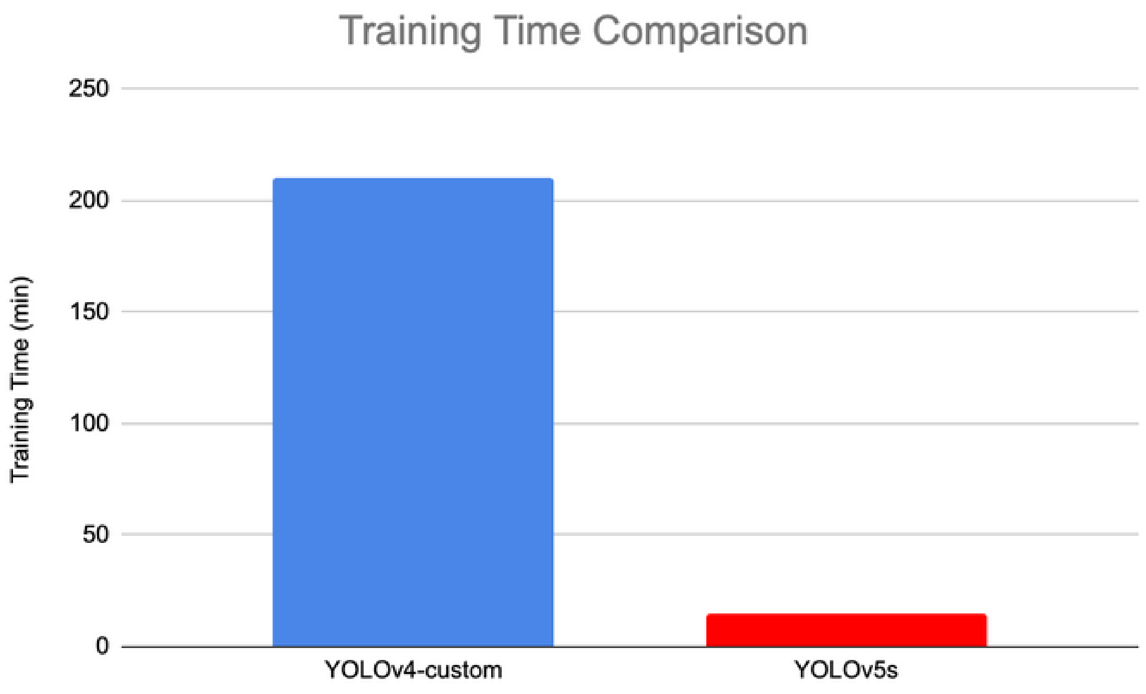


Figure 2.20: YOLOv5 is faster to train than YOLOv4

2.14 Conclusion

In this chapter, we explored the development and application of visual inspection systems for PCBs. These systems are integral to maintaining the quality and functionality of PCBs in modern electronics. We outlined the three main stages of a typical VIS: detecting defects at the track level, verifying the presence and proper placement of components, and assessing the quality of solder joints.

Each stage of the inspection process involves sophisticated modules that work in tandem to capture, preprocess, analyze, and evaluate PCB images. The integration of advanced techniques such as deep learning and neural networks has significantly enhanced the accuracy and efficiency of these inspections. Deep learning, in particular, offers the capability to automatically extract intricate features from large datasets of PCB images, identify subtle defects, and adapt to various scenarios and defect types.

The transition from traditional manual inspection methods to automated deep learning-based systems has brought about numerous benefits, including faster inspection times, increased production throughput, reduced labor costs, and continuous improvement in defect detection accuracy. As we continue to advance in this field, the potential for further enhancements in PCB inspection technology remains substantial, promising even greater precision and reliability in the quality control of electronic components.

Chapter 3

PCBs VI based on advanced image processing and YOLO training

3.1 Introduction

In this chapter, we explore the application of advanced image processing and AI techniques for the automated visual inspection of PCBs, aiming to create a robust system for detecting defects at the track level , verifying component placement, and assessing solder joint quality. The chapter begins with an overview of the system architecture, outlining the interconnections between various components and modules.

It then describes the image acquisition setup, crucial for capturing high-quality images suitable for inspection, followed by the image preprocessing techniques used to enhance these images for further analysis. The core of the chapter focuses on implementing the YOLO algorithm for defect detection at the track level, detailing the training process, dataset preparation, and model performance. Additionally, it covers methods for component localization to ensure correct placement according to design specifications.

The YOLOv5 algorithm's application for solder quality inspection is also discussed, explaining how it identifies and classifies solder joints as acceptable or defective. The integration of these modules into a cohesive system is then examined, emphasizing seamless data flow and interaction between different inspection stages.

Finally, the chapter presents the inspection system's results, including performance metrics such as accuracy, precision, recall, and F1-score, and provides examples of detected defects to illustrate the system's effectiveness.

3.2 System Overview

Our VIS consists of three stages, the first stage detects defects on the track level, the second stage detects the presence and proper placement of components and the last stage checks the solder quality. Each stage is composed of different modules that work together to capture, preprocess, analyze, and assess images of the PCBs.

at the end of the chapter we present the GUI of the three stages.

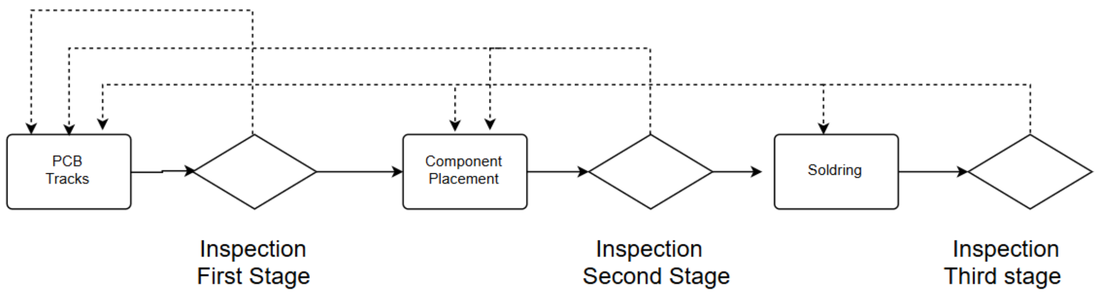


Figure 3.1: High-Level system architecture

3.3 The common component across the three stages

The proposed VIS employs the same initial two components across both the first and second stages:

- Image Acquisition and Preprocessing: This component involves capturing and preparing images for further analysis.
- Mode Selection and Test Image Registration: Here, the system selects operational modes and aligns the test image for registration.

Both of these components heavily depend on the SIFT algorithm for tasks such as image registration, template matching, and image comparison.

In this section, we will delve into these concepts. Following that, we will explain the first two components of both stages:

The system utilizes the SIFT algorithm to achieve three primary objectives:

- Image Registration: Aligning two images for precise comparison.
- Template Matching: Identifying specific patterns or templates within images.
- Image comparison : Comparing and identifying matches within a set of images.

3.3.1 Image registration

Image registration is crucial for aligning the test image with the reference image, enabling accurate comparison and defect detection. This process corrects any misalignment caused by positioning differences during image capture, ensuring precise overlay of the test and the reference image.

Implementation :

For our PCB inspection system, we utilized feature-based and transform-based registration methods to achieve high precision and robustness. The steps involved are:

1. Keypoint Detection: Detect key points in both the test and reference images using the SIFT algorithm.
2. Feature Matching : Match the detected key points using a descriptor matcher .
3. Transformation Estimation: Estimate the transformation matrix using the matched key points.

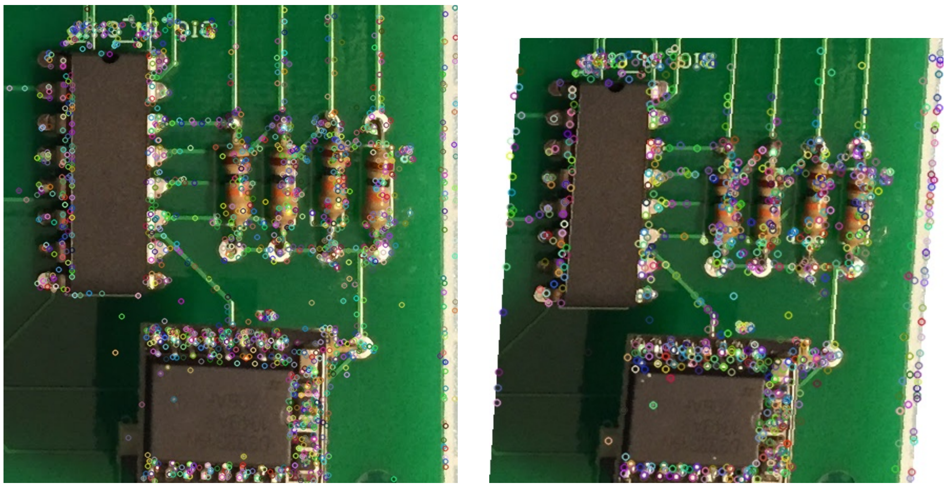


Figure 3.2: Keypoints extraction from the Test and the reference images

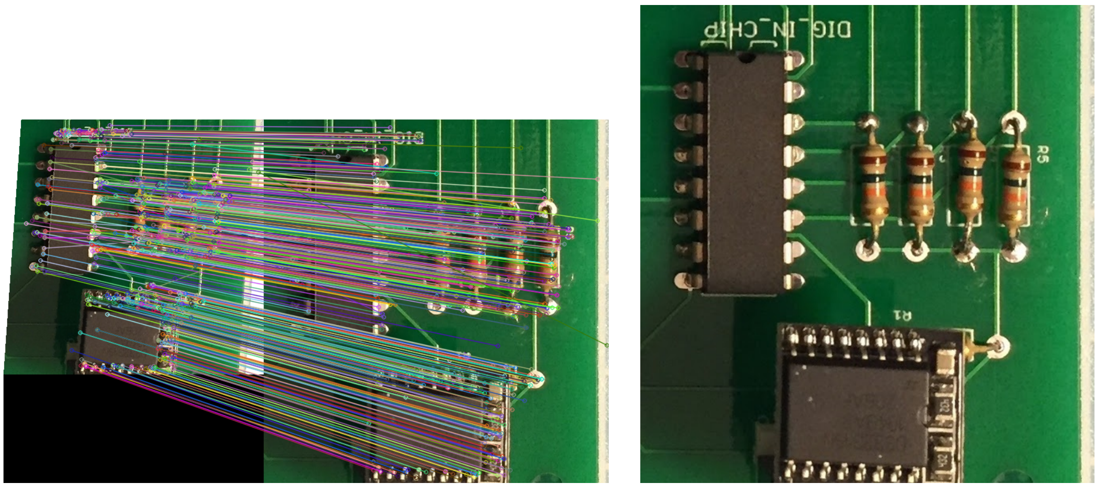


Figure 3.3: keypoints matching and test images registeration

3.3.2 Template Matching

After calculating the transformation matrix H using matched keypoints between the template and reference images, the next step involves applying this matrix to align the images accurately.To find the template in the reference image, we use the transformation matrix H to calculate the transformed coordinates of the four corners of the template . By doing so, we can accurately locate the template within the reference image.

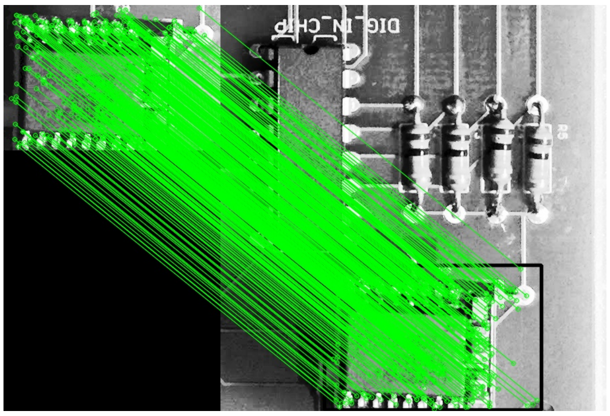


Figure 3.4: SIFT template matching

3.3.3 Image comparison

Finding a lot of good keypoint between two images using the SIFT algorithm indicates that the images share significant visual similarities.In the context of image comparison it means that they represent the same image.

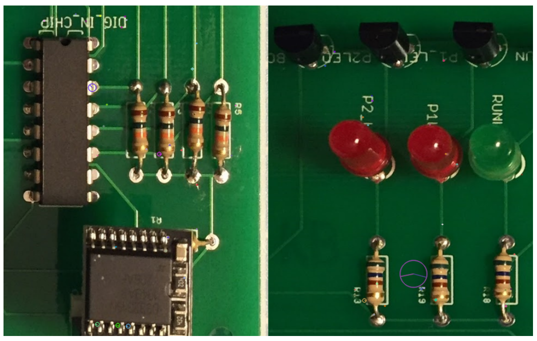


Figure 3.5: Comparing two different images - almost no BKPN -

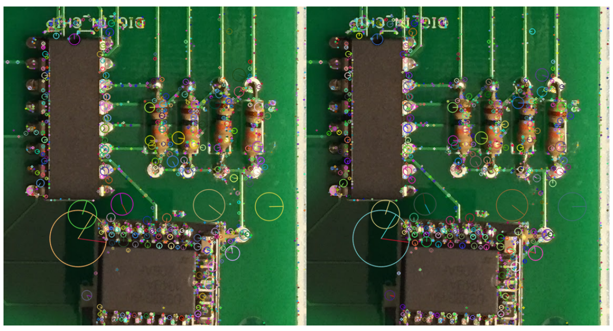


Figure 3.6: Comparing the same image - a lot of BKPN -

In the previous examples, we demonstrated how SIFT can be used for aligning two pictures, performing template matching, and identifying the match of an image within a group of images. To accomplish this task, we need to calculate the transformation matrix H and the number of good points. The Figure 3.7 diagram explains the whole process.

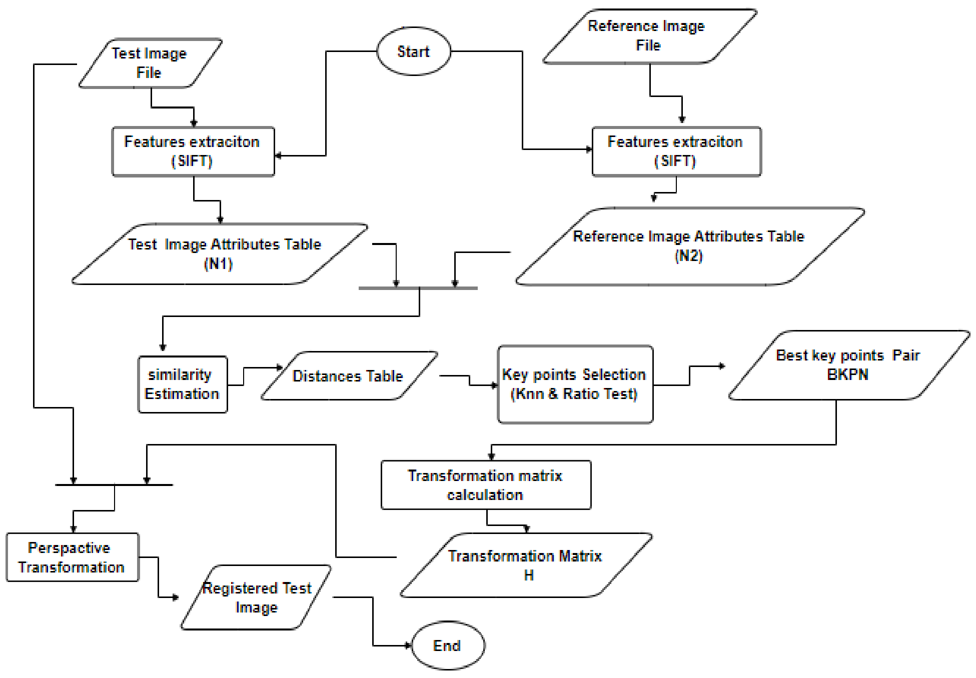

Start
Test Image File
Reference Image File
Features extraciton (SIFT)
Features extraciton (SIFT)
Test Image Attributes Table (N1)
Reference Image Attributes Table (N2)
similarity Estimation
Distances Table
Key points Selection (Knn & Ratio Test)
Best key points Pair BKPN
Transformation matrix calculation
Transformation Matrix H
Perspactive Transformation
Registered Test Image
End

Figure 3.7: SIFT workflow

The process begins with the selection of a Test image File and a Reference Image File.

First, SIFT is applied to both the test image and the reference image to extract features, resulting in two tables: the Test Image Attributes Table and the Reference Image Attributes Table. These tables contain the detected keypoints and their corresponding descriptors for each image. Next, a similarity estimation is performed by comparing the descriptors from the test image with those from the reference image, creating a Distances Table that quantifies the similarity between each pair of keypoints.

To refine these comparisons, the K-Nearest Neighbors (KNN) algorithm is used to find the closest keypoints for each point in the test image. Following this, a ratio test is applied to select the best matches. The ratio test involves comparing the distance of the closest match to the distance of the second closest match and retaining matches that meet a predefined threshold, usually 0.75. This process yields a set of Best keypoint Numver Pairsb(BKPN) that are considered reliable matches which we going to use in the image matching .

These BKPN are then used for image matching, which involves determining the degree of similarity and alignment between the test image and the reference image. Subsequently, a transformation matrix H is calculated. This matrix captures the geometric transformation needed to align the test image with the reference image based on the matched keypoints.

After explaining the SIFT workflow and its applications, we will now delve into the initial two components of the first and second stages: image acquisition and preprocessing, and model selection and test image registration.

3.4 Image Acquisition and preprocessing

The image acquisition setup consists of high-resolution CMOS cameras and LED lighting configured to capture detailed images of PCBs. The cameras are positioned to provide a comprehensive view of the PCB surface, ensuring that even the smallest defects are visible. Preprocessing techniques such as Gaussian filtering for noise reduction, histogram equalization for contrast enhancement, and normalization are applied to the raw images. These steps enhance important features and reduce noise, preparing the images for further analysis.

3.5 Mode selection and Test image registration

Our system operates in two modes, The first mode allows a worker to manually set a reference template, which is then used to compare all subsequent PCBs to identify defects. In the second, automatic mode, the system autonomously sets the reference image by comparing the test image against a dataset of images. This comparison helps in determining the reference image. The autonomous reference image selection is based on the SIFT algorithm introduced in the previous section. The following three diagrams explain the entire process of setting the reference images for defect detection. This concept is used in both the first and second stages.

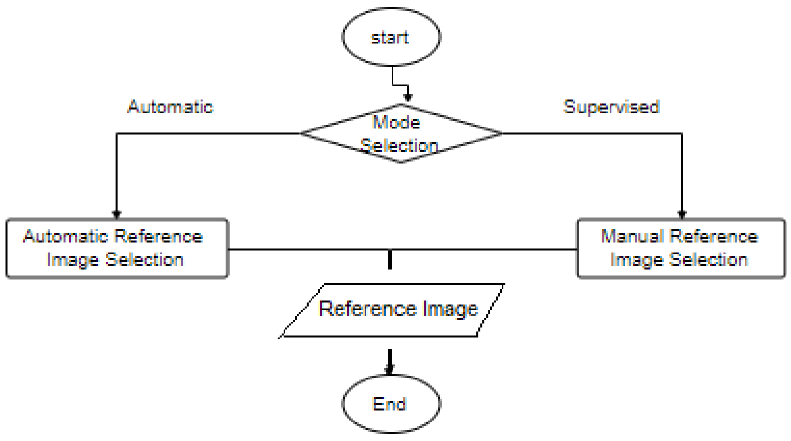

start
Automatic
Mode Selection
Supervised
Automatic Reference Image Selection
Manual Reference Image Selection
Reference Image
End

Figure 3.8: Mode selection

Figure 3.8 shows the mode selection diagram, which takes the mode as an input and then chooses between an automatic mode or a manual mode, both of which are explained in the next two sections.

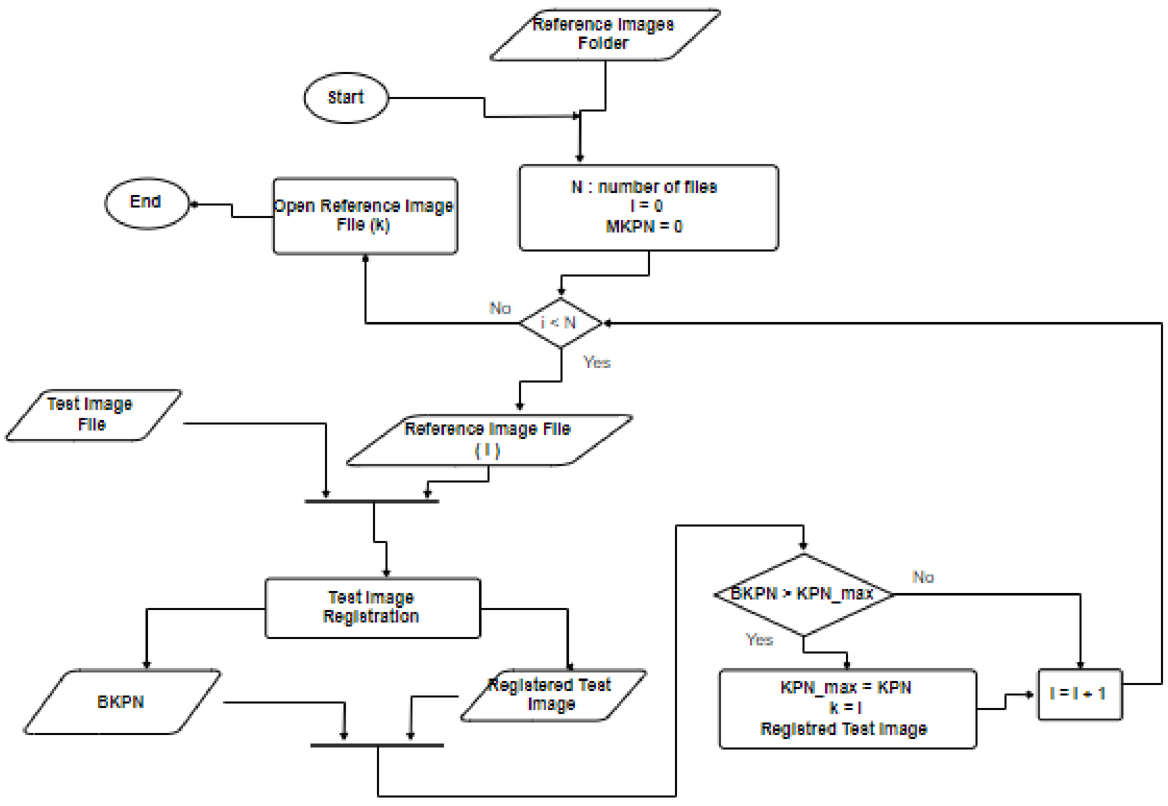


Figure 3.9: Automatic reference image selection

3.5.1 Automatic mode

The process begins with a Test Image File representing the potentially defective PCB and a folder of reference images of known good PCBs. To automatically select the most suitable reference image for the test image, the system iterates through each Reference Image in the folder. For each reference image, features and keypoints are extracted from both the test image and the current reference image. The number of best keypoints pairs (BKPN) between them is calculated.

The system initializes variables for storing the maximum number of BKPN found, referred to as *KPN_max* , and the Registered Test Image. As the loop progresses, if the number of BKPN between the Test Image and the current Reference Image exceeds the current maximum *KPN_max*, the system updates *KPN_max*to the new maximum value. It also updates the registered Test Image to correspond to this reference image, and the variable k takes the value of i.

After looping through all reference images, the system returns K and Reference image (k), representing the most suitable reference image for the test image based on the highest number of keypoints found. This process enables efficient selection of reference images for accurate defect detection and quality control in PCB manufacturing processes.

3.5.2 Manual mode

In manual mode, the process begins with a worker selecting a reference image from a folder containing multiple reference images along with a test image file representing the potentially defective PCB. The chosen reference image is then subjected to feature extraction using the SIFT algorithm, which is also applied to the test image. Subsequently, the number of keypoints extracted from both images is calculated. This count is compared against a predefined minimal valueKPN_min set as a threshold.

If the number of keypoints in the test image exceeds KPN_min, indicating a sufficient level of detail for comparison, the selected reference image is accepted. However, if the number of keypoints falls belowKPN_min, implying insufficient visual information for accurate comparison, an error message is displayed, indicating that the selected reference image is not suitable. The worker receives feedback regarding the acceptance or rejection of the reference image and can choose another reference image from the folder for comparison if necessary.

This iterative process continues until a suitable reference image is accepted, ensuring that only reference images with a satisfactory level of detail are considered for further analysis.

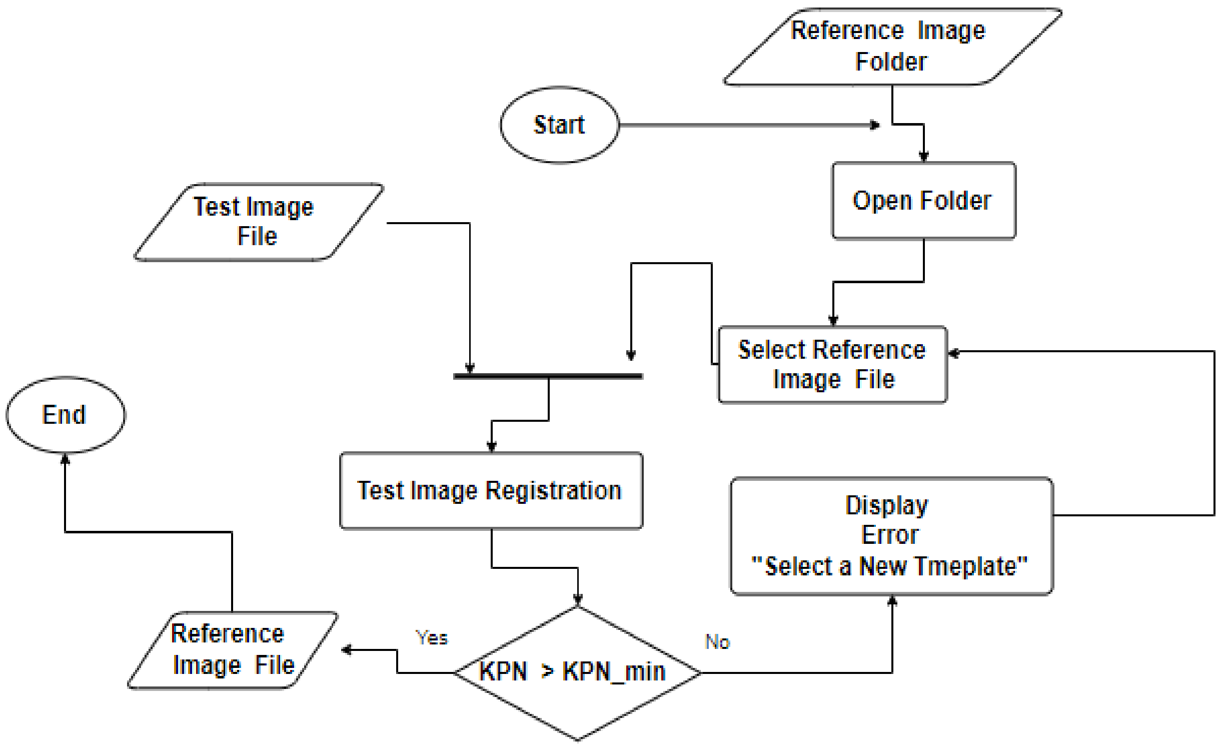


Figure 3.10: Manual reference image selection

3.6 The First Stage: Track defects

The first stage of our VIS focuses on detecting common PCB tracks defects such as shorts, opens, mouse-bites, pinholes, and spurs. This stage combines advanced image processing techniques and deep learning algorithms to ensure high accuracy and reliability in defect detection. The primary methods employed include the YOLOv5 algorithm for real-time object detection and various image processing algorithms, such as absolute difference, to validate and enhance defect identification.

The following diagram shows the working mechanism of the first stage, where image acquisition, mode selection, and test image registration are already mentioned in the section above. In this section, we will discuss defect detection using YOLOv5 and how to enhance accuracy using image processing algorithms such as absolute difference.

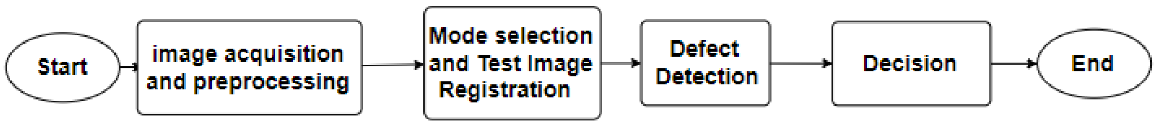

Start
image acquisition and preprocessing
Mode selection and Test Image Registration
Defect Detection
Decision
End

Figure 3.11: First stage global diagram

Defect detection at Track level

The process begins with a reference image and a registered test image , which is an image of a potentially defective PCB that has been aligned with a reference image. Using the YOLO algorithm, the system estimates probable defects and it information .

For each defect, the system defines a Region of Interest (ROI) in the test image. The ROI focuses on the area around the detected defect. The system calculates the absolute difference between the registred test image and the corresponding reference image.

Using the absolute difference calculation results, the system validates each defect. It determines whether the detected defect is real by checking the difference against a threshold. Validation metrics are calculated to assess whether a defect is significant. The validation metric (VM) is compared against a threshold value (S). If the metric exceeds the threshold, the defect is considered validated.

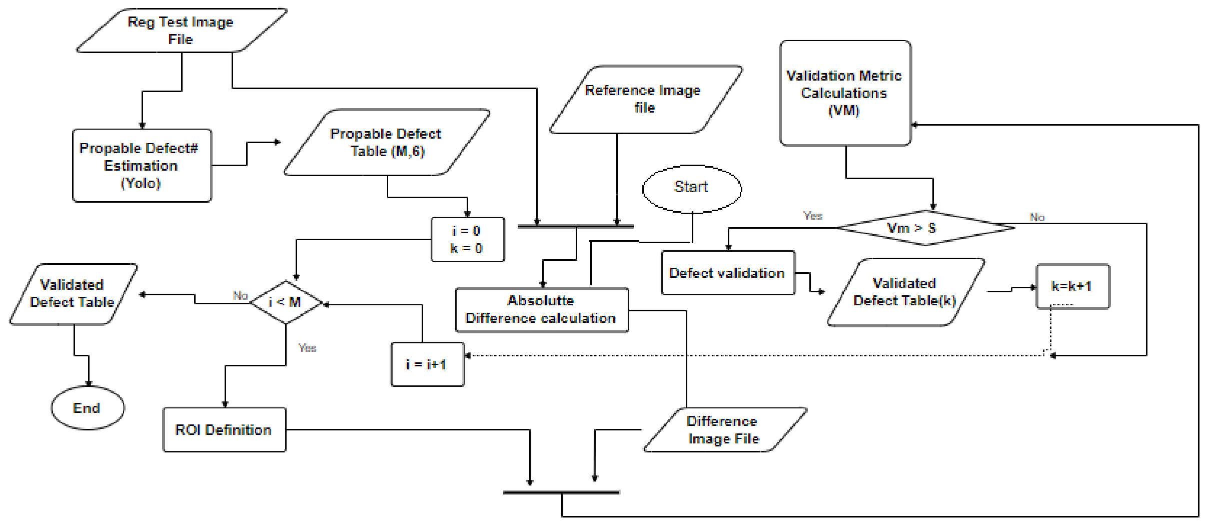
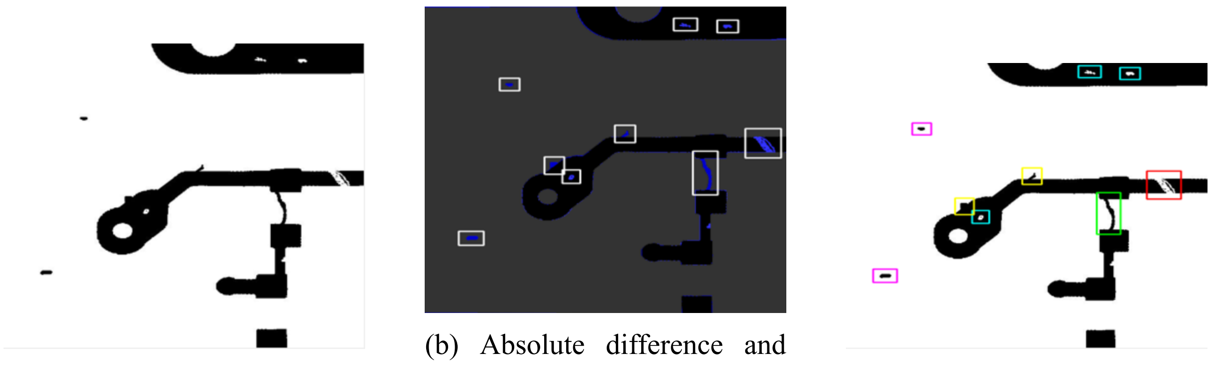


Figure 3.12: Defect detection



(a) Test PCB Image (b) Absolute difference and probable defect (c) Validated defects

Figure 3.13: Comparison of test PCB image, absolute difference, and validated defects

3.6.1 YOLO V5 model Training and Dataset preparation

Model training

The Deep PCB dataset comprises 1,500 pairs of images, each pair including a defect-free template image and a corresponding annotated test image. These annotations mark the locations of the six most common types of PCB defects: mouse bite, open, short, pinhole, spur, and spurious copper.

For the annotated test images, each defect is represented with a bounding box aligned along the axis, and each class of defect is assigned a unique ID . The images in the dataset are captured using a CCD linear scan, achieving a resolution of approximately 48 pixels per millimeter. The defect-free template images are manually checked and cleaned to ensure accuracy.

The original dimensions of the template and test images are approximately 16,000 x 16,000 pixels. To facilitate processing, these large images are divided into sub-images of 640 x 640 pixels. These sub-images are then aligned with the template using image matching methods. Illumination disturbances are mitigated through binarization, with carefully selected thresholds to enhance consistency.

Dataset Transformation for YOLOv5 Training

To train the YOLOv5 model for defect detection in PCBs, we transformed a COCO (Common Objects in Context) dataset to be compatible with YOLOv5 requirements. The COCO dataset format, widely used for object detection tasks, requires conversion to YOLO format to ensure the model can effectively learn and detect the specific defects in PCBs.

COCO Dataset Format

The COCO dataset format is a standard in the computer vision community, featuring images annotated with object instances, including bounding boxes, segmentation masks, and class labels. The annotations are stored in JSON files, detailing each object's position and category.

- **Images:** The COCO dataset contains images of various sizes, each assigned a unique ID.
- **Annotations:** Each image is annotated with objects, represented by bounding boxes and associated class labels.
- **Categories:** A list of object categories, each with a unique ID and name.

Conversion to YOLOv5 Format

The transformation process involves converting the COCO annotations to YOLOv5's required format. YOLOv5 expects annotations in a text file where each line represents an object in the image using the following format: `class_id center_x center_y width height`.

- **Class ID:** The ID of the object class (defect type).
- **Center_x, Center_y:** The normalized coordinates of the bounding box center (values between 0 and 1).
- **Width, Height:** The normalized width and height of the bounding box (values between 0 and 1).

Steps for Conversion:

1. **Parse COCO JSON Annotations:**
    - Extract image IDs, file names, and annotations (bounding boxes and class labels) from the COCO JSON files.
2. **Normalize Bounding Boxes:**

- Convert COCO bounding box format (`x_min, y_min, width, height`) to YOLO format (`center_x, center_y, width, height`).
- Normalize the coordinates by dividing by the image dimensions.

3. **Create YOLO Annotation Files:**
   - For each image, create a corresponding text file with normalized bounding box coordinates and class IDs.

Example of COCO to YOLOv5 Format Transformation

An annotation in the COCO format typically includes bounding box coordinates and a class ID. For example:

```
{
    "image_id": 1,
    "category_id": 3,
    "bbox": [100, 200, 50, 100]
}
```

In the YOLOv5 format, the bounding box is represented with the class ID and normalized coordinates. Given an image of dimensions 1024x1024, the COCO annotation is transformed as follows:

- **Class ID:** 3
- **Center_x:**

$$\text{center\_x} = \frac{x\_min + \frac{width}{2}}{width\_image} = \frac{100 + \frac{50}{2}}{1024} = \frac{100 + 25}{1024} = \frac{125}{1024} \approx 0.1221$$

- **Center_y:**

$$\text{center\_y} = \frac{y\_min + \frac{height}{2}}{height\_image} = \frac{200 + \frac{100}{2}}{1024} = \frac{200 + 50}{1024} = \frac{250}{1024} \approx 0.2441$$

- **Width:**

$$\text{width} = \frac{width}{width\_image} = \frac{50}{1024} \approx 0.0488$$

- **Height:**

$$\text{height} = \frac{height}{height\_image} = \frac{100}{1024} \approx 0.0977$$

Thus, the corresponding YOLOv5 annotation line is:

```
3 0.1221 0.2441 0.0488 0.0977
```

3.6.2 Training

Example of Python Code for SIFT Feature Detection

```
RES_DIR = set_res_dir()
if TRAIN:
    python train.py --data ../data.yaml --weights yolov5s.pt \
    --img 640 --epochs {EPOCHS} --batch-size 16 --name {RES_DIR}
```

- ’–img’ 640 sets the input image size for training. The input images will be resized to a square shape with a width and height of 640 pixels before being fed into the model for training.
- ’–epochs’ specifies the number of training epochs, which in this code is equal to 25.
- ’–batch-size 16’ This argument sets the batch size for training to 16. The batch size determines the number of samples processed simultaneously by the model during each iteration of training
- ’–name {RES_DIR}’ This argument specifies a name for the training run, where {RES_DIR} is a placeholder for a variable representing the name of the directory where the training results will be saved.

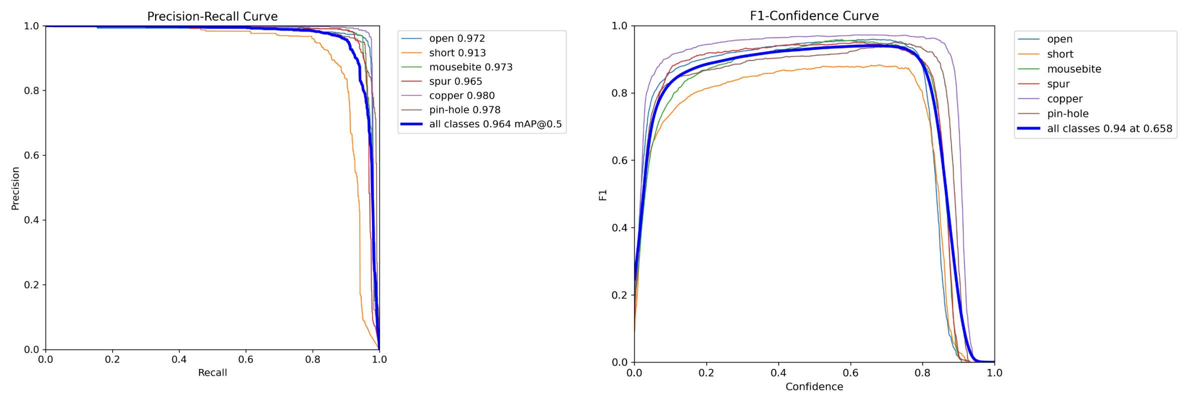


Figure 3.14: Precision-recall and precision curve

The precision-recall curve illustrates the trade-off between precision and recall at various classification thresholds. Precision measures the proportion of true positive detections among all positive predictions, while recall measures the proportion of true positive detections among all actual positive instances in the dataset. A higher precision indicates fewer false positives, while a higher recall indicates fewer false negatives. The precision-recall curve provides insight into how well the model performs across different thresholds and can help determine the optimal threshold for a specific task.

The F1 confidence curve combines precision and recall into a single metric, known as the F1 score, which is the harmonic mean of precision and recall. The F1 score provides a balanced measure of the model's performance, considering both precision and recall. In our case, it's noteworthy that in the confidence curve, it displays for all classes a confidence level of 0.94 at a threshold of 0.658. The F1 confidence curve plots the F1 score against different confidence thresholds, showing how the model's performance changes as the threshold for classifying detections varies. A higher F1 score indicates better overall performance of the model.

3.6.3 First stage overall performance

In testing my VIS, I selected 20 images to serve as test cases and manually introduced defects into each one. This process aimed to simulate real-world scenarios . After introducing defects, I conducted a visual inspection of each image to assess how well the system identified these defects.

I documented my findings in an Excel file, recording details such as image identifiers, descriptions of defects, and the system's performance in detecting them.

This structured approach not only evaluated the system's accuracy in detecting visually perceptible defects but also served as a benchmark for future improvements. By systematically documenting and analyzing the results, I gained insights into the system's strengths and areas for enhancement, ensuring its reliability and effectiveness in practical applications of visual inspection.

the results are calculated using the data table in Table 3.1 and the equations 3.1, 3.2, 3.3, 3.4, and 3.5. The individual sample results are shown in the graph of Figure 3.16, and the overall performance is illustrated in the radar chart in Figure 3.15.

Confusion Matrix Components

These metrics are derived from the confusion matrix, which consists of:

- **TP (True Positives)**: Instances correctly predicted as positive.
- **TN (True Negatives)**: Instances correctly predicted as negative.
- **FP (False Positives)**: Instances incorrectly predicted as positive (actually negative).
- **FN (False Negatives)**: Instances incorrectly predicted as negative (actually positive).

Performance Metrics for AI Models

1. Accuracy

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} = \frac{TP+TN}{TP+TN+FP+FN} \tag{3.1}$$

Description: Measures the proportion of correct predictions out of the total predictions made.

2. Precision

$$\text{Precision} = \frac{TP}{TP+FP} \tag{3.2}$$

Description: Indicates how many of the positively predicted cases (TP + FP) are actually true positives (TP).

3. Recall (Sensitivity or True Positive Rate)

$$\text{Recall} = \frac{TP}{TP+FN} \tag{3.3}$$

Description: Measures the proportion of actual positives (TP + FN) that are correctly identified as such (TP).

4. Specificity (True Negative Rate)

$$\text{Specificity} = \frac{TN}{TN+FP} \tag{3.4}$$

Description: Measures the proportion of actual negatives (TN + FP) that are correctly identified as such (TN).

5. F1 Score

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2 \times TP}{2 \times TP+FP+FN} \tag{3.5}$$

Description: The harmonic mean of precision and recall, providing a single metric that balances between the two.

These metrics provide insights into different aspects of model performance: accuracy gives an overall view, precision and recall focus on the positive class, specificity on the negative class, and F1 score balances precision and recall.

The Table 3.1 show the TP , TN , FP , FN values of the twenty images we tested.

Table 3.1: Performance summary across twenty images

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
TP	5	6	4	7	8	7	6	9	9	10	12	11	13	6	10	3	3	5	6	6
TN	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
FP	0	1	0	0	0	0	0	0	1	0	0	1	0	1	0	1	0	2	0	0
FN	2	0	0	0	0	0	4	1	0	3	2	1	1	0	0	1	1	0	0	1

After performing the calculations using the equations and the data from Table 3.1, we obtained the results for each sample, which are displayed in Graph 3.15

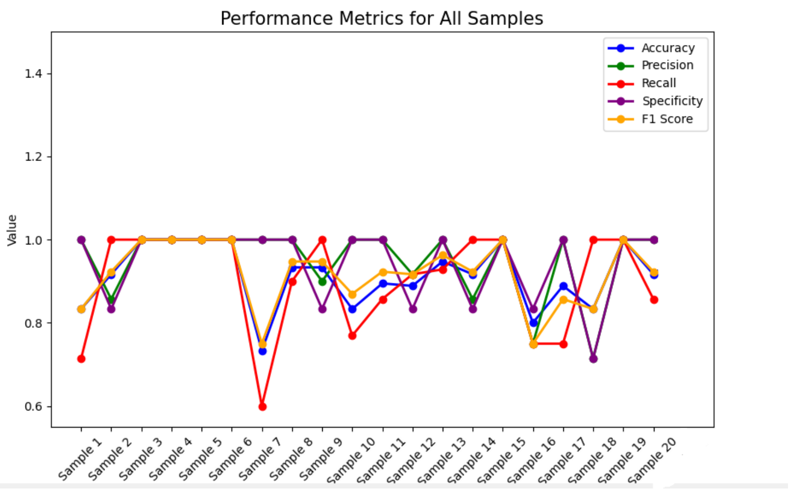


Figure 3.15: Individual performance evaluation

Using the performance metrics of each sample, we concluded the overall performance of the first stage shown in the radar chart Figure 3.16 .

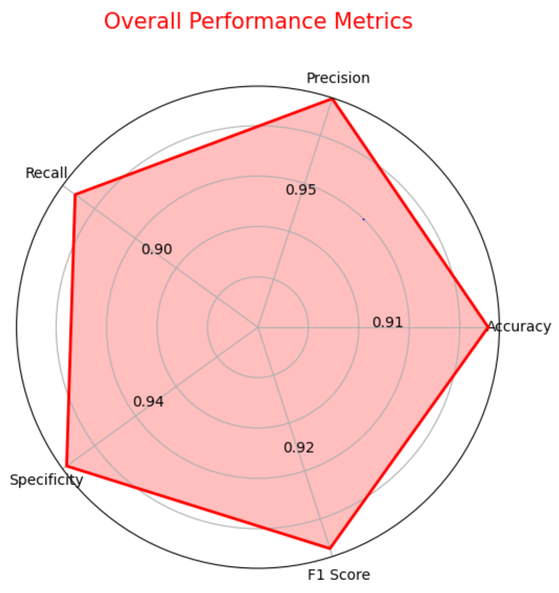


Figure 3.16: First stage overall performance

The model demonstrates a high level of performance in PCB defect detection, with an accuracy of 91.27%, a precision of 95.42%, and a recall of 89.57%. The F1 score, which balances precision and recall, is also high at 92.21%. These metrics indicate that the model is effective in correctly identifying defective PCBs while maintaining a low rate of false positives and false negatives. However, the model's performance might vary depending on the specific application and the relative importance of precision versus recall in that context.

3.7 Second Stage : Individual performance evaluation

In the second stage of the visual inspection process, the focus shifts to verifying the presence and proper placement of components on the PCBs. This stage plays a critical role in ensuring the integrity and functionality of the PCB assembly, as any deviations from the specified component positions can lead to operational failures or performance issues in the final product.

The following diagram illustrates the operational process of the second stage, building upon the previously mentioned steps of image acquisition and preprocessing, mode selection and test image registration, as outlined in the preceding section. Within this stage, we delve into the verification of component presence and correct placement employing template matching techniques. Specifically, we focus on utilizing SIFT for template matching, as well as employing NCC (Normalized Cross-Correlation) for template matching.

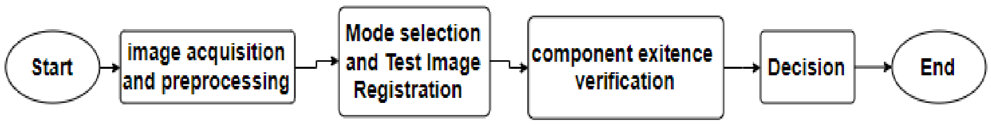

Start
image acquisition and preprocessing
Mode selection and Test Image Registration
component exitence verification
Decision
End

Figure 3.17: Second stage diagram

3.7.1 SIFT template matching

As shown in Figure 3.16, the template matching process begins with a reference template image and a corresponding test image. These images undergo SIFT analysis to compute the number of keypoints (BKPN) and the transformation matrix (H) required for alignment.

Once we have the transformation matrix from SIFT, we need to find out where exactly the template ends up in the test image. To do this, we take the corners of the template in the template reference image and use the transformation matrix to see where they move to in the test image. This transformation considers movements, rotations, and resizes the template to match it up with the test image.

After the transformation, we get the new positions of these corners, which tell us where the template is located within the test image. This helps us compare the template with the test image accurately to check if all components are in place.

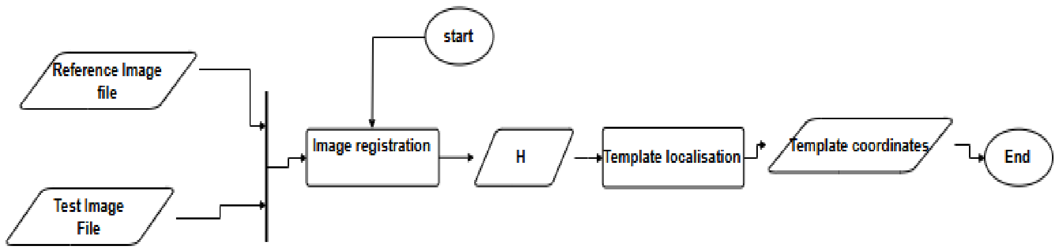


Figure 3.18: SIFT template matching doagram

3.7.2 NCC template matching

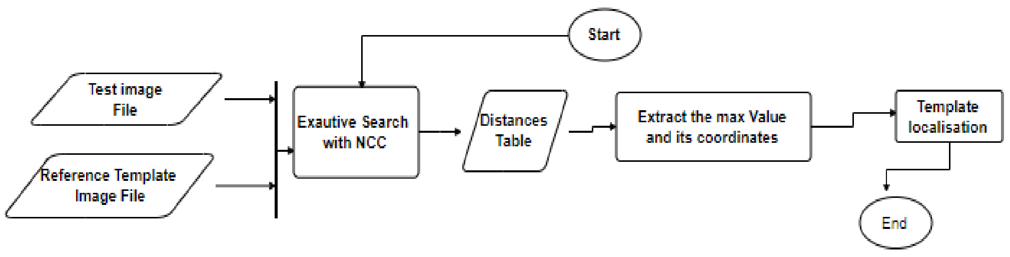


Figure 3.19: NCC template matching diagram

In this step, we begin with a test image and a reference template image, aiming to perform template matching using the NCC technique. The process involves an exhaustive search with NCC across the test image to find the best match for the reference template. The result of this search is represented by a Distances Table, which contains correlation values for each location in the test image.

From this table, we extract the maximum correlation value along with its corresponding coordinates, indicating the location of the template within the test image. Additionally, the maximum correlation value serves as a measure of confidence in the accuracy of the template matching process.

3.7.3 Component existence verification

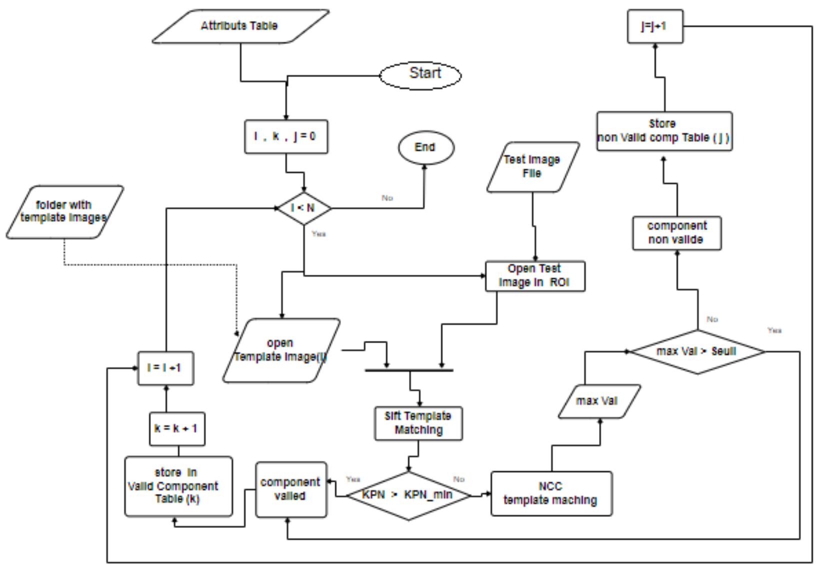


Figure 3.20: Component existence verification

The process begins with a test image file and a folder containing reference template images, along with a table detailing each component's template path and its corresponding coordinates within the test image. Iterating through the table, each line is assigned a variable 'i', prompting the selection of the region of interest (ROI) from the test image based on the specified coordinates.

Using the SIFT algorithm, features and keypoints are extracted from both the reference template image and the ROI. The number of keypoints (BKPN) is compared against a predefined minimum value *KPN_min*. Valid components, with BKPN surpassing *KPN_min*, are logged in a valid component table using an incrementing variable 'k'.

For components with BKPN below *KPN_min*, further validation is performed using the NCC technique. If the NCC value exceeds a predefined threshold (VM), the component is deemed valid and added to the valid component table. Otherwise, it is marked as non-valid and appended to the non-valid component table using an incrementing variable 'j'.

Once all components are processed, the valid and non-valid component tables are returned, providing a comprehensive assessment of component existence within the PCB image.

3.7.4 Minimum number of features for valid registration KPN_min

In our VIS utilizing the SIFT method, we have set the minimum number of good key points required for a registration to be valid to 30. This threshold is supported by extensive empirical testing and relevant literature.

Although the study does not specify an exact threshold, the analysis suggests that using a sufficient number of key points (typically within the range of 20-50) is crucial for achieving reliable matches under various transformations and noise conditions. Therefore, setting the minimum number of good key points(KPN_min)to 30 ensures that our system achieves robust and accurate matching, while remaining computationally efficient.

Descriptor Performance: evaluated the performance of different descriptors under image transformations such as scale changes, rotation, and noise. They found that descriptors like SIFT perform well in maintaining robust matching performance across these variations.

Number of Key Points: While the paper does not specify an exact number of key points required, their experiments typically involve matching hundreds of key points to evaluate performance metrics. This implies the necessity of a reasonably high number of key points to ensure reliable object recognition and matching [23].

3.7.5 Optimal NCC Threshold VM

In our Visual Inspection System , we have set the threshold for NCC to 0.7. This threshold was chosen based on empirical testing and relevant literature, which suggests that a threshold in the range of 0.7 to 0.9 provides a good balance between accuracy and robustness. Empirical studies in image matching and pattern recognition typically use thresholds within this range to ensure reliable matches while avoiding false positives. Therefore, by setting the NCC threshold to 0.7, we ensure that our system achieves a high degree of matching reliability, maintaining both computational efficiency and accuracy.

Research papers and textbooks on image processing and computer vision often recommend NCC thresholds in the range of 0.7 to 0.9. For example, in the book ”Digital Image Processing” [24], NCC is discussed as a similarity measure, and typical threshold values around 0.7 to 0.8 are used in practice for template matching tasks. This range is considered optimal for balancing sensitivity and specificity, ensuring that the system reliably identifies true matches while minimizing the risk of false positives. [25]

3.7.6 The Case for Using IROs in template matching

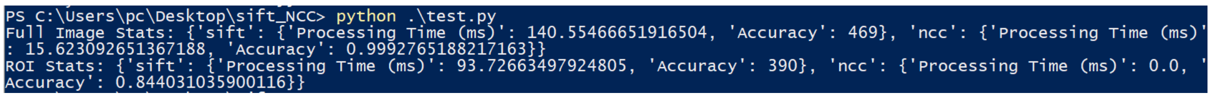
During the implementation of the SIFT and NCC methods, significant delays were observed when comparing the template against the entire image. Both methods require substantial computational resources, leading to slow performance, which is impractical for real-time applications.

The primary reason for the slow performance is the computational complexity of both SIFT and NCC. SIFT involves detecting keypoints and computing descriptors, which is computationally intensive. NCC requires comparing pixel values over the entire image, which further adds to the processing time. When applied to the whole image, the time complexity increases, leading to inefficient processing times.

To address this issue, the comparison was limited to a specific Region of Interest (ROI) where the template is likely to be located. This approach significantly reduces the number of computations required, thereby speeding up the process.

The ROI was determined based on prior knowledge of the PCB layout, which allowed the narrowing down of the area where the component (template) would be located. By extracting this ROI, the SIFT and NCC methods were applied only within this smaller area. This reduction in the area of interest led to a decrease in the number of keypoints and descriptors in SIFT, as well as the number of pixel comparisons in NCC, resulting in faster processing times.

For instance, if a specific component is known to be located in the upper-left quadrant of the PCB, this quadrant is defined as the ROI, and computational processes are restricted to this area. This targeted approach effectively decreases the computational load, enhancing the efficiency of the VIS.



```
PS C:\Users\pc\Desktop\sift_NCC> python .\test.py
Full Image Stats: {'sift': {'Processing Time (ms)': 140.55466651916504, 'Accuracy': 469}, 'ncc': {'Processing Time (ms)'
: 15.623092651367188, 'Accuracy': 0.9992765188217163}}
ROI Stats: {'sift': {'Processing Time (ms)': 93.72663497924805, 'Accuracy': 390}, 'ncc': {'Processing Time (ms)': 0.0, '
Accuracy': 0.844031035900116}}
```

Table 3.2: Template matching statistics

Method	Full Image	ROI
SIFT		
Processing Time (ms)	140.55	93.73
Accuracy	469	390
NCC		
Processing Time (ms)	15.62	0.0
Accuracy	0.9993	0.8440

Using an ROI significantly reduces the processing time for SIFT, highlighting the benefits of focusing computational resources on smaller, relevant areas of the image. For NCC, the accuracy within the ROI drops, indicating that careful selection and definition of the ROI are crucial for maintaining high accuracy.

3.8 Third stage : Solder joint defects

The dataset used in the third stage of the model consists of 10,668 images of bare PCBs, each depicting one of six common defects: missing holes, mouse bites, open circuits, short circuits, spurs, and spurious copper. All images are resized from 600x600 to 608x608 for both training and testing.

We utilize the trained YOLOv5 model to detect open holes in the soldering process. The test images are processed by the YOLOv5 model, which predicts bounding boxes and confidence scores to identify and locate open holes. Detected open holes with confidence scores above a predefined threshold are marked, facilitating efficient and accurate identification of soldering defects. This stage ensures high-quality soldering by highlighting potential defects for further inspection [26].

3.9 Visual Inspection System User Graphical Interface (GUI)

Developing a Graphical User Interface (GUI) for a visual inspection system for PCBs is crucial for enhancing usability, real-time monitoring, data management, and overall efficiency. A well-designed GUI makes the system accessible to users with varying levels of technical expertise, reducing the need for extensive training and minimizing errors. It enables real-time monitoring, allowing users to view the status of inspections, detect defects, and receive immediate feedback. This feature is vital for quickly addressing issues in the manufacturing process. Moreover, a GUI simplifies data management by providing easy access to historical inspection data, facilitating trend analysis and informed decision-making. In this chapter, we will discuss the GUI of our visual inspection system for the three stages.

3.9.1 The main GUI Window

The main GUI (Figure 4.1) for our VIS is designed to make navigating through our PCB inspection stages straightforward. It includes three clear buttons, each for a specific inspection step: detecting defects using YOLOv5 and image processing, verifying component presence and placement with SIFT and NCC template matching, and checking solder quality using YOLOv5. Operators can easily start and monitor inspections with these buttons, which provide immediate visual feedback. This basic but friendly design simplifies operations, improves efficiency, and ensures thorough quality control in PCB manufacturing. Having everything in one place helps streamline workflow and makes it easier for operators to perform accurate inspections.

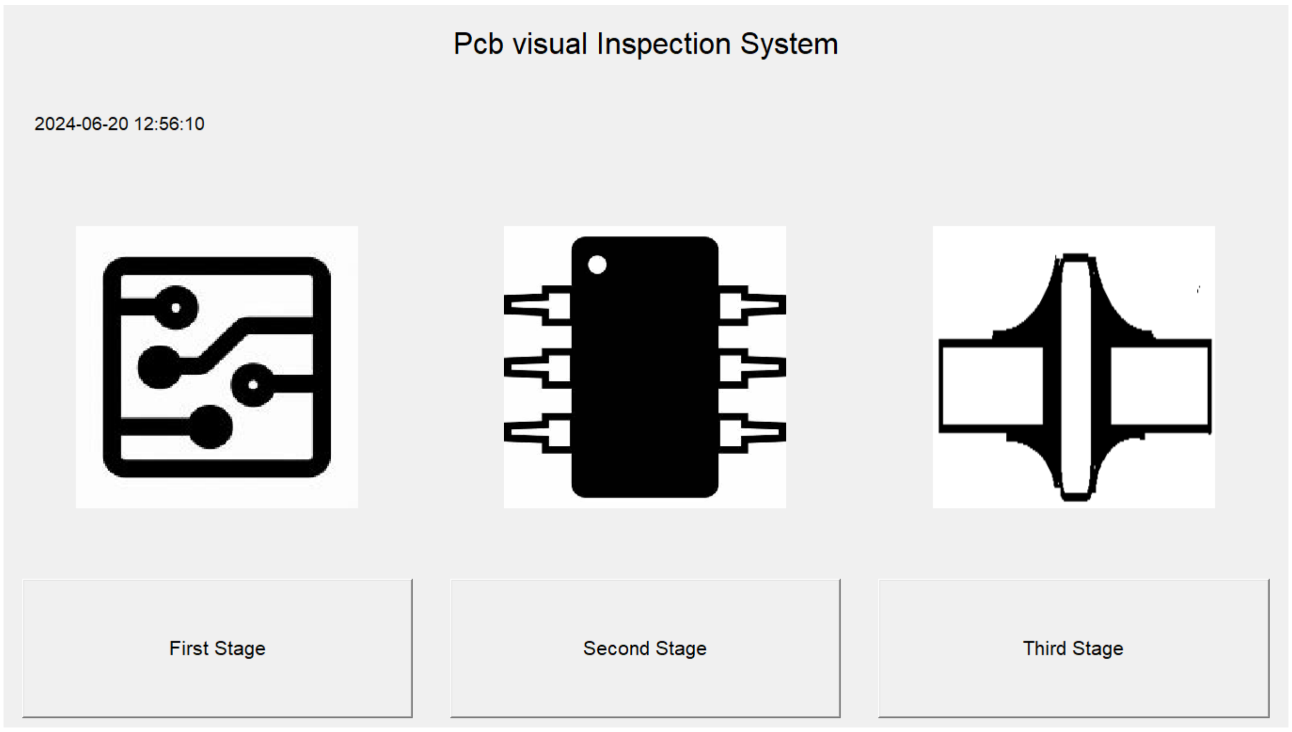
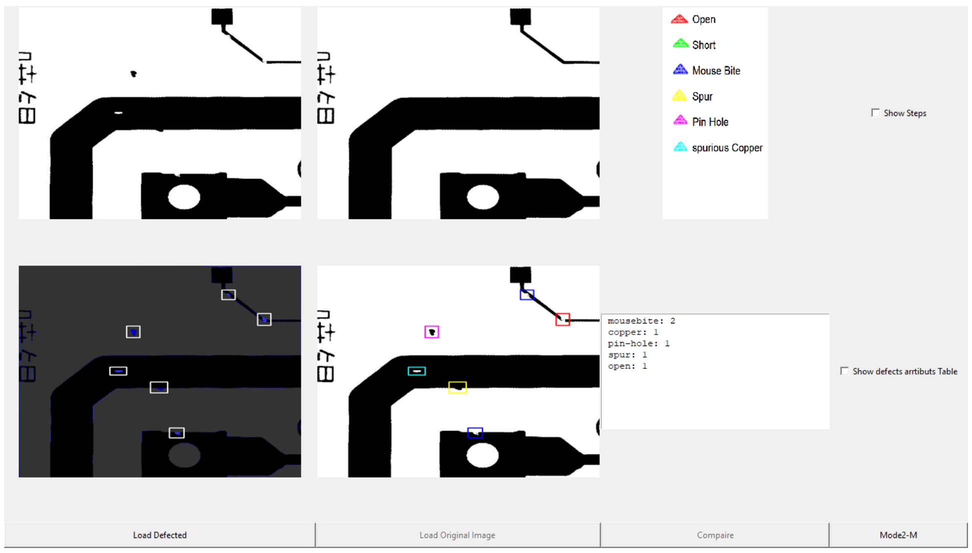


Figure 3.21: Main GUI

First Stage GUI

The GUI for the first stage of our inspection system is designed to detect defects on PCBs. It includes four main image panels: one for the original PCB image, one highlighting detected defects, one showing potential defect locations with color differences, and one displaying confirmed issues. There is also a panel that shows each defect with a colored bounding box for easy identification and classification.

Two checkboxes add more functions: one shows a table with defect coordinates and classifications, and the other provides a step-by-step view of the inspection process. Users can load test and reference images using dedicated buttons, choose between manual and automatic operation modes, and start the inspection with a start button. This setup helps operators efficiently find and confirm PCB defects, ensuring high-quality manufacturing.



```
mousebite: 2
copper: 1
pin-hole: 1
spur: 1
open: 1
```

Figure 3.22: First stage GUI

3.9.2 Second stage GUI

Our GUI for the second stage of the VIS features fouOur GUI for the second stage of the VIS features four image labels: one for the reference image, one for the test image under analysis, one showing the mapped test image with green rectangles for found components and red rectangles for missing or misplaced components, and one for the mapped reference image for comparison. The interface includes four buttons: one to load the test image, one to load the reference image, one to load a CSV file for data input, and a run button to start the inspection process.

Users can select between different algorithms (NCC, SIFT, or both) for component detection using radio buttons. A text area is available to display detailed inspection reports, providing comprehensive feedback on component presence, placement accuracy, and any identified issues. This setup allows for intuitive image comparison, automated detection using selected algorithms, and detailed reporting, facilitating thorough visual inspections and analysis within a single user-friendly interface.

3.9.3 Third Stage GUI

The GUI for the third stage of our inspection system focuses on assessing solder quality for PCBs, providing a straightforward interface for operators. It features two primary elements: a test image label that displays the original PCB image with an emphasis on solder joints, and a test image with labeled defects that highlights areas of concern related to solder quality. Additionally, there is a text box that dynamically updates to list and count detected defects during the inspection process, offering a clear summary of identified issues for efficient review and

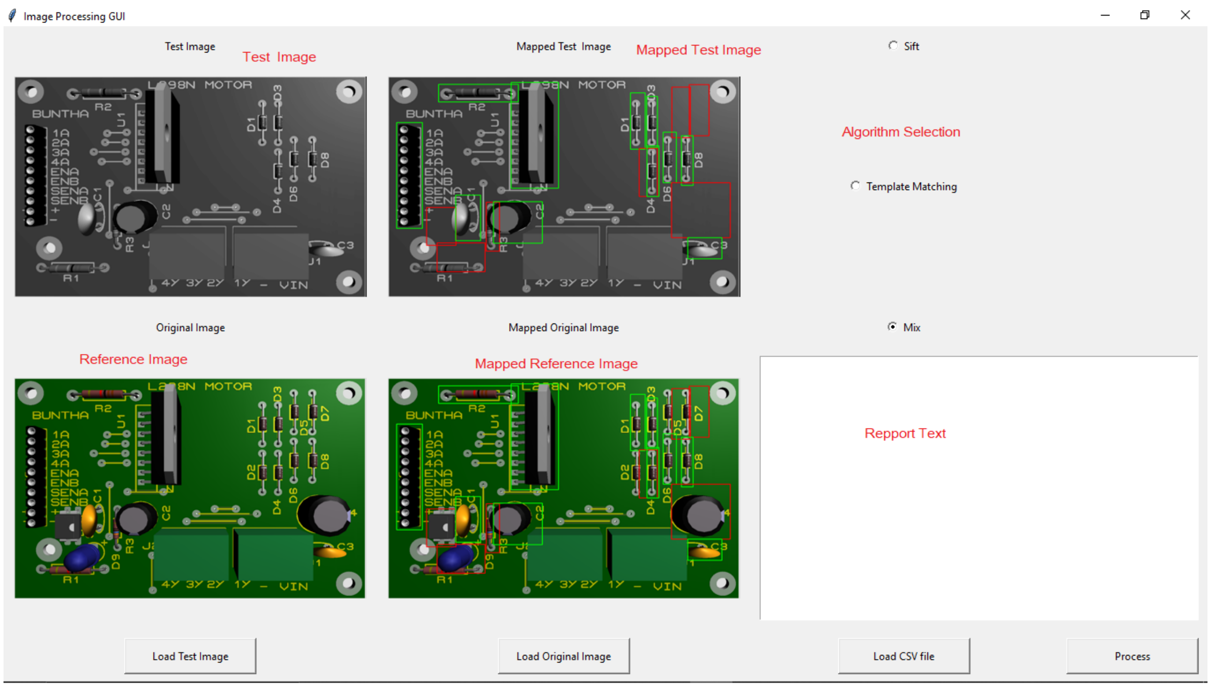


Figure 3.23: Second stage GUI

documentation. This setup allows operators to quickly assess soldering quality, make timely adjustments, and ensure high standards in PCB manufacturing.

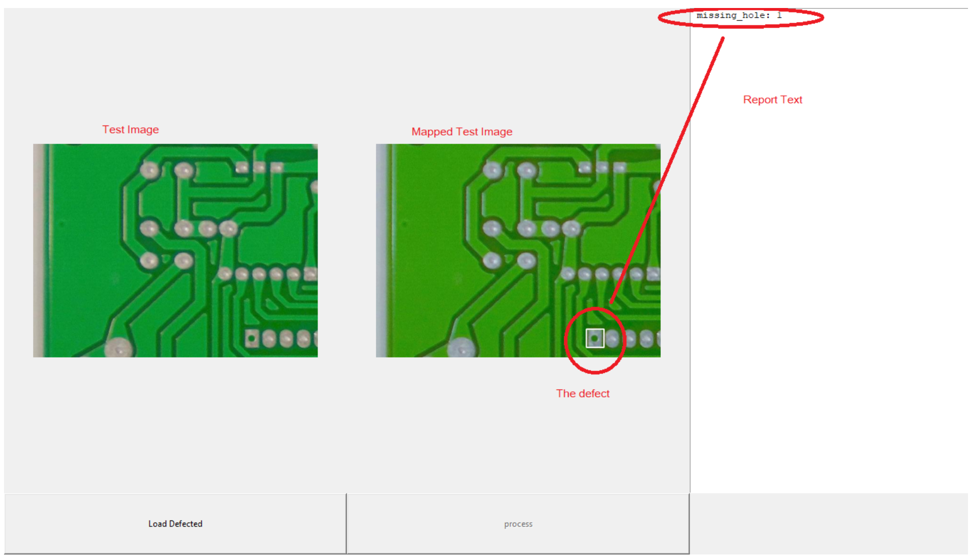


Figure 3.24: Third stage GUI

3.9.4 Conclusion

In this chapter, we explored the practical application of advanced image processing techniques and YOLO algorithms for the automated visual inspection of PCBs. The primary objective was to develop a robust system capable of accurately detecting defects, verifying component placement, and assessing solder joint quality. The system's architecture was detailed, emphasizing the interconnection of various components and modules. Image acquisition and preprocessing were crucial for capturing high-quality images suitable for analysis. The YOLO algorithm was implemented for defect detection at the track level, and methods for component localization were discussed to ensure correct component placement. Additionally, the application of the YOLOv5 algorithm for solder quality inspection was elaborated, highlighting its effectiveness in identifying and classifying solder joints. Performance metrics such as accuracy, precision, recall, and F1-score were provided to demonstrate the system's effectiveness. Examples of detected defects and inspected solder joints illustrated the practical applications and benefits of the system.

Furthermore, the chapter detailed the GUIs developed for the VIS used in PCB manufacturing. The system comprises three main stages, each with its own specific interface to streamline operations and ensure high-quality inspections. The main GUI simplifies navigation through inspection stages, allowing operators to initiate and monitor inspections easily. The first stage GUI focuses on detecting defects on PCBs tracks, providing multiple image views and tools for detailed defect analysis. The second stage GUI assists in verifying component presence and placement , The third stage GUI evaluates solder quality, highlighting defects and providing real-time updates for efficient review. Overall, the GUIs are designed to enhance operator efficiency, improve inspection accuracy, and maintain rigorous quality control standards in PCB manufacturing.

General Conclusion

The development of an advanced visual inspection system for printed circuit boards is crucial for ensuring the quality and reliability of electronic devices. This system leverages high-resolution cameras and sophisticated lighting setups to capture detailed images of PCBs, which are then analyzed using advanced image processing techniques and deep learning algorithms. The primary aim is to automate the defect detection process, enhancing the efficiency and accuracy of quality control in PCB manufacturing.

The first stage of the system addresses defects at the track level, utilizing artificial intelligence (AI) and image processing techniques to accurately locate and classify various types of defects. This includes identifying issues such as missing holes, mouse bites, open circuits, short circuits , spurs, and spurious copper. Advanced algorithms analyze high-resolution images of the PCB tracks to detect these defects with precision.

The second stage focuses on verifying the presence and proper placement of components. This involves checking that all specified components are present on the PCB and correctly positioned according to the design specifications. This stage ensures that no components are missing or misaligned, which is crucial for the proper functioning of the PCB.

The third stage evaluates the quality of the solder joints.

Together, these three stages form a comprehensive visual inspection system that significantly enhances the quality control process for PCBs. By accurately detecting and classifying defects, verifying component presence, and assessing solder quality, the system ensures that only high-quality PCBs are used in electronic devices, improving their reliability and performance across various applications.

Using pre-trained models, particularly the YOLO algorithm, the system can quickly and accurately detect and classify defects. This approach significantly reduces the time and effort required for manual inspections, allowing for real-time monitoring and quality assurance.

Overall, the implementation of this advanced VIS enhances the manufacturing process by ensuring that only defect-free PCBs are used in electronic devices, thus improving the overall quality and reliability of the products. This system not only boosts production efficiency but also helps in maintaining high standards in the electronics industry .

Bibliography

[1] G. K. M. Ali, R. Ravinetto, and A. A. Alfadl, “The importance of visual inspection in national quality assurance systems for medicines,” vol. 13, no. 1, p. 52.

[2] R. Szeliski, *Computer Vision: Algorithms and Applications*. Texts in Computer Science, Springer London, 2011.

[3] R. Parent, *Computer Animation: Algorithms and Techniques*. Amsterdam: Elsevier, Morgan Kaufmann, 3rd ed., 2012.

[4] Editors: Subrahmanyam Murala, B. Bhanu, *Feature Extraction and Image Processing for Computer Vision*. Elsevier, 2020.

[5] S.-H. Huang and Y.-C. Pan, “Automated visual inspection in the semiconductor industry: A survey,” vol. 66, pp. 1–10.

[6] A. Martin and S. Tosunoglu, “Image processing techniques for machine vision,” *Miami, Florida*, 2000.

[7] S. Bagavathiappan, B. Lahiri, T. Saravanan, J. Philip, and T. Jayakumar, “Infrared thermography for condition monitoring – a review,” vol. 60, pp. 35–55.

[8] I. Andreadis, “21 - automated visual inspection systems,” in *Expert Systems* (C. T. Leondes, ed.), pp. 771–800, Academic Press.

[9] E. R. Davies, *Computer and machine vision: theory, algorithms, practicalities*. Elsevier, 4th ed ed. OCLC: ocn761856317.

[10] A. Burduk, E. Chlebus, T. Nowakowski, and A. Tubis, *Intelligent Systems in Production Engineering and Maintenance*. Springer, July 2018. Google Books ID: afpmDwAAQBAJ.

[11] J. Fillmore, “A note on rotation matrices,” vol. 4, no. 2, pp. 30–33.

[12] E. Name, ed., *Handbook of Medical Imaging. Volume 1: Physics and Psychophysics*. SPIE Optical Engineering Press, 2000.

[13] D. G. Lowe, “Distinctive image features from scale-invariant keypoints,” vol. 60, no. 2, pp. 91–110.

[14] I. Rey Otero and M. Delbracio, “Anatomy of the SIFT method,” vol. 4, pp. 370–396.

[15] E. Karami, S. Prasad, and M. Shehata, “Image matching using sift, surf, brief and orb: Performance comparison for distorted images,” *arXiv preprint arXiv:1710.02726*, 2017.

[16] G. S. Cox, “Template matching and measures of match in image processing,” *University of Cape Town, South Africa*, 1995.

[17] R. J. de Jong, J. J. de Wit, and F. Uysal, “Classification of human activity using radar and video multimodal learning,” *IET Radar, Sonar & Navigation*, vol. 15, no. 8, pp. 902–914, 2021.

[18] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, “On the importance of initialization and momentum in deep learning,” in *International conference on machine learning*, pp. 1139–1147, PMLR, 2013.

[19] R. Moradi, R. Berangi, and B. Minaei, “A survey of regularization strategies for deep models,” vol. 53, no. 6, pp. 3947–3986.

[20] M. Vakalopoulou, S. Christodoulidis, N. Burgos, O. Colliot, and V. Lepetit, “Deep learning: basics and convolutional neural networks (cnns),” *Machine Learning for Brain Disorders*, pp. 77–115, 2023.

[21] M. Z. Alom, T. M. Taha, C. Yakopcic, S. Westberg, P. Sidike, M. S. Nasrin, B. C. Van Esesn, A. A. S. Awwal, and V. K. Asari, “The history began from alexnet: A comprehensive survey on deep learning approaches,” *arXiv preprint arXiv:1803.01164*, 2018.

[22] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, “A comprehensive survey on transfer learning,” *Proceedings of the IEEE*, vol. 109, no. 1, pp. 43–76, 2020.

[23] OpenCV, “Introduction to sift (scale-invariant feature transform),” 2020. Accessed: 2024-06-20.

[24] R. C. Gonzalez, *Digital image processing*. Pearson education india, 2009.

[25] D. University, “Image correlation, convolution and filtering,” 2015. Accessed: 2024-06-20.

[26] JiaLim98, “Yolo-pcb: A deep context learning based pcb defect detection model with anomalous trend alarming system.” Accessed: June 20, 2024.

Abstract

Printed Circuit Boards (PCBs) are fundamental in electronic devices, connecting various parts to form functional circuits. Inspecting these boards for defects is crucial for ensuring their performance and reliability. This thesis concerned with the development of a Visual Inspection System for PCBs using advanced image processing and deep learning(DL) techniques.

The system explores high-resolution images of PCBs. These images are then analyzed using pattern recognition, edge detection, and machine learning algorithms to detect defects. The main focus is on leveraging pre-trained deep learning models, particularly YOLO (You Only Look Once), for automatic and early defect detection.

Key words: Printed circuit boards (PCBs), visual inspection, image processing, deep learning, YOLO, defect detection.

ملخص

تُعد لوحات الدوائر المطبوعة (PCBs) أساسية في الأجهزة الإلكترونية، حيث تربط أجزاء مختلفة لتكوين دوائر وظيفية. يُعد فحص هذه اللوحات بحثًا عن العيوب أمرًا بالغ الأهمية لضمان أدائها وموثوقيتها. تتعلق هذه الأطروحة بتطوير نظام فحص بصري للوحات الدارات المطبوعة باستخدام تقنيات متقدمة لمعالجة الصور والتعلم العميق. يقوم النظام بمسح صور عالية الدقة لمركبات ثنائي الفينيل متعدد الكلور. يتم بعد ذلك تحليل هذه الصور باستخدام خوارزميات التعرف على الأنماط واكتشاف الحواف وخوارزميات التعلم الآلي لاكتشاف العيوب. ينصب التركيز الرئيسي على استخدام نماذج التعلم العميق المدربة مسبقاً، ولا سيما نموذج YOLO (أنت انظر مرة واحدة فقط)، للكشف التلقائي والمبكر عن العيوب.

الكلمات المفتاحية: لوحات الدوائر المطبوعة (PCBs)، والفحص البصري، ومعالجة الصور، والتعلم العميق، و YOLO، واكتشاف الأعطال.

Résumé

Les cartes de circuits imprimés (PCBs) sont fondamentales dans les dispositifs électroniques, connectant diverses parties pour former des circuits fonctionnels. L'inspection de ces cartes pour détecter les défauts est cruciale pour garantir leur performance et leur fiabilité. Cette thèse concerne le développement d'un système d'inspection visuelle pour les PCBs utilisant des techniques avancées de traitement d'image et d'apprentissage profond.

Le système explore des images de haute résolution des PCBs. Ces images sont ensuite analysées à l'aide de la reconnaissance de formes, de la détection des contours et d'algorithmes d'apprentissage automatique pour détecter les défauts. L'accent principal est mis sur l'utilisation de modèles d'apprentissage profond pré-entraînés, en particulier YOLO (You Only Look Once), pour la détection automatique et précoce des défauts.

Mots-clés: Cartes de circuits imprimés (PCBs), inspection visuelle, traitement d'image, apprentissage profond, YOLO, détection de défauts.