TECHNISCHE UNIVERSITÄT
CHEMNITZ

Development of a Computer Vision Algorithm for Textile Quality Inspection

Master Thesis

Submitted in Fulfilment of the
Requirements for the Academic Degree
M.Sc.

Dept. of Computer Science
Chair of Computer Engineering

Submitted by: Saivinay Juluri
Student ID: 682044
Date: 23-03-2023

Supervising Tutor: Prof. Dr. Wolfram Hardt
Internal supervisor: M.Sc. Shadi Saleh
External Supervisor: Dipl-Math. Frieder Lamnek

Juluri, Saivinay
Development of a Computer Vision Algorithm for Textile Quality Inspection
Master Thesis, Department of Computer Science
Technische Universität Chemnitz, August 2023

Abstract

Quality inspection is a crucial feature of contemporary businesses, especially in the textile industry, where defect detection is critical. Traditionally, this process relies heavily on human visual inspection, resulting in inefficiencies, substantial labor costs, and considerable time investments. This thesis seeks to address these issues by developing an automated fabric defect inspection system based on computer vision techniques. Specifically, the algorithm will focus on detecting and classifying three types of textile errors. Our approach incorporates two paradigms for error detection. This includes classical image analysis methods and Convolutional Neural Network-based object detection algorithms. Subsequently, we will also compare the performance of these two algorithms. Additionally, the investigation of the potential for deploying the trained algorithms on real-time embedded systems. This aspect will help with the practical implications and feasibility of integrating such an automatic inspection system within the textile industry.

Keywords: Fabric Quality Inspection, Fabric Defects, Computer vision, Blob detection, YOLOv5, Object Detection, Jetson Xavier NX,

Acknowledgement

Firstly, I am deeply thankful to Prof. Dr. Dr. h. c. Wolfram Hardt from the Technical University of Chemnitz for being a constant source of inspiration and guidance during my master's thesis in the Computer Science department. Your remarkable support throughout this journey has been invaluable.

I would want to offer my deepest appreciation to Mr. Shadi Saleh, my supervisor at TU Chemnitz, for his invaluable guidance and expertise in the field of Computer Vision and Artificial Intelligence. His profound knowledge and keen insights have significantly enriched my work and understanding in this domain. The constructive feedback and insightful suggestions he provided have played a crucial role in improving the quality of this thesis.

I am sincerely grateful to Mr. Frieder Lamnek, my supervisor from Unicontrol Systemtechnic GmbH, for his unwavering support and encouragement throughout the Master's thesis journey. During challenging times, his motivating words and confidence in my abilities helped me overcome obstacles and stay focused. His mentorship has been instrumental in shaping this thesis into a piece of academic work that I am incredibly proud of. I can't thank him enough for making it feasible for me to finish my thesis.

My heartfelt gratitude goes out to Unicontrol Systemtechnik GmbH and Mr. Norman Thieme for providing me with the opportunity to conduct my research. Their willingness to share information, provide resources, and offer invaluable insights has been instrumental in the successful completion of this thesis.

To my parents, my sister, my brother-in-law, and all my friends and well-wishers who have accompanied me on this remarkable journey, I extend my profound gratitude. Completing my thesis has offered me great delight and fulfillment. I sincerely appreciate the unwavering support, motivation, and inspiration I have received from each one of you. Your faith in me has been a driving force, and I am deeply thankful for your constant encouragement throughout this process.

Contents

List of Figures

List of Tables

List of Abbreviations

CV Computer Vision

ANN Aritificial Neural Networks

CNN Convolutional Neural Networks

MAP Mean Average Precision

FI Fabric Images

OpenCV Open Source Computer Vision Library

YOLO You Look Only Once

ROI Regions of Interest

SVM support vector machines

MSE Mean Squared Error

BCE Binary Cross Entropy

CCE Categorical Cross Entropy

SCCE Sparse Categorical Cross Entropy

SCE Softmax Cross Entropy

SDK software Development Kit

SOM system On Module

1 Introduction

In the textile industry, inspection of fabric defects plays an important role in quality control. However, The traditional Quality inspection and defect detection of the fabric industry is primarily performed by human inspectors [27]. During the examination, the fabric roll is loaded into a fabric inspection machine, and the fabric roll can be withdrawn and placed on the inspection surface while the fabric is moving. When the fabric inspector detects a flaw in the moving fabric by visual inspection, he or she stops the machine and notes the flaw's position. Although the human visual examination is excellent for fabric inspection [49]. This conventional approach necessitates the use of a trained operator, who must be trained for an extended period of time in order to become a proficient fabric inspector or operator. Visual examination is a time-consuming operation even for an inspector. After lengthy working hours, they often overlook little flaws, and occasionally even major ones. Human inspection is unreliable because a fabric inspector must examine the fabric generally with a breadth of 1.6-2 m on a moving status of roughly 10 m per minute. A fabric inspector's ability to focus for an extended amount of time is quite tough. Thus, manual fabric inspection, on which the existing business is heavily reliant, has an accuracy rate of roughly 60%–75% [30]. As a result, inspection efficiency is low [49]. There are many fabric defects that are occurred during production, some of them are Horizontal lines, shade variation, dirt/stains, nips/knots, and other fabric flaws abound. Picks that are broken, ends that are broken, and ends that are missing Snags, Uneven coloring, thick/thin spots, incorrectly colored fibers, Elastane breakage, thick yarn Holes [17], splicing, needle lines, coarse picks, slubs, oil spot, and so forth these are some of the defects in the fabric industry. Undetected faults generate several difficulties downstream in the manufacturing chain, resulting in lower-quality final goods that are costly and non-profitable. Furthermore, correcting faults is a difficult operation and faulty components are often thrown as wastes that may be recycled or sold at cheap rates (generally 45%–65% of the free defect price). Automatic defect detection systems, on the other hand, provide viable alternatives to traditional human fabric inspection by analyzing fabrics in a systematic manner and aiming for consistent performance. However, the efficiency of these automated systems is determined by a variety of factors, including the quality of the hardware and the analytic algorithm [17].

This thesis explores an extensive exploration of fabric defect detection using computer vision-based techniques. It primarily focuses on implementing two algorithms: the traditional Blob Detection Algorithm and the modern CNN-based Object Detection Model. To enable algorithm development and training, a controlled and scalable dataset creation approach will be adopted, involving the generation of synthetic data. The study will then proceed to evaluate and compare the accuracy of both algorithms in detecting fabric defects. Additionally, real fabric images will be used to validate the algorithm's performance in practical scenarios, providing a realistic assessment. Beyond accuracy evaluation, the research will conduct a comparative analysis between machine learning and classical image analysis approaches, aiming to identify their respective strengths and limitations. Moreover, the impact of different algorithm development workflows on adaptability and

usability in real-world applications will be thoroughly explored. Lastly, the research will assess the algorithms' potential for efficient real-time defect detection by verifying them on an embedded system. This thesis seeks to contribute valuable insights to the fabric defect detection domain, offering implications for enhancing quality control practices in the textile industry and relevant fields.

This master thesis document is structured into eight Chapters. Chapter 2 provides the task description, main objectives, and research questions, establishing a clear framework for the study. In Chapter 3, all the essential basics required for understanding the concepts and implementation are explained. Chapter 4 offers an overview of the thesis, providing theoretical knowledge of both algorithms, Blob detection, and YOLOv5. Additionally, it explains the information on the dataset used in this master thesis. The core of the work is presented in Chapter 5, which covers the implementation process and showcases the training and validation results of both algorithms. In Chapter 7, the results of algorithm development and deployment are discussed in detail, along with an analysis of the outcomes. The Conclusion chapter presents a comprehensive summary of the thesis, highlighting the main findings and contributions. Furthermore, the chapter suggests potential ideas for future research and exploration in the field.

2 Task Description

Task Description

In recent years machine learning technics and artificial neuronal networks have seen more and more use in different fields of industry for computer vision tasks. Here the industry follows the successes in research from the 2010s where convolutional neuronal networks have been shown to solve vision tasks like image classification, object detection, or semantic segmentation with an accuracy matching and even surpassing human abilities. Still, many tasks in quality control rely on human inspection and subjective decisions. In textile manufacturing for example 100% of the product needs to be checked after coloring to remove errors before further processing steps can be performed. In the thesis, the use case of a white fabric with the 3 common errors should be explored. In the first step image analysis methods like blob detection should be used to find the errors. Those methods are commonly used in the industry to solve such tasks but require in-depth knowledge of the expected behavior and are therefore difficult to maintain and adapt to changes. As a comparison machine learning methods should be used to learn the errors from data directly. To add to the challenge acquiring data often is the main concern for using such methods as a good representation of real behavior is needed. In a lot of cases, some errors are more common than others posing a challenge to collecting real data. In some applications, it was shown that models trained on 3D animated data can translate well into real-world applications. Therefore, the use of synthetically generated data should be discussed. A workflow to adapt the algorithms to new error types and color variants should be developed by adding a 4th error type. Often image evaluation with complex algorithms or machine learning models relies on transmitting uncompressed data to a powerful processor or a cloud service. This makes data transmission a bottleneck conflicting with the high-speed real-time requirements of industrial applications. To solve this issue hardware development in recent years has led to specialized systems that can run those kinds of algorithms locally. Therefore, in an optional addition to this survey, implementation of those algorithms on an NVIDIA Jetson Xavier-based system can be done.

Motivation

In this thesis, three typical errors occurring in textile production should be explored in detail. A synthetic dataset of textile images containing those errors as well as a small set of real images will be provided. This reflects difficulties in practice to create a balanced representative dataset and the need to work with synthetic data. The given problem will be solved by implementing both an image analysis algorithm based on the blob detection method and a machine learning algorithm using a convolutional neuronal network with the goal of 95% detection accuracy on synthetic validation data. The methods should also be applied to check real images with the goal of over 80% correct detections. The implementation is to be done in 6 months using the machine learning frameworks like

PyTorch and the python image processing libraries like OpenCV. To be used in the field the algorithms need to be deployed on an Nvidia Jetson-based embedded system and provide the evaluation results in under 100ms.

Objectives

The Objectives of this thesis are:

- Implement the blob detection algorithm and train the CNN-based object detection model.
- Use Synthetic data for algorithm development.
- Evaluate and compare the accuracy of both algorithms for detecting errors in the images and validate synthetic data results on real images
- Comparison of the differences in the algorithm development workflow affects the adaptability and usability of the algorithms in the field.
- Check and verify the developed algorithms on an embedded system.

Research Questions

The following are the Research questions for this Thesis:

- Are there image analysis algorithms that can detect those three errors with high accuracy?
- Can synthetic data be used for algorithms development?
- How do machine learning algorithms compare to classical image analysis algorithms?
- What do differences in the algorithm development workflow affect the adaptability and usability of the algorithms in the field?
- How do those methods perform on an embedded system?

3 Basics

This chapter explains the basic foundation knowledge and basic definitions of the master thesis concept, implementation, evaluation, and also deployment.

3.1 Computer Vision

Computer vision is the conversion of data from a still or video camera into a judgment or a new representation [23]. Computer vision is a subfield of artificial intelligence that teaches computers to interpret and comprehend their visual surroundings. It is concerned with the extraction of high-dimensional data from the real world in order to generate numerical or symbolic information, as well as techniques for acquiring, processing, analyzing, and interpreting digital pictures. In layman's terms, computer vision seeks to recreate human vision via the use of computer software and hardware. Computer vision involves training robots to perceive and interpret the environment in the same way that people use their eyes and minds to comprehend their surroundings.

3.2 Image Analysis

Definition of an Image

A digital picture is a discrete representation of data that processes both spatial and intensity information [42]. A visual representation or portrayal of an item, situation, or concept is referred to as an image. An image is often a two-dimensional array of pixels in the context of digital technology, with each pixel representing a minor aspect of the larger picture. In the case of a color image, each pixel includes color information; in the case of a grayscale image, each pixel contains grayscale intensity.

Image layout

The two-dimensional (2-D) discrete, digital image I(m,n) depicts a sensor's response at a series of fixed locations in two-dimensional Cartesian coordinates (m = 1,2,3...M, n = 1,2,3...N) and is produced from the two-dimensional (2-D) continuous spatial signal I(x,y) via a sampling process known as discretization. Discretization happens naturally in some image sensors and affects a local average of the continuous signal over a limited (typically square) receiving domain region. The indices m and n denote the rows and columns of the picture, accordingly. Individual picture elements or pixels in an image are therefore recognized by their 2-D (m,n) index. I(m,n) represents the response of the pixel in the mth row and nth column beginning from the top-left image origin [42]. Figure 3.1 presents the Image Layout.

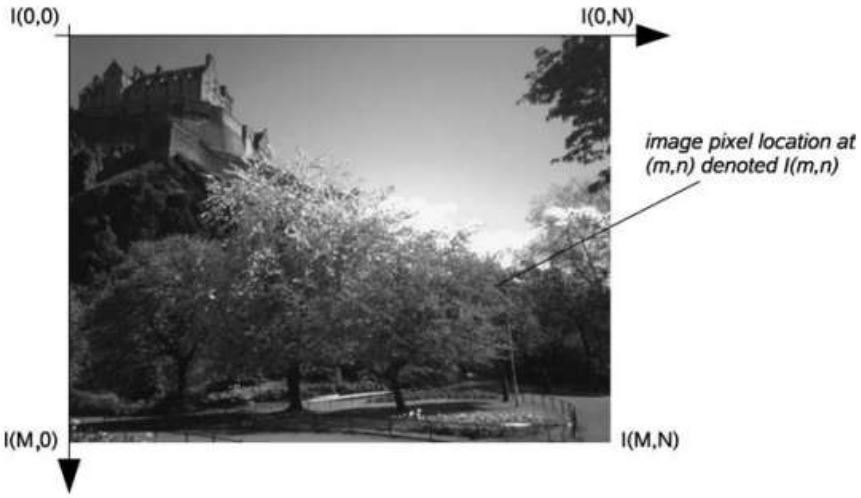

Figure 3.1: Image Layout[42]

Image Color

A picture has one or more color channels that describe the intensity or color at a certain place l(m,n). In its most basic form, each pixel position contains only a single numerical value that represents the signal strength at that location in the image. A color map is used to translate this collection of data into the actual displayed image. A color map assigns a different shade of color to each numerical level in the image to create a visual representation of the data. The translation from this collection of numbers to the actual displayed image is accomplished via a color map. To provide a visual representation of the data, a color map provides a distinct shade of color to each numerical level in the image. Grayscale is the most frequent type of color map. The next section describes the color spaces.

Image Data Types

Binary Images: Binary images are two-dimensional arrays, where each pixel is assigned either a 0 or a 1. They are occasionally known as logical images, with black represented as 0 signifying an 'off' or 'background' pixel, and white represented as 1 indicating an 'on' or 'foreground' pixel. Although only these two values are allowed, making it possible to represent these images as a simple bit stream, they are commonly represented as 8-bit integer images in popular image formats. A facsimile or fax image serves as a practical example of a binary image [42].

Intensity or Grayscale Images: Intensity or grayscale images are two-dimensional arrays in which each pixel is assigned a numerical value that represents its intensity. As mentioned earlier, the range of possible pixel values is determined by the bit depth of the image. These images are stored as N-bit integer images in a specific format [42].

RGB Images: RGB or true-color images represent a three-dimensional array. Each pixel has three numerical values that correspond to the Red, Green, and Blue color channels. They may be thought of as three distinct two-dimensional planes. Commonly, these images are stored in a sequential order of the color channels for example, R0G0B0,

R1G1B1, ... Other color formats are also stored using a similar 3-D array structure. This model can be expanded to include additional image information, like an alpha (transparency) channel, often seen in PNG format images [42].

Floating Point Images: Floating-point images, unlike other image types, store floating-point numbers rather than integer color values. These numbers represent the intensity within a range defined by the image's floating-point precision. Often, they might represent a measurement value beyond mere intensity or color, particularly in scientific or medical imaging [42].

Pixel Values

The content contained within an individual image element, or pixel, is contingent on the data type utilized to express it. Typically, pixel values are composed of binary strings with a length of k, allowing each pixel to denote one of the $2\hat{k}$ diverse values. The variable k is referred to as the image's bit depth, or simply "depth". The specific arrangement of bits within each pixel is determined by the type of image it belongs to, such as binary, grayscale, or RGB color. Below, you'll find a summary of the attributes associated with some commonly encountered image types [16].

It is typical in digital image processing to employ a coordinate system with the origin (u = 0,v = 0) in the upper left corner. The image's columns and rows are represented by the coordinates u and v, respectively [16]. The maximum column number for an image with dimensions M N is umax = M - 1 and the maximum row number is vmax = N.

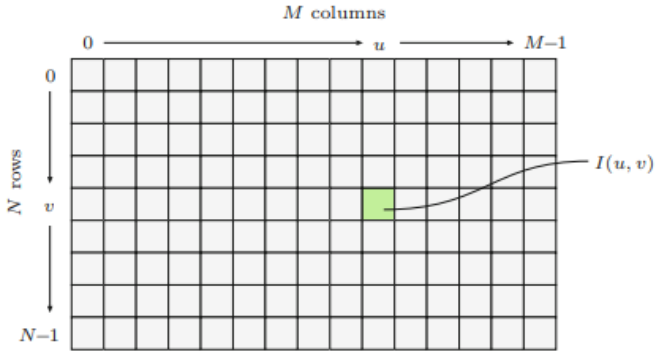

Figure 3.2: Image Coordinates[16]

Channels	bits/pixel	Range
1	1	[0,1]
1	8	[0,255]
1	12	[0,4095]
1	14	[0,16383]
1	16	[0, 65535]

Table 3.1: Grayscale Images Bit Depths[16]

Channels	bits/pixel	Range
3	24	$[0, 255]^3$
3	36	$[0, 4095]^3$
3	42	$[0, 16383]^3$
4	32	$[0, 255]^4$

Table 3.2: Color Images Bit Depths[16]

3.3 Image Spaces

RGB Image Space

The RGB space is the most significant in image processing since color cameras, scanners, and displays are frequently equipped with RGB signal input or output. This color space is the fundamental one that is turned into other color spaces if necessary. The RGB primaries of these devices, however, are not always constant. In RGB space, the color gamut forms a cube (fig 3.1). Each hue is characterized by its RGB components, is represented by a point, and can be located on the cube's surface or within. From black (R=G=B=0) to white (R=G=B=max), all gray colors are positioned on the main diagonal of this cube [39].

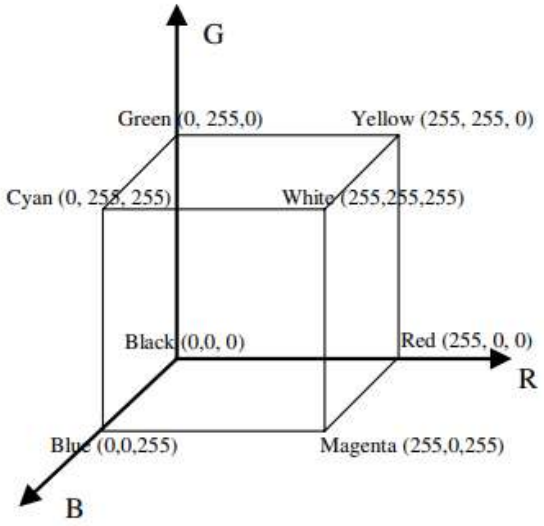

Figure 3.3: 3D Cube Representation[36]

Each color channel in the RGB color space is commonly represented by an 8-bit number ranging from 0 to 255. A pixel's color is determined by combining the intensity levels of the three color channels. An RGB value of (255, 0, 0) for example, symbolizes full-intensity red, (0, 255, 0) for full-intensity green, and (0, 0, 255) for full-intensity blue. Color tints and colors may be created by combining different intensity values for each channel.

A value of 0 in the CMY color space denotes maximum ink intensity, whereas 255 signifies no ink. As a result, a CMY value of (0, 255, 255) indicates full-intensity Cyan (no red or green ink), (255, 0, 255) indicates full-intensity Magenta (no green or blue ink), and (255, 255, 0) indicates full-intensity Yellow (no blue or red ink).

To get the true RGB values from CMY, invert the CMY values, which reflect the lack of color. Subtracting the CMY values from 255 accomplishes this. Because there is no Cyan ink and the maximum intensity of red is present, a CMY value of (0, 255, 255) would be transformed to an RGB value of (255, 0, 0), showing full-intensity red.

True Color Images

In a true-color image, each pixel has the potential to represent any color within its specific color space, provided it lies within the distinct range set for its individual color components. True-color images are particularly suitable when the image encompasses a vast array of colors with slight variations, a characteristic common in digital photography and photorealistic computer graphics. The arrangement of the color components in true-color images can be conducted through two methods: component ordering and packed ordering [16].

Component Ordering

In the method known as component ordering, also called as planar ordering, the color components are arranged in separate arrays, each having the same dimensions. Under this arrangement, one can perceive the color image as a collection of associated intensity images: IR, IG, and IB. To acquire the RGB values of the color image I at a specific location denoted by coordinates (u, v), one needs to access these three component images accordingly [16].

$$I_{comp} = (I_R, I_G, I_B) \tag{3.1}$$

$$\begin{pmatrix} R(u,v) \\ G(u,v) \\ B(u,v) \end{pmatrix} = \begin{pmatrix} I_R(u,v) \\ I_G(u,v) \\ I_B(u,v) \end{pmatrix} \tag{3.2}$$

Figure 3.4 shows the RGB color image in component ordering. The three color components are laid out in separate arrays IR, IG, and IB of the same size.

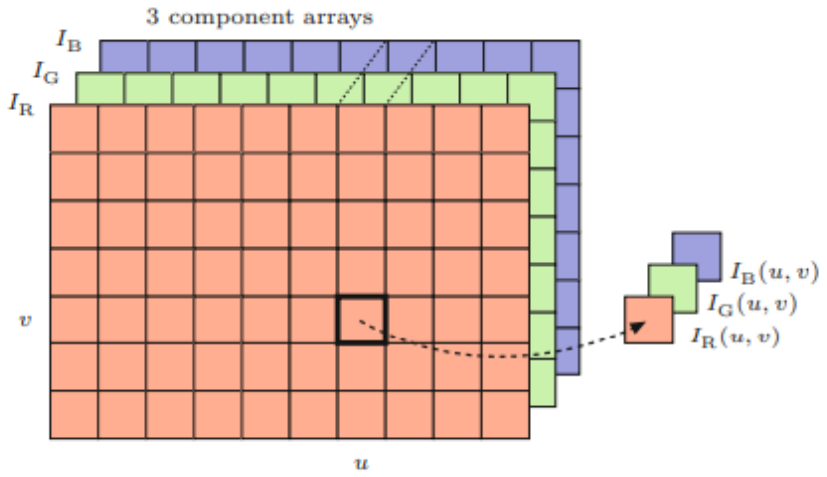

Figure 3.4: RGB Color Image Component Ordering[16]

Packed Ordering

The component values that represent the color of a specific pixel are packed together into a single element of the picture array in packed ordering.

$$I_{pack}(u, v) = (R, G, B) \tag{3.3}$$

The RGB value of a packed image I at the location (u, v) is obtained by accessing the individual components of the color pixel as:

$$\begin{pmatrix} R(u,v) \\ G(u,v) \\ B(u,v) \end{pmatrix} = \begin{pmatrix} \text{Red}(I_{pack}(u,v)) \\ \text{Green}(I_{pack}(u,v)) \\ \text{Blue}(I_{pack}(u,v)) \end{pmatrix} \tag{3.4}$$

Figure 3.5 shows the RGB-color image using packed ordering. The three color components R, G, and B are placed together in a single array element.

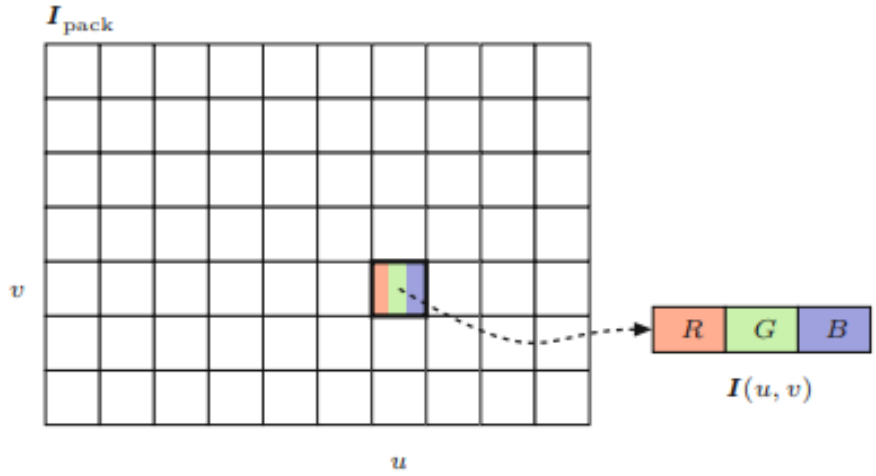

Figure 3.5: RGB Color Image Packed Ordering[16]

RGB to Grayscale image conversion

Converting an image from RGB to grayscale is a simple transformation that is frequently used in image analysis techniques. This strategy is basically a simplification method for reducing the amount of data in the image. Despite the loss of color information, a grayscale image retains the majority of the critical feature-related information, such as edges, regions, blobs, and junctions. Following that, the grayscale images are utilized to train feature recognition and processing algorithms. As shown in hypothetical Figure XX, red and green apples may be identified even in monochrome. Because this conversion approach retains feature discernibility, it is an essential step in many image-processing jobs [42].

An RGB color image, Icolour, is converted to greyscale, Igrey-scale, using the following transformation:

$$I_{\text{grey-scale}}(n,m) = aI_{\text{colour}}(n,m,r) + bI_{\text{colour}}(n,m,g) + cI_{\text{colour}}(n,m,b) \tag{3.5}$$

The coordinates (n, m) relates to a single pixel in the grayscale picture, whereas (n, m, c) indicate the individual color channel in the color image at pixel location (n, m). The 'c' denotes the image's red (r), blue (b), or green (g) channels. According to Equation (3.1), the grayscale picture is a weighted mixture of the red, green, and blue color channels. In accordance with the NTSC television standard, the weighting factors (a, b, and g) are set based on the human eye's relative sensitivity to red, green, and blue light. Specifically, an is equal to 0.2989, b is equivalent to 0.5870, and g is equal to 0.1140.

Because the human eye is more sensitive to red and green light, these colors are given larger weightings in the grayscale image to preserve a comparable intensity balance as in the RGB color image. It is critical to understand that RGB to grayscale conversion is a non-reversible picture change. The particular color information that is lost during this procedure is not easily recoverable.

3.4 Binary Thresholding

3.4.1 Simple Thresholding

A single threshold value is applied to the whole image in simple or global thresholding. If the pixel intensity exceeds the threshold value, it is set to a certain value, generally the maximum intensity value; otherwise, it is set to another value, usually zero. This method assumes that the image has a significant bimodal distribution and that the grayscale histogram has distinct peaks or regions. There are several types of simple thresholding including. As shown in Table 10.2, each threshold type corresponds to a particular comparison operation between the ith source pixel SRCI and the threshold. The destination pixel DSTI may be set to 0, SRCI, or the specified maximum value MAXVALUE depending on the connection between the source pixel and the threshold [23].

1. **Binary Thresholding:** This operation sets the destination pixel intensity to a maximum value if the source pixel intensity is greater than a specified threshold. If the source pixel intensity is not greater than the threshold, the destination pixel intensity is set to zero.

2. **Inverse Binary Thresholding:** This is the inverse operation of the binary threshold. If the intensity of the source pixel exceeds the threshold, the intensity of the destination pixel is set to zero. If the intensity of the source pixel is not greater than the threshold, it is set to the maximum value.

3. **Truncate Thresholding:** If the source pixel intensity is larger than the threshold, the destination pixel intensity is set to the threshold value in this operation. If the intensity of the source pixel is less than the threshold, the destination pixel is set to the same value as the source pixel (i.e., it remains unmodified).

4. **Threshold To Zero:** For this operation, if the source pixel intensity is greater than the threshold, the destination pixel is set to the same value as the source

pixel. If the source pixel intensity is not greater than the threshold, the destination pixel intensity is set to zero.

5. **Inverted Threshold To Zero:** This is the inverse of the 'to zero' thresholding operation. If the intensity of the source pixel exceeds the threshold, the intensity of the destination pixel is set to zero. If the intensity of the source pixel is not greater than the threshold, the destination pixel is set to the same value as the source pixel [23].

Type of Thresholding	**Operation**
Binary Thresholding	$DST_i = (SRC_i > \text{thresh})?\text{MAX VALUE} : 0$
Inverse Binary Thresholding	$DST_i = (SRC_i > \text{thresh})?0 : \text{MAX VALUE}$
Truncate Thresholding	$DST_i = (SRC_i > \text{thresh})?\text{thresh} : SRC_i$
Threshold To Zero	$DST_i = (SRC_i > \text{thresh})?SRC_i : 0$
Inverted Threshold To Zero	$DST_i = (SRC_i > \text{thresh})?0 : SRC_i$

Table 3.3: Types of Simple Thresholding [23]

The horizontal line through each chart in Figure 3.6 depicts a specific threshold level applied to the top chart and its effect for each of the five types of threshold procedures.

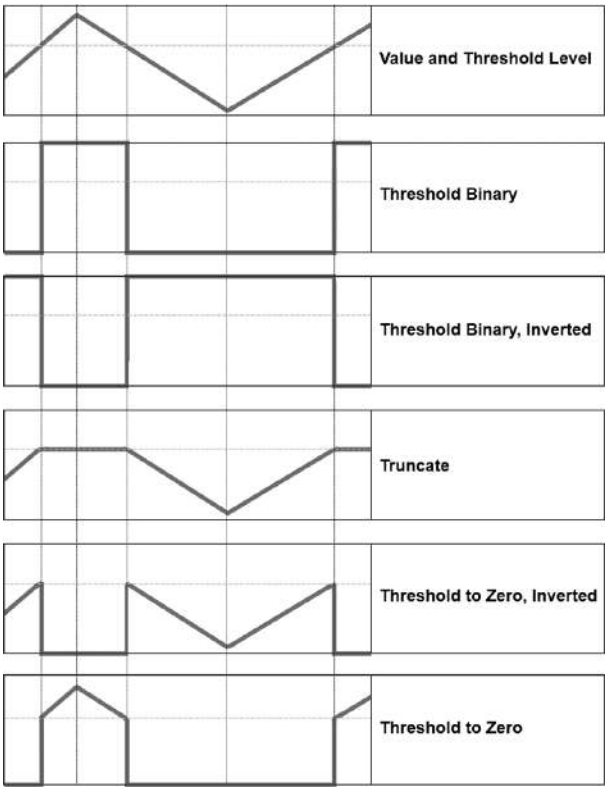

Figure 3.6: Types of Simple Thresholding [23]

3.4.2 Adaptive Thresholding

In Simple thresholding, a single value was used as a threshold for the entire image to identify if a pixel is light or dark. However, this approach may not work optimally if the image has various regions with differing light conditions. Adaptive thresholding provides a solution to this issue. With adaptive thresholding, the algorithm assesses small regions around each pixel to determine its brightness level. As a result, the algorithm can apply different thresholds for different areas of the image. This method proves more effective for images with regions of varied lighting. There are two types of adaptive thresholding methods as explained below:

Mean Adaptive Thresholding: In this method, the threshold value for each pixel is the mean (average) of the pixel intensities in a neighborhood (block) around the pixel, minus a constant value. This technique adjusts the threshold dynamically, which makes it suitable for handling images with different lighting conditions across various areas [23].

Gaussian Adaptive Thresholding: In this method, the threshold value for each pixel is a weighted sum of the neighboring pixel intensities, with the weights determined by a Gaussian window (closer pixels have more weight), again minus a constant. This method is especially beneficial for images with varied lighting conditions in different regions, providing a smoother transition between the differing lighting conditions[23].

3.5 Machine Learning

Machine learning is a subfield of artificial intelligence that focuses on developing algorithms and models that allow computers to learn and make predictions or decisions without being explicitly programmed. It entails developing mathematical models and algorithms that let computers analyze and understand large amounts of data, detect patterns, and make intelligent judgments or predictions[32].

Machine learning involves training computers on a dataset that contains input data and related output labels or intended results. The computer learns patterns and correlations in the data during the training process, allowing it to make predictions or conclusions when presented with fresh, previously unknown data [9].

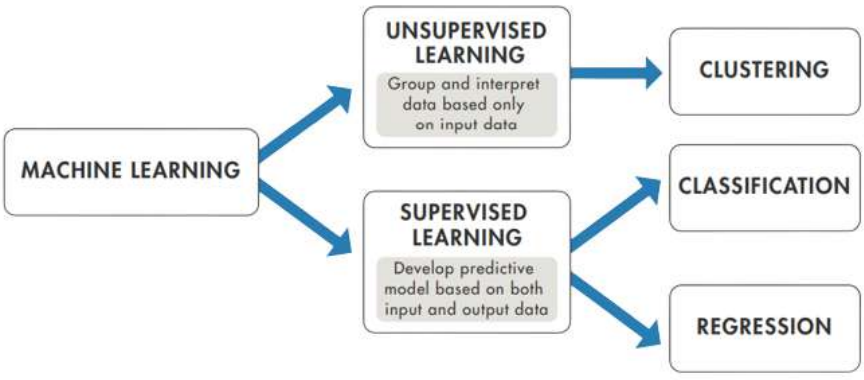

Figure 3.7: Machine Learning Methods [9]

Supervised Learning: A sort of machine learning in which the algorithm learns from labeled data is referred to as supervised learning. The training dataset in this technique comprises input data and the appropriate output labels or target values. The algorithm's purpose is to discover a mapping or relationship between the input characteristics and the output labels. The algorithm analyzes the labeled samples during the training process and attempts to generalize the patterns and correlations in the data. Based on the learned patterns, the system can generate predictions or categorize fresh, unknown data. Supervised learning methods are further subdivided into networks for regression and classification applications [9]. Regression is used to estimate a continuous numerical value, such as property prices, depending on variables such as area, number of rooms, and so on. The purpose of categorization is to assign input data to predetermined categories or groups, such as spamming or not spamming emails depending on their content. Linear regression, decision trees, SVM, and neural networks are examples of supervised learning techniques [40].

Unsupervised Learning: An unsupervised learning strategy is one in which the algorithm learns from unlabeled input. The training dataset in this technique is made up of input data with no matching output labels or goal values. The purpose is to find hidden patterns, structures, or correlations in data. The program examines the unlabeled data to identify commonalities, groupings, or latent variables. It seeks to discover meaningful representations or organize data in such a manner that the underlying structure can be understood. Clustering[19], dimensionality reduction, and anomaly detection may all be accomplished using unsupervised learning methods. Clustering methods bring together related data points, whereas dimensionality reduction approaches try to minimize the number of input characteristics while retaining critical information. Anomaly detection methods discover data points that differ considerably from the dataset's regular trends. k-means clustering, hierarchical clustering, principal component analysis (PCA), and autoencoders are examples of unsupervised learning methods [9] [40].

3.6 Deep Learning Basics

Deep learning is a machine learning approach in which a model learns to do categorization tasks directly from pictures, text, or audio. Deep learning is often implemented using a neural network design. The term "deep" refers to the network's layer count; the more layers there are, the deeper the network. Deep networks include hundreds of layers, whereas traditional neural networks only have two or three. The objective of recreating biological brain systems was the foundation for the field of neural networks, but it has since diverged and evolved into a preoccupation with engineering and getting good results in machine learning tasks. An artificial neuron is a function that accepts input and produces output. Different amounts of neurons are employed depending on the job at hand. It might be as few as two or as many as thousands. Artificial neurons may be linked in a multitude of ways to create a CNN. A feed-forward network is one of these topologies that is often used [41].

Each neuron receives information from other neurons. The weight controls the influence of each input line on the neuron. Both positive and negative weights are conceivable. Understanding the language teaches the whole neural network how to perform useful

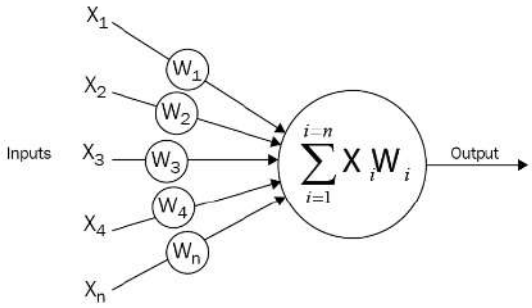

Figure 3.8: Feed Forward Network [41]

object recognition calculations. Neurons can be linked together to form a feed-forward network, in which the output of each neuron in one layer serves as the input to neurons in the next layer. This procedure is repeated until we reach the final output layer. The feed-forward network allows information to travel in a single direction, from the input layer via the hidden levels to the final output, without the use of feedback loops. This architecture is widely used in machine learning and deep learning models for classification, regression, and pattern recognition. One way to put it is as follows:

$$w_1x_1 + w_2x_2 + \ldots + w_nx_n \tag{3.6}$$

$$\sum_{i=1}^{n} w_ix_i = w_1x_1 + w_2x_2 + \ldots + w_nx_n \tag{3.7}$$

3.6.1 Convolutional Neural Networks

A deep neural network comprises several nonlinear processing layers inspired by organic nerve systems, with simple parts running in parallel. It consists of many hidden layers, an input layer, and an output layer. The layers are connected by nodes or neurons, with each hidden layer receiving the output of the previous layer as input [41].

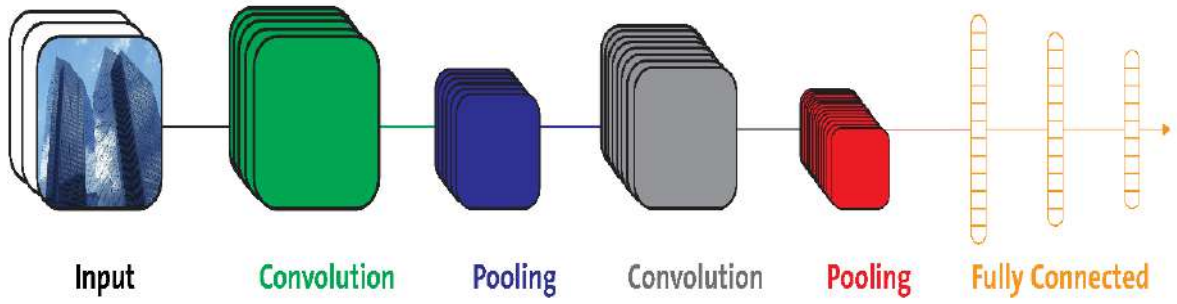

Figure 3.9: CNN Architecture [41]

CNN Feature Extraction

The initial stage of a CNN is feature extraction, in which the network learns to extract meaningful characteristics from the incoming data. This method employs a succession of convolutional layers that execute convolution operations on the input pictures to find patterns and features. Convolutional layers employ learnable filters (also known as kernels) that move across the input data, producing dot products between the filter weights and the input's local receptive fields. This process pulls characteristics of various sizes and captures spatial hierarchies. To incorporate non-linearity and improve the network's capacity to learn complicated representations, the output of the convolutional layers is routed via non-linear activation functions such as ReLU (Rectified Linear Unit). To minimize the spatial dimensions of the feature maps while keeping the most important information, additional processes such as pooling or downsampling are frequently applied. Pooling aids in abstracting and summarizing the learned characteristics, making the network more resistant to translation and scale fluctuations [8].

CNN Classification

The retrieved features are then transmitted to fully linked layers, also known as classification layers, after the feature extraction stage. These layers are in charge of translating high-level characteristics into final output classes or predictions. Each neuron in the fully connected layers is coupled to every neuron in the layer before it, allowing the network to learn detailed connections between extracted qualities and target classes. The last fully linked layer's output is often fed into a softmax activation function, which turns the activations of the final layer into probability scores for each class. The projected class label for the supplied data is thus the class with the highest probability. Throughout the training process, both the feature extraction and classification layers are changed using optimization methods (e.g., gradient descent) to reduce the difference between the predicted outputs and the ground truth labels. Backpropagation modifies the network's weights and biases, allowing it to learn discriminative features and enhance classification performance [8].

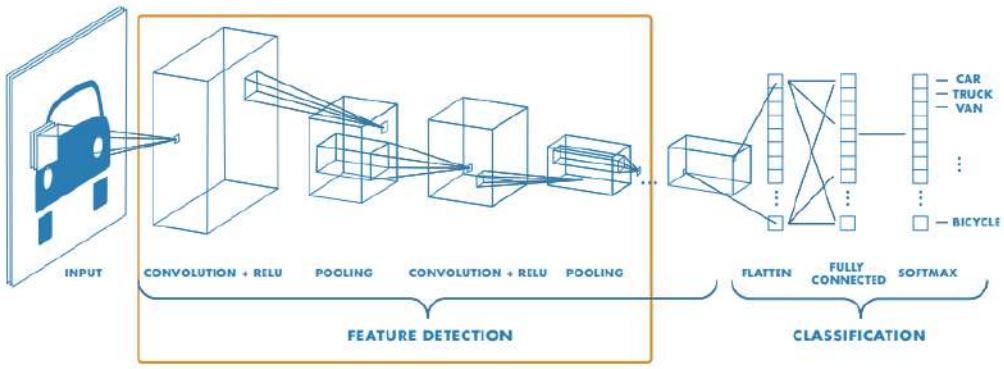

Figure 3.10: CNN Classification and Feature Extraction[8]

Input Layer

The picture data is stored in the input layer. The input layer in the diagram below has three inputs. The neurons connecting two neighboring layers in a fully connected layer

are completely linked pairwise but do not share any connections within a layer. In other words, all activations in the preceding layer are fully connected to the neurons in this layer. As a result, their activations may be determined using simple matrix multiplication and, optionally, a bias factor. A convolutional layer varies from a fully connected layer in that its neurons are connected to a local region in the input and share parameters [41].

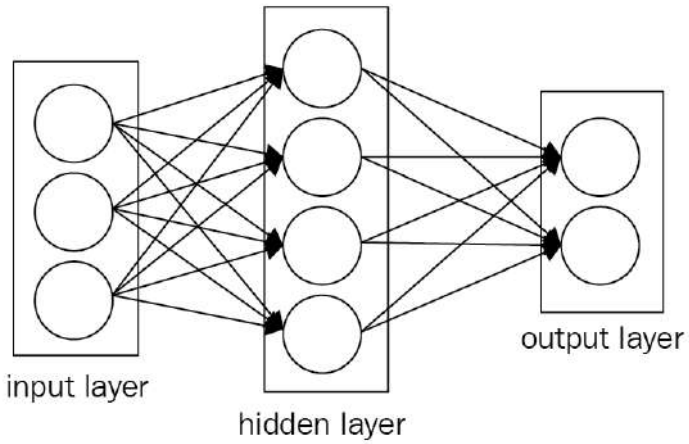

Figure 3.11: Input Layer[41]

Convolutional Layers

The name of the convolutional layer comes from the convolution operator. This layer's goal is to extract picture features. By learning visual attributes, convolution preserves the spatial link between pixels [41] using small squares of input data. For feature recognition and extraction, each convolution layer employs several filters such as Edge recognition, Sharpen, Blur, and so on. Kernels and feature detectors are other names for these filters. After swiping the filter across the image, we receive a feature map matrix. The convolution in the first convolution layer is between the input picture and its filters. The neuron weights are represented by the filter values (see Figure 3.4). The size of the feature map is determined by three criteria.

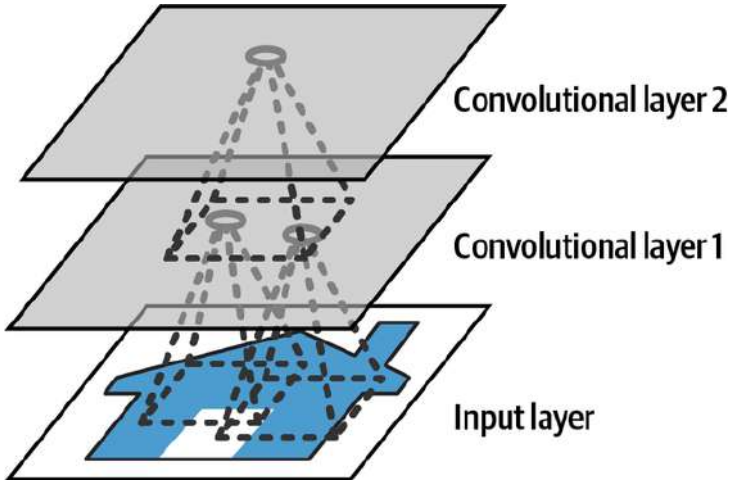

Figure 3.12: CNN layers with rectangular local receptive fields[19]

1. Depth: Depth relates to the number of filters employed during the convolution process [26].

2. Stride: The number of pixels by which we slide our filter matrix across the input matrix is referred to as the "stride." Smaller feature maps emerge from a longer stride.

3. Zero-padding: It is practical to pad the input matrix with zeros all the way around the border in order to apply the filter on the bordering components of the input image.

Pooling Layers

A convolutional layer is a collection of feature maps, one for each filter. Convolution becomes more dimensional as more filters are added. More parameters are indicated by higher dimensionality. As a consequence, the pooling layer prevents overfitting by gradually minimizing the representation's spatial dimension, reducing the number of parameters and computation. The pooling layer typically receives input from the convolutional layer. Max pooling is the most often used pooling method. Pooling units can perform other purposes outside max pooling, such as average pooling. We may modify the behavior of the convolutional layer in a CNN by adjusting the size of each filter and the number of filters. We can increase the number of filters in a convolutional layer to increase the number of nodes, and we can increase the size of the filter to increase the size of the pattern. There are a few other hyperparameters that can be tweaked. One of them is the convolution's stride. The amount by which the filter glides over the picture is referred to as the stride. A stride of 1 advances the filter horizontally and vertically by one pixel. In this case, the convolution becomes the same width and depth as the input picture. If the filter goes outside the picture, we can disregard these unknown values or replace them with zeros. This is referred to as padding [41].

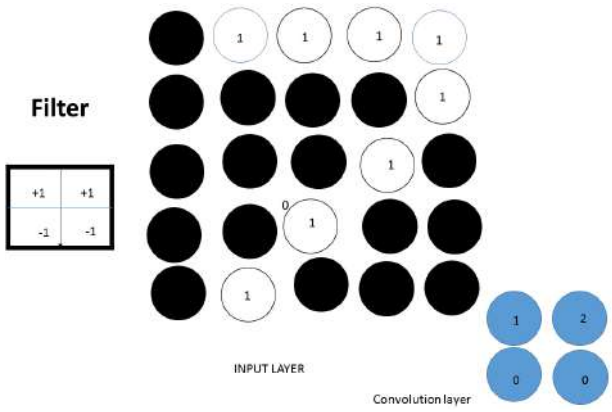

Figure 3.13: Pooling Layer[41]

Fully Connected Layer

A regular fully connected layer, also known as a FNN or dense layer, is placed to the top of the stack; it functions similarly to an MLP, which may be made of several fully connected layers, such as +ReLUs. The prediction, for example, is output by the last layer. A softmax layer that produces estimated class probabilities for multiclass classification is

one example. Every neuron in one layer is connected to every neuron in the next layer via fully connected layers. Although fully linked FNNs may be used to learn features as well as categorize data, using this architecture to photos is impractical [41].

Activation Funtions

To inject non-linearities into the network, activation functions are mathematical functions applied to the output of each neuron or convolutional layer. Activation functions determine a neuron's output depending on its input and whether or not the neuron should be activated. The use of activation functions in CNNs is intended to establish non-linear mappings between the neural network's input and output, allowing the network to learn complicated patterns and generate more expressive predictions. [41].

1. Rectified Linear Unit (ReLU): One of the most often utilized activation functions in CNNs is ReLU. It resets all negative numbers to zero while leaving positive values alone. It is written as $f(x) = max(0, x)$. ReLU mitigates the vanishing gradient problem and encourages sparse activations.

2. Sigmoid: The sigmoid function transforms the input into a number between 0 and 1. It is defined as $f(x) = \frac{1}{1+e^{-x}}$. Sigmoid is often used in the final layer of a binary classification CNN to produce probability-like outputs.

3. Hyperbolic Tangent (Tanh): Tanh is similar to the sigmoid function but maps the input to a value between -1 and 1. It is defined as $f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$. Tanh is often used in hidden layers of CNNs.

4. Softmax: Softmax is often utilized in the output layer of CNNs for multi-class classification. It transforms the input into a probability distribution across multiple classes, ensuring that the sum of all class probabilities is 1.

Pretrained Weights

The learning parameters of a neural network that have been trained on a big dataset for a certain task or area are referred to as pre-trained weights. These weights are generated after the network has been trained on a huge amount of data using advanced computing resources. A neural network is trained by fine-tuning its parameters such as weights and biases to minimize a specific loss function. This optimization procedure entails feeding input data to the network and modifying the weights depending on the computed error during training cycles. Pretrained weights are useful because they capture the learned representations and patterns from the training data, which can be used for other related tasks or as a starting point for additional fine-tuning [41]. Pre-trained weights from a CNN trained on a large-scale dataset like as ImageNet, for example, can be utilized as a starting point for different vision-related tasks such as object identification, picture classification, or object detection. These pre-trained weights give the network a favorable start, allowing it to converge faster and perform better on subsequent tasks with less training data [41].

Loss Functions

In Convolutional Neural Networks, loss functions, also known as objective functions or cost functions, measure the discrepancy between the predicted output of the network and

the true target values [41]. The choice of a specific loss function depends on the task at hand, such as classification, regression, or object detection. Here are some commonly used loss functions in CNNs:

Mean Squared Error: MSE is a popular loss function in regression problems. It computes the average squared difference between anticipated and actual values. The formula for MSE is:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_{\text{pred}_i} - y_{\text{true}_i})^2 \tag{3.8}$$

where y_{pred} represents the predicted values and y_{true} represents the true values. Binary Cross-Entropy: BCE is commonly used for binary classification tasks. It measures the difference between the predicted probabilities and the true binary labels. The formula for BCE is:

$$\text{BCE} = -\frac{1}{n}\sum_{i=1}^{n} y_{\text{true}_i}\log(y_{\text{pred}_i}) + (1 - y_{\text{true}_i})\log(1 - y_{\text{pred}_i}) \tag{3.9}$$

where y_{pred} represents the predicted probabilities and y_{true} represents the true binary labels. Categorical Cross-Entropy: For multi-class classification jobs, CCE is employed. It calculates the difference between the expected and true class probabilities. The formula for CCE is:

$$\text{CCE} = -\sum_{i=1}^{k} y_{\text{true}_i}\log(y_{\text{pred}_i}) \tag{3.10}$$

where y_{pred} represents the predicted class probabilities and y_{true} represents the true class labels (usually one-hot encoded). Sparse Categorical Cross-Entropy: SCCE is identical to CCE except that it is used when the genuine class labels are integers rather than one-hot encoded vectors. It avoids the need for one-hot encoding of the labels. The formula for SCCE is:

$$\text{SCCE} = -\sum_{i=1}^{k} y_{\text{true}_i}\log(y_{\text{pred}_i}) \tag{3.11}$$

Softmax Cross-Entropy: SCE combines the softmax activation function and cross-entropy loss. It is often used in multi-class classification tasks, where the softmax function is applied to the predicted logits to obtain class probabilities. The formula for SCE is:

$$\text{SCE} = -\sum_{i=1}^{k} y_{\text{true}_i}\log\left(\frac{e^{y_{\text{pred}_i}}}{\sum_{j=1}^{k} e^{y_{\text{pred}_j}}}\right) \tag{3.12}$$

3.6.2 Deep Learning Approaches

Image Classification

Classification is a fundamental task in deep learning where the goal is to assign a label or class to an input sample. This task involves training a model to recognize and categorize inputs into predefined classes. It is commonly used in image classification, where the model predicts the presence or absence of specific objects in an image [7].

Figure 3.14: Image Classification[7]

Object Detection

Object detection is a more complex task that involves both classification and localization. The goal is to detect and localize multiple objects within an image or a video frame and classify them into specific classes. Object detection models typically output bounding boxes around the detected objects along with their corresponding class labels [7].

Figure 3.15: Object Detection[7]

Image Segmentation

Image segmentation is a computer vision problem in which an image is divided into meaningful and semantically coherent sections or segments. The purpose is to divide the image into discrete sections based on attributes such as color, texture, or object borders. Each segment represents a separate object or region of interest within the image [7].

Figure 3.16: Image Segmentation[7]

3.6.3 Object Detectors

There are different types of object detectors such as Two-Stage Detectors, Single-Stage Detectors, Anchor-Free Detectors, One-Stage Detectors with Dense Prediction, Hybrid Detectors, etc. In this master thesis, the Single stage object detector type YOLO model is implemented the basic idea of this single-stage object detector is explained here [7].

Single Stage Object Detectors

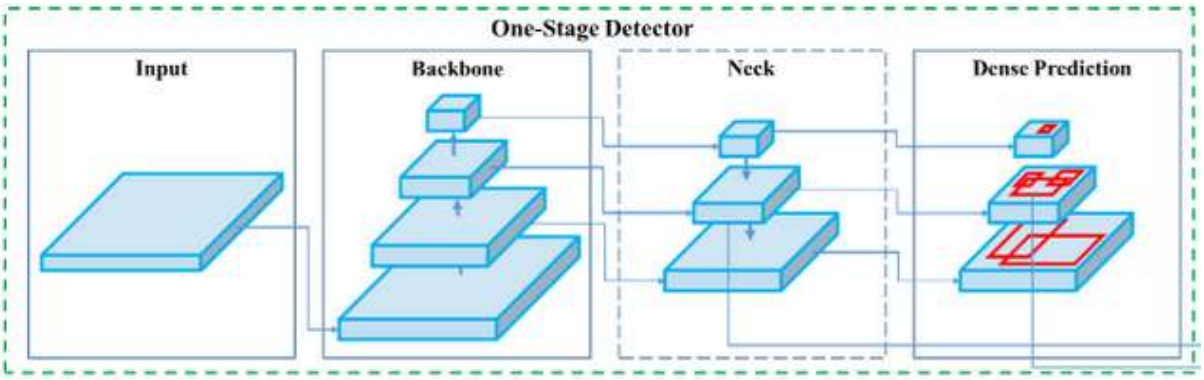

Figure 3.17: Single Stage Object Detector [13]

Model Backbone: The backbone in single-stage object detection acts as a feature extractor, providing rich and hierarchical representations of the input image. These representations serve as the foundation for subsequent layers in the object detection pipeline, such as the neck and dense prediction layers, which further refine and process the features to predict bounding boxes, objectness scores, and class probabilities [13].

Model Neck: The model neck in single-stage object detection acts as an intermediate processing stage that refines and enhances the features extracted by the backbone. It leverages techniques like feature fusion, contextual modeling, spatial attention, and additional convolutional layers to improve the model's ability to localize and classify objects accurately [13].

Model Head: The model head in single-stage object detection is responsible for generating the final predictions based on the extracted features from the backbone and the

refined representations from the neck module. It performs dense prediction tasks, including bounding box regression, objectness score estimation, and class probability prediction [13].

3.7 Evaluation Metrics

This subsection explains the metrics for the master's thesis evaluation. The creation of Blob Detection and the Yolov5 algorithm, as well as its implementation into embedded systems, have several types of evaluation metrics that can be covered in this section.

Algorithm Evaluation in the Development of an Algorithm

True Positive: This refers to the instances that are actually positive and are correctly detected as positive by the model.

True Negative: This refers to the instances that are actually negative and are correctly detected as negative by the model.

False Positive: This refers to when instances that are actually negative are incorrectly detected as positive by the model.

False Negative: This refers to the instances that are actually negative and are correctly detected as negative by the model.

Recall: The recall, also known as the sensitivity or True Positive Rate (TPR), represents the percentage of properly categorized positive samples and is computed as the ratio of successfully classified positive samples to all samples allocated to the positive class [20].

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

Precision: Precision is the fraction of accurately predicted positive cases out of all positive instances predicted by the model. It also assesses the accuracy of a model's positive predictions [19] [20].

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

F1 Score: The F1 score is the harmonic mean of accuracy and recall, which means that excessive levels of either are penalized. This metric is not symmetric between classes, which means that it relies on which class is regarded as positive and negative [20].

$$F1\ score = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Mean Average Precision: Mean Average Precision (mAP) is a commonly used evaluation metric in object detection and information retrieval tasks. It measures the average precision across multiple classes or queries. Similarly, in information retrieval, mAP is computed by calculating the average precision for each query and then taking the mean across all queries [51][19].

$$mAP = \frac{1}{N} \sum_{i=1}^{N} AP_i$$

Where N is the total number of classes. AP i represent the Average Precision (AP) for each class i.

Accuracy: The accuracy is defined as the proportion of properly identified samples to the total number of samples in the assessment dataset. "Number of Correct Predictions" refers to the instances that were classified correctly by the model, and the "Total Number of Predictions" represents the overall number of instances the model made predictions on [20].

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

Accuracy provides a straightforward measure of how well the model predicts the correct class labels. It gives an indication of the model's overall correctness in classifying instances. However, if the dataset is unbalanced, meaning that the classes are not fairly represented, accuracy might be deceptive. Other evaluation measures, such as accuracy, recall, or F1 score, may offer a more thorough assessment of the model's performance in such instances.

Algorithm Evaluation in Deployment of an Algorithm

Picture to Picture interference Time: In embedded systems, picture-to-picture interference refers to the visual interference or degradation that occurs when multiple pictures or frames are displayed simultaneously or sequentially.

3.8 Workflow of Traditional image processing and Deep Learning

Traditional image processing workflow

The traditional image processing pipeline consists of a series of steps, starting with the input image. In this approach, manual extraction plays a crucial role, where domain experts design and implement specific algorithms and filters to process the image. These algorithms are based on mathematical operations and predefined rules to manipulate the pixel data and extract meaningful information [31]. After manual extraction, relevant features are obtained from the processed image, representing distinctive patterns or characteristics such as edges, corners, texture, or color information. These extracted characteristics are then passed into a classifier, which uses the learned patterns to label or recognize the image's content. The output of the traditional image processing pipeline is the result of the classification process, such as identifying objects, defects, or patterns in the image. While this approach can be effective for well-defined tasks with limited variations, it may lack the adaptability and robustness required for complex and diverse real-world scenarios. With the advent of deep learning, which can automatically learn features from data, traditional image processing has been complemented and, in some cases, replaced by more powerful and versatile deep learning-based approaches in computer vision tasks.

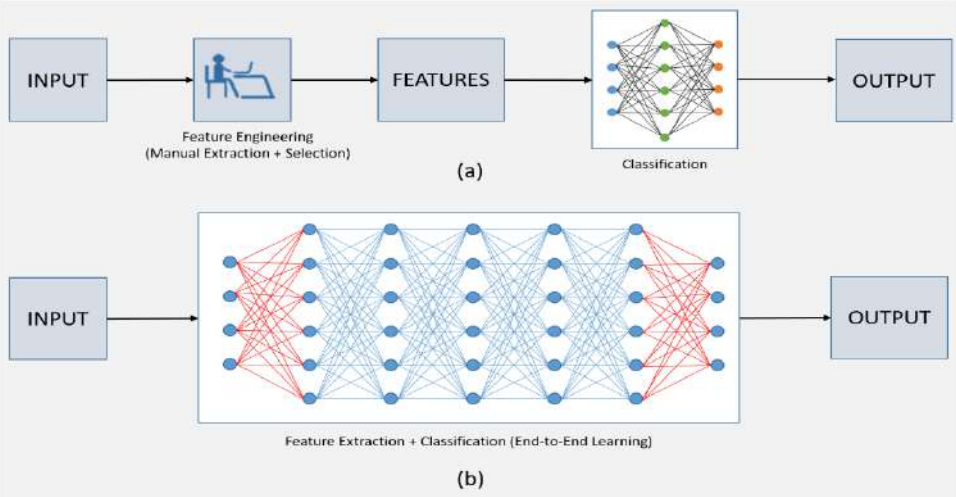

Figure 3.18: (a) Traditional Image processing, (b) Deep Learning processs[31]

Deep Learning Approach Workflow

Deep learning models have emerged as a powerful and transformative approach to image processing and analysis. Unlike traditional methods, which rely on manual feature extraction and handcrafted rules, deep learning models take a data-driven approach and learn to automatically extract relevant features directly from raw input images. This end-to-end learning process enables them to capture intricate patterns and representations, making them highly effective in tasks like image classification, object detection, segmentation, and more. Through convolutional layers, activation functions, pooling, and fully connected layers, deep learning models can learn hierarchical representations from the data, empowering them to generalize well to diverse and complex real-world scenarios.

Their adaptability and ability to handle large-scale datasets have led to state-of-the-art performance in various computer vision applications. While deep learning models may require more labeled data and computational resources for training, their superior performance often surpasses traditional image processing methods, making them a preferred choice in modern image analysis tasks. As technology continues to evolve, the dominance of deep learning in computer vision research and applications is expected to grow, revolutionizing how we interact with images and enabling advancements in various fields [31].

3.9 Model Optimization for Hardware Deployment and Inference

NVIDIA's TensorRT is a high-performance deep learning inference optimizer and runtime framework. It is intended to optimize and speed up the deployment of deep neural networks on NVIDIA GPUs, such as those present in the Jetson Xavier platform. TensorRT provides a range of optimizations and optimizations to maximize inference performance while maintaining accuracy [10]. Figure 3.12 shows the TensorRT optimization diagram.

Precision Calibration: TensorRT employs techniques like dynamic range calibration and layer-wise precision calibration to quantize models to lower precision, such as INT8 while maintaining accuracy. This reduces memory usage and increases computational throughput [10]. Layer and Tensor Fusion: TensorRT optimizes the computational graph by fusing layers and operations together, reducing memory bandwidth requirements and improving inference speed.

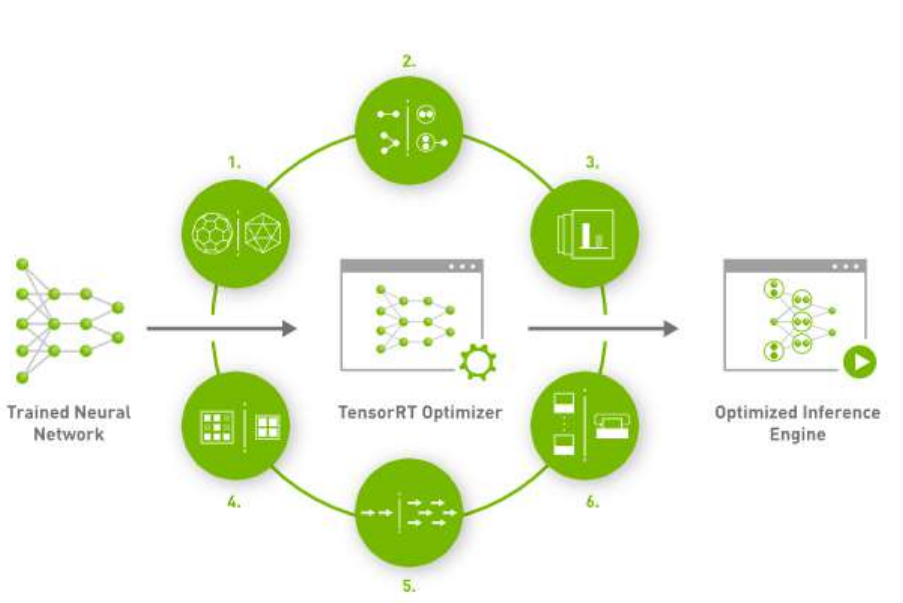

Figure 3.19: TensorRT Conversion [10]

Kernel Auto-Tuning: TensorRT automatically tunes convolution and matrix multiplication operations for the target GPU architecture, optimizing performance for the specific hardware.

Dynamic Tensor Memory: TensorRT optimizes memory allocation by reusing memory across layers, reducing memory consumption, and improving inference throughput[10].

Multi-stream Execution: TensorRT supports multi-stream execution, allowing concurrent inference on multiple streams of data. This enables parallel processing of multiple input data streams, which can significantly improve throughput in scenarios where multiple inputs need to be processed simultaneously [10].

Time Fusion: TensorRT uses dynamically generated kernels to optimize recurrent neural networks over time steps. It fuses the operations performed at each time step into a single layer, reducing the overhead associated with executing recurrent computations in a step-by-step manner [10].

4 State of the Art

4.1 Literature Review

The main objective of the thesis is to develop two different computer vision algorithms, the literature survey is similarly separated into two sections: classical image analysis techniques and deep learning approaches. The overall literature review is covered in this section.

Classical Image analysis based approaches

Traditional textile texture patterns or structures based on the spatial distribution of gray values, such as morphological operations, bi-level thresholding, and edge detection, were analyzed by early low-level feature-based algorithms to identify flaws. However, these methods typically fall short when identifying color defects in textiles with uneven texture or discerning fuzzy flaws that do not change the image [18].

Peng et al. [33] suggested a quick and effective approach for finding fabric defects such as holes, stains, wrinkles, hook wire, and broken weft. The proposed method by this author for feature point recognition, contour identification, and grayscale integration employs the Blob, Canny, and Rotating Integral algorithms. The author was able to achieve a 98 percent accuracy rate using this strategy, demonstrating the effectiveness of these strategies. The main issue with this approach is that the author employed category-specific methods to identify these three categories of flaws, which implies that these three sorts of defects were identified in three different ways.

Mak et al [28] have suggested a special method for inspecting woven fabrics that uses morphological filters (MF) to identify defects. The pre-trained GWN (Gabor wavelet network) used in this proposed technique has been used to extract fabric features from the textile, and these features have then been used in the structuring element to perform additional morphological operations to distinguish the defects and to remove the background. The usage of morphological operations has been made due to the prior research study's conclusion that MF can lower the false alarm rate. 78 images obtained from the Manual of Standard Fabric Defects in the Textile Industry which contain the fabric error like the soiled end, warp float, burl, knot, harness breakdown, foreign fiber, a big knot, gout, etc. which contain many fabric defects, extensive offline experiments have been conducted to evaluate the proposed method. The following evaluation parameters are measured: true detection (TD), false alarm (FA), and missed detection (MD). Additionally, this technique has been tested for real-time analysis on a prototype defect inspection system (which is a low-cost system that has been developed), where 276 image frames from an experiment using twill weaving fabric with numerous defects were used and examined. The proposed method had a 91 percent detection rate [28].

A visual saliency-based approach for flaw detection in both patterned and plain fabrics was proposed by Li et al. in Li2019fabric. To find fabric errors, saliency maps' computed features are employed. The technique builds saliency maps in the first phase to distinguish

between regions with defects and regions without defects. In the second stage, saliency histogram features are extracted and chosen to help distinguish between fabric photos with faults and those without them. Last but not least, classification is carried out using a binary support vector machine (SVM) that was trained on samples of both defective and undamaged cloth. The suggested method can be used for identifying errors in both patterned and plain fabric images, and it produced accurate detection when compared to state-of-the-art research [24]. Although this method outperformed all others and provided an accuracy of 95.5%, histogram-based algorithms are noise sensitive and have a low detection rate in non-regular textures.

Deep Learning-based Approaches

Wei et al [47] recommended a faster regional-based convolutional neural network (faster RCNN) based on the VGG net topology. While employing the fabric defect benchmark, the network is adjusted to improve performance. To lessen the effect of input data, the fabric dataset is enlarged. To extract the picture feature map, the ReLU activation function is combined with pooling and convolution layers. To compute proposals, a region proposal network (RPN) is employed to generate bounding box regression and foreground anchors. Finally, the computed suggestions are given to the ROI pooling layers so that the network of the softmax layer may classify the images. Vertical and horizontal flips are utilized to supplement the textile dataset to minimize overfitting. Fabric flaws discovered include broken pick, felter, drawback, sundries, broken ends, and oil stains. To examine the suggested approach, the authors built their own dataset with flaws. The dataset developed consists of 06 classes, each with 135 fabric flaws. The suggested research is contrasted to established approaches to computer vision, and experimental findings demonstrate the effectiveness of the proposed quicker RCNN model the Detection accuracy achieved by this model is 95.8%. One disadvantage of the suggested quicker RCNN model is its high training cost.

The main limitations of using deep learning-based techniques, according to Liu et al [25] are the high computing cost and training time. The applications of deep learning-based models for embedded and portable devices are constrained by these drawbacks. these authors suggested a CNN-based framework called YOLO-LFD that uses a reduction in feature dimensions to lower the computational cost. Ground truths and initial anchor boxes are computed using K-means dimension reduction. The proposed study offers a compression of the YOLO-v2 model from 241MB to 8MB with a 2.6x multiplicative boost in detection speed. In this study, the deep network is retrained using the COCO dataset, and the network is then fine-tuned using a fabric defect dataset, which addresses the issue of insufficient training samples. The suggested research is evaluated using fabric image benchmarks with 3000 samples and five classes such as holes, surface debris, oil stains, longitude, and weft breakage.1000 images were chosen for testing while 2000 were used for training. the results are compared with SSD, YOLO-v3, and Tiny-Yolo-v3 models using the same environment and parameters. With regard to input parameters and defect detection time and also proposed method also produced 97.2% accuracy. the proposed research has excelled in the current state of the art. It will be safer to draw the conclusion that the proposed research can be employed in an embedded device to identify fabric faults in the industry based on the experimental results [25].

A method for defect classification utilizing CNN with compression sensing based on tiny sample sizes was proposed by Wei et al. Wei et al claim that despite using tiny sample

sizes, CNN has yet to produce accurate classification results; for this reason, CNN is employed in conjunction with the compressive sensing approach. The experiment uses the Fabric Defect Image (FDI 500) dataset, whose images were collected from industrial monitors at textile manufacturers. There are normal, slub, oil stain, broken end, felter, double flat, and mispick photos among the sample images. The operating system Ubuntu 14.04 with TensorFlow 0.8.0, 128 GB RAM, and 4 NVIDIA GeForce 1080 GPUs are used for the method's implementation and assessment. Calculated classification accuracy has been compared against a wide range of other models. The proposed model's claimed classification accuracy is 97.9% [47].

A novel approach for the detection of faults based on a sophisticated deep-learning approach was developed by Jeyaraj et al [21] According to these authors, the suggested method also localizes tiny faults. The proposed method uses a ResNet512-based CNN to learn the features from images. An i7 processor with an 8GHz (frequency), 1TB hard drive, and NVIDIA GPU are used for the research. Numerous quantitative metrics are generated in experiments using the TILDA textile dataset. The method's accuracy is calculated and validated using classification accuracy, and the findings have been compared to those of other classifiers, including SVM and Bayesian. Missing yarn, Slub, Dirt, Foreign yarn, and Pattern defect comprise the dataset for this method. The accuracy of the proposed method was measured, and the process yielded about 96.5% accuracy [21].

In order to preprocess the most likely defective patches from the original image, Wen et al. introduced a CNN-based approach patch extractor triple matrices (PETM), which is a self-similar estimate algorithm, for the detection of minor defects. A collection of 100 no-defective and 100 minor defective images called the minor fabric defect images (MFDI) dataset is created. It uses a patched size of 4040 with Np=10 and n=4. 901 normal patches and 99 defective patches are extracted from 100 defective fabric images. The assessment parameters utilized are the true-positive rate (TPR), true-negative rate (TNR), and accuracy (Acc). Accuracy has been evaluated with a few other approaches. Each component of the proposed method has a reported accuracy of 79.32 percent [48].

The Mobile-Unet approach based on CNN was proposed by Jin et al. for the detection of fabric defects. The suggested approach is effective because it achieves pixel-level defect categorization and works best for online FDD. The median frequency balancing loss function is used to enhance the detection process and accuracy. This process produced the 92% of accuracy. A platform for this study includes a Windows 10 OS, Intel Core i7-7700K 4.2GHz, 128GB RAM, NVIDIA GeForce 1080Ti, and PyTorch deep learning framework. The suggested approach is trained and tested using the benchmarks Fabric Images (FI) and Yarn-dyed Fabric Images (YFI). As evaluation matrices, recall, precision, F1-measure, and intersection over union (IoU) are employed [22].

An enhanced YOLOv5 object detection technique for fabric defects is proposed by wang at al. to address the issues of uneven forms and numerous little objects. The Adaptively Spatial Feature Fusion feature fusion method is used to mitigate PANet's negative impact on multi-scale feature fusion in order to increase the detection accuracy of small objects. The network can concentrate on valuable information by using transformer techniques to improve fused features. According to the experimental results, the improved YOLOv5 object detection algorithm's mean average precision for detecting fabric defect maps is 71.70%. The enhanced method can quickly and precisely increase the accuracy of defect localization and fabric defect detection [46]

.

4.2 Related Work

This section explains the state of the art of fabric defect detection, which covers the significance of quality inspection in the fabric industry, as well as traditional and automated quality inspection of fabric defect detection and their influence.

Fabric defect detection and their impact on the textile quality

As described in the introduction chapter, fabric companies rely heavily on quality inspection. Quality inspection refers to the process of inspecting fabrics to ensure that they fulfill specified quality standards and specifications. This inspection is often performed at several stages of production, such as incoming raw materials, intermediate manufacturing stages, and finished goods, to identify any errors, flaws, or deviations from the specified requirements [11]. The Impact of the Quality Inspection in fabric industry:

1. Quality inspection assists in the identification of defects such as fabric flaws, irregularities, color variations, stains, holes, or weaving inconsistencies. By detecting these problems producers can take corrective efforts to avoid the creation of faulty fabrics. minimizing waste and improving overall quality [1].

2. Quality inspection can assure color, texture, weight, aesthetic consistency, and uniformity. This is especially crucial for making fabrics for uses requiring uniformity, such as garments or household textiles[1].

3. Quality inspections help ensure that fabrics satisfy industry standards, regulations, or the needs of customers. Fabric strength, shrinkage, colorfastness, and other physical and chemical qualities are all considered. These requirements must be followed in order to meet consumer expectations[1].

4. Quality checks assist in finding and correcting defects early in the manufacturing process. This decreases the possibility of mass-producing substandard fabrics, which can be costly in terms of raw material waste, and rework. This can save costs and increase profitability.

5. Fabric manufacturers can avoid shipping defective or damaged fabrics to consumers by undertaking quality inspections. Providing high-quality items increases customer happiness, develops trust, and improves long-term connections with consumers [1].

Traditional fabric Quality inspection techniques

Conventional methods and procedures used in the textile industry to inspect the quality of fabrics are referred to as traditional fabric quality inspection. These procedures typically entail manual inspection by skilled workers who use visual inspection methods and simple measuring equipment. The detection of manual fabric defects [11] can be done using a variety of procedures, including visual inspection, manual measuring techniques, sampling techniques, manual sorting or grading, etc. Traditional cloth defect identification is seen in Figure 4. as skilled workers identify the errors.

There are some advantages and disadvantages of the traditional quality inspection. they discussed below.


Figure 4.1: Traditional Fabric Quality inspection [2]

Advantages of the traditional quality inspection techniques

1. Familiarity: Industry professionals have become used to and knowledgeable about traditional inspection techniques. They have been in use for several years, and fabric producers and inspectors are familiar with the defects.

2. Flexibility: Different fabric types, sizes, and production phases can be accommodated by traditional inspection techniques. They enable customization and flexibility in fabric inspection based on particular needs or client preferences.

3. Assessment based on knowledge: Manual inspection relies on the knowledge and skills of qualified inspectors who can visually spot flaws and quality problems. During the inspection process, their knowledge and abilities help with accurate assessment and decision-making.

Disadvantages of the traditional quality inspection techniques

1. Time-consuming: Traditional inspection techniques can be laborious, especially when producing large amount of quantities. Each fabric piece must be manually examined and measured, which takes a lot of time and can delay production schedules and extend lead times. Under 20 m/min is the traditional detecting speed.

2. Human error: Manual inspection is susceptible to human error, including fatigue, oversight, or distraction. Inspectors may miss certain defects or make mistakes in their assessment, compromising the accuracy and reliability of the inspection process [35].

3. Labor-intensive: It takes a lot of human labor to discover fabric faults using traditional methods. Trained inspectors visually inspect fabrics and manually spot defects. The labor- and time-intensive nature of this manual inspection method can reduce production.

4. Traditional quality inspection has its benefits, but it also has an array of limitations that can be overcome by using computer vision-based methods.

Computer vision-based Automated quality inspection

Automated fabric defect detection is the technique of automatically identifying and categorizing fabric flaws using cutting-edge technology including computer vision, machine learning, and image processing algorithms. These methods examine fabric images and find deviations from the specified quality requirements using digital imaging equipment, cameras, and specialized software [2].

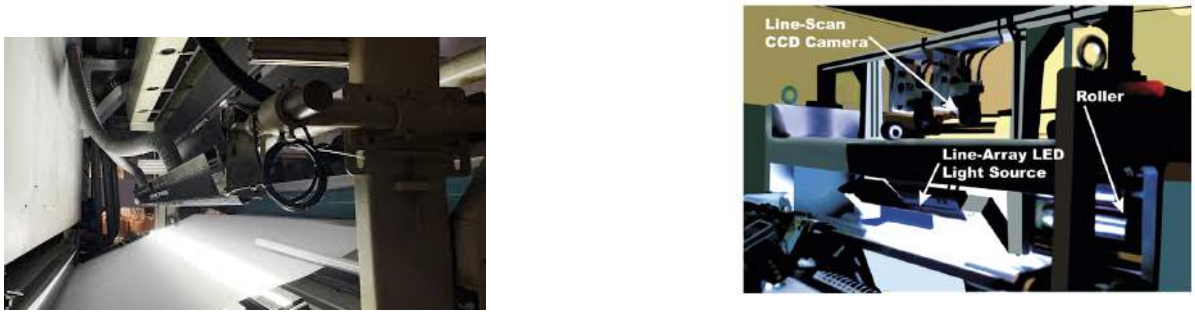

Figure 4.2: Computer-vision-based automated fabric Quality inspection [37]

Advantages of Automated quality inspection

1. Efficiency: Automated systems for detecting fabric defects process are quick and accurate. When compared to manual procedures, they can quickly examine huge quantities of fabrics, thus lowering inspection time. This accelerated pace improves output efficiency and facilitates quicker decision-making[2].

2. Accuracy and Consistency: Computer vision-based methods for flaw detection provide higher accuracy and consistency. In order to learn patterns, textures, and defect characteristics, algorithms can be trained on a big dataset of fabric photographs. As a result, fewer false positives and negatives can occur during the accurate identification and classification of problems.

3. Sensitivity to Defects: Computer vision algorithms are capable of spotting little flaws that can be difficult to spot with the eyes alone. They have the ability to detect even tiny differences in color, pattern, or texture in fabric images at the pixel level. This improved sensitivity improves the ability to find defects, which raises the bar for quality.

4. Data Logging and Analytics: Effective data logging and analytics are made possible by computer vision-based methods. The systems' ability to store and analyze inspection data offers useful insights into fault patterns, trends, and potential for process improvement. Predictive maintenance and quality control optimization can both benefit from this information.

5. Cost-effectiveness: Automated systems for detecting fabric defects may require a larger initial investment than conventional ones, but they can save money in the long run. Defect detection has increased accuracy and efficiency, which lowers waste, rework, and customer refunds. Furthermore, automation reduces the requirement for sizable personnel devoted to human examination [2].

5 Design and Methodology

The master thesis' overall flow, from dataset preparation through deployment onto an embedded system, is explained in this chapter. As mentioned in the objectives, this master's thesis's algorithm development is based on two approaches: a deep learning-based algorithm that uses the YoloV5 object detection-based algorithm and a classical image analysis-based algorithm that uses blob detection. Both of these concepts and their theoretical basis are discussed here. This also explains the concept of hardware deployment and its theory details. Additionally, this helps in describing programming languages, libraries, etc.

5.1 Overview

Figure 5.1 illustrates the master thesis's overview as a block diagram. The following subsections describe the steps involved in the development of this master's thesis.

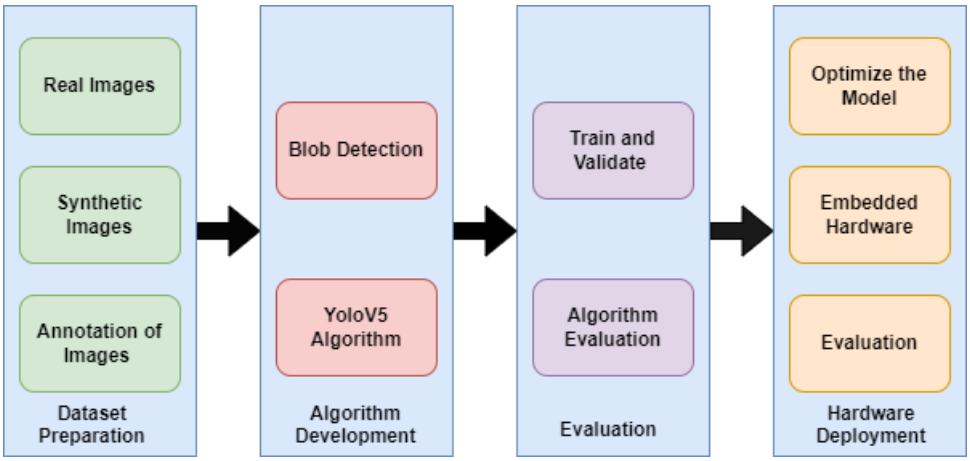

Figure 5.1: Overview Block Diagram

Dataset Preparation: The preparation of datasets is an essential phase in the creation of computer vision algorithms. This master's thesis focuses on collecting real images of fabric defects and creating synthetic images from these real images. Annotating is another important step in the algorithm development.

Algorithm Development: After the preparation of both the real and synthetic image datasets, the subsequent step involves algorithm development. In this study, the classical image analysis-based algorithm known as blob detection, as well as the deep learning-based algorithm YOLOv5 object detection algorithm, are selected. These algorithms have been chosen based on previous research findings documented in the literature.

Evaluation: The next step in this process is the evaluation of both the developed algorithms. The evaluation metrics of this algorithm are Precision, Recall, and Accuracy for the blob detection algorithm and for YOLOv5 Precision, Recall, and mAP_0.5 for the YOLOv5 algorithm.

Hardware Deployment: The final step in this master thesis is to deploy the models into the targeted hardware system. The embedded system deployment can be done after optimizing the model into tensor rt, which can be deployed into Jetson Xavier NX. After testing the model evaluation can be performed to find the metrics called picture-to-picture interference time detection.

5.2 Dataset

There are various colors of clothing, such as white and black, and various types of materials, such as silk, velvet, cotton, and so on, available in the fabric sector. In addition, there are so many errors present in the fabric industry such as holes, stains, slubs, horizontal lines, broken picks, wrong fiber, etc [33].

The dataset preparation decision and reasons for this thesis are presented in the table below.

Given	**Decision**	**Reasons**
The textile of different materials and color	Focus on one type of white cloth	• High contrast • A high percentage of production throughput
Error catalog of 20+ error types	Focus on 3 production-caused errors Horizontal, similar size, and high frequency of appearance.	• High importance for the implementation. • Easier synthetic data attainable.

Table 5.1: Data Collection Criteria

5.2.1 Real Image Dataset:

The Real images are the images that are captured from the real-time fabric manufacturing industry, These images are very difficult to collect from the industries, The real images available for this thesis are only 10-20 with three different error types, The error types are discussed in the next section. The main purpose of these real images is to test the developed algorithms and find the defects of the fabrics. Figure 5.2 is a real defect image which is obtained from the industry.

Figure 5.2: Original Fabric Defect Image

5.2.2 Synthetic Image Dataset:

As mentioned in the above section obtaining a large number of real images is extremely difficult, particularly of some rare types, which results in data imbalance or even complete failure of traditional supervised methods. To deal with these challenges, numerous researchers have made substantial efforts. In this research, the 3D Synthetic images are generated using HOUDINI software and these images are generated based on the original image which is used as an Input dataset for Model training, validation, and testing. Figure 5.3 is a replica of the real defect image which is synthetic.

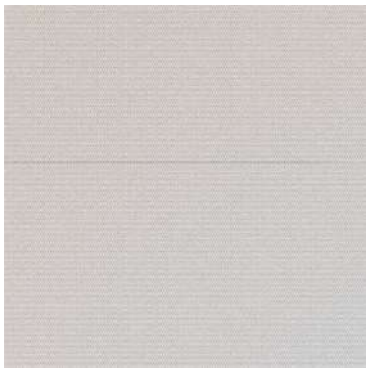

Figure 5.3: Synthetic Fabric Defect Image

5.2.3 Error Types

As mentioned in Table 5.1, There are 3 error types that are selected in this thesis which are similar in shape and size. The three errors are explained below:

Foreign Fiber of Wrong Color

Foreign Fiber Colored refers to the presence of wrong-colored foreign fibers within the fabric. These foreign fibers may be of a different color, composition, or origin than the surrounding fabric. Detecting Foreign fiber colors is essential in ensuring the quality and uniformity of the fabric, as the presence of foreign fibers can affect the appearance, durability, and performance of the fabric. Foreign Fiber of Wrong Color is an English word that translates in German as Fermdfaser falscher Farbe. this error will be denoted as class "FF".

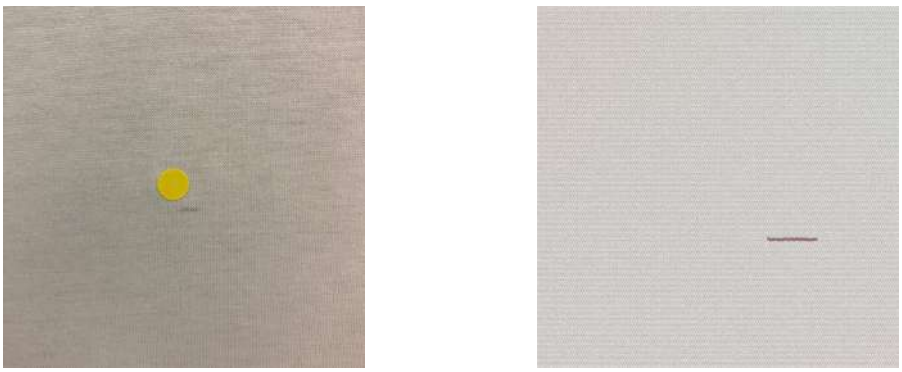

Figure 5.4: Real and Synthetic images of Foreign Fiber of Wrong Color

Thick Yarn

In fabric defect detection, Yarn thickening refers to an irregular thickening of the yarn used in the fabric. It occurs when certain sections of the yarn become thicker than the rest, resulting in an uneven appearance and potential quality issues in the fabric. Detecting and identifying Yarn thickening is important in quality control processes to ensure that the fabric meets the required standards and specifications. The thick yarn in fabric defect detection is commonly referred also thick places or thickened yarns. Thick yarn is an English term that translates to Garnverdickung in German, This error is denoted as the class "DG".

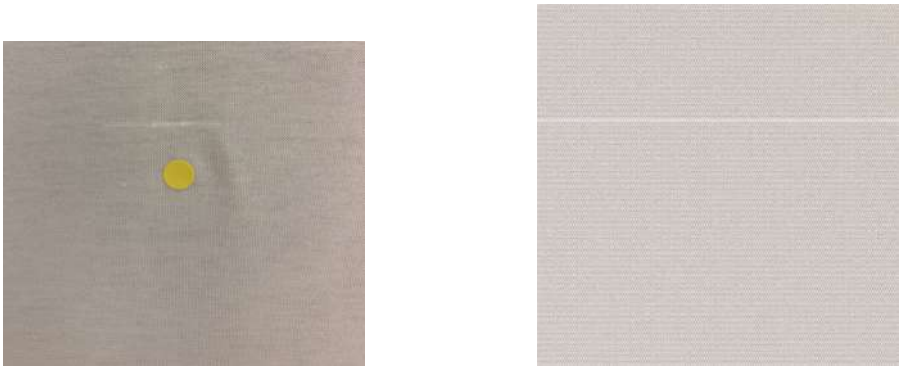

Figure 5.5: Real and Synthetic images of Thick Yarn

Elastane Breakage

In fabric defect detection, Elastane breakage refers to the occurrence of breakages or tears in the elastane fibers present in the fabric. Elastane breakage indicates that the elastane fibers within the fabric have experienced damage, resulting in visible breaks or tears. This defect can significantly affect the functionality and appearance of the fabric, as it compromises the fabric's stretch and recovery properties. Elastane breakage is an English term that translates into German as Elastaneplatzer. This error can be denoted as class "ELA".

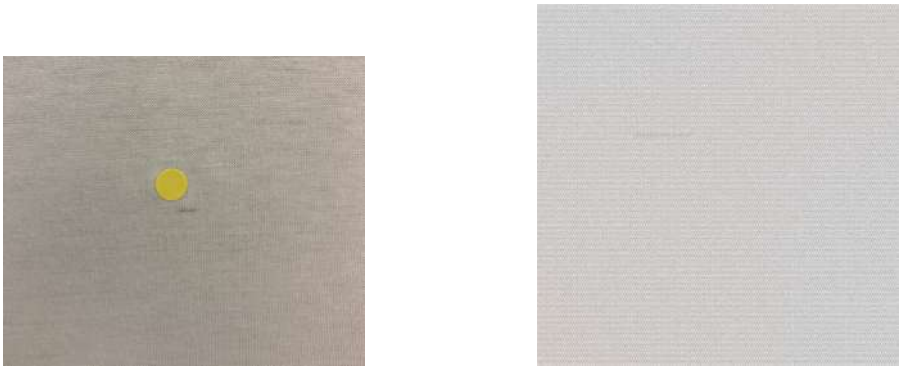

Figure 5.6: Real and Synthetic images of Elastane Breakage

5.2.4 Dataset annotation

The practice of labeling data with appropriate tags to make it easier for computers to comprehend and analyze is known as data annotation. This information can take the shape of photos, text, audio, or video, and data annotators must categorize it as precisely as possible. Annotating data can be done manually or automatically utilizing powerful machine learning methods and tools. There are several types of image annotation methods[5] they are image classification, object recognition/detection, semantic segmentation, and instance segmentation. for this thesis image recognition or object detection model of annotation is used.

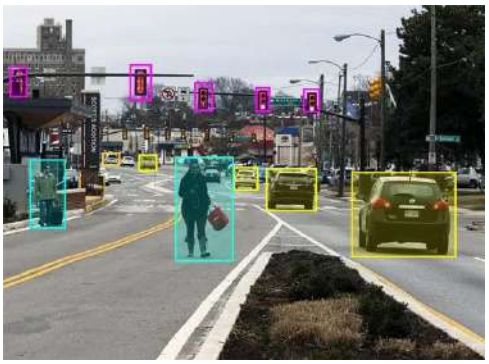

Figure 5.7: Sample object detection annotation

Object recognition/detection is a further version of image classification[6]. It is the proper description of the image's numbers and exact placements. Unlike image classification, which assigns a label to the whole image, object recognition assigns a label to each

object in the image. labels entities separately. For example, with image classification, the image is labeled as defect fabric and defect-free fabric. whereas in object detection it will tags individually various error types of the fabric along with the location of the error on the fabric.

Advantages of the dataset annotation

- By providing labeled data, annotations enable algorithms to learn patterns, relationships, and correlations between input features and desired outputs. This labeled data acts as a training set, allowing the model to generalize and make accurate predictions or classifications on new, unseen data [5].
- Dataset annotation plays a vital role in evaluating and validating the performance of computer vision models. Annotated datasets provide ground truth labels that serve as a benchmark for comparing the model's predictions. By measuring the accuracy, precision, recall, or other evaluation metrics against the annotations[5].

5.3 Concept and Methodology

This segment explains into the comprehensive scientific understanding of both the Blob detection and YOLOv5 algorithms

5.3.1 Blob Detection Algorithm

Blob detection techniques in computer vision are used to identify regions of a digital image that have different characteristics from the surrounding areas, such as brightness or color. A blob is an informal term for a section of a picture where certain qualities are constant or almost constant; all the points in a blob can be thought of as being somewhat similar to one another [3]. Blobs can be seen as a region of pixels in an image that shares some common property and is thus grouped together[14].

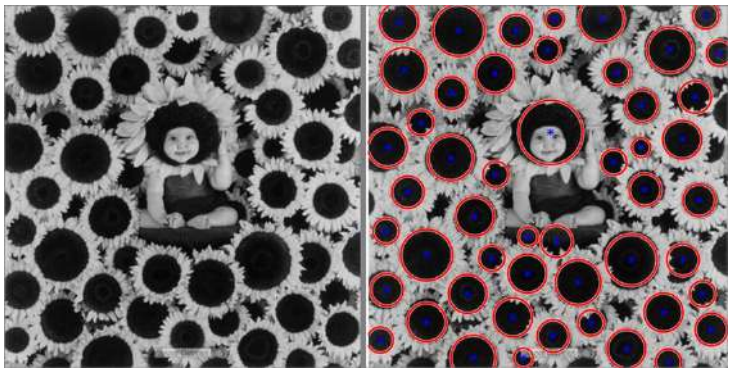

Figure 5.8: Blob Detection [34]

Parameters:

The Blobs can be filtered by different parameters which are explained below.

Thresholding: In image processing, a threshold is a critical level that segregates pixels based on intensity. Pixels with intensity levels above or below the threshold are classified differently. This is a common method used in image binarization where all pixels below a certain threshold are turned black and all pixels above the threshold are turned white. In blob detection, the threshold parameter can be used to control the minimum contrast required for detecting blobs [4].

$$I'(x,y) = \begin{cases} 1 & \text{if } I(x,y) > T, \\ 0 & \text{otherwise.} \end{cases} \tag{5.1}$$

Here, I(x,y) is the intensity of the pixel at location (x,y), T is the threshold, and I'(x,y) is the resulting binary image. **Color:** or color images, blobs can be detected based on color channels. For example, blobs with similar colors can be grouped together. Blob detection can also be performed based on the intensity of the blobs.

Area: In blob detection methods can filter detected blobs based on their area. This means the algorithm will ignore blobs that are smaller or larger than the specified area. The area of a blob is often defined as the number of pixels within that blob. This is

useful when blobs of interest have a specific size range, and you want to ignore smaller or larger blobs.

Circularity: Circularity or roundness is a metric that describes how close a blob is to a perfect circle. It's defined as

$$Circularity = \frac{4\pi \times Area}{Perimeter^2} \tag{5.2}$$

Perfectly circular blobs have a circularity of 1, while elongated or irregular blobs have a circularity of less than 1 [4].

Inertia Ratio: inertia (also referred to as the second moment or moment of inertia) provides information about the spatial distribution of pixel intensity in the blob, revealing the blob's shape and orientation. The resistance of an item to changes in rotation is measured by its inertia. In image processing, the concept of inertia is used to describe the distribution of the pixels in a blob around its center of mass. High inertia values typically indicate elongated blobs, while low inertia values may suggest round or regularly shaped blobs [4].

If f(x, y) is the intensity of a pixel at position (x, y) and (x_c, y_c) is the center of mass of the blob, then the second-order moments of inertia Ixx, Iyy, and Ixy are often defined as follows:

$$I_{xx} = \sum f(x,y) \cdot (x - x_c)^2 \tag{5.3}$$

$$I_{yy} = \sum f(x,y) \cdot (y - y_c)^2 \tag{5.4}$$

$$I_{xy} = \sum f(x,y) \cdot (x - x_c) \cdot (y - y_c) \tag{5.5}$$

The inertia tensor for the blob can then be defined as a matrix using these moments:

$$I = \begin{bmatrix} I_{xx} & I_{xy} \\ I_{xy} & I_{yy} \end{bmatrix} \tag{5.6}$$

Convexity: It is defined as the blob's area divided by the area of its convex hull. A blob's convex hull is the smallest convex polygon that encloses all of its points (or pixels). Convexity is expressed as a number ranging from 0 to 1. A perfect circle or a rectangle would have a convexity of 1, as for these shapes, the area and the convex hull are the same. In contrast, an irregularly shaped blob or a blob with indentations would have a convexity value of less than 1 [4].

If A represents the area of the blob and A_hull represents the area of the blob's convex hull, then the convexity C is defined as:

$$C = \frac{A}{A_{\text{hull}}} \tag{5.7}$$

By Scale: The scale parameter is often associated with scale-space theory, which represents an image as a one-parameter family of smoothed images, with the parameter being the scale or width of the smoothing kernel. In blob detection, it refers to the standard deviation of the Gaussian that is convolved with the image. Different scales can display blobs of various sizes. Scale selection is a critical aspect of the Difference of Gaussians (DoG), Laplacian of Gaussian (LoG), and Determinant of Hessian (DoH) blob detection methods. In blob detection algorithms like DoG, LoG, or DoH, a Gaussian kernel with standard deviation (the scale parameter) is convolved with the image. The equation for a 2D Gaussian is shown below:

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right) \tag{5.8}$$

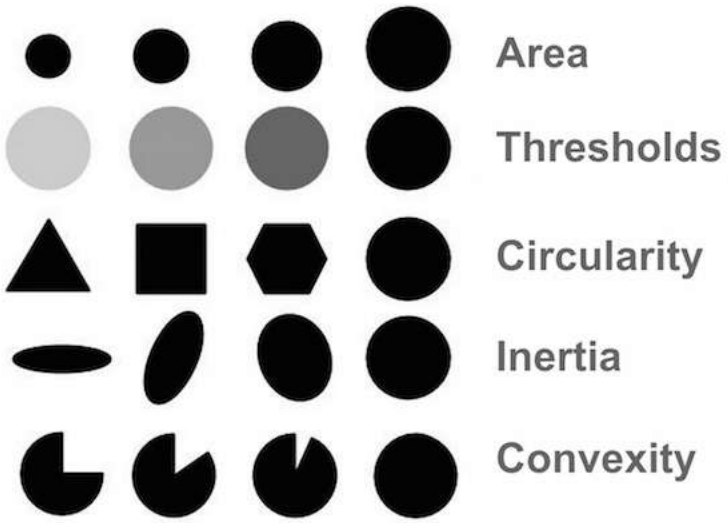

Figure 5.9: Blob Detection Parameters [4]

Contour Detection

Contours can be defined as curves that connect all continuous boundary points with the same color or intensity. They play a crucial role in the analysis of shapes and in the detection and recognition of objects. [23] The fundamental principle behind contour detection is that within an object, the color or intensity stays relatively constant, but at the edges or boundaries of the object, a sharp change or gradient in color or intensity occurs. So, by looking for these sharp changes, you can detect the contours of objects. Once the contours are detected, they can be used for a variety of applications like object detection, object recognition, and shape analysis. In image processing and computer vision, contours are generally understood as a set of points that form a curve outlining an object within an image. These points are often derived from the image's pixel data, typically based on discontinuities in pixel intensity of grayscale value or color.

Contour approximation Methods

Contour approximation is a technique used in computer vision to reduce the number of points in a detected contour while still preserving the overall shape. The goal is to simplify the contour so it's easier to process without losing the essential structure [15].

1. Douglas-Peucker Algorithm: The Ramer-Douglas-Peucker algorithm, commonly known as the Douglas-Peucker algorithm, is a method for reducing the number of points in a series-of-points-approximate curve. It does this by "simplifying" the curve, lowering the amount of memory necessary to store the curve as well as the amount of computation required to process it. The technique is well suited for vector graphics and geographic data processing.

2. Chain Approximation Methods: These methods are usually used for representing contours in a more compact manner. There are several types of chain approximation methods

   - CHAIN_APPROX_NONE: This method stores all the contour points That is, any two subsequent points in the contour will be either horizontal, vertical, or diagonal neighbors [15].

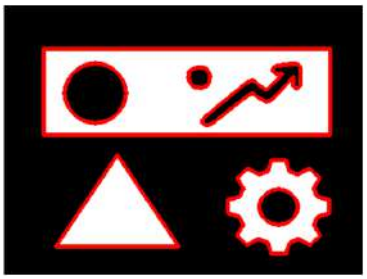

Figure 5.10: None Approximation [15]

   - CHAIN_APPROX_SIMPLE: This method compresses the contour by removing redundant points and leaving only the endpoints of straight-line segments. For example, suppose you have a straight line at all angles (not just horizontal, vertical, or diagonal). In that case, it doesn't store all the contour points but retains only the endpoints of the line.

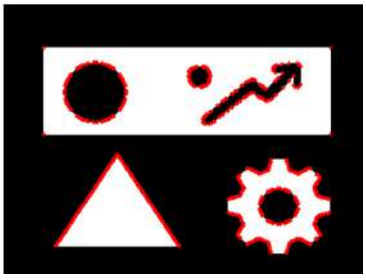

Figure 5.11: Simple Approximation [15]

- CHAIN_APPROX_TC89_L1: This version of the Teh-Chin chain approximation algorithm uses L1 metrics in its calculations. In essence, the L1 metric, also known as Manhattan distance, calculates the distance between two points as the sum of the absolute differences of their coordinates. For example, the L1 distance between two points in a 2D plane would be $d_{L1} = |x_1 - x_2| + |y_1 - y_2|$. This metric effectively calculates the shortest path when only orthogonal (vertical and horizontal) movements are allowed [**?**].

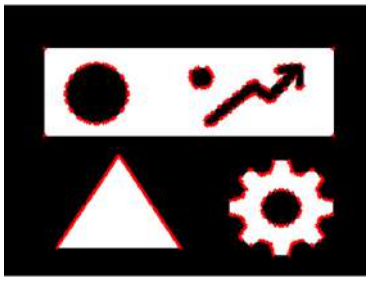

Figure 5.12: TC89_L1 Approximation [15]

- CHAIN_APPROX_TC89_KCOS: This method also uses the Teh-Chin chain approximation algorithm, but instead of L1, it uses kernel cosine (KCOS) as its metrics. Without diving deep into the math, the KCOS metric uses the cosine of the angle between two vectors and applies a kernel function to weigh these vectors. This can create a smoother approximation of contours compared to L1.

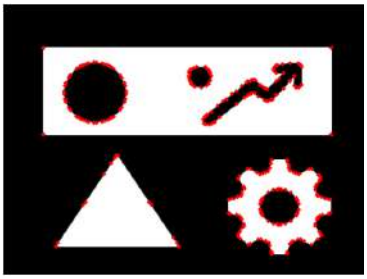

Figure 5.13: TC89_KCOS Approximation [15]

5.3.2 YOLOv5 Algorithm

Yolo is a cutting-edge, real-time object detector, and YOLOv5 is built on YOLOv1-YOLOv4. It outperformed on two official object detection datasets: Pascal VOC (visual object classes) [32] and Microsoft COCO (common objects in context). Yolo is well-known for its speed and accuracy, and it has been used in a wide range of applications like as video surveillance, self-driving cars, augmented reality, and so on. The YOLO version 5 is employed in this master thesis. There are Five models which are available in YOLOv5. They are nano (YOLOv5n), small(YOLOv5s), medium(YOLOv5m), large(YOLOv5l), and extra large(YOLOv5x) size models. there is no difference between the five models in terms of architectural operations but there is a difference in the performance and speed which shows in Figures 5.9 and 5.10.

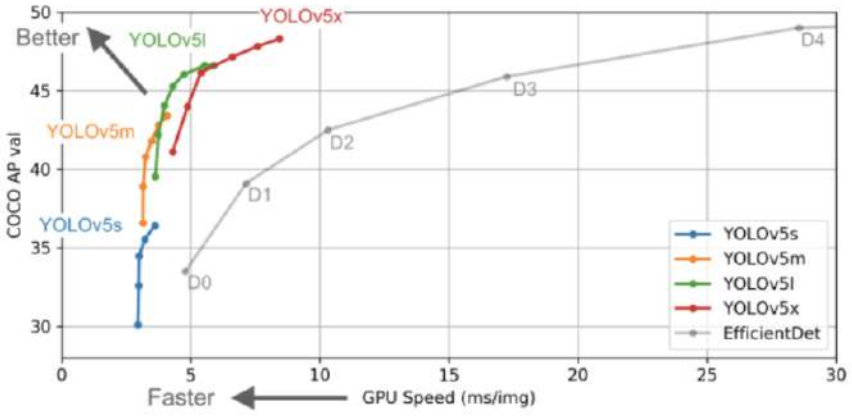

Figure 5.14: Performance of YOLOv5 Different Size Models [44]

Model	AP^{val}	AP^{test}	AP_{50}	$Speed_{GPU}$	FPS_{GPU}	params	FLOPS
YOLOv5s	36.6	36.6	55.8	**2.1ms**	**476**	7.5M	13.2B
YOLOv5m	43.4	43.4	62.4	3.0ms	333	21.8M	39.4B
YOLOv5l	46.6	46.7	65.4	3.9ms	256	47.8M	88.1B
YOLOv5x	**48.4**	**48.4**	**66.9**	6.1ms	164	89.0M	166.4B

Figure 5.15: YOLOv5 Models Evaluation Metric Comparison [44]

The architecture of all YOLOv5 models can be represented by Figure 5.11, which illustrates the conceptual division of the YOLOv5 algorithms into three main components. These components include the CSP-Darknet53 [29] as the backbone, SPP (Spatial Pyramid Pooling), and PANet (Path Aggregation Network) as the neck and the head which remains consistent with the head used in YOLOv4 [43]. The backbone's primary task is to perform feature extraction by extracting deep-level features from the input data. This process involves capturing relevant and informative characteristics from the input to enable subsequent analysis and prediction. The neck component plays a crucial role in architecture by combining information from various depths. This process is often referred to as feature aggregation, where the features extracted by the backbone are consolidated and integrated from different layers of the network. This aggregation helps in effectively

incorporating multi-scale information and enhancing the model's ability to capture diverse spatial characteristics [38]. Finally, the heads in the YOLOv5 architecture are responsible for making predictions [29]. These heads utilize the aggregated features to generate predictions regarding the presence, location, and classification of objects within the input data. The head component remains consistent with the one used in YOLOv4, ensuring compatibility with the established approach for object detection and recognition. This modular design enables efficient processing and analysis of input data, leading to accurate and robust object detection capabilities.

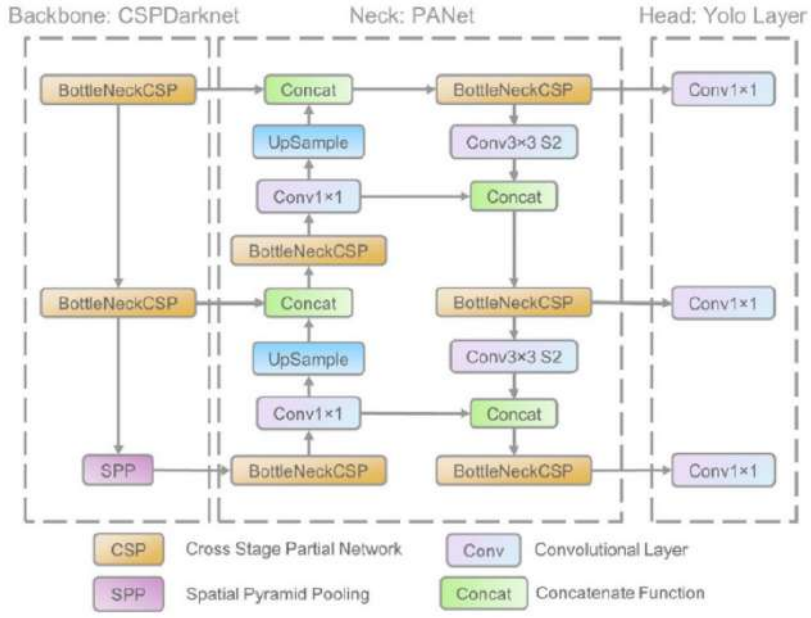

Figure 5.16: YOLOv5 Architecture [45]

Backbone

YOLOv5 adopts CSP-Darknet53 as its backbone, which is based on the convolutional network Darknet53 used in YOLOv3. The authors of YOLOv5 have incorporated the Cross Stage Partial (CSP) network strategy into Darknet53. The CSP network strategy addresses the issue of redundant gradients that can arise when using dense and residual blocks in deep networks. These blocks are effective in enabling information flow to deep layers and mitigating the vanishing gradient problem[45]. However, they can also introduce a significant amount of redundant gradient information. The CSPNet method solves this problem by truncating the gradient flow. YOLOv5 separates the feature map of the base layer into two parts and then combines them via a cross-stage hierarchy using the CSPNet technique. This partitioning and merging process helps preserve the feature reuse characteristics of DenseNet, ResNet while reducing the redundant gradient information[45].

The BottleNeckCSP is an essential component employed in the YOLOv5 architecture. It combines the benefits of both bottleneck and Cross Stage Partial (CSP) strategies to improve the performance and efficiency of the model. The bottleneck structure, derived from the ResNet architecture, is a compact representation that reduces computational

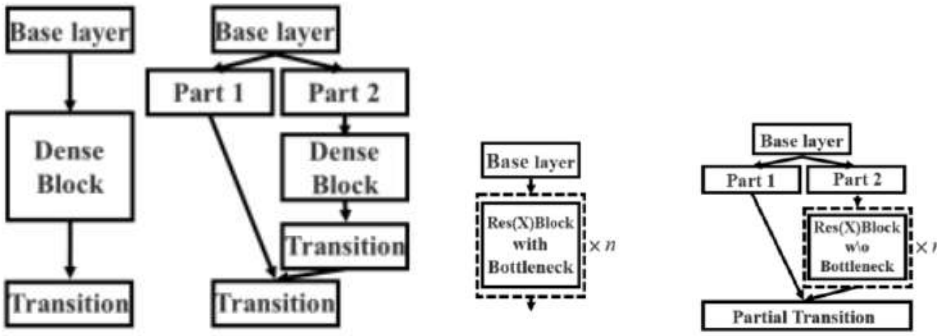

Figure 5.17: Applying CSPNet to DenseNet and ResNet [45]

complexity while preserving the representational power of the network. By using a bottleneck structure, YOLOv5 can efficiently capture and extract essential features in a more compact form.

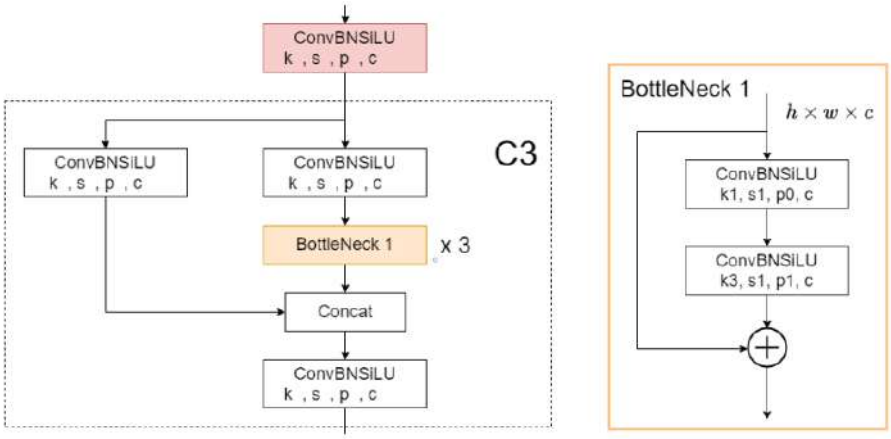

Figure 5.18: BottleNeck CSP Architecture [44]

The incorporation of the CSPNet strategy in YOLOv5 offers significant advantages. It reduces the number of parameters and computational requirements (measured in FLOPS), leading to enhanced inference speed—an important consideration in real-time object detection models. This optimization contributes to the overall efficiency and effectiveness of YOLOv5 for object detection tasks.

Neck

YOLOv5 introduced two significant changes to the model's neck component. The first change involves the utilization of a modified variant of Spatial Pyramid Pooling (SPP), while the second change involves the incorporation of BottleNeckCSP into the architecture of the Path Aggregation Network (PANet). The PANet serves as a feature pyramid network, previously employed in YOLOv4, to enhance information flow and aid in precise pixel localization for mask prediction[50]. In YOLOv5, the PANet has been further enhanced by integrating the CSPNet strategy, as depicted in the network architecture diagram. On the other hand, the SPP block, which aggregates input information and outputs a fixed-length representation, plays a crucial role in increasing the receptive field

and extracting relevant contextual features without compromising network speed. While previous versions of YOLO (YOLOv3 and YOLOv4) employed the SPP block to separate important features from the backbone, YOLOv5 (6.0/6.1) introduces SPPF, a variant of the SPP block designed specifically to improve network speed.

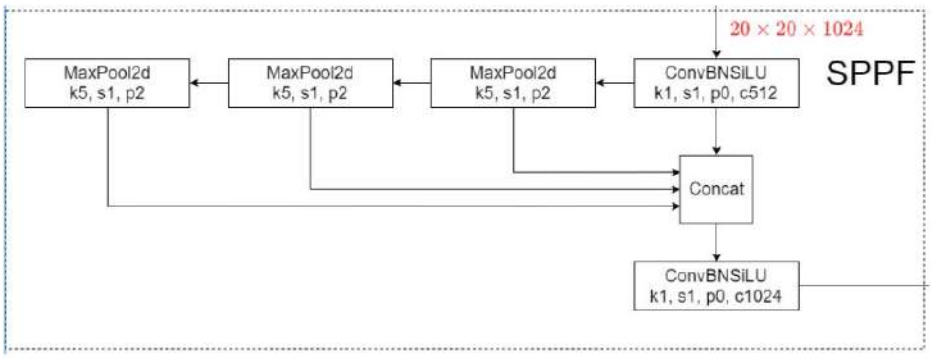

Figure 5.19: YOLOv5 Neck [44]

Head

In YOLOv5, the head component used for bounding box prediction is shared with YOLOv3 and YOLOv4. This head is composed of three convolutional layers and is responsible for predicting the location of the bounding boxes in terms of their x and y coordinates, height, and width. Additionally, the head predicts the object scores, which indicate the likelihood of an object being present in a given bounding box, and the classes of the detected objects. This shared head architecture across YOLOv3, YOLOv4, and YOLOv5 demonstrates a continuity in the design and enables consistency in terms of the bounding box predictions and class predictions among these versions. By utilizing three convolutional layers in the head, the model can capture and process the relevant spatial information necessary for accurate object detection and localization[50]. The equation to compute the coordinate bounding boxes is shown below.

$$b_x = (2 \cdot \sigma(t_x) - 0.5) + c_x \tag{5.9}$$
$$b_y = (2 \cdot \sigma(t_y) - 0.5) + c_y \tag{5.10}$$
$$b_w = p_w \cdot (2 \cdot \sigma(t_w))^2 \tag{5.11}$$
$$b_h = p_h \cdot (2 \cdot \sigma(t_h))^2 \tag{5.12}$$

The YOLOv5 introduced a first-time Focus layer. The Focus layer replaced the first three layers of the YOLOv3 algorithm. Using the focus layer reduces the amount of CUD A memory required, reduces the layer size, and improves forward propagation and backpropagation. Spatial pyramid pooling is a pooling layer that is used to eliminate the network's fixed size limitation [43].

Activation Function

The selection of an appropriate activation function is a critical decision in designing deep learning models, and for YOLOv5, the authors have chosen the SiLU and Sigmoid activation functions. SiLU, which stands for Sigmoid Linear Unit, is also commonly referred to as the swish activation function. In YOLOv5, SiLU is employed in conjunction with convolution operations within the hidden layers of the network. The SiLU function offers advantages such as non-linearity and smoothness, which can aid in capturing complex relationships and gradients during the training process. By employing SiLU and Sigmoid activation functions in YOLOv5, the authors aim to leverage the strengths of each function in different parts of the model. SiLU facilitates the extraction of complex features within the hidden layers, while Sigmoid aids in producing meaningful probability-based predictions in the output layer[44].

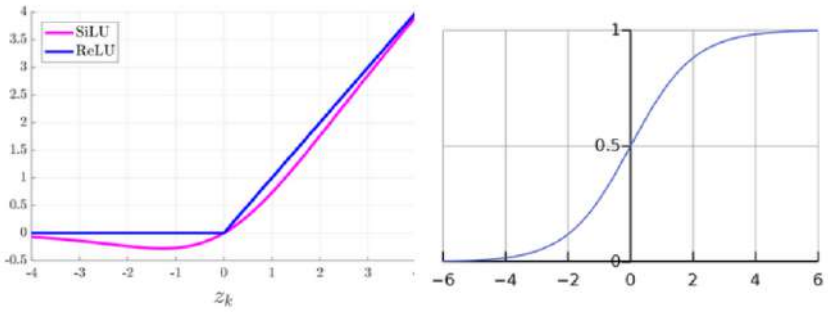

Figure 5.20: Graphs of Activation (a) Silu Function and (B)Sigmoid Function[44]

Loss Function

The YOLOv5 model generates three outputs for each detected object: the predicted classes, the corresponding bounding boxes, and the objectness scores. To compute the losses associated with these outputs, YOLOv5 utilizes different loss functions. For the class loss and the objectness loss, YOLOv5 employs the Binary Cross Entropy (BCE) loss. The BCE loss measures the dissimilarity between the predicted classes and the ground truth classes, as well as the discrepancy between the predicted objectness scores and the ground truth objectness labels. On the other hand, for the location loss, YOLOv5 utilizes the Complete Intersection over Union (CIoU) loss. The CIoU loss computes the difference between the predicted and ground truth bounding boxes, taking into account both their spatial overlap and their forms. By using the CIoU loss, YOLOv5 encourages accurate localization of objects and enhances the precision of the bounding box predictions.

$$\text{Loss} = \lambda_1 L_{\text{cls}} + \lambda_2 L_{\text{obj}} + \lambda_3 L_{\text{loc}} \tag{5.13}$$

5.4 Hardware Deployment System

The Jetson Xavier is built on the NVIDIA Volta GPU architecture, featuring a custom-designed embedded system-on-module (SoM). The SoM integrates an octa-core ARM CPU, a GPU with 512 CUDA cores, and dedicated hardware accelerators for deep learning inference. This architecture enables efficient execution of neural networks and computationally demanding AI algorithms on the edge device itself. NVIDIA provides comprehensive software support for the Jetson Xavier, including a SDK, libraries, and frameworks optimized for AI and deep learning tasks. Developers can leverage popular deep learning frameworks such as TensorFlow, PyTorch, and Caffe, as well as NVIDIA's own CUDA parallel computing platform, to build and deploy AI models on the Jetson Xavier platform[52]. Benchmarking tests demonstrate the impressive performance of the Jetson Xavier Figure 5.22 shows the Jetson Xavier Deep Learning Benchmarks. It delivers exceptional processing capabilities, capable of handling complex deep-learning models with high accuracy and low latency. Furthermore, the Jetson Xavier exhibits remarkable power efficiency, consuming minimal energy while executing AI workloads, making it well-suited for battery-powered and resource-constrained edge devices. The Jetson Xavier finds applications in various domains, revolutionizing industries by bringing AI capabilities to the edge. It enables real-time object detection, image recognition, natural language processing, and other AI tasks in robotics, enabling autonomous navigation, manipulation, and perception. In autonomous vehicles, the Jetson Xavier powers advanced driver assistance systems (ADAS) and enable intelligent decision-making at the edge. It also enhances surveillance systems by enabling real-time video analytics and anomaly detection, leading to improved security and situational awareness.

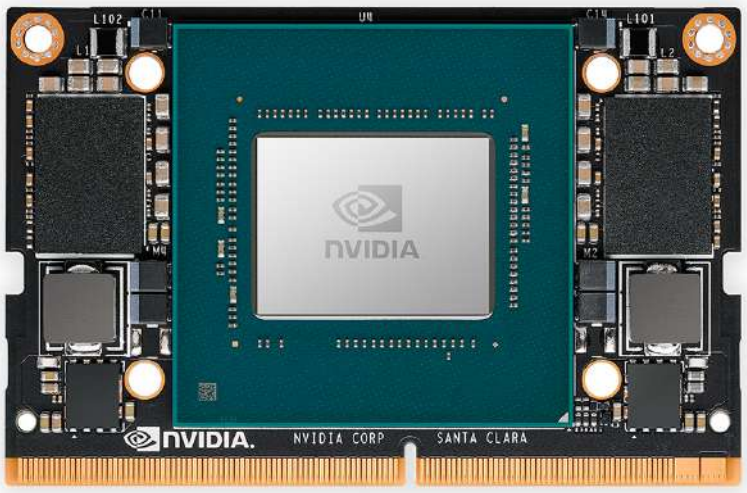

Figure 5.21: Nvdia Jetson Xavier NX[52]

The Jetson Xavier NX Deep Learning Model Benchmarks

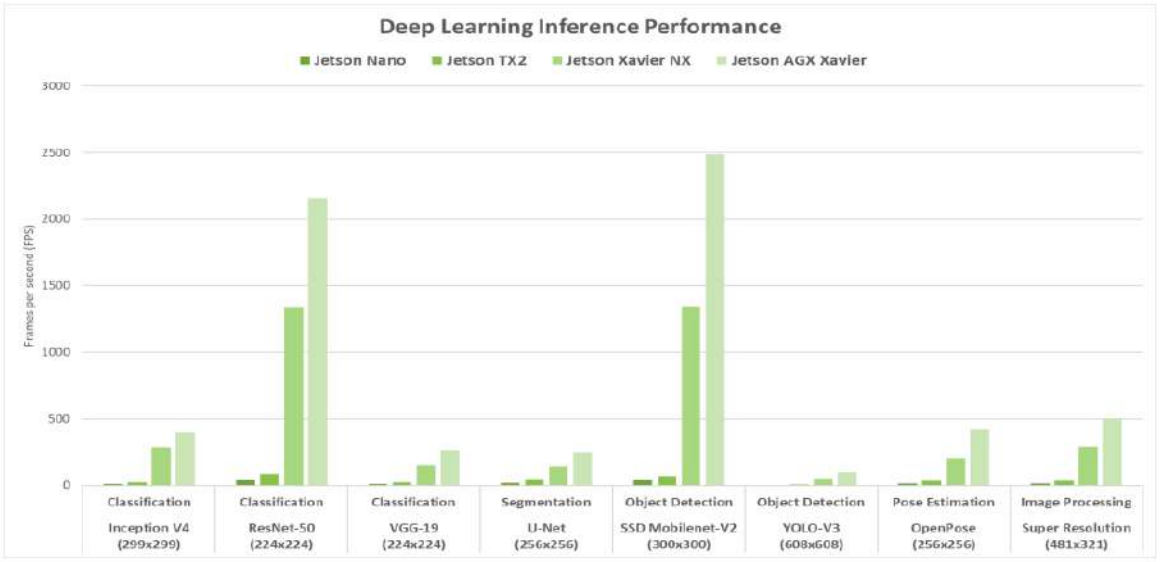

Figure 5.22: Jetson Xavier Deep Learning Bemchmarks[52]

5.5 Implementation Platform

Programming Language: YOLOv5 and Blob Detection can be implemented using Python, a versatile and widely-used programming language for machine learning and computer vision tasks. For deployment of algorithms in Jetson Xavier the C++ programming language is used.

Deep Learning Framework: YOLOv5 is typically implemented using the PyTorch deep learning framework. PyTorch provides a powerful set of tools and functions for building and training neural networks, making it well-suited for YOLOv5 implementation.

Libraries: The implementation of YOLOv5 and Blob Detection often relies on libraries such as

i) OpenCV: OpenCV is used for image and video processing tasks. It provides various functions for loading, manipulating, and saving images, which are often required during the pre-processing steps of YOLOv5. The Blob detection algorithm is developed using OpenCV, the preprocessing, image conversions, Thresholding, finding the counters, and detecting the blobs can be developed using this library.

ii) Numpy: NumPy is an essential Python package for scientific computing. It allows for efficient numerical operations on multidimensional arrays. NumPy is frequently used in YOLOv5 for handling image data and manipulating bounding boxes. The NumPy is used in developing both the algorithms for handling the image data and manipulating the boundary boxes.

iii) Matplotlib: Matplotlib is a plotting library used for creating visualizations in Python. It is often employed in YOLOv5 for visualizing training and evaluation

results, such as displaying bounding boxes and object detection predictions.

Development Environment: This thesis project was implemented on Google Colab. It is a cloud-based Jupyter Notebook environment that provides free access to GPUs and allows users to run and experiment with Python code, collaborate with others, and perform data analysis and machine learning tasks without the need for local installation or high-end hardware.

Hardware Acceleration: For optimal performance, utilizing hardware acceleration, such as GPUs (Graphics Processing Units), is beneficial, especially for YOLOv5, which involves complex deep learning models. Google Colab provides free access to GPUs, enabling faster training and inference. Tesla T4 is used for the YOLOV5 algorithm implementation, using a Normal CPU the Blob detection algorithm is developed.

Test Setup: The fabric defect detection test setup, as illustrated in the figure below, comprises a roller to process the fabric for defect identification using a Basler Camera. The Basler Camera is connected to Jetson Xavier, which deploys the defect detection models. These models effectively detect fabric defects and showcase the results on a user-friendly GUI.

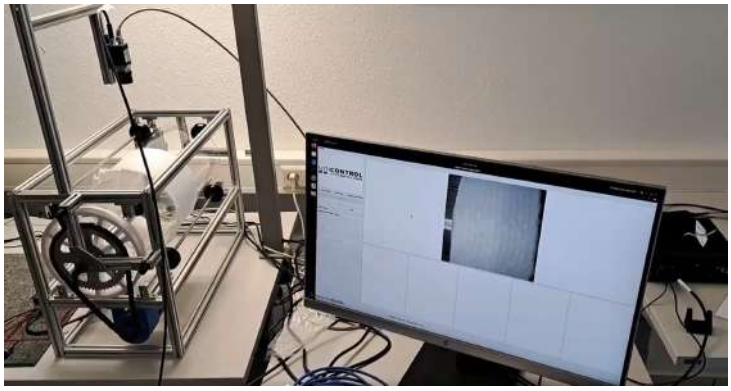

Figure 5.23: Fabric Defect Detection Test Setup

6 Algorithm Implementation

6.1 Preparation and Annotation of Dataset

Dataset Preparation

The synthetic dataset is typically divided into 3 subdirectories(Test, Train, Val) which are split into 70% for training, 20% for validation, and 10% for testing[Own illustration]. The table below shows the synthetic dataset preparation for the training of the algorithms for iteration 1. The size of these images is based on Yolo image sizes 640*640. and the real images are only used for Testing.

Error Types	No of Synthetic Images	No of Real Images
Images contain FF Class	800	11
Images Contain ELA Class	800	2
Images contain DG Class	800	3
Images contain FF & ELA classes	500	-
Images contain FF & DG class	500	-
Images contain FF, DG &ELA class	600	-

Table 6.1: Dataset Preparation

Figure 6.1: FF, DG, ELA Error Types

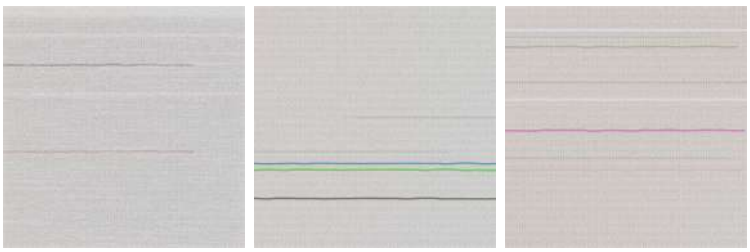

Figure 6.2: Mixed Error Type Dataset

Dataset Annotation

Here in the data annotations of this particular project, there are two types of annotations, for the CNN-based algorithm the Yolo-based annotations in the .txt file, and for blob detection the .xml-based annotations. the makesenseai tool is used for the data annotation process. The below figure can show the input fabric image with their annotated image along with the type of the error. In the image, there are 3 errors which are FF, DG, ELA they are annotated with 3 different colors of boundary boxes, the purple color annotation is for the ELA class, and the Green color annotation is for FF, and the Pink color represents the DG class. The Figure 6.3 shows the Image with its Annotated image.

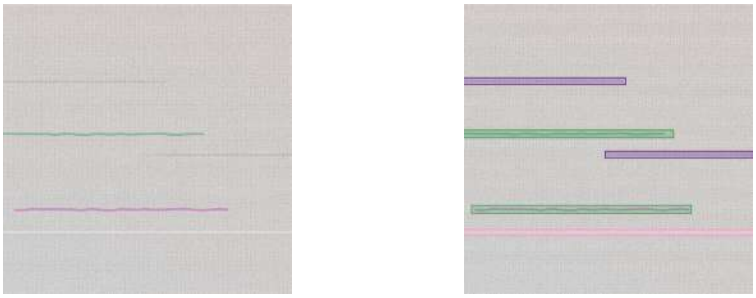

Figure 6.3: Original Image and Its Annotation

YOLO Format for YoloV5 object detection

In YOLO format, each annotation is saved in a separate text file with the same name as the corresponding image file. Each line in the text file represents an object annotation and follows the format:

<class_label><x_center><y_center><width><height>,

where

1. <class_label>is the object category,
2. <x_center>and <y_center>are the normalized coordinates of the bounding box center and
3. <width>and <height>are the normalized width and height of the bounding box.

PASCAL VOC Format for Blob Detection

1. Each <annotation>represents a single annotation file.
2. The <filename>element specifies the image file name.
3. The <size>section provides the image dimensions with <width>and <height>.
4. Each <object>element represents an annotated object.
5. The <name>tag denotes the class label of the object.
6. The <bndbox>section contains the bounding box coordinates, including <xmin>, <ymin>(top-left corner), <xmax>, and <ymax>(bottom-right corner).

6.2 Implementation of Blob Detection Algorithm

In Figure 6.4 the Flow chart explains the implementation of the Blob detection algorithm. For algorithm implementation, The 3 datasets are used for implementation, The implementation code is OpenCV-based Python code, The process is outlined in Figure 6.4. Before applying the Blob detection, the pre-processing step converts the original image to grayscale and applies binary thresholding. Chapter 5 provides details about the Blob detection parameters and the contour detection process. Fine-tuning these parameters helps identify blobs in error areas. Contour detection is used to draw the merged regions of interest on the contour image, visualizing the detected defects. Each contour is represented as a list of (x, y) points outlining a connected region in the binary image. The algorithm then creates boundary boxes around nearby contours, merging them into one contour to form a merged region of interest.

After contour detection, post-processing is performed. This step involves drawing bounding boxes enclosing the merged regions of interest from the contours of Section 5.3.1 explains the different types of contour approximation methods all four methods can be tested during the feature extraction. These bounding boxes define the top-left corner (x, y) as well as the width and height of the regions. Additionally, the algorithm determines the defect type during post-processing. This is achieved by analyzing the mean color values within the regions of interest. If the region contains mainly black pixels, it represents the "ELA" class. If it contains mostly white pixels, it is labeled as the "DG" class. Regions with different colored lines are treated as the "FF" class. The detected images, along with annotations of the boundary boxes and defect classes, can be saved for further analysis. these annotations can be stored in the PASCAL VOC format.

To find the metrics provided by the Python script, Compare the detected bounding boxes and the defect type with the ground truth bounding boxes to calculate metrics like Precision, Recall, F1 Score, and Accuracy.

Overall, the implementation involves preprocessing the image, detecting blobs and contours, post-processing to create boundary boxes and determine defect types, and finally, saving the results and annotations for evaluation and visualization purposes. Fine-tuning the Blob detection parameters is crucial to achieve accurate defect detection in the images. After implementing the defect detection and localization algorithm, evaluation metrics can be calculated by comparing the detected defects in the output images with the ground truth values of the original images.

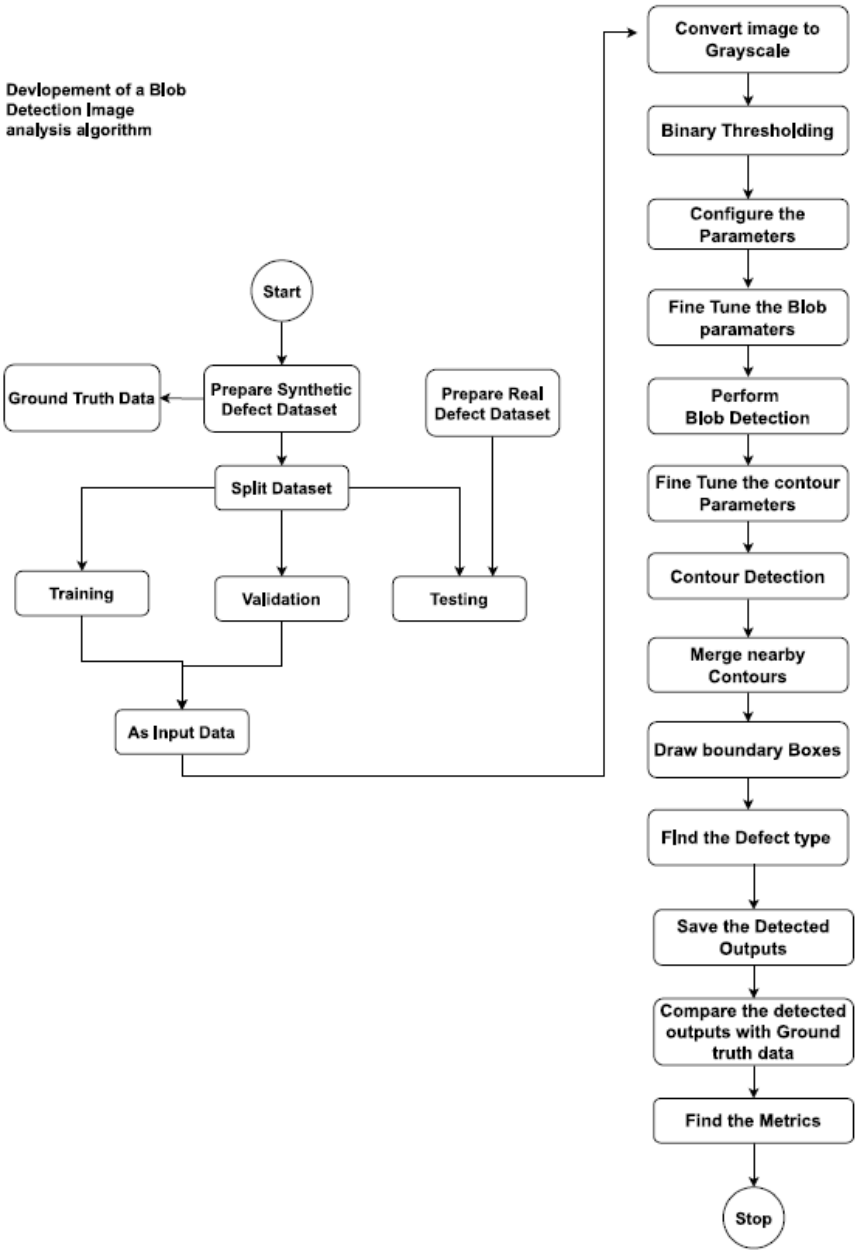

Figure 6.4: Blob Detection Algorithm

In classical image analysis, algorithms are typically rule-based and do not learn from data or adjust their parameters based on training examples like machine learning models do. However, when comparing results and evaluating the performance of classical image analysis algorithms with CNN-based algorithms a similar dataset preparation style is often used, involving the division of data into Train, Validation, and Testing sets.

As mentioned in the evaluation criteria, The evaluation metrics of the Training and validation dataset are shown below,

Metrics	Training Dataset	Validation Dataset
Precision	66.67%	62.67%
Recall	56.48%	68.73%
F1 score	61.17%	65.55%
Accuracy	44.40%	49%

Table 6.2: Evaluation Results of Blob Detection on Training and Validation Dataset

In section 3.7 of the Basics Chapter, the performance metrics for evaluating models were discussed, including True Positives, True Negatives, False Positives, and False Negatives. These metrics are essential for understanding the model's effectiveness in classification and object detection tasks. To explain this some of the images are taken from the train and validation dataset, for every image there is one Original image on the right side, the middle image is the ground truth image and the left image is the detected output. Let's illustrate the concept using Figure 6.5. this demonstrates a scenario of True Positive detection. The image in question contains instances of the "FF" class, which are marked as ground truth annotations (the correct labels). The algorithm or model being evaluated is capable of detecting these instances perfectly, as evidenced by its correct predictions that align with the ground truth. In this scenario, both instances of the "FF" class are accurately detected by the model, leading to two True Positives (TP = 2). This indicates that the model correctly identified both instances of the "FF" class in the image.

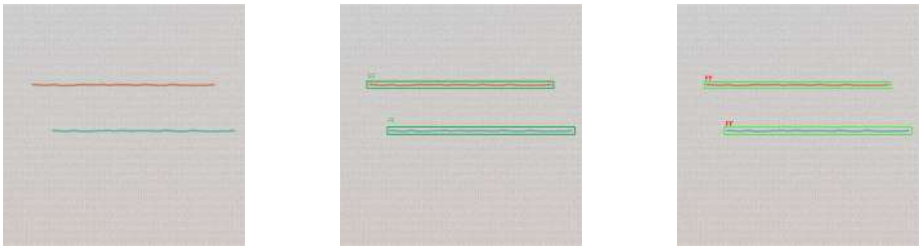

Figure 6.5: True Positive Detection

Figure 6.6, The Original contains two DG class defects, The algorithm is able to detect the defect type correctly but not able to draw the correct boundary box, this treated the False Positive.

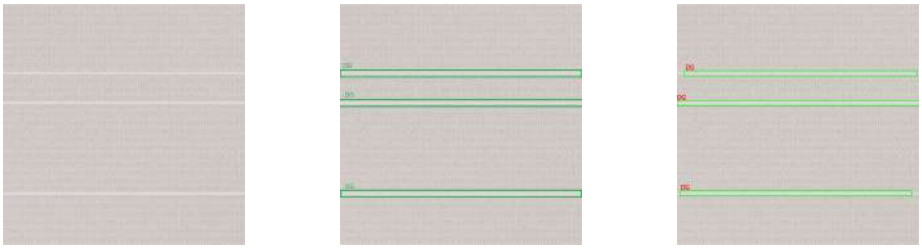

Figure 6.6: False Positive Detection

In Figure 6.7, there are two different defect types present in the image: one belonging to the "DG" class and the other belonging to the "FF" class. The algorithm correctly detects the defects of the "DG" class, which can be considered as True Positives. However, the algorithm fails to detect the defects of the "FF" class, resulting in two missed detections. These missed detections of the "FF" class defects are known as False Negatives.

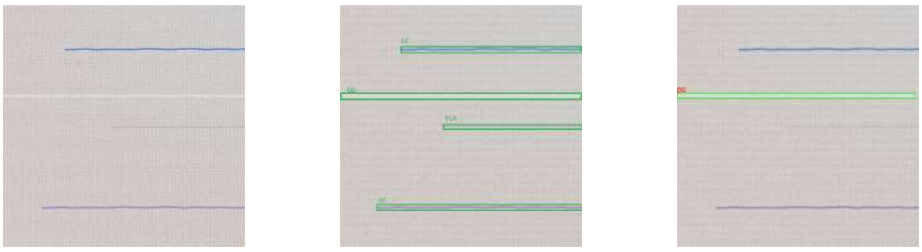

Figure 6.7: False Negative Detection

6.3 Implementation of Yolov5 Algorithm

The Implementation of the YOLOv5 algorithm involves several steps. First, the dataset needs to be prepared and annotated, with bounding boxes around the objects of interest. The source code of this YOLOv5 is obtained from the YOLOv5 repository from Ultralytics [12] Next, the YOLOv5s, YOLOv5m, YOLOv5l, and YOLOv5x pre-trained models are fine-tuned using this custom dataset. Training typically consists of 50 epochs, which are iterations over the entire dataset. Once the model is trained, it is validated and evaluated using ground truth data to assess its performance. After this process, the trained model can be used for object detection on the testing dataset. Evaluation metrics are provided to measure the performance of each YOLOv5 model variant. The flow chart in Figure 6.4 likely illustrates the overall implementation process of YOLOv5.

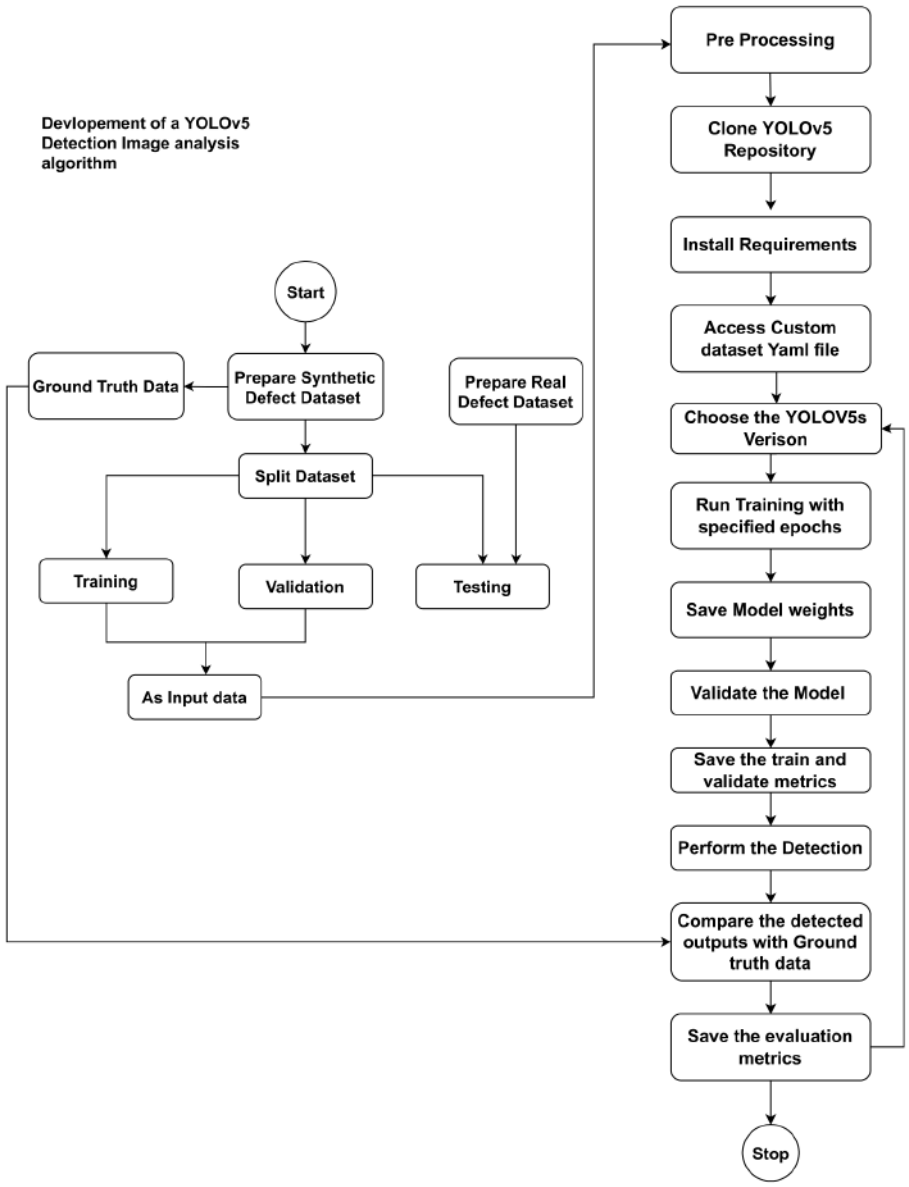

Figure 6.8: Flow chart of YOLOv5 Implementation

Iteration 1

There are 4000 total images in iteration 1, and these images are split up into 3 datasets. The images comprise 70% of the training data and 20% of the validation data and 10% is for Testing the four different model versions, including YOLOv5s, V5m, V5l, and V5x. Table 7.1 provides an explanation of how the four models were implemented during this iteration and also this table can explain the batch size, no of epochs for training and validation.

The Training and Validation Losses at the 50th Epoch are shown in Table 7.2. Training loss and object loss are discussed in Basics Chapter. The object loss and class loss graphs of the four YOLOv5 models are shown in Figures 7.1, 7.2, 7.3, and 7.4. for both the validation and training sets. These losses are 0.06, 0.03 at the first epoch, and 0 at 50 th epoch for training. also 0.06, 0.03 at the first epoch, and 0.0 for the validation set at the 50th epoch. This ensures that the model for all Four YOLOv5 models is doing

well on training and validation datasets. No object and class loss exists, hence all Four different models can provide precise object localization and accurate object recognition are all features of all 4 models.

Model	**Training**		**Validation**	
	Object Loss	Class Loss	Object Loss	Class Loss
YOLOv5s	0.024755	0.0029129	0.022703	0.0017782
YOLOv5m	0.023481	0.0026481	0.016612	0.001604
YOLOv5l	0.023091	0.0025718	0.016264	0.0016141
YOLOv5x	0.022888	0.0024768	0.016857	0.0016583

Table 6.3: Training and Validation Loss of Iteration 1

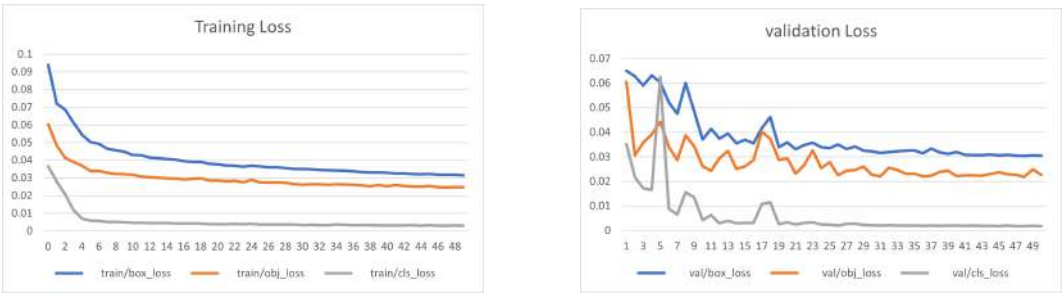

Figure 6.9: YOLOv5s Training and Validation Loss of Iteration 1

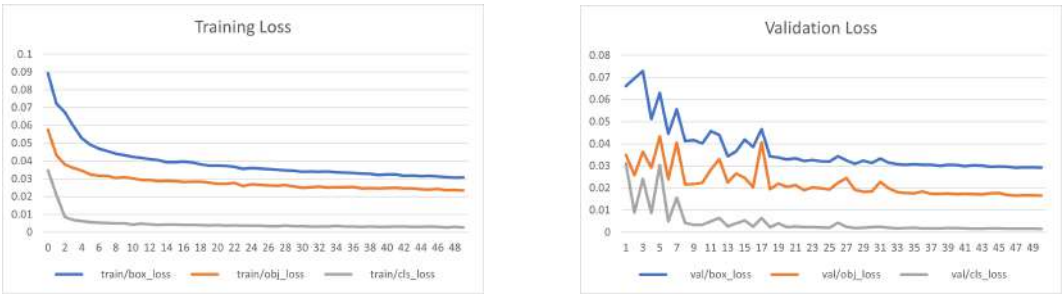

Figure 6.10: YOLOv5m Training and Validation Loss of Iteration 1

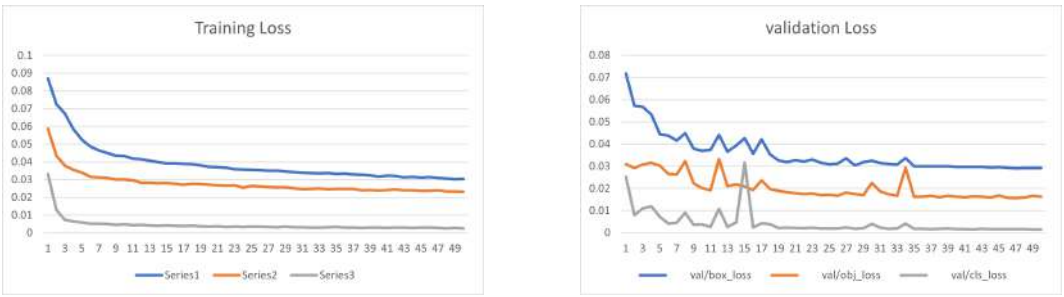

Figure 6.11: YOLOv5l Training and Validation Loss of Iteration 1

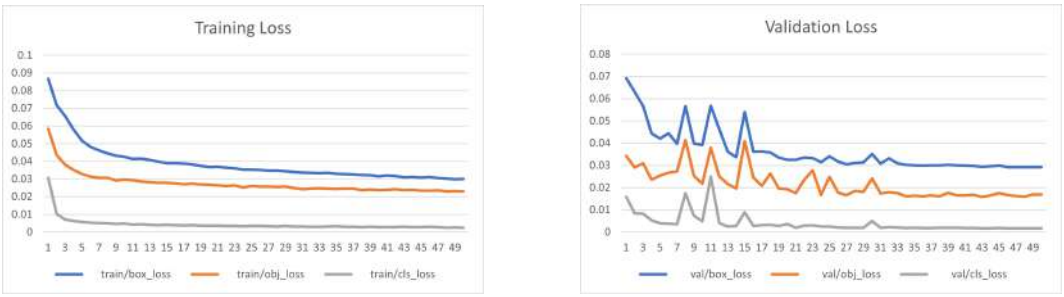

Figure 6.12: YOLOv5x Training and Validation Loss of Iteration 1

Iteration 2

In the second iteration, To overcome detection issues in iteration 1, 100 new images are added which are very tiny wrong-colored errors. These images are also split into 3 individual datasets and added to the first iteration dataset.

The validation set object loss and class can be seen in Figures 7.11, 7.12, 7.13, and 7.14. The values of these losses increase during the first set of 20 epochs, but after that, they approach zero and remain there until the 50th epoch. The training set object and class losses are zero at the 50th epoch.

Model	**Training**		**Validation**	
	Object Loss	Class Loss	Object Loss	Class Loss
YOLOv5s	0.023581	0.0030425	0.019469	0.0018675
YOLOv5m	0.022507	0.0028482	0.015084	0.001616
YOLOv5l	0.022182	0.0026553	0.014567	0.0016048
YOLOv5x	0.022027	0.0026048	0.014793	0.0016669

Table 6.4: Training and Validation Loss of Iteration 2

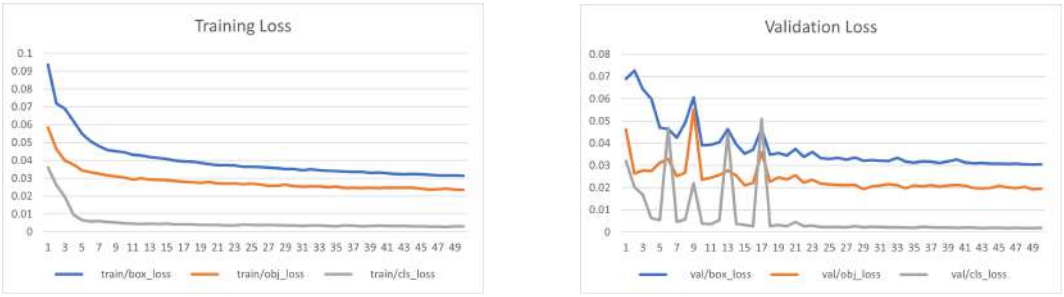

Figure 6.13: YOLOv5s Training and Validation Loss

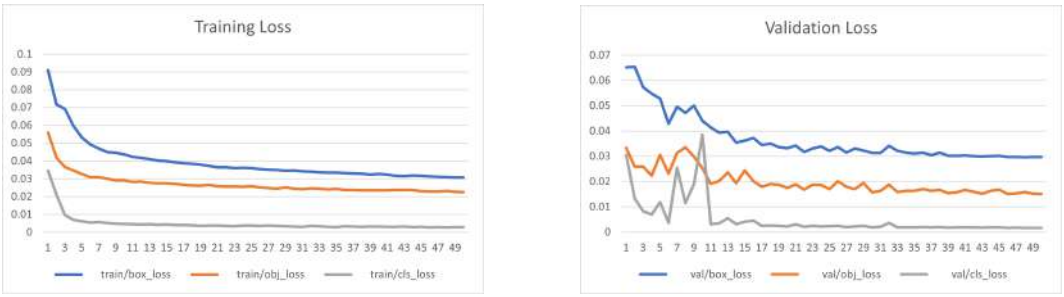

Figure 6.14: YOLOv5m Training and Validation Loss of Iteration 2

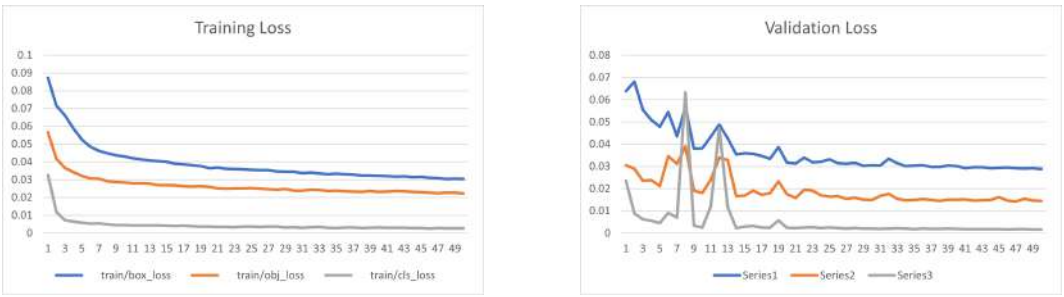

Figure 6.15: YOLOv5l Training and Validation Loss of Iteration 2

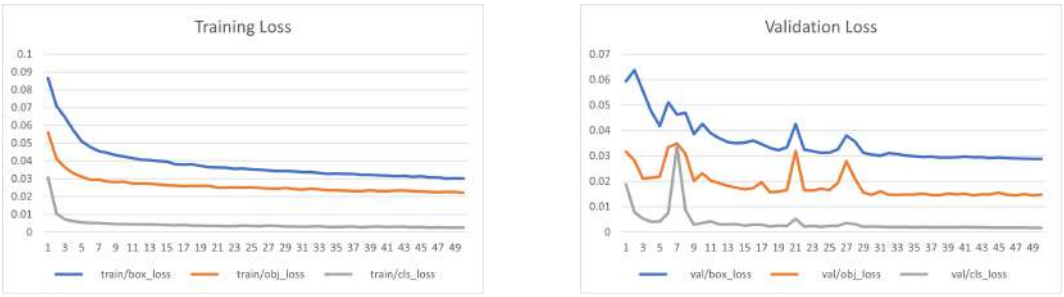

Figure 6.16: YOLOv5x Training and Validation Loss Iteration 2

6.4 Algorithm Deployment on Jetson Xavier NX

The deployment of the YOLOv5 algorithm and Bloc detection algorithm in Jetson Xavier can be explained here with the help of a flowchart.

6.4.1 Deployment of YOLOv5 Algorithm

Figure 6.13 shows the Flowchart of the YOLOv5 Alogirthm deployment in Jetson Xavier.

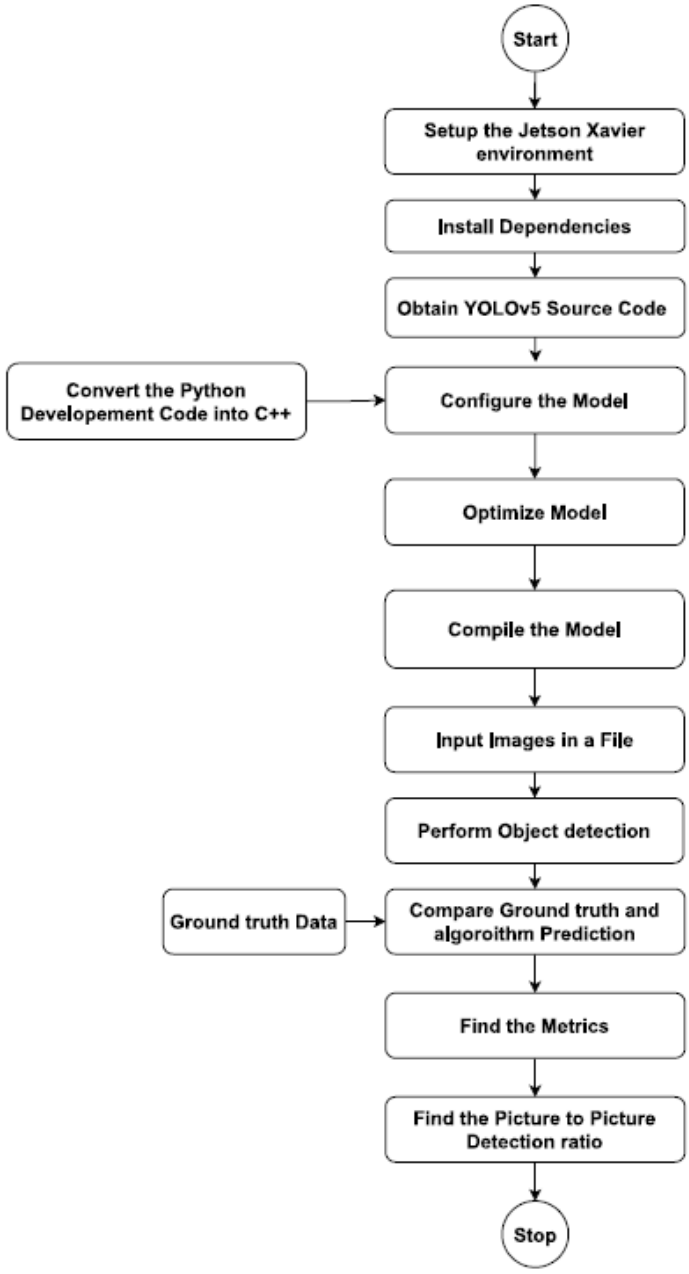

Figure 6.17: YOLOv5 Algorithm Deployment on Jetson Xavier NX

The deployment of YOLOv5 on NVIDIA Jetson Xavier involves several key steps. First, the Jetson Xavier platform needs to be prepared by ensuring it is powered on and equipped

with the necessary software and drivers. Next, dependencies such as CUDA, cuDNN, and OpenCV must be installed to facilitate the efficient execution of YOLOv5. The YOLOv5 source code is obtained, either by downloading it from the official repository or a reliable source. Configuration of the YOLOv5 model is then performed, involving the specification of model size, weights, and other relevant parameters. For custom-trained models, conversion to an optimized Tensor RT [10] format compatible with Jetson Xavier's GPU architecture is necessary this conversion part is explained in the 3.7 section, and also performs the INT8 Quantization Subsequently, the model is compiled to leverage the GPU capabilities of the Xavier device, ensuring it is optimized for deployment. Loading the model onto the Xavier device is essential for subsequent inference tasks. To test the deployment of the 410 images of the Synthetic test dataset, 16 images of the Real image dataset are provided for object detection. Utilizing the loaded YOLOv5 model, the input data is processed to predict bounding boxes, class labels, and confidence scores for detected objects. Finally, the results can be displayed with overlaid bounding boxes or stored for further analysis, depending on the specific application requirements. The performance results of both the synthetic dataset and the Real Dataset can be explained in the Deployment Result section.

6.4.2 Deployment of Blob Detection Algorithm

Figure 6.14 shows the flow chart of the blob detection algorithm on Jetson Xavier NX.

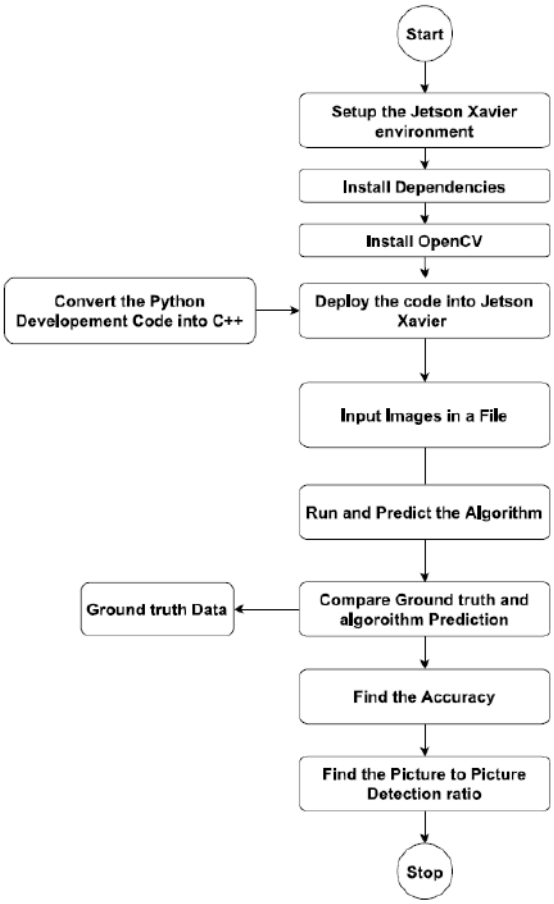

Figure 6.18: Blob detection Algorithm Deployment on Jetson Xavier NX

To deploy the OpenCV blob detection code on the Jetson Xavier, several key steps need to be followed. First, ensure that the Jetson Xavier is properly set up and configured and Convert the Opencv python code into C++ code. Install the dependencies and OpenCV on the Jetson Xavier, Once the code is ready, transfer the OpenCV code files to the Jetson Xavier using file transfer protocols like SSH or USB storage devices. Next, navigate to the code directory on the Jetson Xavier and compile the code using the appropriate compilation commands, making sure to link against the installed OpenCV libraries. After successful compilation, execute the code by running the compiled binary from the command line, providing any necessary command-line arguments or inputs. To optimize performance, consider leveraging the GPU capabilities of the Jetson Xavier by utilizing OpenCV's GPU modules and functions. Monitor and debug the code execution to identify and resolve any issues or performance bottlenecks. Keep in mind deployment considerations such as power consumption, memory usage, and real-time processing requirements. Optimize the code for Jetson Xavier's hardware architecture and capabilities to ensure efficient and effective deployment.

7 Results and Discussion

7.1 Blob Detection Results

In this section, the evaluation metrics of the Test dataset and the detected outputs are discussed.

7.1.1 Algorithm performance on Synthetic Dataset

The blob detection algorithm was subjected to testing on 410 images, resulting in an accuracy of 46%. However, the obtained accuracy and other evaluation metrics fell significantly short of the thesis objectives, revealing the limitations of the current approach. The results underscore the necessity for substantial improvements to achieve more reliable and accurate detections. The evaluation metrics of this algorithm can be shown in Table 7.1,

Metrics	Testing Dataset
Precision	68.05%
Recall	58.42%
F1 score	63.15%
Accuracy	46.14%

Table 7.1: Evaluation Results of Blob Detection on Testing Dataset

The Algorithm heavily relies on manual feature extraction. It performs well when the image contains only one error, successfully detecting and localizing it. However, its performance degrades significantly when the image contains multiple errors. In such cases, the algorithm struggles to detect and properly localize all three errors.

As discussed in the implementation section, post-processing is used for localization after identifying the defective areas. Figure 7.1 presents an example of accurate defect detection and successful localization achieved through this process for the FF class.

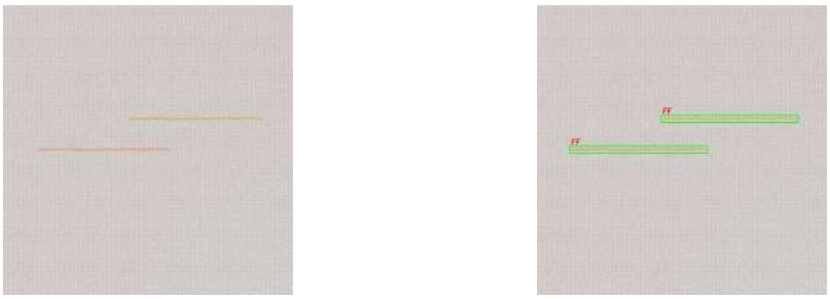

Figure 7.1: Original Image and Detected Image of FF Defect

The algorithm exhibits excellent performance in detecting and localizing the DG defect accurately. Figure 7.2 presents both the original image and the corresponding detected

image, showcasing the algorithm's ability to correctly label and precisely locate the DG defect.

Figure 7.2: Original Image and Detected Image of DG Defect

The algorithm demonstrates the capability of detecting the ELA defect. However, it struggles to correctly localize the error within the images. Figure 7.3 is the reference image to show this output, While the algorithm can successfully identify the presence of the defect, its localization accuracy falls short, highlighting the need for improvements in the localization process to achieve more precise results.

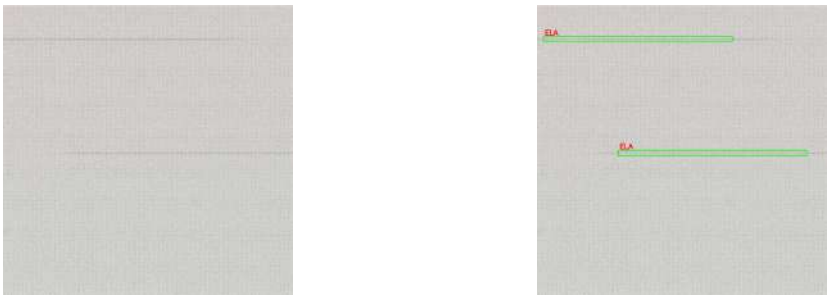

Figure 7.3: Original Image and Detected Image of ELA Defects

In cases where the image contains multiple errors, the algorithm struggles to detect and accurately localize them. Figure 7.4 illustrates this issue, displaying the original image alongside the detected image of multiple errors. The algorithm's performance is notably compromised in such complex scenarios, leading to incorrect or incomplete detections, and making it evident that improvements are required to handle images with multiple errors effectively.

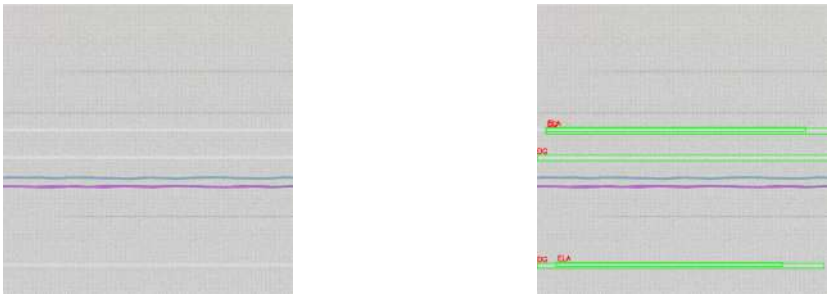

Figure 7.4: Original Image and Detected Image of Multiple Error

Despite the algorithm's limitations with multiple errors, it demonstrates promising results when handling single-error images. The post-processing step plays a crucial role in refining the defect localization, thereby enhancing the algorithm's overall performance.

7.1.2 Algorithm Performance on Real Dataset

As mentioned in Section 6.1, there are 16 real images that include the three error classes. The blob detection algorithm is tested on these images. Out of the 16 images evaluated, the algorithm failed to make correct detections in 13 images. For Reference, there were two images where the algorithm made incorrect detections, assigning them to the wrong classes. In Figure 7.5, the Detected output of the FF class image is shown. In the right-side image, there were two instances of the FF class, but only one defect was detected, while the other defect was not detected at all. Similarly, in the left-side image, there were two instances of the FF class, but the algorithm detected the defects without proper localization. Furthermore, despite there being no instances of the DG class, the algorithm erroneously detected some DG defects. These results indicate the limitations and shortcomings of the current algorithm, highlighting the need for further improvement and fine-tuning to achieve more accurate and reliable detections.

Figure 7.5: Detected Output of FF Class

The algorithm's performance is suboptimal as it fails to detect any instances of the ELA and DG classes in the images. Figure 7.6 and 7.7 illustrates the output of the detection using blob detection, revealing the absence of detections for these specific classes.

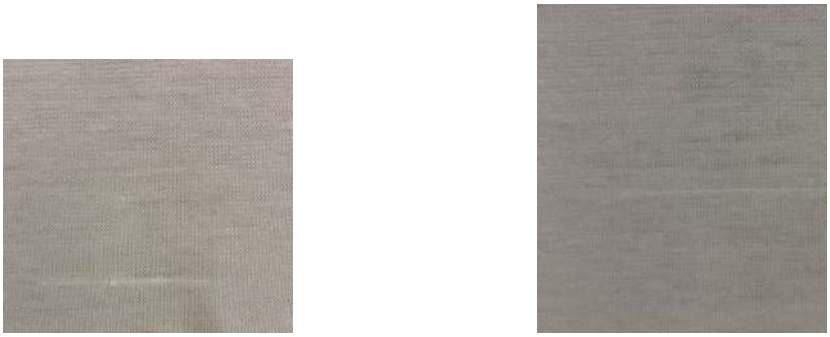

Figure 7.6: No Detection on DG Class Images

Figure 7.7: No Detection on ELA Class Images

Discussion

Classical image processing methods, like blob detection, have inherent challenges that limit their effectiveness in identifying different defect types accurately. One of the key issues is the need for manual feature extraction and selection for each defect type for the FF type they required one set of parameters to identify and for the other two types DG and ELA require different parameters, making the process time-consuming and challenging. Additionally, as mentioned methods heavily rely on parameter settings, leading to sensitivity to parameter tuning, where small adjustments can cause significant variations in the results. This reliance on parameters also makes the algorithms susceptible to manipulation, reducing their efficiency in detecting certain defects. Another observation of classical image analysis algorithms is their lack of adaptability. They are specifically designed for particular tasks, making it difficult to handle new defect types or accommodate variations in images. Adapting these algorithms to new challenges requires substantial modifications, consuming significant time and effort, and limiting their flexibility for various image analysis tasks. Furthermore, classical algorithms often lack robustness, particularly in real-world scenarios with challenging conditions. Factors like noise, varying lighting conditions, or occlusions can negatively impact their performance, leading to reduced accuracy and reliability. Consequently, these algorithms may not be as effective in practical applications where real-world challenges are prevalent. Overall, classical image processing methods face multiple limitations, including manual feature extraction, sensitivity to parameter tuning, limited adaptability, and lack of robustness. These limitations have spurred the development of alternative approaches like deep learning, which can overcome many of these challenges and offer more efficient and flexible solutions for image analysis tasks.

7.2 YoloV5 Object Detection Results

7.2.1 Algorithm Performance on Synthetic Dataset

In this section, the YOLOv5 model evaluation results and the Test outputs of two Iterations are explained.

Iteration 1

The evaluation metrics of the YOLOv5 algorithm are discussed in Chapter 3. The evaluation results of the YOLOv5 algorithm with four different models are displayed in Table 7.2. Among all four models, the YOLOv5x was well performed and obtained the highest precision, recall, and mAP value. In the first iteration of the YOLOv5 implementation, all four versions crossed the threshold mAP which is 95% of all classes. All four models also got more than 95% Precision and Recall. Figures 7.8, 7.9, and 7.10, illustrate the first iteration Precision, Recall, and mAP_0.5 of each of the four models, which were trained over 50 epochs.

Models	Precision	Recall	mAP_0.5	mAP_0.5:0.95
YOLOv5s	95.77%	97.11%	97.74%	57.80%
YOLOv5m	96.66%	97.66%	98.33%	60.25%
YOLOv5l	96.75%	97.67%	98.51%	60.12%
YOLOv5x	96.81%	98.20%	98.52%	59.99%

Table 7.2: Iteration 1 Evaluation Results of Validation Dataset

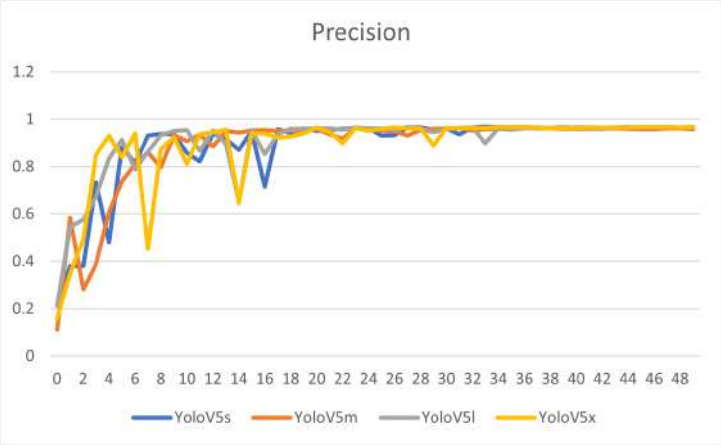

Figure 7.8: Precision Graph

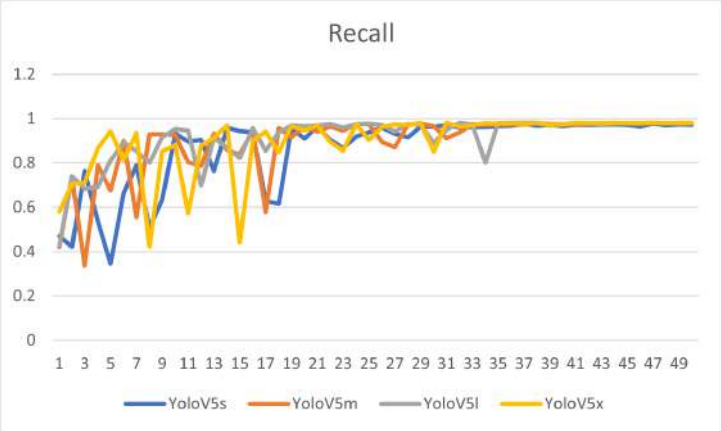

Figure 7.9: Recall Graph

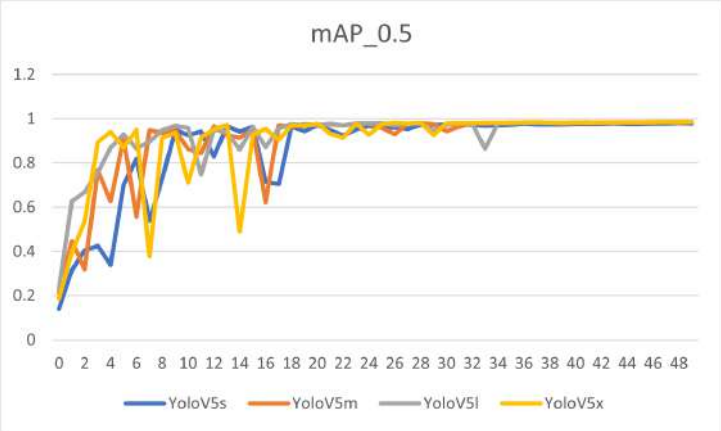

Figure 7.10: mAP_0.5 Graph

The four models of YOLOv5 algorithms are tested on the test dataset. The 400 images with 3 different types were detected. in this dataset, there are two images that have tiny wrong color errors, these two images were not detected out of these two images, one image and its corresponding ground truth is shown in Figure 7.11, the output of four models of this image is also shown in 7.12 and 7.13. Ideally, there is no detection is performed on any of the models.

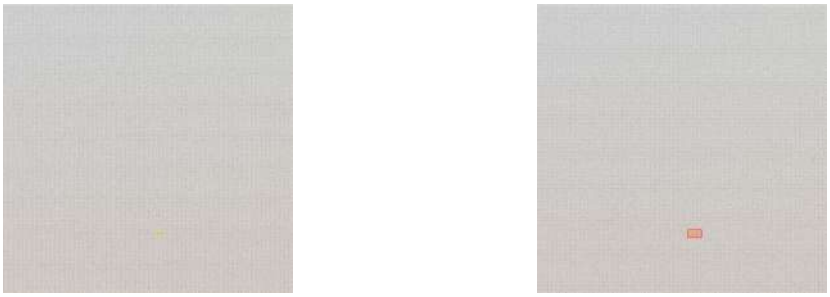

Figure 7.11: Error and Ground Truth Image

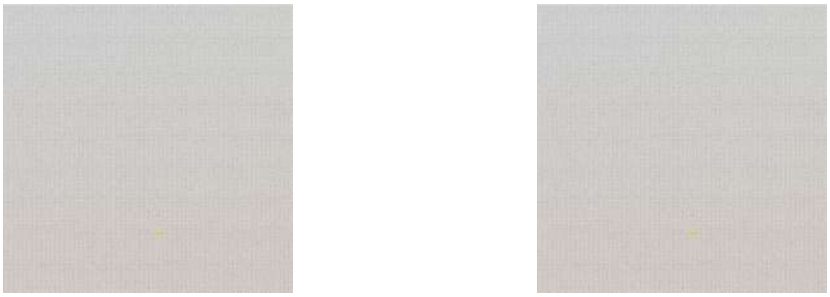

Figure 7.12: YOLOv5s, YOLOv5m Detection

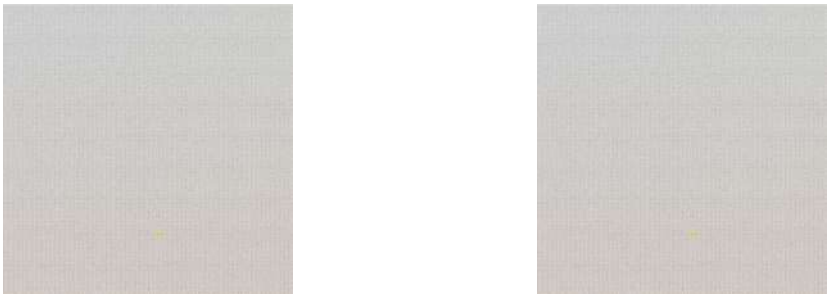

Figure 7.13: YOLOv5l, YOLOv5x Detection

Other 398 images were correctly detected with correct error objects. For example, in Figure 7.14 Right side image has three types of error classes left side image shows the corresponding ground truth image. object detection is performed by developed models on the image where algorithms are able to detect, recognize and localize correctly. Figures 7.15 and 7.16 shows the detected output.

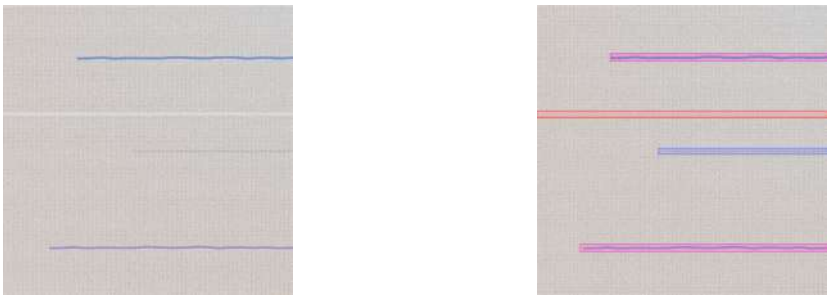

Figure 7.14: Original Image and Ground Truth Image

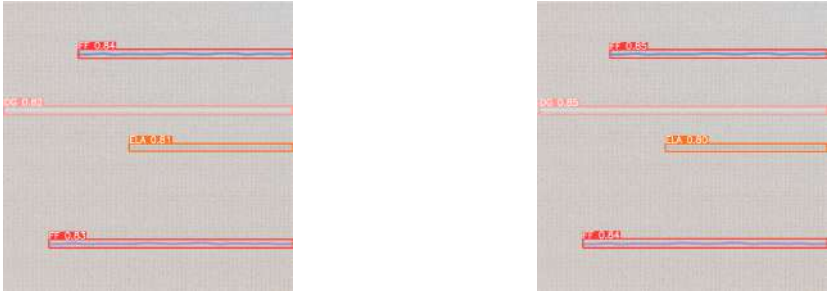

Figure 7.15: YOLOv5s and YOLOv5m Detection

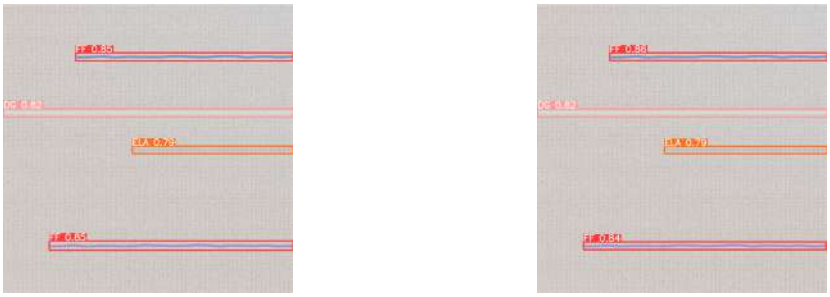

Figure 7.16: YOLOv5l and YOLOv5x Detection

Iteration 2

Since the tiny errors of the wrong color are not detected with the first iteration of YOLOv5 models, So new images were added to the dataset which has tiny wrong colored errors. The evaluation results of this second iteration are shown in Table 7.3. All the metrics, including Precision, Recall, and mAP_0.5, exceeded those values of the first iteration. Figures 7.17, 7.18, and 7.19 shows a graphical representation of these metrics. In this Iteration, the model YOLOv5x model did well among all four models achieving 98.33% percent of the mean average precision, 96.68% of precision, and 98.33% of recall value of all 3 classes. all four YoloV5 models performed well and delivered findings that exceeded the thesis's 95% accuracy goal for synthetic datasets.

Models	Precision	Recall	mAP_0.5	mAP_0.5:0.95
YOLOv5s	96.43%	97.33%	98.05%	58.83%
YOLOv5m	96.47%	98.10%	98.51%	59.45%
YOLOv5l	96.66%	98.25%	98.67%	59.97%
YOLOv5x	96.68%	98.33%	98.78%	60.17%

Table 7.3: Iteration 2 Evaluation Results of Validation Dataset

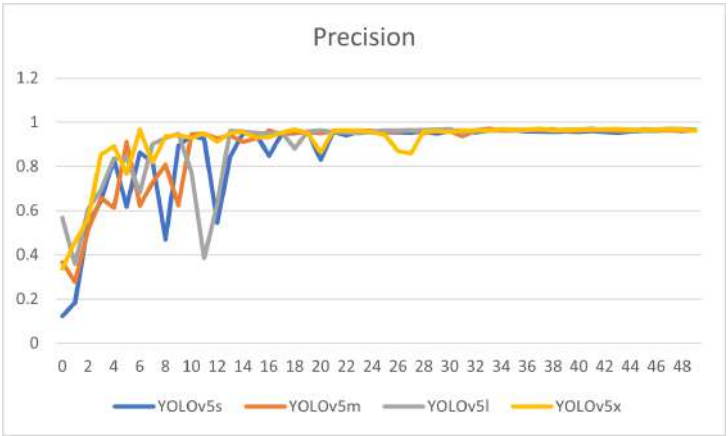

Figure 7.17: Precision Graph

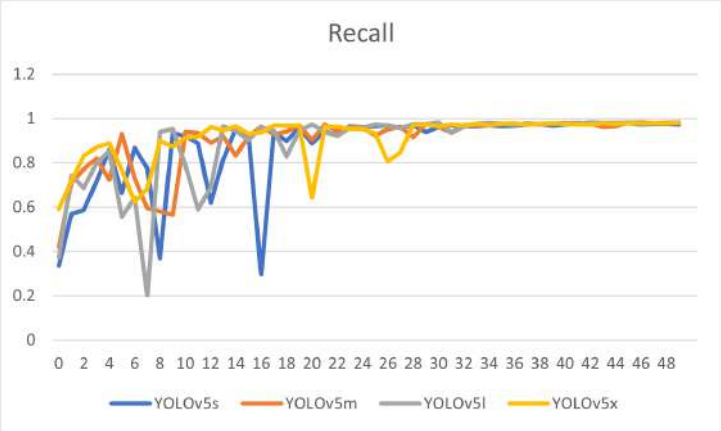

Figure 7.18: Recall Graph

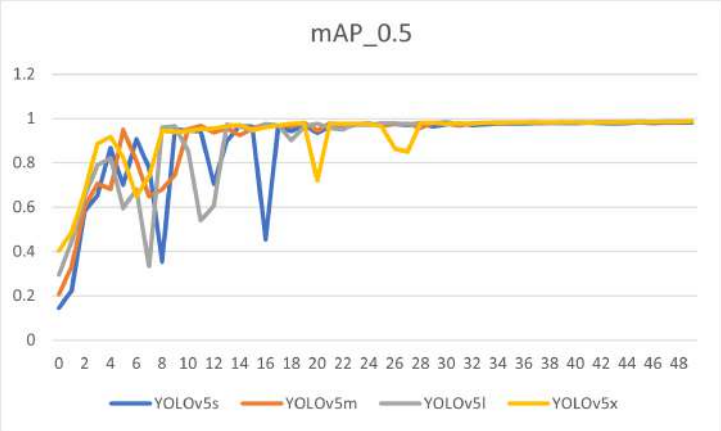

Figure 7.19: mAP_0.5 Graph

The second iteration algorithms, such as Yolov5s, v5m, v5l, and v5x, are able to detect the tiny wrong-colored error shown in Figure 7.7 shows the original error image and its corresponding ground truth image of the tiny wrong-colored error. The detected output is shown in Figures 7.20 and 7.21.

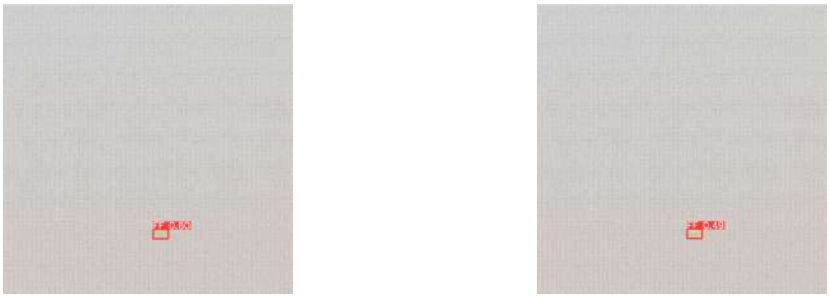

Figure 7.20: YOLOv5s, YOLOv5m Detection

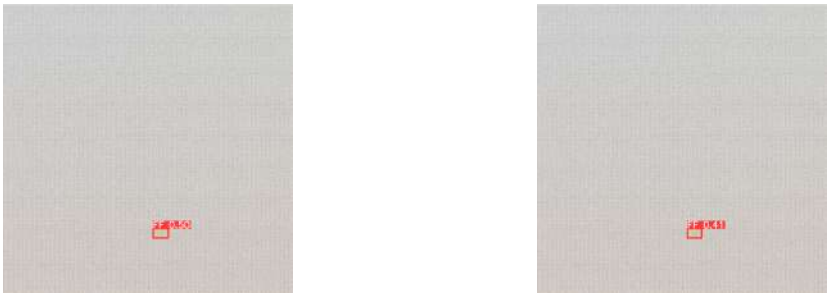

Figure 7.21: YOLOv5l, YOLOv5x Detection

Discussion

A tiny wrong-colored error is not detected by any other YOLOv5 model algorithm in Iteration 1. This can be observed for several reasons those are Insufficient Training Data, Imbalanced Classes, and Poor Quality Annotations to overcome this issue there are some solutions that can adding new training data with specified error types, checking the annotated ground truth dataset, fine-tuning training parameters to check over-fitting or under-fitting issues and also can use augmentation techniques. As per the recommendations, the new training data is added to the dataset and performed the Second Iteration of Training, all these new images are wrong-colored images with tiny sizes. These Issue with Detecting the Tiny wrong-colored error is resolved, After the second iteration all four YOLOv5 models are able to detect the tiny wrong-colored errors this can conclude that all four Algorithms are able to detect all the errors now the 410 images were detected properly. The Evaluation metrics of this test dataset are provided below. The Evaluation Results of the test dataset with Iteration 2 models is shown in Table 7.4

Models	Precision	Recall	mAP_0.5	mAP_0.5:0.95
YOLOv5s	96.3%	96.6%	96.7%	59.2%
YOLOv5m	96.4%	98.1%	97.5%	60.2%
YOLOv5l	96.5%	98.5%	97.7%	60.7%
YOLOv5x	96.5%	97.7%	97.8%	61.6%

Table 7.4: Iteration 2 Evaluation Results of Test Dataset

even though the 410 images are detected properly with the correct class the evaluation metrics are still not 100% this can be observed due to several reasons such as mislabeled data, IOU threshold selection, object size, etc.

7.2.2 Algorithm Performance on Real Dataset

All four YOLOv5 models were evaluated using 16 real images, which contained instances of the three classes. Specifically, Figure 7.22 and Figure 7.23 demonstrate the detection results for the "FF" class. In both cases, all four models successfully detected and localized the errors associated with the "FF" class.

Figure 7.22: FF Class YOLOv5s and YOLOv5m Detection

Figure 7.23: FF Class YOLOv5l and YOLOv5x Detection

Among the 16 images evaluated, two of them contain instances of the "DG" defect type. However, the detection results for the DG Defect were not favorable for either of these images. In one of the images, none of the models were able to detect the DG defect, while in the other image, all of the models provided incorrect detections and localizations for the defect. Specifically, Figure 7.24 and Figure 7.25 illustrate the problematic detection and localization of the defect.

Figure 7.24: DG Class YOLOv5l and YOLOv5x Detection

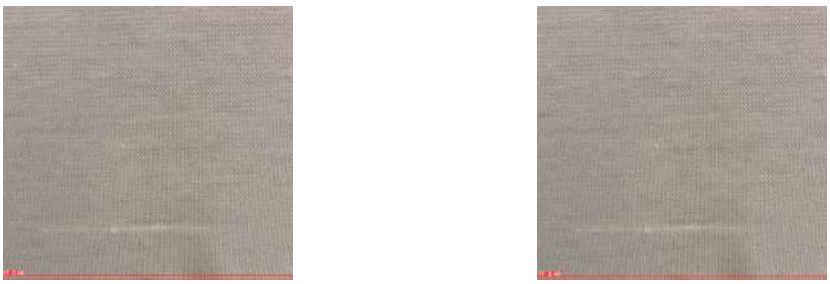

Figure 7.25: DG Class YOLOv5s and YOLOv5m Detection

Figure 7.26 and 7.27 explains the ELA class images, The real images the Model v5s detected the error as Ff and ELA, and Model v5x detected the defect type as FF and the other two models did not detect anything.

Figure 7.26: ELA Class YOLOv5s and YOLOv5m Detection

Figure 7.27: ELA Class YOLOv5l and YOLOv5x Detection

There are different datasets available in fabric defect detection this dataset is found in the Literature Jeyaraj et al [21], In that there are different error types, here this algorithm is tested on some of the Datasets which has similar error types. TILDA Dataset, and Fabric Defect Dataset These datasets have similar errors discussed in the literature summary. Some of the images were selected from the Fabric defect dataset and the second iteration YOLOv5x Algorithm is tested on these images. The results of these images were shown in Figure 7.28, the original image has a thick Yarn that looks like an error the algorithm is able to detect and localize it correctly. it shows the detected image, in this image, there is an elastane breakage the algorithm is able to detect and localize correctly.

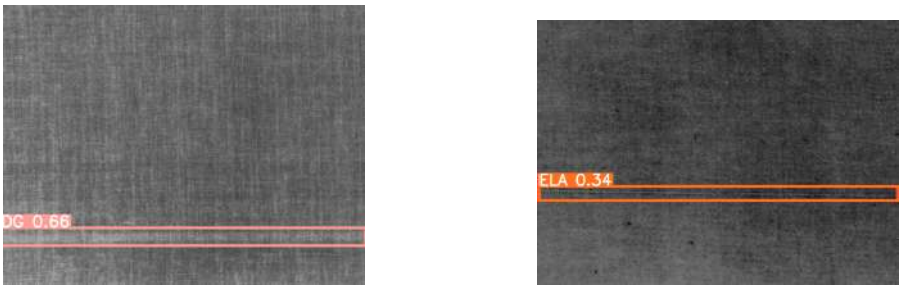

Figure 7.28: Detected Output of test Images from Fabric Defect Dataset

The Algorithms are also tested on TILDA dataset images, which have images having horizontal line error classes, but the system is not able to detect or localize correctly. the detected outputs were shown in Figure 7.29.

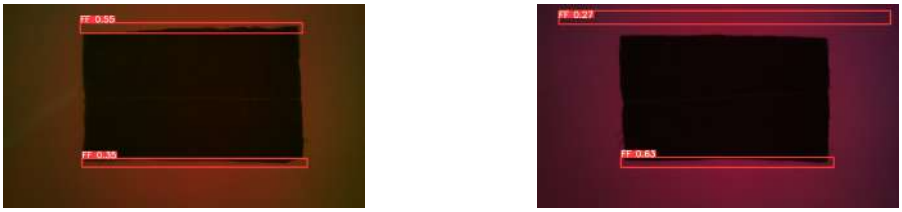

Figure 7.29: Detected Output of test Images from TILDA Dataset

Figure 7.30 shows the image with two different colors, the Model is able to find the Wrong colored error. but the model also detects the DG class even though there is no such kind of error present.

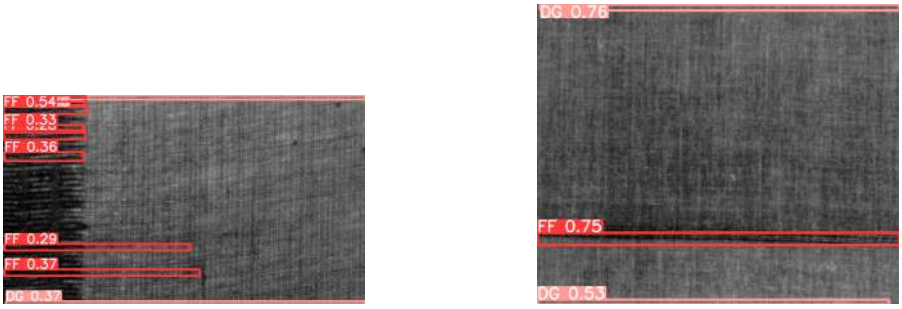

Figure 7.30: Detected Output of test Images from TILDA Dataset

Discussion

The models are able to detect the FF defect correctly, but the models are not accurately finding the ELA and DG classes, Even some of the images were detected as FF instead of ELA. The Improvements in the synthetic data of ELA can be able to overcome this issue. Deep learning models have shown significant advantages over traditional image processing methods in defect detection tasks. One of the key benefits is their ability to automatically learn relevant features from the data during the training process. Unlike traditional approaches that require manual feature engineering and parameter tuning for each defect type, deep learning models can adapt and generalize well to various defect scenarios. this can make the detection process more efficient and less prone to human error. Moreover, incorporating new defect types in traditional image processing can be complex and require an overhaul of the feature extraction pipeline. On the other hand, deep learning models offer a simpler approach where adding a new detection type involves modifying the model's configuration file and training it with relevant defective images. This advantage makes deep learning models more adaptable and flexible when it comes to handling new defect types. Our experimental findings have indicated that deep learning models outperform traditional image analysis algorithms, particularly in accurately detecting and localizing defects. The deep learning models' ability to learn from data and adapt to different defect scenarios enables them to overcome the limitations of traditional methods, such as sensitivity to parameter tuning and reliance on parameters. The good performance of deep learning models highlights the effectiveness of automatic feature learning in image analysis tasks. By automatically extracting meaningful features from images, these models deliver reliable and efficient defect detection results and also consume less time for development, surpassing what traditional methods can achieve.

This makes deep learning a promising and highly effective approach in the field of defect detection for the fabric industry in real-time situations also.

7.3 Deployment Results

This chapter discusses the deployment of the YOLOv5 model on the embedded hardware, as it outperformed the Blob detection algorithm significantly. The chapter presents the results and performance metrics of the YOLOv5 model, highlighting its effectiveness in various computer vision tasks compared to the previous algorithm.

7.3.1 Synthetic Image Dataset

In the previous chapter, the deployment implementation of the YOLOv5 object detection algorithm was discussed. The results of the deployed algorithm on Synthetic images are presented here. Table 7.5 explains the evaluation metrics of the algorithm tested on 410 synthetic images using four YOLOv5 models on Jetson Xavier NX.

All four models achieved an mAP value of over 90%, indicating their strong performance. Among them, YOLOv5l outperformed the other three algorithms. Additionally, the inference times for YOLOv5s, v5m, and v5l were all below 100 ms, which aligns with one of the objectives of this master's thesis – ensuring that the picture-to-picture inference time is less than 100 ms.

On the other hand, although the v5x model achieved an impressive accuracy of 95%, its inference time was measured at 200 ms, making it slower compared to the other models. As a result, despite its high accuracy, the v5l model is the most optimal choice for real-time applications where low inference time is crucial.

Models	mAP_0.5	Picture to Picture Inference time
YOLOv5s	93.98%	20 ms
YOLOv5m	95.76%	48 ms
YOLOv5l	95.86%	75 ms
YOLOv5x	95.50%	200 ms

Table 7.5: Evalution Metrics of Synthetic dataset

During the evaluation of the deployment results, two reference images were utilized. Figure 7.31, 7.32, 7.33, and 7.34. The left side image contained all three types of defects such as FF, DG, ELA, and the models of YOLOv5s, v5m, v5l, and v5x demonstrated accurate detection and localization of these defects. The models effectively identified the presence of the defects and provided precise bounding box coordinates around them.

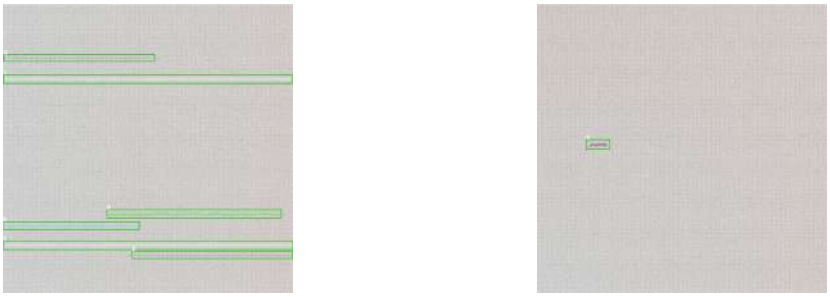

Figure 7.31: YOLOv5s Detection

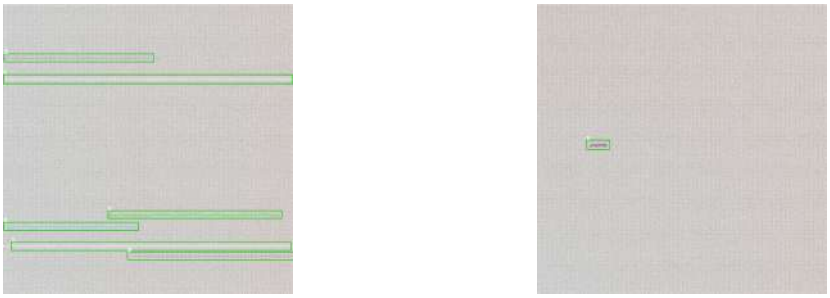

Figure 7.32: YOLOv5m Detection

In the Right side image, a small defect belonging to the FF class was present, and the models successfully detected and localized it as well. This indicates that the models possess the capability to effectively detect and localize defects, even when they are relatively small or belong to specific defect classes. In the image, 0 represents the FF, 1 represents the DG, and 2 represents the ELA.

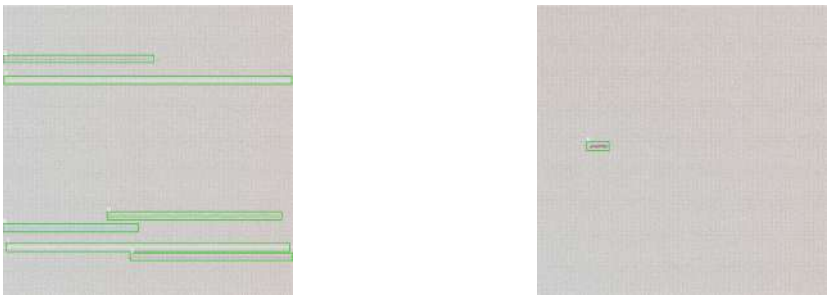

Figure 7.33: YOLOv5l Detection

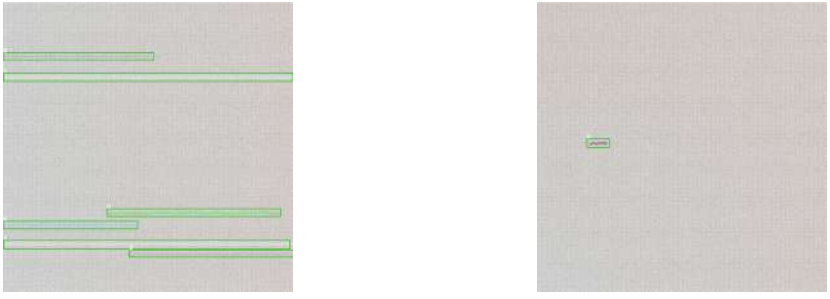

Figure 7.34: YOLOv5x Detection

7.3.2 Real Image Dataset

The performance metrics of real image data are summarized in Table 7.6. The evaluation metric used for deployment is the Picture to Picture inference time. As previously mentioned, the inference time for all YOLOv5s, v5m, and v5l models must be less than 100ms. The results show that these models indeed meet the requirement, achieving inference times within the desired range.

The real images with detections are shown in the figures below. Specifically, the images from the FF defect type were tested on an embedded system, and all four models (YOLOv5s, v5m, v5l, v5x) successfully detected and precisely localized the defects within the images Figure 7.35 and 7.36 displays the Output. This demonstrates the models' effectiveness and reliability in defect detection on the embedded system on FF defect.

Models	mAP_0.5	Picture to Picture Inference time
YOLOv5s	21.42%	20 ms
YOLOv5m	30.92%	75 ms
YOLOv5l	31.13%	75 ms
YOLOv5x	22.00%	200 ms

Table 7.6: Evalution Metrics of Real dataset

Figure 7.35: YOLOv5s and YOLOv5m Detection

Figure 7.36: YOLOv5l and YOLOv5x Detection

8 Conclusion and Future work

8.1 Conclusion

This thesis has successfully explored and compared fabric defect detection techniques using computer vision-based approaches. The research and development revealed that the machine learning-based object detection model performed well, delivering excellent results in detecting fabric defects. In contrast, the traditional blob detection-based algorithm fell short of meeting the expected expectations. The utilization of synthetic data proved to be a valuable asset in the machine learning algorithm's development, resulting in impressive performance. However, when tested with real image datasets, certain defects, such as ELA DG, posed challenges for the machine learning approach. Despite this limitation, the machine learning model excelled in identifying the FF defect with high accuracy. The research highlighted the advantages of the deep learning model over the blob detection-based image processing system. The deep learning model demonstrated superior performance and required less time for development, making it a more efficient and effective solution for fabric defect detection. The implementation of the machine learning-based defect detection system on an embedded system produced excellent results, reaffirming its practical applicability for real-time defect detection in industrial settings. In conclusion, this thesis has made significant achievements in the field of fabric defect detection. The successful implementation and performance of the machine learning-based object detection model underscore its potential in enhancing quality control practices in the textile industry. While the traditional blob detection algorithm may have shown limitations, the findings reinforce the value of embracing modern computer vision techniques for improved accuracy and efficiency.

This research offers valuable insights into the significance of synthetic data for training, and the suitability of machine learning-based systems for real-time applications. The study's accomplishments contribute to the advancement of fabric defect detection technology, providing a foundation for future research and development in the field. Overall, the thesis has made a meaningful contribution to the fabric defect detection domain and holds implications for optimizing quality control processes in the textile industry and beyond.

8.2 Future Work

- Enhancement of Synthetic Data Quality: To enhance the accuracy and realism of synthetic data specifically for DG and ELA defects, continuous improvements can be made to the data generation process. Advanced data augmentation techniques can be explored, focusing on DG and ELA defect-specific transformations to create diverse samples. Additionally, integrating more complex fabric textures that closely resemble those associated with DG and ELA defects can improve the synthetic

dataset's authenticity. Incorporating variations in lighting conditions and perspectives specific to these defect classes will further enhance the dataset's representation of real-world scenarios, making it more effective for training and evaluating fabric defect detection models.

- Further Algorithm Optimization for Blob Detection: Focus on algorithm optimization by exploring and implementing advanced techniques by using different filtering techniques to fine-tune the parameters and feature extraction process of the blob detection algorithm. By doing so, the algorithm's accuracy in detecting both single and multiple defects in fabric images can be significantly improved

- Increasing Real Image Samples for Algorithm Enhancement: In order to enhance the robustness and reliability of the algorithm, it is essential to increase the number of real images in the dataset, particularly for the DG and ELA classes. The limited number of samples in these defect categories may not fully represent the diversity of real-world defects, leading to potential performance gaps

Bibliography

[1] Benefits and importance of quality control in textile industry. Fashion Apparel https://fashion2apparel.com/importance-of-quality-control-in-textile-industry/, [Accessed on 05 Feb, 2023]

[2] Benefits the users can get out of this bullmer technology ai 01. Apparel resources https://apparelresources.com/technology-news/manufacturing-tech/new-ai-driven-fabric-defect-identifier-bullmer-fabric-inspection-process/

[3] Blob detection https://en.wikipedia.org/wiki/Blob_detection#:~:text=Incomputervision,blobdetection,color,comparedtosurroundingregions, [Accessed on 12 Feb, 2023]

[4] Blob detection using opencv (python, c++). Learn OpenCV https://learnopencv.com/blob-detection-using-opencv-python-c/, [Accessed on 10 Feb, 2023]

[5] Data annotation in 2023: Why it matters top 8 best practices. Data , Data Labeling https://research.aimultiple.com/data-annotation/, [Accessed on 10 Feb, 2023]

[6] Data annotation tutorial: Definition, tools, datasets. V7 https://www.v7labs.com/blog/data-annotation-guide, [Accessed on 10 Feb, 2023]

[7] Image classification vs. object detection vs. image segmentation. Medium https://medium.com/analytics-vidhya/image-classification-vs-object-detection-vs-image-segmentation-f36db85fe81

[8] Introducing deep learning with matlab. Mathworks https://de.mathworks.com/content/dam/mathworks/ebook/gated/80879v00_Deep_Learning_ebook.pdf, [Accessed on 25 Jan, 2023]

[9] Introducing machine learning. Mathworks https://www.mathworks.com/content/dam/mathworks/tag-team/Objects/i/88174_92991v00_machine_learning_section1_ebook.pdf, [Accessed on 20 Jan, 2023]

[10] Nvidia tensorrt. NVIDIA TensorRT https://developer.nvidia.com/tensorrt#benefits

[11] Scope of fabric testing — different types of fabric testing. Textile Learner https://textilelearner.net/scope-of-fabric-testing/, [Accessed on 05 Feb, 2023]

[12] Train custom data. Ultralytics https://docs.ultralytics.com/yolov5/tutorials/train_custom_data/#12-create-labels_1, [Accessed on 05 March, 2023]

[13] Yolo v5 model architecture [explained] `https://iq.opengenus.org/yolov5/`, [Accessed on 15 Jan, 2023]

[14] Blob detection. Computer vision Robotics (Oct 15, 2019), `https://medium.com/image-processing-in-robotics/blob-detection-309226a3ea5b`, [Accessed on 21 Feb, 2023]

[15] Automation, B.: Vision. `https://infosys.beckhoff.com/english.php?content=../content/1033/tf7xxx_tc3_vision/13363346955.html&id=`, accessed: 2023-06-29

[16] Burger, W., Burge, M.J.: Digital Image Processing: An Algorithmic Introduction. Springer Nature (2022)

[17] Eldessouki, M.: Computer vision and its application in detecting fabric defects. In: Applications of computer vision in fashion and textiles, pp. 61–101. Elsevier (2018)

[18] Fan, J., Wong, W.K., Wen, J., Gao, C., Mo, D., Lai, Z.: Fabric defect detection using deep convolution neural network. AATCC Journal of Research 8(1_suppl), 143–150 (2021)

[19] Géron, A.: Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow. " O'Reilly Media, Inc." (2022)

[20] Hicks, S.A., Strümke, I., Thambawita, V., Hammou, M., Riegler, M.A., Halvorsen, P., Parasa, S.: On evaluation metrics for medical applications of artificial intelligence. Scientific reports 12(1), 5979 (2022)

[21] Jeyaraj, P.R., Nadar, E.R.S.: Effective textile quality processing and an accurate inspection system using the advanced deep learning technique. Textile research journal 90(9-10), 971–980 (2020)

[22] Jing, J., Wang, Z., Rätsch, M., Zhang, H.: Mobile-unet: An efficient convolutional neural network for fabric defect detection. Textile Research Journal 92(1-2), 30–42 (2022)

[23] Kaehler, A., Bradski, G.: Learning OpenCV 3: computer vision in C++ with the OpenCV library. " O'Reilly Media, Inc." (2016)

[24] Li, M., Wan, S., Deng, Z., Wang, Y.: Fabric defect detection based on saliency histogram features. Computational Intelligence 35(3), 517–534 (2019)

[25] Liu, Z., Cui, J., Li, C., Wei, M., Yang, Y.: Fabric defect detection based on lightweight neural network. In: Pattern Recognition and Computer Vision: Second Chinese Conference, PRCV 2019, Xi'an, China, November 8–11, 2019, Proceedings, Part I. pp. 528–539. Springer (2019)

[26] Loussaief, S., Abdelkrim, A.: Convolutional neural network hyper-parameters optimization based on genetic algorithms. International Journal of Advanced Computer Science and Applications 9(10) (2018)

[27] Mahajan, P., Kolhe, S., Patil, P.: A review of automatic fabric defect detection techniques. Advances in Computational Research 1(2), 18–29 (2009)

[28] Mak, K., Peng, P.: Detecting defects in textile fabrics with optimal gabor filters. International Journal of Computer Science 1(4), 274–282 (2006)

[29] Nepal, U., Eslamiat, H.: Comparing yolov3, yolov4 and yolov5 for autonomous landing spot detection in faulty uavs. Sensors 22(2), 464 (2022)

[30] Ngan, H.Y., Pang, G.K., Yung, N.H.: Automated fabric defect detection—a review. Image and vision computing 29(7), 442–458 (2011)

[31] O'Mahony, N., Campbell, S., Carvalho, A., Harapanahalli, S., Hernandez, G.V., Krpalkova, L., Riordan, D., Walsh, J.: Deep learning vs. traditional computer vision. In: Advances in Computer Vision: Proceedings of the 2019 Computer Vision Conference (CVC), Volume 1 1. pp. 128–144. Springer (2020)

[32] Paluszek, M., Thomas, S.: MATLAB machine learning. Apress (2016)

[33] Peng, D., Zhong, G., Rao, Z., Shen, T., Chang, Y., Wang, M.: A fast detection scheme for original fabric based on blob, canny and rotating integral algorithm. In: 2018 IEEE 3rd International Conference on Image, Vision and Computing (ICIVC). pp. 113–118. IEEE (2018)

[34] Poulopoulos, N., Psarakis, E.Z.: A blobs detection algorithm based on a simplified form of the fast radial symmetry transform. In: 2017 22nd International Conference on Digital Signal Processing (DSP). pp. 1–5 (2017)

[35] Prof. Neeta Ingale, Atika Ansari, S.Y.S.T.S.S.: Fabric defect detection. International Journal of creative research thoughts(IJCRT) (2023)

[36] Ramella, G., SANNITI DI BAJA, G.: A new technique for color quantization based on histogram analysis and clustering. International Journal of Pattern Recognition and Artificial Intelligence 27(03), 1360006 (2013)

[37] Rasheed, A., Zafar, B., Rasheed, A., Ali, N., Sajid, M., Dar, S.H., Habib, U., Shehryar, T., Mahmood, M.T.: Fabric defect detection using computer vision techniques: a comprehensive review. Mathematical Problems in Engineering 2020, 1–24 (2020)

[38] Saleh, H., Saleh, S., Toure, N.T., Hardt, W.: Robust collision warning system based on multi objects distance estimation. In: 2021 IEEE Concurrent Processes Architectures and Embedded Systems Virtual Conference (COPA). pp. 1–6. IEEE (2021)

[39] Sangwine, S.J., Horne, R.E.: The colour image processing handbook. Springer Science & Business Media (1998)

[40] Sarah, G., et al.: Introduction to machine learning with python (2022)

[41] Sewak, M., Karim, M.R., Pujari, P.: Practical convolutional neural networks: implement advanced deep learning models using Python. Packt Publishing Ltd (2018)

[42] Solomon, C., Breckon, T.: Fundamentals of Digital Image Processing: A practical approach with examples in Matlab. John Wiley & Sons (2011)

[43] Sunduijav, C., Bayasgalan, Z., Tudevdagva, U., Hardt, W., Bilegt, D.: Improvement of insulator image processing using deep learning algorithms. In: 2022 23rd International Conference on Computational Problems of Electrical Engineering (CPEE). pp. 1–4. IEEE (2022)

[44] Ultralytics: YOLOv5: YOLOv5 in PyTorch. `https://github.com/ultralytics/yolov5`

[45] Wang, C.Y., Liao, H.Y.M., Wu, Y.H., Chen, P.Y., Hsieh, J.W., Yeh, I.H.: Cspnet: A new backbone that can enhance learning capability of cnn. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops. pp. 390–391 (2020)

[46] Wang, Y., Hao, Z., Zuo, F., Pan, S.: A fabric defect detection system based improved yolov5 detector. In: Journal of Physics: Conference Series. vol. 2010, p. 012191. IOP Publishing (2021)

[47] Wei, B., Hao, K., Tang, X.s., Ren, L.: Fabric defect detection based on faster rcnn. In: Artificial Intelligence on Fashion and Textiles: Proceedings of the Artificial Intelligence on Fashion and Textiles (AIFT) Conference 2018, Hong Kong, July 3–6, 2018. pp. 45–51. Springer (2019)

[48] Wen, Z., Zhao, Q., Tong, L.: Cnn-based minor fabric defects detection. International Journal of Clothing Science and Technology 33(1), 1–12 (2020)

[49] Wong, W., Jiang, J.: Applications of computer vision in fashion and textiles. pp. 47–60. Elsevier (2018)

[50] Xu, R., Lin, H., Lu, K., Cao, L., Liu, Y.: A forest fire detection system based on ensemble learning. Forests 12(2), 217 (2021)

[51] Yadav, P.K., Thomasson, J.A., Searcy, S.W., Hardin, R.G., Braga-Neto, U., Popescu, S.C., Martin, D.E., Rodriguez, R., Meza, K., Enciso, J., et al.: Assessing the performance of yolov5 algorithm for detecting volunteer cotton plants in corn fields at three different growth stages. Artificial Intelligence in Agriculture 6, 292–303 (2022)

[52] Zheng, B.Z., Pyke, J.: Optimizing nvidia tensorrt conversion for real-time inference on autonomous vehicles. Nvdia DEVELOPER (May 07, 2020), `https://developer.nvidia.com/blog/optimizing-nvidia-tensorrt-conversion-for-real-time-inference-on-autonomous-vehicles/`