TECHNISCHE UNIVERSITÄT
CHEMNITZ

Fakultät für Informatik

CSR-24-02

Image Classification for Drone Propeller Inspection using Deep Learning

S. M. Rizwanur Rahman · Wolfram Hardt

August 2024

Chemnitzer Informatik-Berichte

TECHNISCHE UNIVERSITÄT
CHEMNITZ

Image Classification for Drone Propeller Inspection using Deep Learning

Master Thesis

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

Dept. of Computer Science
Chair of Computer Engineering

Submitted by:
Name: S. M. Rizwanur Rahman
Student ID: 510561
Date: 17.04.2023

Supervising tutor:
Prof. Dr. Dr. h. c. Wolfram Hardt
Dr. Ariane Heller
Mohamed Salim Harras, MSc.

Acknowledgement

First of all, I express my gratitude and am overwhelmed by all those who have assisted me in turning these ideas into something tangible that goes much beyond the superficial dimension to conduct my master's thesis.

I would like to express my profound gratitude to Mohamed Salim Harras for his supervision and guidance at every stage from conceptualization to the evaluation of results. Without his invaluable assistance and suggestions, this thesis would not have been successful. His patience, tenacity, and ambition enabled me to successfully complete my thesis. Additionally, I want to thank Mr. Shadi Saleh for his prudent remarks on the active deep learning framework architecture were a great help to me. Moreover, I sincerely like to thank my academic supervisor, Prof. Dr. Dr. h. c. Wolfram Hardt and Dr. Ariane Heller. The insightful remarks were given by Prof. Dr. Dr. h. c. Wolfram Hardt, on my concept presentation, helped me to comprehend scientific report writing in a better way. Furthermore, the guidance provided by Dr. Ariane Heller during the time-to-time meeting kept me motivated, and her thoughtful comments on this report pointed me in the right direction.

Finally, I am also thankful to Mr. Asif Ahmed, Mr. Soaibuzzaman and my family for their encouragement, support, and inspiration.

Abstract

Drones have become an integral part of today's rescue operations. This field involves the risk of human life which is time and safety sensitive. Most of the time, drones are found to be faulty when needed, leading to accidents during operations. Among them propeller fault is the most frequent cause. Therefore, an important aspect of drone maintenance before the drone's flight performance is the diagnosis of the propellers. Traditional propeller inspection methods can be time-consuming and labor-intensive. A solution is proposed for inspecting drone propellers using image-based deep-learning techniques. The system utilizes to classify the propeller images into healthy and broken categories. In this research study, state-of-the-art classification models such as VGG-19, ResNet-50, and YoloV5m (medium) Classifier are applied. YoloV5m classifier achieved the highest Top-1 accuracy of 96.7% using passive learning. Furthermore, YoloV5 (medium, small), and YoloV7 detector are also trained. Based on the results, YoloV5m obtained the highest mAP among them with 97.1%. Following the active deep learning strategy, the YoloV5m and YoloV7 detector models are trained through an iterative process. According to the results, YoloV5m obtained the maximum mAP of 97.00%. On the other hand, YoloV7 is achieved 90.70% of mAP. Finally, our proposed solution is based on the maximum accuracy of the detection results, which helps to reduce accidents during rescue operations by detecting propeller faults in time.

Keywords: Propeller inspection, Image classification, VGG-19, ResNet-50, YOLOV5m classifier, Object detection, YoloV5, YoloV7, Deep learning, Active learning.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

UAV Unmanned Aerial Vehicle

UAS Unmanned Aircraft Systems

GPS Global Positioning System

DWT Discrete Wavelet Transform

IMU Inertial Measurement Unit

ANN Artificial Neural Network

CNN Convolutional Neural Network

DCNN Deep Convolutional Neural Network

FCN Fully Convolutional Network

FPN Feature Pyramid Network

RCNN Region-based Convolutional Neural Network

RFCN Region-based Fully Convolutional Network

ROI Region of Interest

RPN Region Proposal Network

SPP Spatial Pyramid Pooling

YOLO You Only Look Once

NMS Non-Maximum Suppression

1 Introduction

A drone, more formally known as an unmanned aerial vehicle (UAV) or unmanned aircraft system (UAS), can act as a flying robot. These drones have the capability to fly autonomously which is controlled remotely by utilizing software-controlled flight plans embedded in their systems. Additionally, they are equipped with onboard sensors and a global positioning system (GPS) to navigate. In recent years, drones have become an increasingly popular and versatile advanced technology for various applications because of their unique capabilities which have made them useful tools for specific tasks, such as aerial photography[29], surveillance and monitoring [30], emergency rescue [31, 32], inspection routines and more. Previously, drones were most often associated with military activities, but now they are also used in a range of civilian rules [33, 34, 35]. The low cost and low power consumption of these vehicles, paired with their high-resolution cameras, make them effective tools for civilian tasks. This has opened a wide range of research opportunities in these areas.

Using drones allows individuals to avoid dangerous situations without physical exposure. It also improves work efficiency and productivity by providing fast responses, which reduces working time and costs. However, drone failures can result in devastating consequences if they crash. According to an Airprox report[1], 30% of drone incidents occurred from 2013 to November 2018, and the number continues to rise. This is a concerning trend that highlights the importance of proper drone safety measures. To decrease the number of drone incidents, it is important to inspect drones before operation and identify any damaged parts. Common sources of physical damage on drones include kinetic energy, propellers, and storage batteries. Propellers are particularly susceptible to damage and are a major cause of drone incidents.

Propeller defects, such as cracks, deformations, and wear, can significantly reduce the efficiency and safety of drone operations and even lead to catastrophic failures. The traditional methods for inspecting drone propellers, such as manual visual inspection and ultrasonic testing, are time-consuming, labour-intensive, and may not be suitable for certain types of defects. In figure 1.1 depicts the majority of propeller fault inspection solutions in drones fall under acoustic, charge flow (current), and vibration-based approaches. Lee et al.[1] proposed a technique for detecting faults in UAV motors using steady-state model and an infrared sensor to measure angular speed. Another approach [36] uses angular speed and current data to detect speed

[1] `https://www.uasvision.com/2019/01/25/`

and failure in brushless motors, with simulations of unbalanced propellers and loose motors for faulty data. Although the current-based method is effective in detecting drone anomalies, it is more suitable for detecting faults in electrical components such as propeller motors and electronic speed controllers. The sound-based approach can

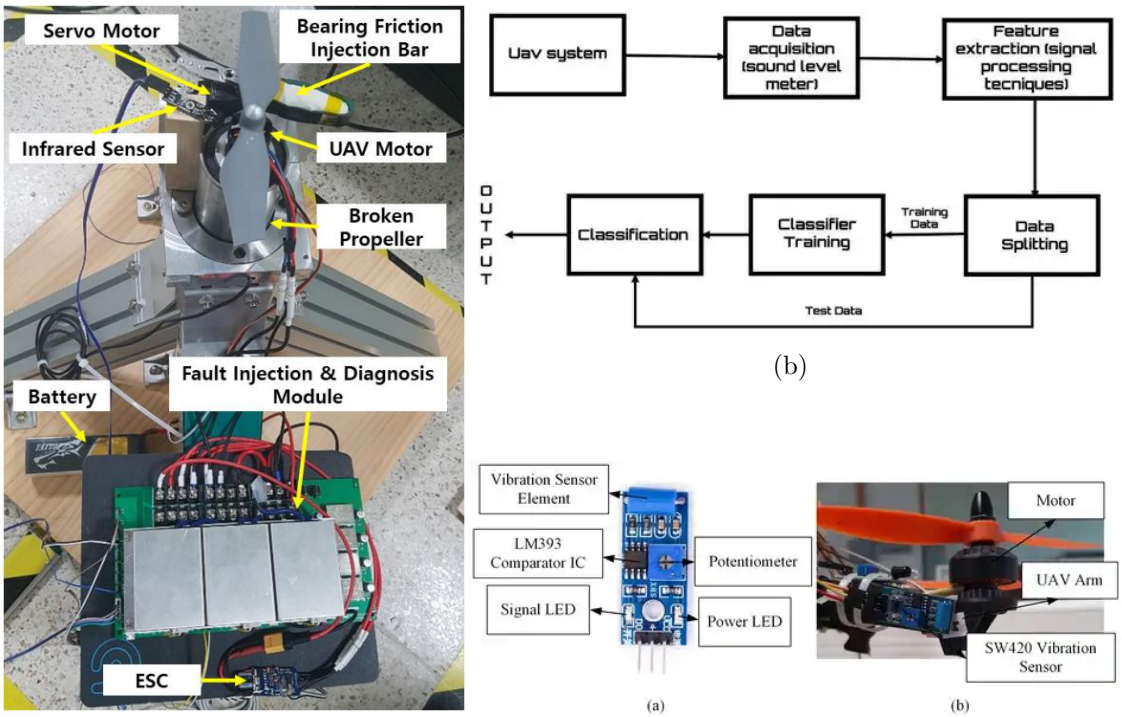


Figure 1.1: Propeller fault detection techniques (a) steady state model-based [1] (b) Sound-based [2] (c) Vibration-based inspection[3]

detect more faults and is easier to acquire signals than current-based methods. In [37, 2], have applied the noise from drones to detect the propeller faulty blades using a neural network model and techniques like Fourier transform and Discrete wavelet transform (DWT) [38] have also been used. However, sound-based techniques have a drawback in that they are susceptible to background noise and can be affected by sounds from other motors. Vibration-based methods [3] are based on acceleration data which is got from the Inertial Measurement Unit (IMU), built-in or external accelerometer and then classify the signals of rotor blades. In this approach, healthy propellers produce smooth signals compared to broken ones.

However, deep learning has evolved in recent years as a strong tool for image-based tasks such as object detection, classification, and segmentation. The task of object classification and detection, being the foundation of image understanding and computer vision, has laid the groundwork for solving complex vision problems. It enhances a range of applications including human-computer interaction, machine

vision and content-based image retrieval. And so far, no work has been done on propeller fault inspection using computer vision and deep learning technologies. Hence our proposed method is also taken a step forward by applying an image-based deep learning solution for inspecting the propeller fault.

1.1 Problem Statement

Several difficulties have been acknowledged in the context of the "Rescue Fly" project, and one of the most investigated areas is ensuring the healthy state of the drone propellers. This research work aims to develop an automated system that can classify or detect the propeller's health condition before going to a rescue operation, and the problem statement of this research work is image classification for drone propeller inspection using deep learning. After conducting research and testing, several problems were attempted to be identified in order to design the solution. The problems are described below –

In most cases inspection of the propeller's health condition was carried out by manual visual inspection or applying ultrasonic testing. The manual inspection process is very time-consuming, and it might be dangerous for the inspector at times. On the other hand, the problem has emerged more acute due to the distant location of drone hangars. In most emergencies, releasing the drone on a rescue attempt is not feasible unless a propeller mechanical issue is carefully monitored. Besides, the expense of inspection also increases due to the long distance. Ultrasonic testing, such as acoustic, charge-flow (current) and vibration-based solutions are also expensive due to their sensor. Moreover, the essential problem with this type of solution is its accuracy. The accuracy rate is not satisfactory which would allow the drone to fly for rescue operations. Overall, these approaches are not suitable for getting an accurate inspection result on the basis of time, accuracy and cost. Ensuring the safety of healthy propellers is the uttermost priority before going to the rescue operation. In that sense, the diagnosis of the propeller by using an ultrasonic approach is not a trustable and worthy solution. Therefore, it is a big challenge to develop a solution with a new method that will give maximum accuracy compared to other methods for inspecting the propellers.

Another major problem with this research work is the scarcity of propeller image data. Despite extensive research, no propeller-related datasets were found, either online or offline, that could be used for further research. This has been a significant challenge and causes concern in preparing a high-quality dataset.

Finally, the challenge is to find a solution that is not only effective but also practical for use in real-world scenarios, and that can provide results with a high degree of credibility. The proposed solution must be reliable and precise, as accurate inspection of drone propeller faults is crucial to ensure the safe operation.

1.2 Motivation

This thesis aims to secure the flying safety of the drone before a rescue operation. In order to ensure the flying condition of the drone, it has to be assured that the drone components are functioning absolutely. Among these components of the drone, the propeller's health must be ensured first and foremost. A drone must have a healthy propeller and it is a pre-condition for rescue flight operation. Hence, the inspection of propellers is done by applying the traditional methods in hangers. Most of the methods are very time-consuming and expensive. Besides, they are dependent on different kinds of sensors. It is a very challenging task to provide an instant solution when issues happen during the diagnosis time and sometimes act as a risk factor for the non-technical inspector.

Currently, several kinds of industrial inspection have been done by applying image processing techniques. Although the automated inspection system has already achieved an accepted range of accuracy, it still demands more research to reach the highest level of the expected result. Furthermore, the field of deep learning has become a new era to provide a more accurate solution for inspection tasks. Regarding this aspect, there has been no research on making an automated system for drone propeller inspection based on image processing and deep learning techniques. Moreover, not only for the propeller diagnosis system but also for it is possible to inspect other drone components by using these current trend technologies. Scientifically it has already been proved that image-based deep-learning solutions can predict the maximum accuracy for other diagnosis or inspection-based systems.

The principal motivation for this research work is to create an automated image-based inspection system for detecting drone propeller faults before a rescue mission which could be able to control the disastrous consequences of drone crashes. Furthermore, this system will help to control all the remote hangars from a centralized base station. This system will reduce the physical attachment of humans as well as from risky circumstances. Since the solution is being built on a centralized base solution, it will save both time and cost. For developing an image-based inspection system, a good and balanced dataset needs to be created to solve the data scarcity problem. Besides, it can provide the starting point for future research related to leveraging more accuracy of propeller fault inspection. A better model is built and eventually, the traditional drone propeller inspection method can replace by using computer vision and deep learning methods. So overall, the proposed method can diagnose the propeller fault and increase the work efficiency for the inspector by developing a central based solution. It is also competent to reduce the outlay compared to others existing solutions.

1.3 Thesis Structure

This section gives an overview of this research topic and is organized into several chapters that cover everything from the principles of computer vision and deep learning. The chapter introduction aimed to denote the entire project's scope and explain how this research project is worthy of it. The following is a brief description of each chapter's objectives:

Chapter 1. Introduction: provides a context and current researched solution trends on this thesis topic. This chapter explains the main problem statement to focus on during the research work and expresses the motivation for a fruitful outcome.

Chapters 2. Background Knowledge and Fundamentals: discusses the core fundamentals of computer vision, image processing, and CNN. It also explains the object classifier and detector architecture along with its components. It includes a general overview of the deep learning strategies, including learning strategies, which would be essential to comprehend the thesis's objective.

Chapter 3 - State of The Art: includes a review of relevant literature and research areas that focus on topics related to the thesis. It clarifies a comparative analysis of the benefits and drawbacks of the various approaches covering defect inspection.

Chapter 4 - Methodology: explains the current technology and architectural concept. The proposed solution based on the literature review and the state of the art is explained thoroughly in this chapter. The data preprocessing section provides an explanation of dataset creation. The selected classifiers and detectors section provides an overview of the DNN architecture employed in this research work and also discusses the active deep-learning framework. Finally, the software deployment section is explained to deploy the solution in the production label.

Chapter 5 – Implementation: explains the implementation details of propeller fault inspection. A brief overview of system requirements for both hardware and software used to enforce the model is provided. Furthermore, different computing platforms utilized to train, test, and evaluate the models are also explained.

Chapter 6 - Results and Evaluation: describes the results of analyzing and evaluating the output obtained after implementation. The accuracy of trained models following different strategies is calculated and explained in this chapter. Additionally, the execution and inference times of different platforms are also mentioned.

Chapter 7 - Conclusion: summarizes the overall thesis work and explains potential future scopes based on the outcomes of the implementation.

2 Technical Background

To understand the concept of this thesis work, one needs to grasp the technical background of all the methods used. This chapter explains the basic knowledge of convolutional neural network's (CNN) architecture briefly in this chapter. CNN is the backbone of object classifiers and detectors. Here, the architecture of object classifier networks and object detector networks are discussed along with their major components. The deep learning strategies for the training model, such as active learning and passive learning are also described in this chapter.

2.1 Convolutional Neural Network (CNN)

A powerful technology that refers to a multi-layered artificial neural network (ANN) known as deep learning (DL) has evolved over the past few decades. The remarkable advancements in deep learning technology have been driven by convolutional neural networks (CNNs) or "ConvNet". Nowadays, CNN is frequently used for computer vision-related tasks like image classification, object detection, pattern recognition, and so forth [39].

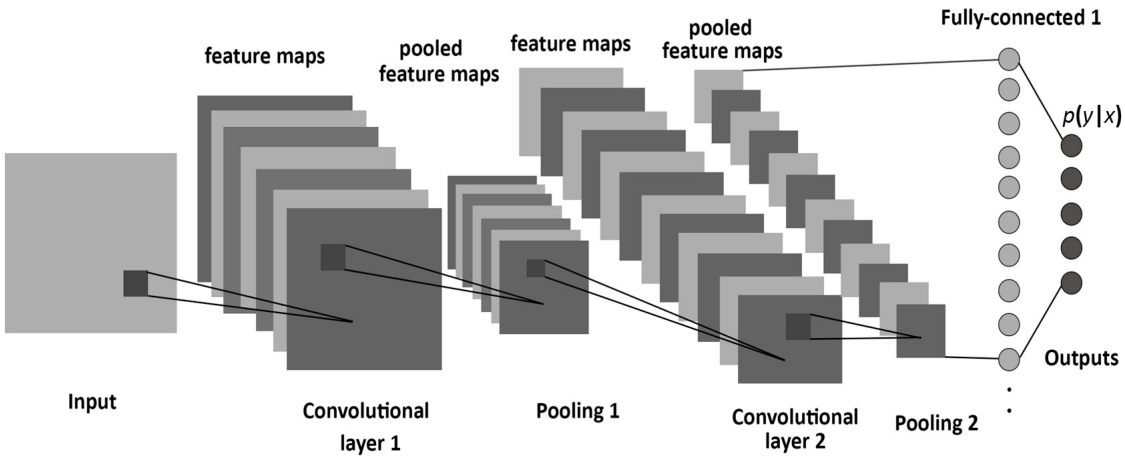


Figure 2.1: The architecture of a CNN, consisting of basic three layers [4]

An overview of a convolutional neural network's general architecture is shown in figure 2.1. It analyze massive volumes of input in a grid structure and extract important granular key features for detection and classification. CNNs generally have three layers: a convolutional layer, a pooling layer, and a fully connected layer.

Each layer has a distinct function, executing a specific task on the data it collects and learning progressively complex things. A brief discussion has been given below about the convolutional neural network layers.

2.1.1 Convolutional Layer

A convolutional layer is the key component or a core building block of a CNN. The design philosophy of this layer is to extract various features from an input image using the kernels or filters and give information such as edges and corners. This feature extraction is usually accomplished using linear and nonlinear operations, namely the convolution operation and the activation function.

Convolution

A convolution operation is a special type of linear operation which is a dot product between a filter and an image. These filters are also referred to kernels. Similarly, an image is also called a tensor. The convolutional kernels split the tensor into small pieces known as receptive fields[40]. At each position of the tensor, an element-wise product is computed between each element of the kernel and the input tensor, and the resulting value is added to the corresponding position of the output tensor, producing a feature map.

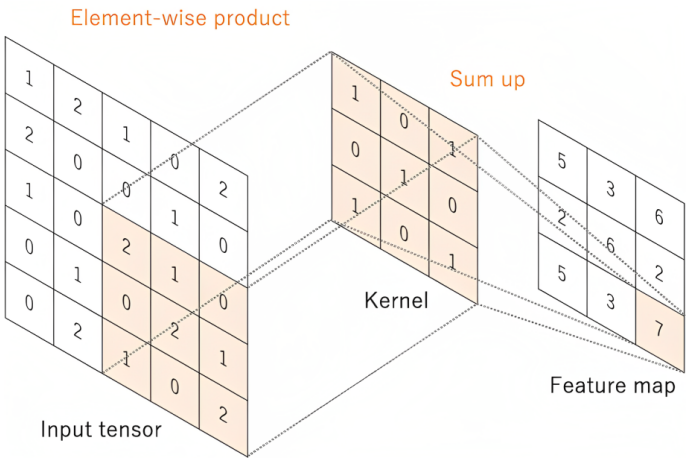


Figure 2.2: A convolutional operation example[5].

Figure 2.2 represents a simple input image dimension of a 5x5 matrix. The size and number of kernels are two essential hyperparameters that regulate the convolution operation. A kernel, dimension of 3x3 matrix has been utilized as an example in above figure.

$$cf_l^k(p,q) = \sum_c \sum_{x,y} i_c(x,y) e_l^k(u,v) \tag{2.1}$$

Here, the input or image tensor is I_C and the element of this tensor $i_c(x, y)$ is multiplied by $e_l^k(u, v)$ index of k_{th} convolutional kernel of the lth layer, called element-wise multiplication and $cf_l^k(p, q)$ is defined as a feature matrix of the elements of all channels[40].

The structure of the convolution layer and the volume of the feature map rely on three hyperparameters: depth, stride, and zero-padding[41, 8]. The number of filters applied in the preceding layer is denoted by the depth of the output volume. In the

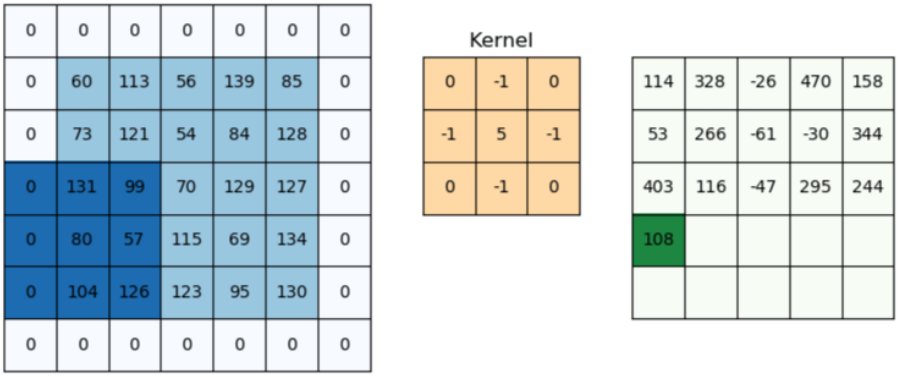


Figure 2.3: A convolution operation example with stride 1 and zero padding[6].

first convolutional layer, the raw tensor is given as an input and different neurons along the depth dimension can be activated in response to differently oriented color blobs or edges. The filter slides depending on the stride value. If the stride is 1, the filter slides one pixel at a time, and so on. In the above figure 2.3, it adds zero rows and columns to the outside of input tensor and retain the equivalent dimension for output feature map through convolutional operations.

Activation Function (Nonlinear)

A nonlinear activation function is then applied to the output of convolution or linear operation. The activation function acts as a decision-making function and assists the understanding of complex features. The learning process of convolutional neural network (CNN) relies on the appropriate non-linearity function. The activation function, which is convolved with feature map can be described by a mathematical equation.

$$T_l^k = g_a(F_l^k) \tag{2.2}$$

Here, F_l^k is the result of a convolution that is given to the activation function $ga(.)$ with adding non-linearity and the output of T_l^k, which is transformed and returned to the lth layer. There are different types of activation functions are available, such as tanh, sigmoid, rectified linear unit (ReLU), Leaky ReLU, maxout, ELU, Softmax, SWISH and so on[7, 40]. The most commonly used nonlinear activation function is ReLU, which calculates the maximum value between 0 and x[5].

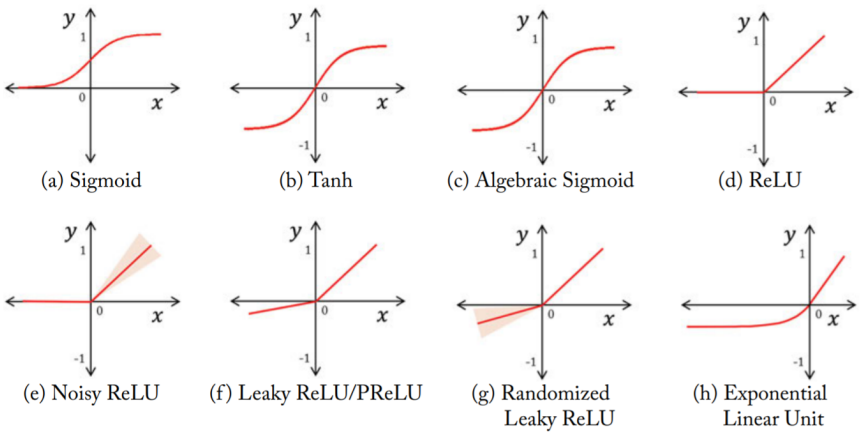


Figure 2.4: Common activation functions applied in CNN[7]

2.1.2 Pooling Layer

After obtaining a feature map in the convolutional layer, a pooling layer is added to downsampling the input tensor. This layer is used periodically in between succeeding convolutional layers. The purpose of this layer is to reduce the size of the spatial domain or convolved feature map, which means reducing the subsequent learnable parameters on the input and increasing the computational speed in the network.

Max pooling and average pooling are the two types of pooling that are available. In the max pooling operation, the filter sweeps over the entire input tensor and returns the maximum values. On the other hand, the average pooling operation returns average or mean values across the receptive fields.

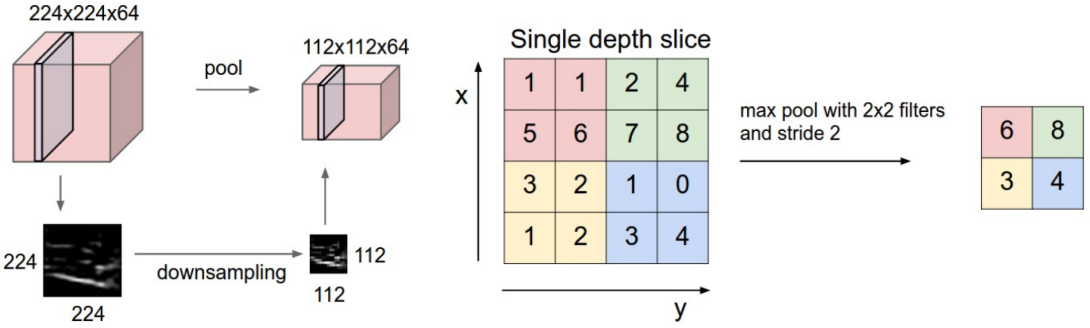


Figure 2.5: Pooling Layer[8].

A representation of the pooling layer has shown in figure 2.5. The input image size in 2.5(a) is (224x224x64) pixels. A 2x2 size filter was applied for the pooling operation to create an output tensor (112x112x64) pixel in size. There is no change

in the depth (64) of the volume. Max-pooling down sampling operation with a stride of 2 has been demonstrated in 2.5(b). Each operations was carried out over 4 numbers, which is (2x2) square[8].

2.1.3 Fully Connected Layer

A fully connected (FC) layer consists of several final layers of a CNN or ConvNet architecture, combining the activation of all previous layers. The high-label feature map of all convolution or pooling layers is then flattened into a 1D vector and connected with one or more FC layers. These layers are also known as dense layers. All connections between input and output are connected by their weights matrix and applied a linear transformation in the dense layer. The fully connected layer classifies the input probabilities and provides a range of probabilities from 0 to 1 using a nonlinear activation function called softmax[5, 42]. A mathematical equation has given below to understand fully connected layer:

$$y_{jk}(x) = f(\sum_{i=1}^{n_H} W_{jk} X_i + W_{j0}) \tag{2.3}$$

In the above equation, weight matrix (W) is multiplied by the input vector (x) using the dot product, and also bias is added to the nonlinear function[43].

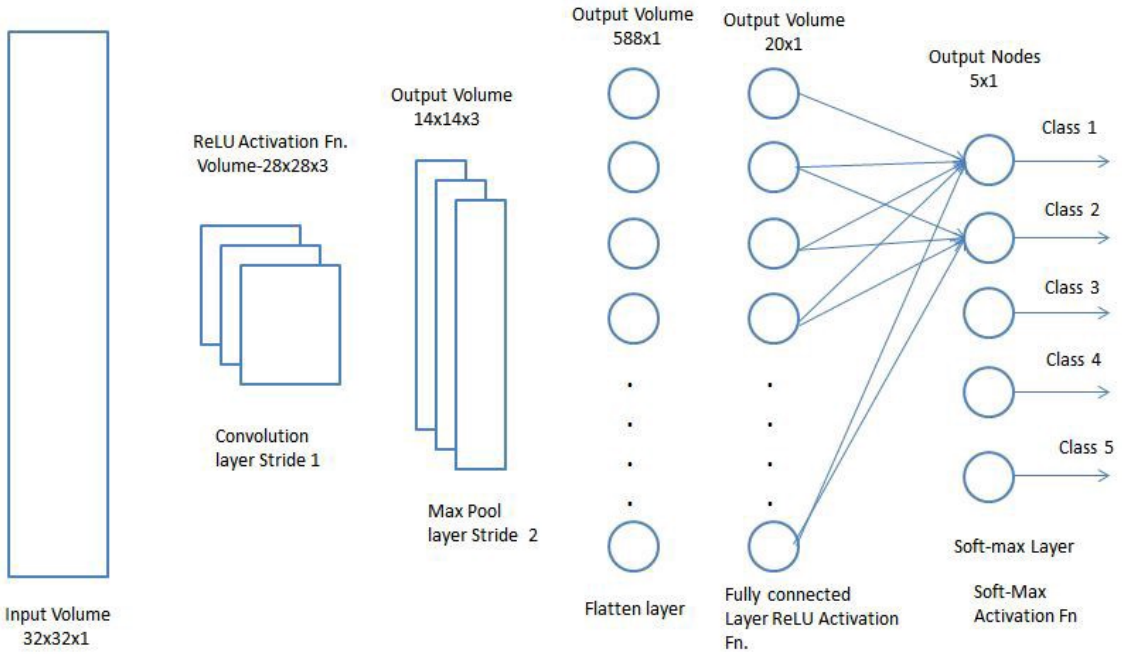


Figure 2.6: An overview of a fully connected layer. [9].

Figure 2.6 provides an illustration of a fully connected layer. It can be seen that a fully connected layer resembles a convolution layer or a normal neural network

layer. The convolution layer's connection to the input is limited to a very small input segment, which is the main difference with fully connected layer. Both of the layers calculate the matrix dot products, since they have the similar linear functionalities in their computation.

2.2 Object Classifier and Detector

Due to the incredible technological developments, researchers have pursued significant use of artificial intelligence. Nowadays, classification and detection applications are the most reliable solutions for all security monitoring and inspection systems. The task of this research is also to provide an automated system for drone propeller inspection through the most dependable solution based on a comparison of experimental results between classification and detection algorithms. A brief discussion of classification and detection algorithms has given below.

2.2.1 Object Classification

The method of categorizing and identifying sets of pixel values inside an image in accordance with applying some specific rules is known as classification. In other words, it is a process to ensure that unclassified object of images are classified into specific categories. It is possible to develop the classification criteria using spatial frequencies or image features.

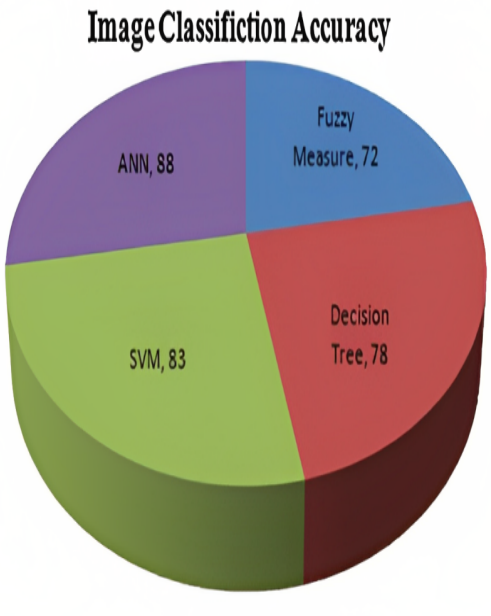


Figure 2.7: Comparison between classification techniques[10].

There are several techniques for image classification, such as artificial neural network (ANN), support vector machine (SVM), fuzzy logic, and genetic algorithm. All algorithms have both benefits and limitations in performance, and these individual techniques are always applied depending on the specification or requirements of the problem statement. According to the figure 2.7, it is shown that the artificial neural networks has gained the highest accuracy in comparison to others. However, the accuracy of the current algorithm depends on having sufficient labelled data[10]. Graph-based learning algorithms yield good accuracy but have complex computation. The accuracy of an algorithm is based on the selection of samples and taking spectral information. DNN-based classification algorithms and their architectures will be discussed in detail later in Chapter 4.

2.2.2 Object Detection

Object detection refers to the classification including localization of objects in an image or video frame. Specifically, localization refers to the location of an object by a rectangular bounding box. In a nutshell, the object detection model takes an input image and returns the same image with bounding boxes and labels to all detected objects. Here, the below figure 2.8 depicts the relation of collective computer vision-related tasks, such as object recognition, detection, and segmentation.

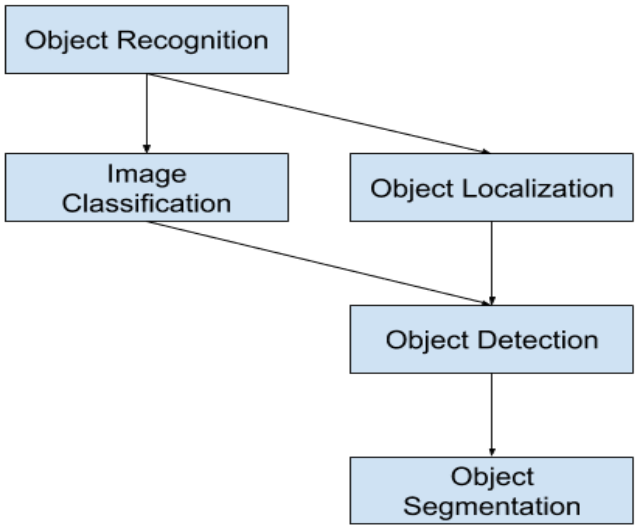


Figure 2.8: Difference between object recognition, detection, and segmentation[11].

In machine learning, traditional and deep learning-based object detection models are available. In this research study, deep learning-based models have been applied and a brief discussion of its key components is given below.

Core Components

The key elements or building blocks of deep learning approaches for image-based object detection are: 1. Backbone Networks 2. Region Proposals 3. Anchor 4. Object Classification 5. Bounding Box Regression 6. Loss Function 7. Non-Maximum Suppression (NMS)[44].

Backbone Network

In general, the backbone network refers to the primary network architecture that is typically employed to extract high-level information or features in a given task. The input data is first passed through the backbone network, which is typically comprised of several layers of convolutional neural networks (CNN). These layers are designed to gradually extract more abstract and high-level features from the input data using various filters and transformations. Deep neural image classifier networks are used to build backbone networks. They eliminate the final classification layer and use the remaining layers as the backbone of a network. Detection layers are added to backbone networks for creating comprehensive object detectors. The backbone network is typically combined with additional specialized layers such as region proposal networks (RPNs) or fully convolutional networks (FCNs). High computing efficiency and detection accuracy are the main goals of backbone networks. Some well-known networks are VGGNet, ResNet, GoogleNet and etc [44].

Region Proposal

Region proposal is one of the key components of deep learning. It alludes to the method of identifying the interest of object localizations within an image. It takes an image input and gives the output of the region of interest (ROI) and a confidence score for object classification[44]. The architecture of RPN is based on the small networks on top of the convolutional layers. It acts as a sliding window to find the region proposals of the features. Then these features are fed into the classification and box regression layer that predicts the binary classification for objects in the selected regions[45, 46].

Anchors

Anchors use the feature map from the output of CNN instead of using filter pyramids to identify the different shapes of bounding boxes from an input image. They are a collection of pre-determined bounding boxes with different scales and ratios that are frequently located on the feature maps. Anchors are projected onto different points on the feature map to match the input images to the ground-truth bounding box. [44].

Object Classification and Bounding Box Regression

The aim of object classification is to estimate the probability of several class labels of an object or a given region of interest (ROI). The object classification component is kin to a standard classification problem applied to each RoI rather than the entire images because the localization problem is dealt with via bounding box regression. Producing effective results for the object classification and bounding box regression tasks share the same backbone network.

Loss Function

A mathematical function called a loss function or cost function, is used to calculate the difference between the predicted output and actual output. There are different types of loss functions available. For example, the softmax loss function is applied to minimize the loss of classification and smoothL1 is used for calculating the bounding box regression loss[44]. Also, the focal loss is used to improve the balance between the foreground and background of an object [47].

Non-maximum Suppression (NMS)

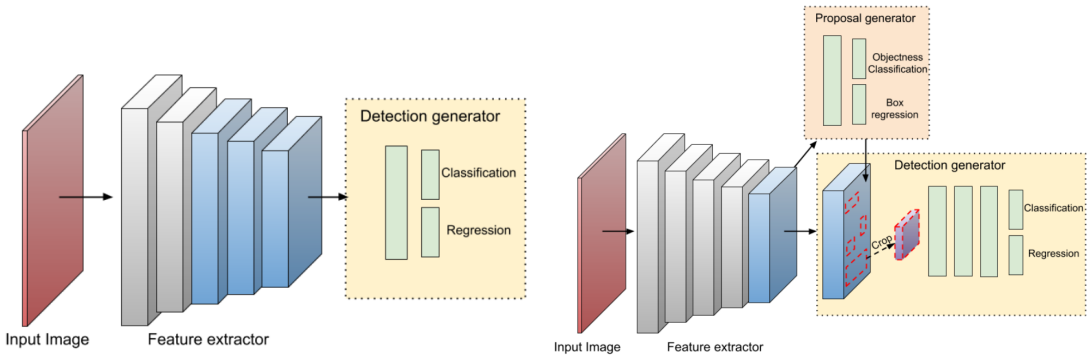
NMS is a method for eliminating redundant, overlapping bounding boxes and keeping only those with the highest confidence score in tasks involving object detection and localization. It is used as a post-processing step in the inference phase[44]. NMS is applied to remove unnecessary boxes and keep only the one with the maximum confidence score. It contributes to reducing the number of false positives and improving accuracy.

2.2.3 Anchor-based Detectors

Anchor-based detectors rely on pre-defined anchor boxes, which are rectangular areas of various sizes and aspect ratios, to recognize objects in an image. They are classified into two state-of-the-art categories: one-stage and two-stage detectors.

One-stage Object detectors

One-stage or single-stage detector is suggested as the simplest method for object detection. In this method, the detector consists of a single feed-forward FCN that can generate bounding boxes and classify objects[12]. As a result, a one-stage algorithm flow is more straightforward because it does not require developing a region proposal as it is described in figure 2.9(a). The detection problem is treated as a regression-based problem, which is its main feature. As a result, both inference and training become faster and more effective [48].



(a) One-stage detector (b) Two-stage detector

Figure 2.9: Anchor-based (a) One-stage and (b) Two-stage object detector[12].

Two-stage Object detectors

The detection problem is divided between the region proposal stage and the detection stage in a two-stage object detector. This method was presented by the pioneer of deep learning-based R-CNN[49] that combines CNN features with the region. In the first stage, it suggests or proposes a number of object candidates known as regions of interest (RoI) by applying anchors or reference boxes. This is known as the region proposal stage and the objective is to generate regions where objects can exist. The second stage involves classifying the suggested ROIs and optimizing their localization as shown in figure 2.9(b). The candidate region proposals are categorized into different classes throughout this stage. It generates the probability of the class as well as refines the region locations[44].

2.3 Deep Learning Strategies

The purpose of deep learning is to assess data using a logical framework resembling the human brain based on a set of inputs commonly referred to as training data. The training process can be done by applying different learning strategies. Supervised learning is best known and requires labelled training data. Several techniques are available for supervised learning, depending on how the dataset will be used to train the model, such as passive learning and active learning. This section describes the learning strategies and working principles, as well as their benefits and drawbacks.

2.3.1 Passive Learning

Deep learning has evolved as complex and sophisticated computational models, which are made up of several processing layers, to learn feature representations from training data at multiple abstraction levels [50]. Traditional or passive learning is the most common strategy used to train deep learning models. According

to this approach, a sizable dataset is acquired, labelled by humans, and then fed to the model for training. The model is trained on the complete dataset, and the parameters are adjusted to minimize the difference between the expected and true results.

The positive aspect of passive learning is that it is simple to implement and requires little human involvement. However, the main drawback is that it demands a substantial amount of labeled data, which can be costly and time-consuming to acquire, and it takes into account the exponential growth of data in the modern world[13]. Figure 2.10 shows the exponential growth of data for the current decade, which will lead to data labeling problems.

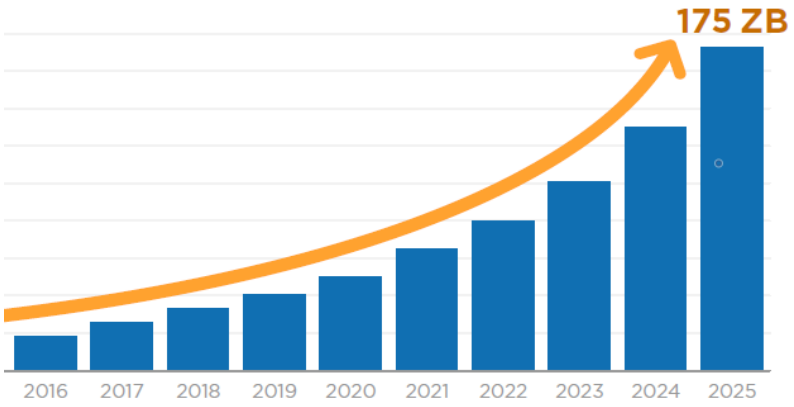


Figure 2.10: Data growth forecasting for the current decade [13].

2.3.2 Active Learning

Active learning (query learning)[14] is a more advanced technique for training deep learning models, which involves selecting the most informative data samples for labelling to improve the model's accuracy. In this approach, the model is trained on a small initial set of labelled data, and then the model is used to predict the labels of unlabelled data. The most informative data points are then selected for labelling by humans, and the model is retrained on the newly labelled data. This process is iteratively repeated until the model achieves the desired accuracy. The aim of the active learning method is to decrease the number of labelled data points required while maintaining or improving model accuracy.

Three categories of active learning strategies can be distinguished based on the application scenario: membership query synthesis, stream-based selective sampling, and pool-based active learning. Figure 2.11 has shown the scenarios of active learning strategies.

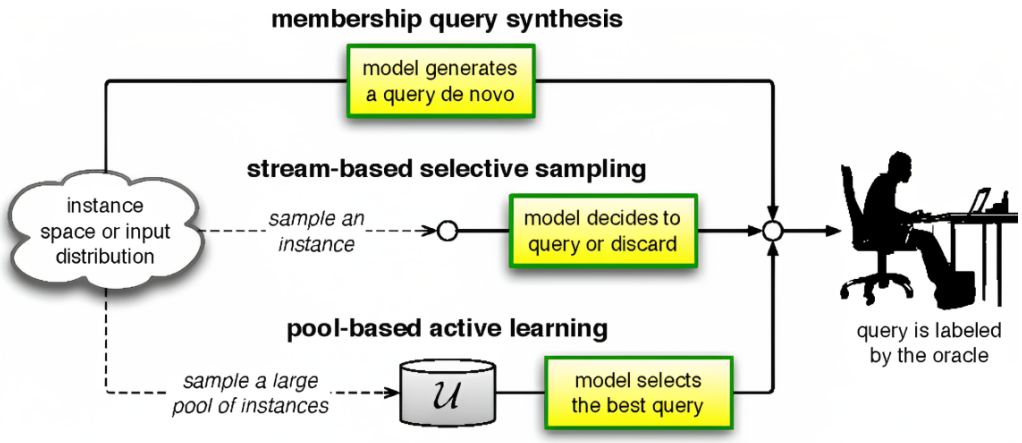


Figure 2.11: Three main active learning scenarios [14].

Membership query synthesis is among the earliest active learning techniques to be studied. In this scenario, the learner has the option of requesting labels for any unlabeled instances in the input space, adding queries that the learner produces entirely from scratch as compared to those that are derived from some underlying natural distribution. The upside of this approach is its flexibility with challenges where it is very simple to create a data instance. However, the downside is the strong possibility of data misidentification[51].

Another strategy called stream-based selective sampling assesses all unlabelled data points by individual ones and correlates their informativeness to the query parameters[14]. For each data point, the learner freely chooses whether to utilize a label or ask the teacher a query. This kind of approach is frequently applied in lightweight solutions, such as embedded devices with constrained resources. Despite being an inexpensive strategy, this methodology has a significant drawback, namely the confined efficiency because of discrete decisions.

On the other hand, the most popular scenario involving active learning is the pool-based sampling. In this strategy, the algorithm makes an effort to assess the complete dataset before choosing the most effective query or collection of queries[51]. It is common practice to train the active learner algorithm on a completely labelled portion of the data, which is then used to decide which instances would be most useful to add to the training set for the following active learning loop. The disadvantage of this strategy is that it may demand a large amount of memory with a sufficiently computing capable machine [51, 14].

2.4 Chapter Summary

For a better understanding of this thesis work and subsequent chapters, a general explanation of the technical knowledge used in this chapter is provided.

The beginning of this chapter discusses an overview of Convolutional Neural Network (CNN) architecture. It generally has three layers. Each layer has been described in detail to understand the basic mathematical calculation. In the convolutional layer, convolve the image using kernels. Dependable hyperparameters such as depth, stride, and zero padding are used for convolution operation. After that, a nonlinear activation function Relu is applied to the output of the convolution operation. Next, the pooling layer is used to downsample the input image. It reduces the size of the spatial domain or convolved feature ma and it returns the mean value. The fully connected layer consists of several final layers of architecture. The tensor is flattened into a 1D vector and connected with one or more Fully-connected layers. These are also known as a dense layers. Finally, This layer classifies the input probabilities and provides a range of probabilities from 0 to 1 using the Softmax.

Next, the difference between object classification and object detection algorithms is discussed along with their utilization. The core components of object detection, such as backbone, region proposal, anchor, object classification and bounding box regression, loss function, and Non-Maximum Suppression (NMS) are also explained. Furthermore, two categories of anchor-based object detectors: one-stage and two-stage detectors are briefly discussed. One stage or single-stage detectors is a single feed-forward fully connected that generate the bounding box and classify objects. This detector treated detection problem as a regression-based problem. Two-stage detectors divide the region proposal stage and detection stage. In the first stage, it calculates Region of Interest(RoI) by using anchors. The second stage, it classifies the suggested RoIs and optimize the localization.

Finally, the different learning strategies, such as traditional or passive learning, and active learning are covered in this chapter. The machine learning paradigm provides a variety of learning methods depending on how the dataset is used. The whole dataset is used in the traditional deep learning method. On the opposite, active learning provides a better architecture that enables greater accuracy to be achieved while utilizing less data. These different learning strategies are described in this chapter along with their guiding principles, advantages, and disadvantages.

3 State of The Art

This chapter will describe state-of-the-art machine vision solutions for inspection where the previous Chapter 2 elaborated the fundamental knowledge of architectural components. The agenda of this chapter is to explain in detail on the effort to formulate the base for today's well-fitted inspection solution that combined with machine-vision technologies.

Section 3.1 will give an overview of the image-based inspection system. Section 3.1.1 will explain the traditional approaches and how it has transformed the revolutionized evolution of artificial intelligence (AI). Section 3.1.2 will cover a details exposition of current state-of-the-art efforts using deep learning or machine vision.

3.1 Overview of Image-Based Inspection

With the beginning of image processing and computer vision techniques, the revolution of the world has drastically changed. This sector has gained attraction to scientists over the last decades, and it is still a complex, significant and long-standing problem in the current research area. Although it has achieved remarkable advancement to a certain extent from the perspective of state-of-the-art solutions.

The advent of image-based inspection relies on image feature extraction, representation, classification, detection and so on as shown in the figure which is the cornerstone of high-label or complex solutions in computer vision, and machine learning-related tasks [21]. Considering the aspect of a potential number of notable state-of-the-art vision-based detectors' algorithmic performance and characteristics, they can be categorized into two distinct approaches. One is traditional and the other is deep learning approaches. Figure 3.1 illustrates the comparison between these approaches. Here, in (a) traditional approaches take input from different sources or domains. After that, the feature engineering is followed by the manual extraction and selection process and preparing the features to feed the classifier model with their shallow structure and predict the final output. on the contrary, in (b) deep learning approaches, both feature learning and classification models work through an End-to-End learning process without the help of an oracle (human) and give the final output. As it is very clear that the traditional approaches have limitations as compared to the deep learning approaches but these are still very common and popular solutions in a specific field. The key concern of this research project is to create an image-based inspection system using deep learning.

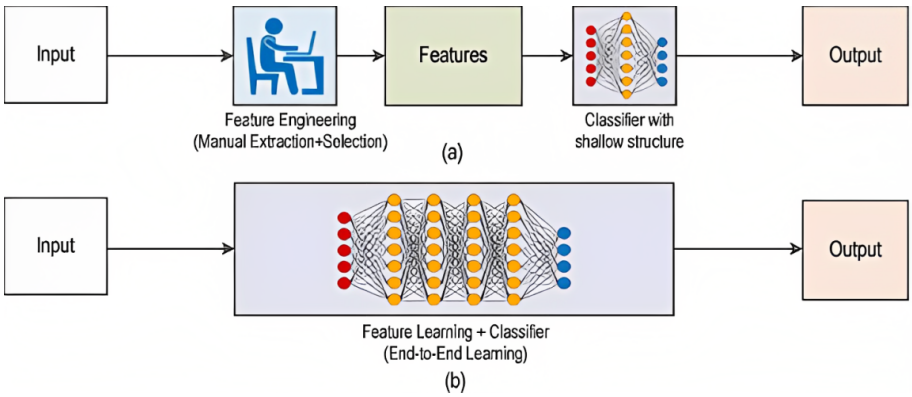


Figure 3.1: (a) traditional approaches, (b) deep learning approaches [15].

It is necessary to understand the approaches with respect to the respective field of work according to the various state-of-the-art methods. Hence, both approaches are described below comparing their working principle and performance in terms of various state-of-the-art methods.

3.1.1 Traditional Approaches

Before the era of deep learning, traditional approaches were widely used and popular for solving machine vision tasks. They are mainly included with hand-crafted feature extraction and the construction of the model in a separate way. In the field of computer vision, traditional object detector algorithms are mainly involved with the combination of region selection, and image feature extraction with classification.

The Bag of Visual Words (BoVW)[52] model is one of the most extensively utilized conventional approaches. With this approach, local features are extracted from images, clustered to create a visual vocabulary, and then each image is represented as a histogram of the frequency of visual words in the image. A classifier, such as SVM[53] or kNN[54], can then be used to classify the image. The Scale-Invariant Feature Transform (SIFT)[55] is another well-known approach. It is a feature extraction technique which is resistant to changes in lighting or illumination, scale, and rotation. It entails locating key points in the input image, extracting the local descriptors surrounding each key point, and then comparing these descriptors between images. The tasks of object recognition and image retrieval have seen extensive use of this strategy. In principal component analysis (PCA)[56], it transforms data from a high-dimensional to a lower-dimensional setting while preserving the most crucial image details. It can be used to reduce the dimensionality of image features and make them simpler to classify in the context of images.

Apart from these, traditional approaches also use several classification algorithms including logistic regression, Naive Bayes, and decision trees. These algorithms are

utilized to classify images according to their attributes or features. Despite being commonly utilized in image-based classification problems, they have some drawbacks when compared to deep learning methods. However, they can be effective in cases of limited labelled data or when a simpler model is desired.

Figure 3.2: An architecture overview of traditional object detector algorithm[16].

As described in the figure 3.2, the architectural design of the detection-based traditional approach has three prime components: region selector, feature extractor, and classifier [16]. Here, the region selector primarily employs sliding windows[57] of various sizes and ratios in order to slide on the image from the left to the right and top to bottom by a specific step size. After that the feature extractor such as SIFT[55], FAST[58], HOG [59], Haar [60], and so on are used to get the feature representative information. The final part of the classifier algorithm such as KNN [54], SVM [53], or AdaBoost [61] is used to categorize the object from an image. There are commonly used traditional handcrafted features detection algorithms, such as Deformable Part-based Model (DPM)[62], Oxford- Multiple Kernel Learning (MKL) [63], Selective Search [64], etc. The below figure 3.3 has illustrated the evolution of object detection algorithms that are transformed from traditional to deep learning based approaches.

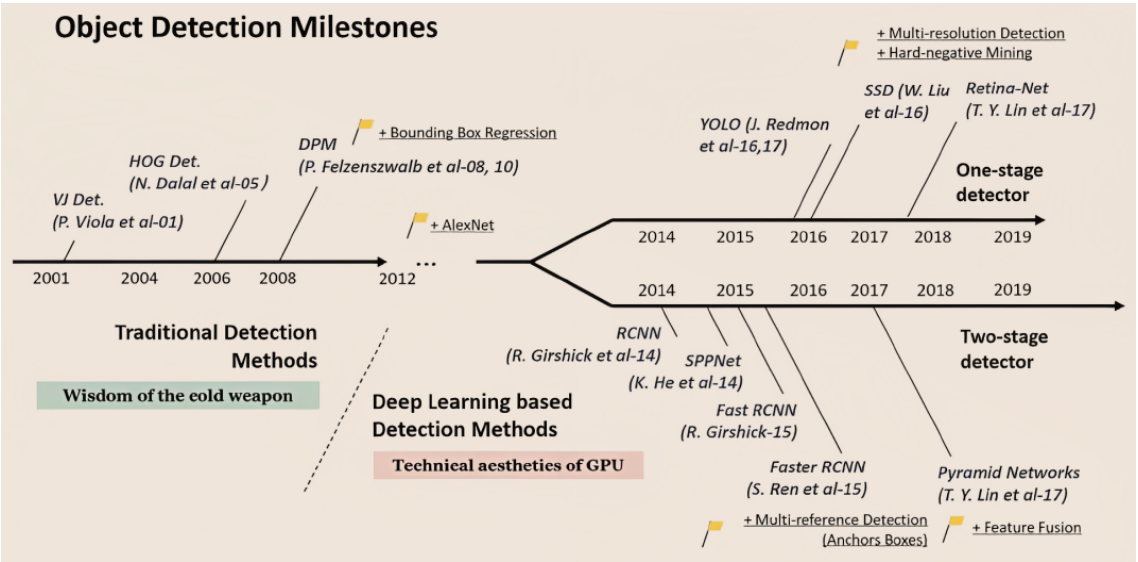


Figure 3.3: The progression of object detection milestones[17].

From the above figure 3.3, one of the most extensively used traditional detection approaches is the Viola-Jones detection algorithm introduced by Paul Viola and Michael J. Jones[65] (2001). Compared to available methods at that time, this

detector achieved enormous detection accuracy. The feature computation speed of this method is fast because of integral image properties. It involves using Haar-like features to detect objects in an image. Haar-like features are simple rectangular features that can be used to distinguish between different parts of an image. The algorithm also involves training an AdaBoost classifier to differentiate between these features and using a sliding window approach to detect objects in an image. The HOG is known as a feature descriptor and was introduced by Triggs and Dalal [59] in 2005. It made substantial progress in terms of SIFT and context of shape. It utilizes the local contrast of the overlapping normalization technique to boost accuracy while controlling the invariance of feature and non-linearity. The computation of equal space of dense grid cells is the primary task for this method. For detecting varied sizes of objects, it iteratively rescales the visual representation of the image while keeping the sliding window's constant size. The drawback of this detection algorithm is that the objects are consistently upright, with some partial occlusions and little variation in pose, appearance, illumination, and background. It does not perform well in non-rigid subjects.

The extension of the HOG detector, called deformable parts models (DPMs) is created by training on collections of deformable parts. The work of Felzenszwalb et al.[62] (2008) brought this strategy to the forefront by demonstrating that it can produce comprehensive, accurate findings on challenging data sets. Deformable parts models (DPMs) are built on the premise that objects can be thought of as collections of components. As a result, finding the components and assessing how they interact should be sufficient to detect objects. This can be accomplished by recognizing the components and respective bounding boxes, and then combining both of them into broader bounding boxes which are representing objects. After discovering the bounding boxes of objects, no maximum suppression is applied to these regions to discard the additional analysis. For practical purposes, it assigns a score to each prospective bounding box, retaining the one with the highest score, and ignoring any that would have a significant percentage of overlap with the remaining bounding box. This model is also established as a "star model"[17].

By following the DPM model, later R. Girshick (2009-2012) conducted more experiments and developed several extended DPM models called "Mixture Models" [62, 66, 67, 68]. These models aim to detect objects more precisely in real-world scenarios. A combined root filter and some part filters are the core of the DPM detector. Context priming, hard negative mining, and bounding box regression are a few of his formulated techniques. Additionally, he developed a cascade architecture that enabled significantly faster inference without compromising accuracy [17].

Multiple Kernel Learning (MKL)-based detectors were proposed by Vedaldi [63]. A single or a set of kernels with adjustable parameters is used. Finding the best combination of kernels is determined by the optimization technique. Several regularizes, including entropy-based, mixed normalization, L1 and Lp normalization

have been proposed to learn an efficient kernel combination. It may reduce unnecessary and noisy kernels. Besides, it can produce a sparse solution. L1 normalization is the one that is most frequently utilized. This method made an effort to address two important problems with object detection, namely the improvement of learning effectiveness and classifier accuracy. Although the traditional detection algorithms had inherent flaws, they were still rather advanced. However, the sliding window-based region selection method has computational costs and excessive window redundancy. Also, the morphological variability in appearance, the variation of illumination conditions, and the dense background make it difficult to construct robust features.

3.1.2 Deep Learning Approaches

In recent years, deep learning methods are gaining reliance to provide robust solutions in real-time applications such as image classification, recognition, and detection. CNN has achieved a tremendous state-of-the-art performance for classification and detection tasks. In this section, CNN-based deep learning approaches will be discussed in detail which are deeply related to this research project.

For classification and detection-based problem-solving issues, the feature is extracted using deep convolutional neural networks, known as backbone networks by applying an end-to-end learning process. These backbone networks are very complex and hungry for memory consumption. Therefore, they are trained by GPU-accelerated computers.

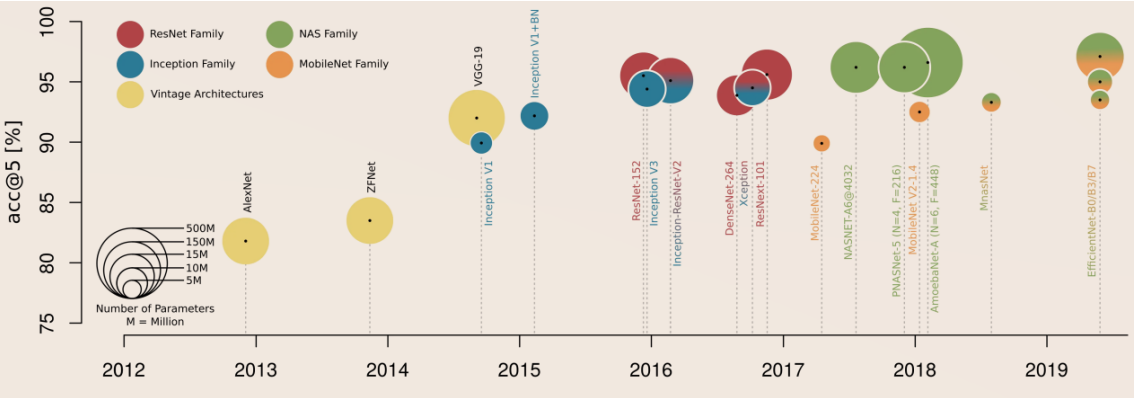


Figure 3.4: The Progression of backbones with accuracy metrics based on the ImageNet 2012 dataset[18].

Here, figure 3.4 depicts the progression of feature-extracting backbone networks with accuracy metrics and parameter counts based on the ImageNet 2012 dataset. The various colours depict the relationships between the network architectures; mixed colours denote that concepts from two distinct network families are combined.

Feature Extraction Networks

Deep convolutional neural models are composed of convolutional layers that work for extracting features consecutively by diminishing the dimensionality from the image as input data. In addition, these convolutional layers are well-known as a backbone networks. Deep learning refers to the process of building models that are deeper as more layers are added, allowing the learning of more complicated information or features. It can be classified as a Complex network(3.1) or a lightweight network according to the convolutional layers and their parameters. As their name implies, complex networks are more intricately designed. On the other hand, lightweight networks contain a small number of layers.

Backbone Network	Parameters (M)	Complexity (FLOPs)	Top-1 acc(%)	Top-5 acc(%)	Top-1 err(%)	Top-5 err(%)
AlexNet [69]	60	0.72G	57.2	80.3	42.8	16.4
ZFNet [70]	58	-	60.0	85.2	35.7	11.7
VGGNet-16 [24]	138	15.3G	71.5	89.8	28.5	9.9
GoogleNet [71]	6.8	1.5G	69.8	93.3	-	7.89
Inception-V2 [72]	12	1.9G	79.9	95.2	21.2	5.6
Inception-V3 [73]	23.6	5.72G	82.7	96.5	18.7	4.2
ResNet-50 [25]	23.4	38.32G	79.3	96.4	20.74	5.25
ResNet-101 [25]	42	-	80.1	96.4	19.87	4.60
DarkNet-53 [74]	41.57	7.14B	76.50	92.8	-	-
CSPDarkNet-53 [75]	28	5B	77.76	93.0	-	-

Table 3.1: A summery of several complex backbone networks performance is measured on ImageNet dataset.

The architecture AlexNet, developed by Alex Krizhevsky et al.[69], outperformed the preceding LeNet-5 to win the ILSVRC in 2012. It has eight layers, including five convolutional layers with max-pooling layers, and three fully connected layers with 60M parameters. The last fully connected layer produces class probabilities. The input size of the network is 227 x 227. AlexNet was widely recognized for using rectified linear units (ReLU) as an activation function and dropout regularization in the fully connected layers for solving gradient dispersion. It also used a data augmentation technique to extend the dataset for minimizing the overfitting problem. It included a parallel multi-GPU computing system to build layer communication and increase the training speed. It remains a landmark in the field of deep learning, and its success has paved the path for the development of different cutting-edge CNN architectures.

In 2013, ZfNet was released as an upgrade to AlexNet developed by Fergus and Zeiler [70], and in the same year, it won the minimum top-5 error rate at ILSVRC. The architecture consists of five convolutional layers, the first two of which differ

from AlexNet's first two layers in that they have smaller filters and a shorter stride for reducing the down-sampling rate, allowing for the preservation of more low-level information.

The Visual Geometry Group (VGG) first unveiled the deep convolutional neural network architecture known as VGGNet[24] in 2014. It was created to enhance CNN performance on image classification tasks by extending the network's depth while preserving a straightforward and standard architecture. The representation of features in the network was improved by increasing the depth of AlexNet to 16–19 layers and VGG16, and VGG19 are the two common network architectures. It utilizes a 3 x 3 kernel and stride value 1. The advantage of using a short kernel is that the network's depth is increased while the receptive field is kept constant. After the parameters are decreased, the network model's feature representation capabilities are improved. VGGNet has significantly influenced CNN development overall and is still a well-liked architecture for many computer vision applications.

GoogleNet[71] referred to as Inception v1, is a deep convolutional neural network architecture created by Google in 2014. The architecture comprises 22 layers, including a cutting-edge Inception module that enables the collateral processing of various filter sizes and pooling operations within a single layer. This module improves the visualization of the input image and facilitates the extraction of features at different scales. The term "network-in-network" or "micro-architecture" was also introduced, which refers to a tiny convolutional filter size 1 x 1 to conduct a sort of dimensionality reduction and enhance the nonlinearity of the network. It reduces the network parameters as well as increases network effectiveness. On the ILSVRC in 2014, it attained state-of-the-art performance, and since then, other deep learning architectures have been influenced greatly.

As an expansion of the first GoogleNet (Inception-v1)[71] design, InceptionNet-v2 [72] and InceptionNet-v3 [73] are both deep convolutional neural network architectures. The key feature of Inception is that it accumulates data across the various scales of the image using multiple convolutional kernels and then concatenates them to produce a better visual representation. Two key aspects set Inception-v2 apart from Inception-v1. The first one requires splitting a 5 x 5 convolution into two 3 x 3 convolutions. The second is the reduction of a n x n convolutional kernel's size into two convolutions of n x 1 and 1 x n respectively. Batch Normalization is mainly utilized by Inception-v3. Both discrete convolution and a reduction in the spatial resolution were included in Inception-v3. Inception-v4 adds a specific reduction block for changing the network width and height. Inception-v4 [76] contains more inception modules than Inception-v3, as well as a more cohesive and straightforward architecture.

ResNet, short for "Residual Network," [25] was first presented by Microsoft researchers in 2015. It was developed to deal with the issue of disappearing gradients

in deep networks of neurons, which can make the network perform less optimally as layers are added. It presents a revolutionary idea "skip connections" also known as "identity mappings," which permits information to be transferred from one layer to the next without being altered by the intermediary layers. This mitigates the vanishing gradient problem and enables the training of much deeper neural networks. It is built by the several numbers of "residual blocks," each block contains two or more convolutional layers, batch normalization, and skip connections[25]. As a result of the skip connections, the block's input is added to its output, ensuring that the gradient signal can easily travel across the network. It has been extended to ResNet-50 [25], ResNet-101 [25], and ResNet-152 [25] which vary in the complexity and number of layers, and is used as a premise architecture for many other deep learning applications.

Network	**VOC-07 (test)**	**VOC-12 (test)**	**MS COCO**	
			mAP@.5	**mAP@[.5,.95]**
VGGNet-16	73.2	70.4	41.5	21.2
ResNet-101	76.4	73.8	48.4	27.2

Table 3.2: Performance comparison between VGG-16 and ResNet-101.

In table 3.2, ResNet has achieved a better performance for both object detection Pascal VOC 2007, 2012 and COCO datasets. This test has been done by adopting Faster R-CNN object detection method. The performance has increased 3.2% in Pascal VOC 2007 dataset, 3.4% in Pascal VOC 2012 in test data accordingly. Furthermore, the mAP has 6% increased in COCO dataset detection challenges.

Darknet-53 [74] was introduced as a replacement for the previous Darknet-19 [77] architecture used in YOLOv2 by Joseph Redmon in 2018. There are 53 convolutional layers in the architecture, and they all have residual connections, just like ResNet. In contrast to ResNet, this network applies 1 x 1 convolutional layers to the feature maps to make them lower dimensional before using larger convolutional filters. It assists in decreasing the network's computational expense while maintaining excellent accuracy. In addition to the residual connections, it also has skip connections, which allow data to move directly from one layer to the next without being altered by the intervening layers. The vanishing gradient problem is decreased. As a result, the gradient flow is improved. It is a highly efficient and effective neural network architecture and contributes significantly to computer vision. Figure 3.5 illustrates the comparison of the top 1 and top 5 error rates between several networks and DarkNet-53. The lowest error rate is achieved by the DarkNet-53 network.

The architecture CSPDarkNet-53 is also called CSPNet [75], which introduces the notion of "cross-stage partial connection" which divides the network into two

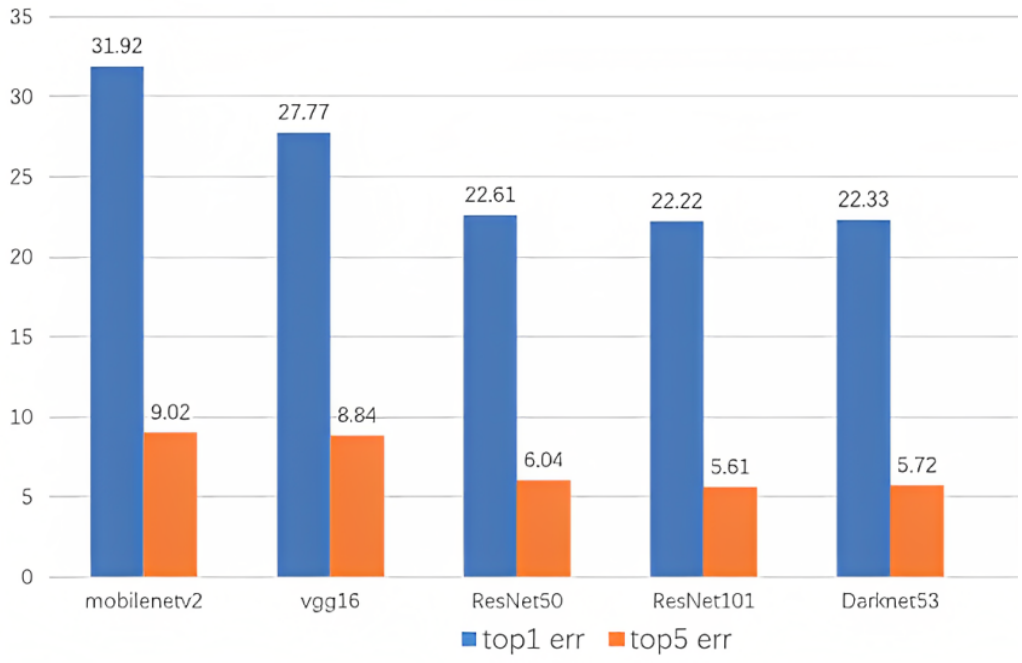


Figure 3.5: Darknet-53 network top1 and top5 error rate in CIFAR-100 dataset [19].

channels at specific stages and recombines them at a later time using partial convolutions. As a result, the gradient flow is enhanced and the cost of computation is decreased while keeping high accuracy. The fundamental idea of CSPNet is to partition the gradient flow and make it propagate over various network paths. By switching the concatenation and transition phases, it creates a significant correlation difference. Moreover, this network can significantly minimize the computation and increase both inference speed and accuracy. CSPNet-based detectors have also solved several problems such as enhancing the learning ability and eliminating bottleneck computation with memory cost reduction. In order to gather multi-scale features, CSPDarknet uses spatial pyramid pooling (SPP).

Backbone	**Top-1**	**Top-5**
ResNet-50	76	91.6
VGGNet-16	68.5	88.8
DarkNet-53	76.5	92.8
CSPDarkNet-53	78.7	94.8

Table 3.3: Accuracy comparison between complex networks for classification.

In table 3.3 has shown the accuracy comparison in imagenet dataset,while CSPDarkNet-53 backbone network has achieved the highest accuracy of top1 78.7% and top5 94.8% accordingly. Overall, 2% accuracy is improved compared to Darknet-53.

This network has contributed significantly to the field of deep learning by achieving cutting-edge performance on a number of computer vision challenges. A detailed architectural explanation will be given in the next chapter.

Applications of Backbone Networks

Among these complex backbone networks, several networks are extracting features for both image classifiers and object detectors. The table below 3.4 indicates the backbone networks involved in image classification and object detection task.

Backbone Network	**Applications**
AlexNet [69]	image classification
VGGNet [24]	image classification
GoogleNet [71]	image classification
Inception [72]	image classification
ResNet [25]	image classification
Inception-ResNet-V2 [76]	image classification, object detection
DarkNet-53 [74]	object detection
CSPDarkNet-53 [19]	image classification, object detection

Table 3.4: A summery of backbone networks according their applicable task.

Object Classifiers

As we can see in table 3.4, there are very promising and effective feature extraction networks available now and still are emerging. The depth and width of the network must be increased when effectively obtaining the representation of the features is much expected.

As this research project is closely related to a "Rescue Fly" operation, we need to develop an automatic inspection system that is capable of providing best accurate results and capable of fast computing. Considering these aspects, we can see that YOLOV5-classifier [20] model is comparatively very effective. The feature extractor or backbone of this network is CSPDarkNet-53[75]. Table 3.3 has already represented the highest achievement of accuracy compared to other complex networks. Recently, YOLOV5 has published their classification model and compared with ResNet and EfficientNet family.

Figure 3.6 illustrated the performance between the Yolov5 classifier and ResNet models. The SGD optimizer is used to train all checkpoints to 90 epochs with lr0=0.001 and weight decay=5e-5 at image size 224 with all default settings[20]. The YoloV5x-cls has achieved the highest top-1 79.4% and top-5 94.4% accuracy

Model	size (pixels)	acc top1	acc top5	Training 90 epochs 4xA100 (hours)	Speed ONNX CPU (ms)	Speed TensorRT V100 (ms)	params (M)	FLOPs @224 (B)
YOLOv5n-cls	224	64.6	85.4	7:59	**3.3**	**0.5**	**2.5**	**0.5**
YOLOv5s-cls	224	71.5	90.2	8:09	6.6	0.6	5.4	1.4
YOLOv5m-cls	224	75.9	92.9	10:06	15.5	0.9	12.9	3.9
YOLOv5l-cls	224	78.0	94.0	11:56	26.9	1.4	26.5	8.5
YOLOv5x-cls	224	**79.0**	**94.4**	15:04	54.3	1.8	48.1	15.9
ResNet18	224	70.3	89.5	**6:47**	11.2	0.5	11.7	3.7
ResNet34	224	73.9	91.8	8:33	20.6	0.9	21.8	7.4
ResNet50	224	76.8	93.4	11:10	23.4	1.0	25.6	8.5
ResNet101	224	78.5	94.3	17:10	42.1	1.9	44.5	15.9

Figure 3.6: Performance comparison between YoloV5 classifiers and ResNet [20].

with 15.9 billion floating point operations (BFLPOS) and 48.1 million (M) parameters where ResNet-101 has achieved 78.5% and 94.4% accuracy respectively. The Yolov5m-cls takes 26.9ms through ONNX format in CPU and 0.9ms in TensorRT V100 with 12.9M parameters and only 3.9 billion FLOPS for processing. While Resnet50 and 101 have taken much time and used many parameters and BFLOPS almost more than twice of yolov5m-cls parameters and FLOPS. As we know accuracy is a very important factor for this research project but also have to consider that speed is imperative simultaneously.

Object Detectors

The reinstation of convolutional neural networks (CNN) and deep learning for image classification evolved the field of visual perception by ALexNet[69], winning the ILSVRC 2012 image recognition challenge. It has given the spirit of light to conduct further research for computer vision applications. Figure 3.7 has illustrated the modern evolution of deep learning approaches that success to omit the manual feature extractions for object detection algorithms. The invention of deep learning-based methods has led to significant advancements in computer vision. Modern GPUs' increased computing power has also enabled researchers to create extremely deep convolutional neural networks, which have emerged as the most efficient way to extract accessible information from images.

Recognizing and localizing are two separate problems that comprise the object detection tasks. Based on these tasks, the state-of-the-art object detection models

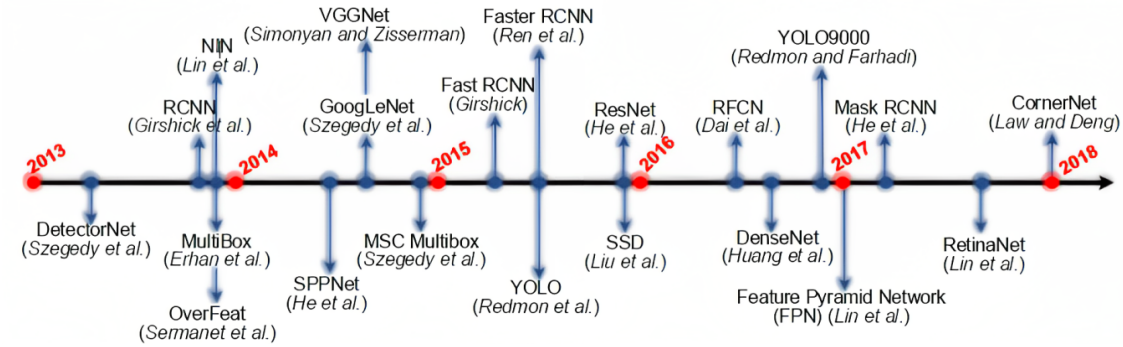


Figure 3.7: Evolution of deep learning based object detectors [21].

are categorized into two groups: 1) Two-stage detectors, and 2) One-stage detectors (shown in figure 2.9). Two-stage detectors typically achieve better accuracy but at a higher computational cost than one-stage detectors. The primary difference between these two groups of detectors is their accuracy and speed. However, this result mostly depends on the characteristics of feature-extracting backbone networks and hyperparameter tuning in the model, which is a mathematically complex task.

Two-stage Detectors

The two-stage detectors mainly distinguish two different tasks; one is for the location task, and the other is for the classification task. First, it produces the region proposals, and then the classification task is applied to that region. R. Girshic et al. proposed a network named RCNN (Regions with CNN) [49] and developed it by motivating the AlexNet [69] feature extractor backbone network. A selective search algorithm is used for selecting region proposals and giving them as input into the DCNN. The SVM algorithm is utilized for feature classification, and a bounding box regressor with NMS is applied for region refinement. RCNN significantly improved the performance compared to conventional methods with a mean average precision (mAP) accuracy of 58.5%. Slow operation is the main disadvantage of this network and takes up a lot of space due to the discrete extraction process [16].

Following a few problems with RCNN, K. He et al. proposed SPPNet[78]. A fixed-size input image was required by the previous approach, which can be troublesome when working with images of various sizes. This issue is resolved by SPPNet and employing a spatial pyramid pooling layer that generates a fixed-size output regardless of the size of the input image. This pooling layer divides the image into regions of different sizes without rescaling and pools the features in each region individually, resulting in a fixed-size representation of the image that can be fed into FC layers by avoiding an iterative calculation of the convolutional features. It has shown a better performance than the RCNN model. The drawback of this network is slow algorithmic performance, no end-to-end training, indicated expensive training time, and computation as well[16].

Detector Name	Backbone (DCNN))	Train set	Speed (fps)	VOC 2007 mAP(%)	VOC 2012 mAP(%)
RCNN [49]	AlexNet	7	<0.1	58.5	53.3
SPPNet [78]	ZFNet	7	<1	60.9	-
Fast RCNN [79]	VGG16	7+12	<1	70.0	68.4
	VGGM	7+12	<1	70.0	68.4
Faster RCNN [45]	VGG16	7+12	<5	73.2	70.4
	ResNet101	7+12	<2.5	76.4	-
	ResNet101	7+12	<5.0	76.4	-
	ZFNet	7+12	<18.0	62.1	70.4
RFCN [80]	ResNet101	7+12	<10	80.5	77.6
	ResNet101	7+12	<5.8	83.6	82.0
Mask RCNN[81]	ResNet101	7+12	<5	50.3	-
	ResNext101	7+12	<5	50.3	-

Table 3.5: Two-stage detectors performance in PASCAL VOC dataset.

With the advancement of RCNN and SPPNet, R. Girshick presented Fast RCNN in 2015[79], allowing the simultaneous training of a bounding box regressor and a detector using the same network architecture. The ROI pooling layer was added to the region proposal features. With Fast RCNN, the mAP for VOC 2007 dataset increased from 58.5% to 70.0% as well as for VOC 2012 until 68.4% has increased (as shown in table 3.5). However, there is no room for end-to-end training due to the sluggish extracting of region proposals[16].

Later that year, S. Ren et al. proposed the Faster RCNN[45], developed the Region Proposal Network (RPN), a quick replacement for the slow selective search technique and shared feature map with backbone network[16]. It provided an opportunity for comprehensive training. In addition, the speed of detection has improved and on the PASCAL VOC 2007 dataset and the PASCAL VOC 2012 dataset, it earned mAPs of 73.2% and 70.4%, respectively (Table **??**). Despite still having trouble detecting multi-scale and small objects, faster RCNN dramatically enhances object detection.

Following that, J. Dai et al. created Region-based Fully Convolutional Networks (RFCN) in 2016 [80], and K. He et al. created Mask RCNN [81] in 2017. By computing the FC of the entire image, RFCN reduces the amount of work needed for each ROI, speeding up the process. For the PASCAL VOC 2007, and 2012 datasets, it obtained a mAP of 83.6% and 82.0%, respectively. Mask RCNN is the expansion of the previous model FRCNN. It has increased the detection accuracy by replacing with the ROI align pooling layer which gains pixel-level alignment. Missing the real-time requirements for detection is the main drawback of this model[16].

One-stage Detectors

Single Shot Multibox Detector or SSD is one of the one-stage algorithm developed by Liu et al. VGG-16 has been used as a backbone network but not fully connected layers. Feature is extracted by following many scales and decrease the size of input gradually in every layer for better accuracy. Confidence loss and location loss both are calculated to get total loss. It has achieved 74% mAP at 59 fps on Pascal VOC and COCO dataset.

Detectors	Backbone Network	Size	AP[0.5:0.95]	AP[0.5]	fps
YOLO [26]	GoogleNet (Modified)	448	-	57.90%	45
SSD [82]	VGG16	300	23.20%	41.20%	46
YOLOV2 [83]	DarkNet-19	352	21.60%	44.00%	81
RetinaNet [47]	ResNet-101-FPN	400	31.90%	49.50%	12
YOLOv3 [84]	DarkNet-53	320	28.20%	51.50%	45
EfficientDet-D2 [85]	Efficient-B2	768	43.00%	62.30%	41.7
YOLOV4 [74]	CSPDarkNet-53	512	43.00%	64.90%	31
YOLOV5 [20]	CSPDarkNet-53	640	45.4%	64.1%	140
YOLOV7 [23]	E-ELAN	640	51.4%	69.7%	161

Table 3.6: Performance comparison for one-stage detectors using PASCAL VOC and COCO datasets.

YOLO (You Only Look Once) was introduced by J. Redmon in 2015 as a solution for an end-to-end one-stage detector which creates the probabilities for class and bounding box regression [26]. The image is split into a grid cell, and each cell must determine whether the object's centre is in it. This design allowed it to increase its speed significantly. According to the table 3.6, it detects an average precision of 57.9% with 45 fps on VOC 2012 and COCO datasets. It achieved real-time object detection speed but not accuracy due to its localization and recall issues. To increase the trade-off accuracy and speed, YOLOv2 improves over YOLOv1 by using DarkNet-19 as the backbone network [83]. The introduction of batch normalization and the concept of anchors has accelerated network convergence and improved detection over its predecessor.

In comparison to earlier methods, Redmon et al. introduced YOLOv3[84], which offers greater accuracy and real-time detection performance. The feature extractor DarkNet-53 [17] network, which has 53 convolutional layers, has been substituted for the YOLOv2-like overall architectural framework of this approach. It is considerably deeper and has more residual connections. A 3x3 convolutional layer with a stride of two between the two blocks is used instead of the max-pooling layer. Additional FPN has also been used for creating various sizes of feature maps and

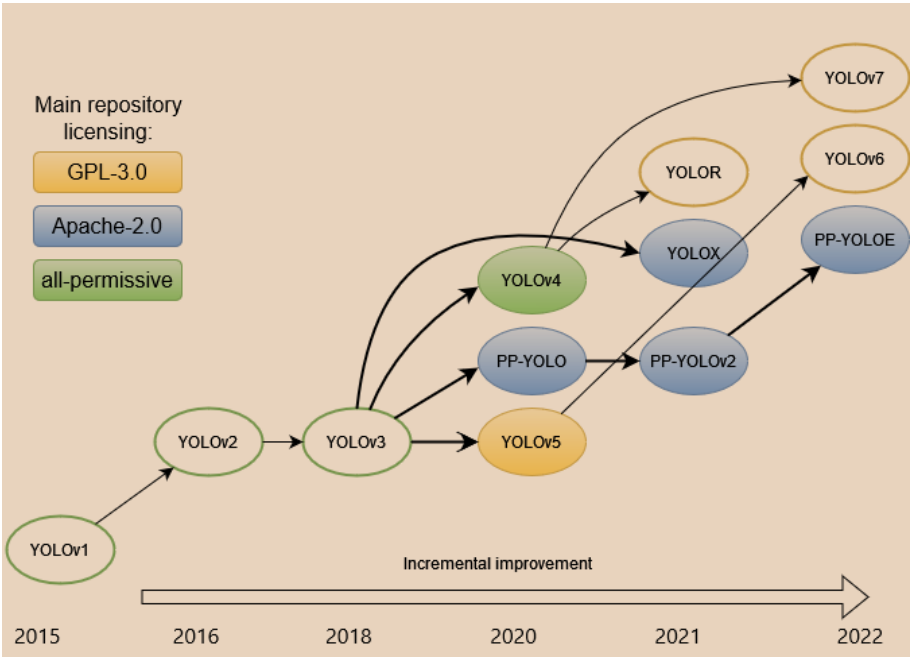


Figure 3.8: YOLO family tree [22].

achieved 45 fps detection speed with 51.5% of mAP.

YOLOv5[20] is a state-of-the-art real-time object detection algorithm developed by Ultralytics. It is the fifth version of the You Only Look Once (YOLO) family of object detectors as shown in figure 3.8, and it has been trained using a highly efficient and accurate architecture. It has used CSPDarkNet-53 as the backbone. It also uses a unique anchor-free design called YOLOv5-P6, which does not rely on predefined anchor boxes and is more adaptable and ideal for recognizing objects of different sizes and aspect ratios. YOLOv5 achieves state-of-the-art results on various object detection benchmarks while being computationally efficient and easy to use. It has reached 64.1% of mAP with 140 fps. However, it should be noted that YOLOv5 is not an official version of YOLO and has not been peer-reviewed or published in a scientific journal.

The latest version of YOLOV7 was proposed by WongKinYiu which is developed on the basis of Extended efficient layer aggregation networks (E-ELAN) [23]. YOLOv7 is a highly efficient model, achieving state-of-the-art results on popular object detection benchmarks while maintaining a fast inference speed. Its architecture is optimized for both accuracy and speed, making it suitable for various real-world applications. It has achieved 69.7% accuracy and 161 frames per second. Model re-parametrization and scaling techniques have been used. This model has used 75% fewer parameters with 36% less computation than YOLOV4 and increased 1.5% average precision [23]. Figure 3.9 has shown the highest performance of YOLOV7

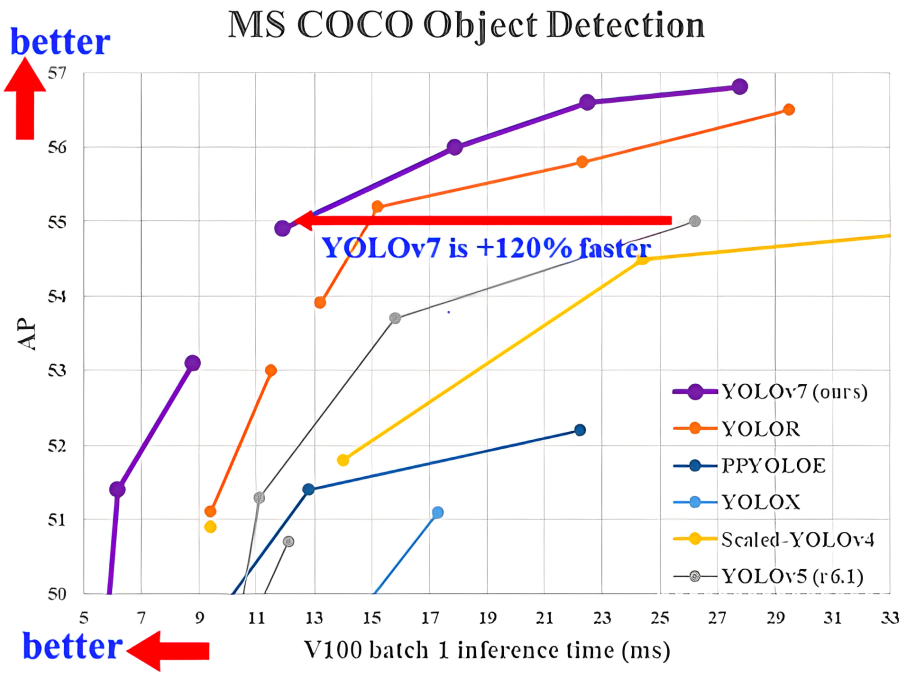


Figure 3.9: Performance comparison between YOLO's [23].

in MSCOCO object detection datasets and inference speed is +120% faster than others. In a later chapter, there are more details discussion will be given about YOLOV5 and V7 detection models.

3.2 Fault Inspection

As we know the reality of rescue fly operation, it is extremely emergent to ensure the safety of the drone components before going to operation. In most cases, drone propellers are damaged. Recently, researchers have been interested in deep learning to consider how to ensure the safety of these propellers. The traditional methods of manual visual inspection and ultrasonic testing are very time consuming and labour intensive. Sometimes, it may not be able to detect the small defect on propellers. Although several techniques such as current- based, sound-based, and vibration-based inspection were done in before. But the result is not sufficient to ensure the propellers health condition. After revolution of artificial intelligence, researchers and scientists have given their focus in this field for solving all kinds of surveillance, monitoring and fault inspection. As a result, several acoustic based solution has been given using deep learning technology. Since there is no direct research paper about image-based drone propeller fault inspection using deep learning, in that sense a similar classifier task such as Wind turbine propeller fault inspection or other detection technique is considered to search for a well-fitted and optimized solution for drone propeller fault inspection. This section discusses the state-of-the-art solutions

of traditional and deep learning based solution for propeller fault inspection.

The steady state model is proposed by Lee et al.[1] for inspecting the UAV motors fault. The classification and diagnostics of motors is tested based on the predicted nonlinear parameters of a steady-state model. DC motor is used to simplify the model assumption due to the facility of removing derivatives. The model performance is unstable due to transient state. The fault classification is tested by injecting different faults.Another drawback is about the current sensor noise. Another author G.Ciaburro[2] proposed a model based on the vibration sensor signal which is set on the propeller. After collecting data, model is trained and compared the performance with principal component analysis(PCA) and wavelate analysis. The sound-based approach is easier to acquire signals than current-based methods. In [37, 2], have applied the noise from drones to detect the propeller faulty blades using an ANN model, Fourier transform and Discrete wavelet transform (DWT) [38]. However, sound-based techniques have a drawback due to susceptible background noise.

Vibration-based methods [3] are based on acceleration data which is got from the IMU, built-in or external accelerometer and then classify the signals of rotor blades.The disadvantage of this model is to get the wrong data from different measurement sensor and the final output is classified by vibration anomaly. Another approach [36] BLDC, based on angular speed and failure data using brushless DC motors. It is calculated chaos from the system model by using the state variables. The speed is estimated and detected anomalies in the propellers motor. This chaos-based model is evaluated and compared to traditional methods such as Hall-effect sensors and back-EMF. It achieved a good result based on precision and reliability. The drawback of this model is about sensor data measurement.Sometimes, it might be given the wrong data, which leads the false prediction of this model.

According to Liu Yajuan et al.[86], they proposed a three-layer based cloud-edge-end framework for detecting blade surface damage and chose a lightweight custom YOLOV3-MobileNet-PK model. In the architecture label, the backbone Darknet-53 is replaced with MobileNet and applied the knowledge distillation process, known as transfer learning. YOLOV3-ResNet34 has been trained as a teacher model and then YOLOV3-MobileNet has been pruned as a student model. Finally, the result showed a 4% increase with 94.9% accuracy. The inference time was 35ms and the hardware Xeon 4214 (CPU) and NVIDIA Quadro 16GB RTX 5000 (GPU) has been utilized.

In M. Cruse et. al[87] proposed a model that combines the weakly and strongly supervised segmentation technique. Here, ResNet-34 is selected as backbone. In the weak segmentation part, class activation mapping has been chosen and combined with the Mask-RCNN strong segmentation part. 92% accuracy has been achieved by using this model and was trained in NVIDIA Gforce 1060 GPUs.

In [88], have developed a solution by using Faster R-CNN object detector. They have shown a comparison by replacing four backbone networks consecutively. These are Inception-v2, ResNet-50, ResNet-101, and Inception-ResNet-V2. Traditional data augmentation and pyramid patching augmentation are utilized during the model training. The highest accuracy has been achieved 81.10% of mAP@0.30. The average inference time was 2.11s per image and the hardware Gfroce GTX 1080 is used.

Another paper [89], has proposed a solution using Mask R-CNN with the backbone network of ResNetX-101. They have applied white balance, contrast enhance, gray scale, geometric transformation with the loss convergence of 0.08 and achieved 86.74% of accuracy for classification.

Sakar et. al[90] unveiled a solution using SRCNN with YOLOV3 and CSPDarkNet-53 is used as a feature extractor network. In this solution, Laplace variance distribution is applied for image separation, SRCNN model is utilized for reconstructing the blurry images and manual data annotation is done for data labelling. They have achieved 96% accuracy with 0.20s inference time per images. In another paper which is proposed by A. Foster et. al [91], used YOLOV5 model. The idea was to select image aspect ratio adaptively by applying manual data balancing and labelling technique. They have taken heavily uneven dataset for their training purpose. Finally, they have achieved 79.37% of accuracy with 82.39% precision and recall 82.12% and the fps has gained 17.6.

In shihavuddin et. al [92], another solution has been done by using YOLOV5. Geometric transformation and color temperature for augmentation technique has been utilized with combined datasets. After training the model, it has achieved only 87.3% accuracy with a two second inference rate.

D. Liao et.al [93] have developed a solution by using customized YOLOV5 model. They have applied data enhancement technique, a weighted bidirectional feature pyramid network is proposed which is replaced with PAN, FPN networks in neck. CSPDarkNet-53 has been used as a backbone network. In the neck, multi-scale features are fused by gathering different convolutional kernels. Accuracy has been achieved 97.41% by using this model.

Zhang, Rue et. al [94] proposed a solution, named as SOD-YOLO using a customized CSPDarkNet backbone network. In this solution, they have applied foreground segmentation and Hough transformation. Additionally a micro-scale detection layer has been added in backbone. The anchor box is re-clustered through K-means and CBAM attention has been used for all feature fusion layer. The accuracy of 95.1% has been achieved through this network with 41.3 frames per second. They have used NVIDIA GTX 1650 GPU for training model.

Further, other sectors are conducting research for developing an automated fault or damage detection system. For example, in [TUC1], they are doing research on insulator and vibration damper detection using YOLO, insulator burn mark detection by using cutting-edge deep learning solution and achieved a tremendous performance according to their developed models result. In [TUC2] this research project, they have experienmented YOLOV4-tiny model with CSPDarkNet-53 backbone network for insulator detector edge cutting platforms application. According to the paper, they have collected data and then categorized them. Then select a number of percentage random data from that categorical datasets and apply data augmentation while labelling the data. After training the model, the solution has been tested in embedded Jetson Nano and Xavior NX device. They have achieved 98.9% accuracy for detection.

Another research has been conducted under the Automotive Software Engineering department in TUC. In [TUC3] this paper, they have applied YOLOV5 model for experimenting the burn mark detection based on active deep learning. According to the paper, they have applied data preprocessing and augmentation techniques. After then manual data annotation was done to prepare train data. They have compared the training result by applying the active deep learning strategy during model training in each iterative step. Finally, they have achieved a tremendous accuracy of 99.4% which is inspired to apply this strategy in this research project.

Upesh Nepal et al.[95], have conducted a research about autonomous landing spot detection in faulty UAVs.According to this paper, the prime task is to find a safety location for emergency landing incase UAV faces the in-flight failure problem. They have used different version of YOLO object detection model and made a fruitful comparison between Yolov3, Yolov4, and Yolov5l. All the models are trained with large aerial image dataset, known as DOTA. After model training, they have checked the feasibility of selected models. They have got highest precision 73%, recall 41% in YoloV3, 69% and 57% in Yolov4, 70.7% and 61.1% in Yolov5l respectively. As it is shown that Yolov3 needs more improvement while v4 and v5l comparatively better. But the efficient performance is dependent on the balance of precision and recal value which is measured by F1 score. In this way, yolov5l has achieved highest 65.5% f1 score. Finally, the authors has found the best model Yolov5l which is developed by PyTorch framework.

In this research project, it is decided to apply three distinct backbone networks for image classification and compare their result by using full datasets. Additionally, two state-of-the-art solution YOLOV5 and YOLOV7 object detector will be trained by following active deep learning strategy and compare their performance using minimal data.

3.3 Chapter Summary

This chapter contains the literature reviews of state-of-the-art methods of deep learning technologies which is related to this research topic. An overview has been given about the backbone networks that works as a feature extractor or backbone networks for various image classification problems. Traditional and deep learning approaches has been described in details and shown their evolution according to their performance. A general discussion has been given about object classifiers and object detectors. Several comparison details has also been given according to the image classification challenges on different datasets. Finally, the implementation idea for this research project has been introduced which is taken from this chapter.

At first, the image-based inspection system depends on the process of feature extraction from an image. This can be done by applying traditional and deep learning approaches. In traditional, feature engineering is completed by the help from oracle (human) and the feature is fed to the machine learning models and classify the images. After evolution of deep learning, the feature engineering and model training is done by applying the end-to-end learning process and finally got the output. The traditional algorithms has three prime components. These are region selector, feature extractor, and classifier. Region selection is calculated using sliding windows and extracted features from the image tensor by using different feature extractor such as SIFT, HOG, Haar and so on. The SVM, KNN or AdaBoost is used for classification. In deep neural network, feature is extracted by using complex backbone network and the classify the image.Here, several backbone networks performances are discussed and we got that CSPDarkNet-53 has achieved the heighest Top-1 and Top-5 accuracy.Finally, Object classifiers and detectors performance has been discussed based on the different dataset results. RFCN has achieved the highest mAP among the two-stage detectors. On the other hand, YoloV5, and YoloV7 has achieved a very good average precision in different dataset and the backbone of this network is CSPDarkNet-53, and E-ELAN.

Finally, we discussed several papers related to propeller fault inspection. A brief discussion has been given about the traditional approaches as well as deep learning approaches. In traditional way, we found steady state model that is used to classify and diagnostics of UAV motors based on the nonlinear parameters. For reducing the derivatives from the current signal, DC motors are used due to its simplicity. The test has been done by injecting fault.Besides, other approaches like vibration-based or sound-based has given a good result but the disadvantages of its sensor data. There is a possibility to loss the signal data. Some solutions have been found, where image-based deep neural network is utilized. Most of the authors worked with either two-stage detector or one-stage detector. According to the papers, YoloV5 model has achieved upto 99% accuracy for the different type of fault inspection system. Besides, the cutting-edge solution like transfer learning, active learning strategies are also applied in few solutions.

4 Methodology

This chapter represents the methodology or conceptual idea of the research project. In our case, we proposed two methods: Passive learning and Active learning. Both methods are mainly divided into three stages. The first stage is data preprocessing, the second is the training stage and the last is the deployment process. Further details are given in the upcoming section of this chapter. The workflow of the proposed methodologies are illustrated in figure 4.1 and 4.2 below.

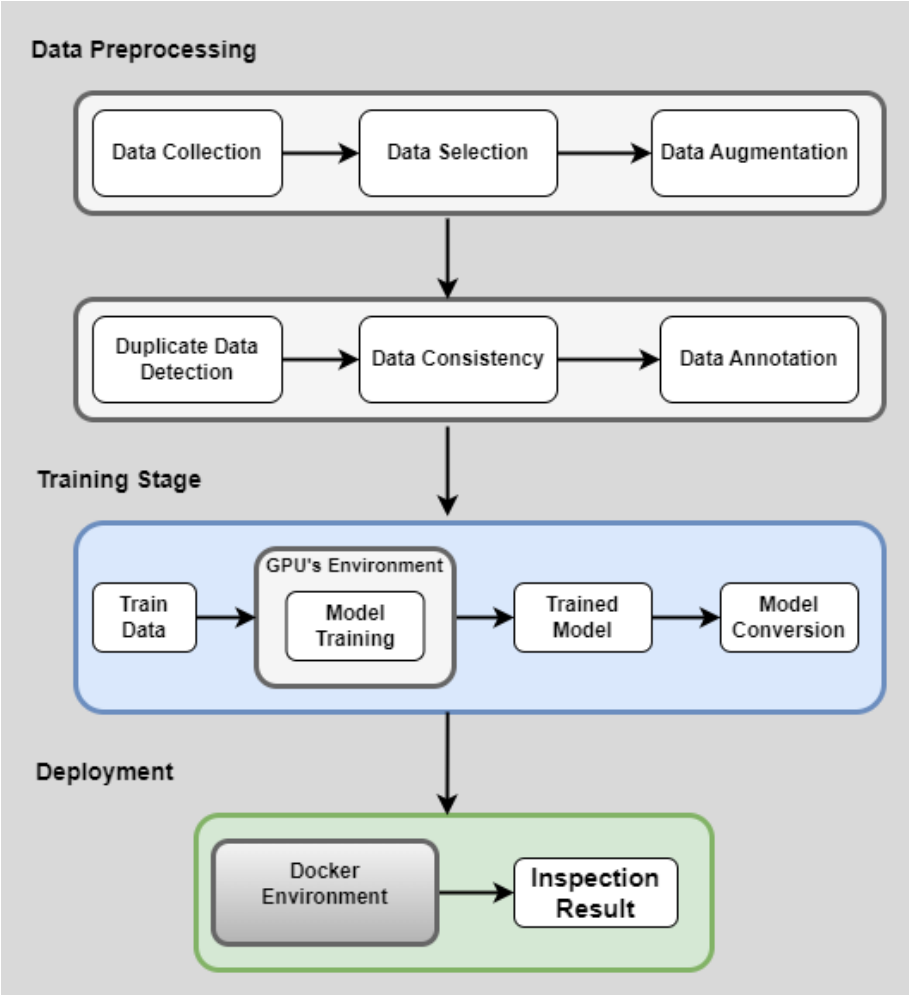


Figure 4.1: Passive learning workflow of the proposed methodology.

Figure 4.1 depicts the proposed methodology for passive learning where data is

prepared in the first stage and then send to the training stage and after finishing the training, model is deployed using Docker in the final stage.

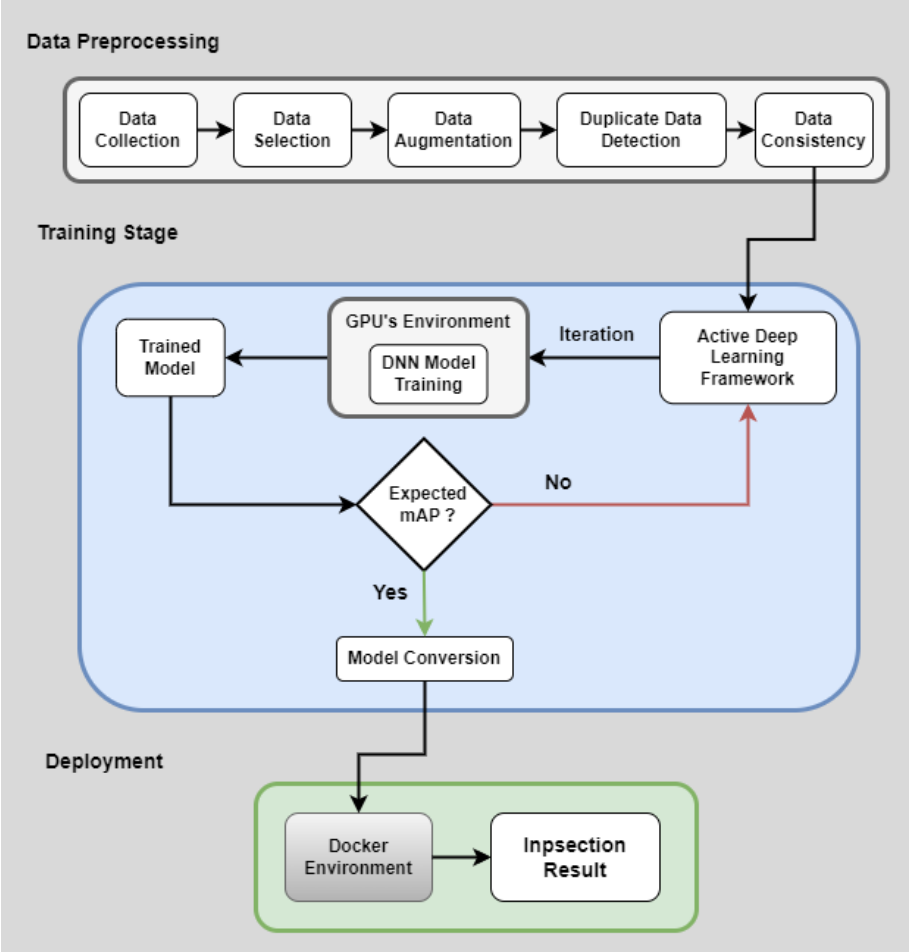


Figure 4.2: Active learning workflow of the proposed methodology.

Figure 4.2 depicts the workflow of active learning methodology where active deep learning framework is used in the training stage after data preprocessing and checked the best model performance through the iteration process. Finally, the model is converted into ONNX format and deployed via Docker to be used in any environment on the production label.

4.1 Data Preprocessing

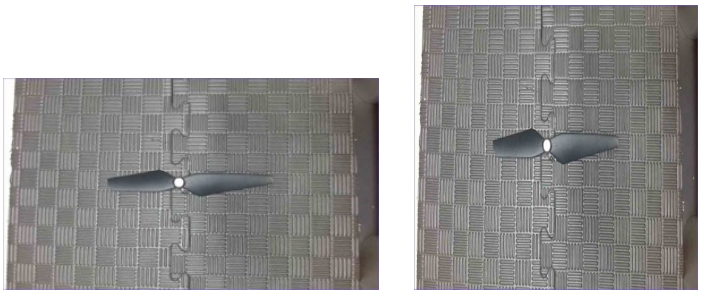
This thesis project aims to develop a solution for drone propeller inspection using deep learning. We know that learning a deep neural model requires a large volume of data. This research project deals with image-based solutions. There are no propeller related datasets available online or offline. In that sense, it is imperative to create a

proper image dataset that is playing a vital or leading role of this research project. There are many issues that are of concern while creating a dataset. The quality and resilience of the deep learning model are mostly determined by the precision in which the input data is processed. In addition to being important, cleaning up the data and getting it ready for a deep learning model enhance the model's precision and efficiency. Several approaches are used to build the model's dataset are explained in detail in this section. These are related to the collection of data, data selection, augmentation for enhancing data, duplicate data detection and remove, and data balancing of each class. The following subsections provide more in-depth information on each of these strategies.

4.1.1 Data Collection

In the Rescue Fly project, there is no prior image-based work related to the drone propeller fault inspection. Furthermore, there are no available online datasets as we can get access to utilize in our research project. So, a Logitech camera has been mounted on the indoor environment where the drones will take place before and after rescue operation. The placing of the camera is on top of each propeller with a minimal distance that covers the whole area of propeller. The images are captured considering the different light conditions and backgrounds.

It is very important to look at the shape of the object in the image while feeding these images to the selected model. Most models resize the input images according to their specified size requirements and specifically prefer square image size. In our case, we cannot feed any kind of aspect ratios of propeller images because when it resides in the model, the object of the images will shrink.

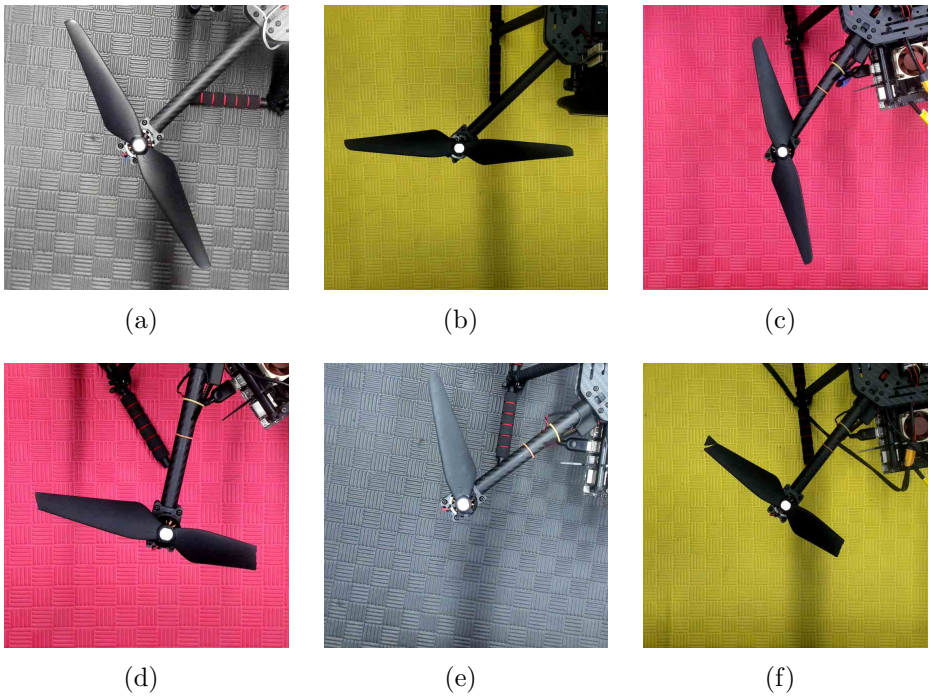


(a) Rectangular image (b) Barrel distortion

Figure 4.3: Problems of rectangular images for the selected model, a) original image b) distorted image

Since our propeller size is maintained only by the symmetrical in length, it distorts the original shape of object in the image. The one and only solution for this

problem is to choose square aspect ratio and spatial selection of an image. Figure 4.3 illustrated the problems about resizing the original rectangular aspect of propeller image which is converted to the barrel shape of distorted image in the model. Also other distortions can occur while taking images such as pincushion distortion, mustache distortion and so on. These group of distortion are also known as radial distortion. Furthermore, another distortion might be happened due to optical design in camera which is known as camera distortion. It generates the image deformity because of curvilinear bending light. The pinhole camera model is contrasted with the deviation lines. As a result, curved lines are created instead of straight lines. Therefore, it is very important task to check camera before taking images.



(a) (b) (c)

(d) (e) (f)

Figure 4.4: 720x720 size of propeller images that (a-c) represents healthy and (d-f) represents broken in different light condition and background.

We have set the square frame 720x720px. So that we do not have to face image distortion while training our model. If we set the rectangular image frame, it creates barrel distortion in the image because of propeller size. Since our model will train by converting into 640x640 sizes of images, we have considered the specific square resolution of the camera. Moreover, indoor images are acquired inside the Indoor Flight Center IFC) lab at TU Chemnitz. Several random healthy and broken propeller images with different backgrounds and lighting have been taken from TUC Lab and shown in the above figure 4.4.

4.1.2 Data Selection

The selection of the image data is a crucial phase in the dataset development process since it impacts the quality and applicability of the data used to train and evaluate algorithms based on deep learning. The purpose of data, diversity, quality, and consistency of the images are some important points to consider when creating an image-based dataset. The selected images must be appropriate for the specific task for which the dataset is being generated. If the dataset is intended for the classification or detection task, the object of interest must be featured clearly in the image. Images from a variety of contexts and variations should be included in the dataset because these are the types of things the DNN model is likely to see in the actual world. Variations in lighting, camera angles, size of objects, background, and other factors must always be considered. For learning the model properly from the chosen images, they must be of good quality and have enough resolution and clarity.

The images in the collection should be consistent in terms of format, size, and other relevant features. This can assist in making sure that the model can consistently process the selected images. After collecting thousands of raw propeller images, need to be sorted according to their quality. Although it is concerned about a good resolution of the images, the images had to be sorted in good, average, and bad categories as per their resolutions. Few images were not in the correct position or the condition of the propeller due to the different lighting, background, and cluttering environment. Bad categorical images were not considered to make a propeller image dataset. This would bias the model to give a wrong prediction and for this reason, these images were manually removed from the dataset.

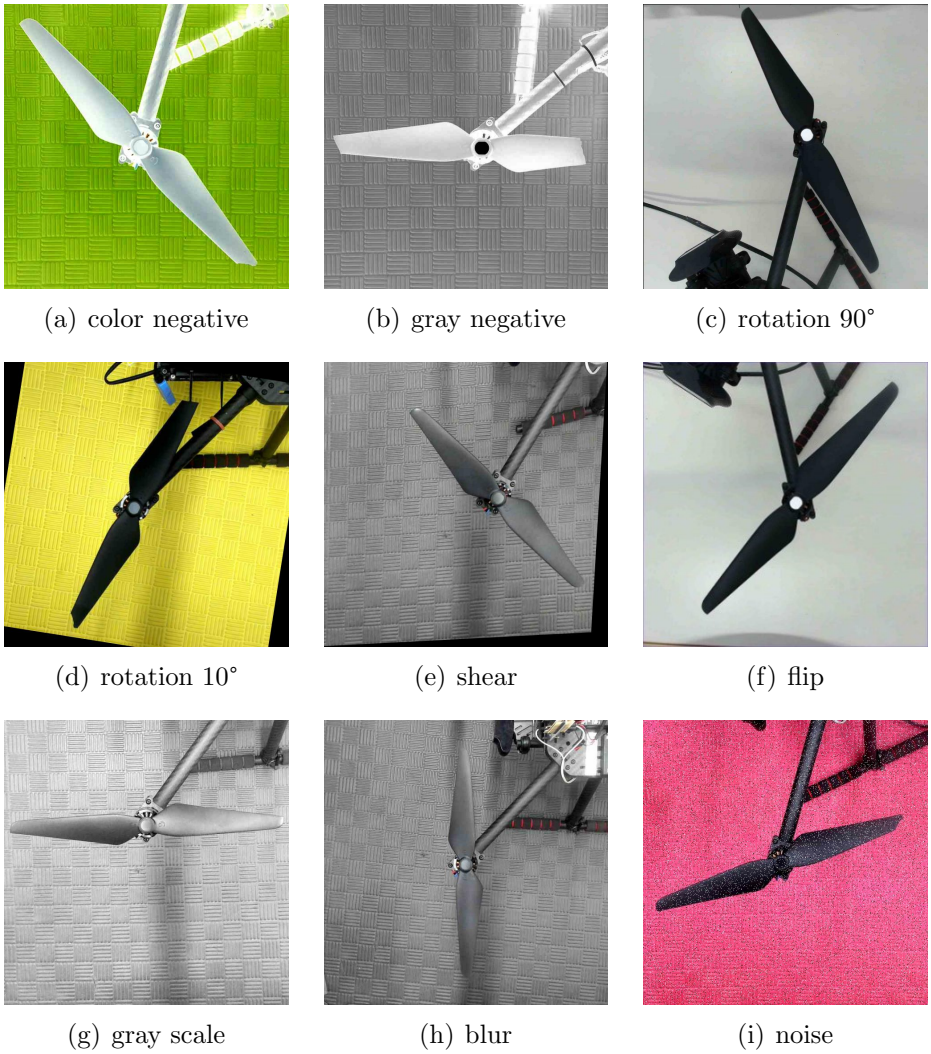
4.1.3 Data Augmentation

Data augmentation is a strategy that uses modified versions of current data samples to improve the size and diversity of a dataset. It assists in avoiding overfitting, inconsistency of class data and increase the model performance with generalization. Moreover, it is very effective for the cost reduction of data collection and labelling. Geometric and photometric transformation are applied to create new images in this research project.

Negative Transformation: It is formed by inverting the image with the point processing operation. In image inversion, the original pixel value is replaced by the subtraction between the maximum pixel value and each pixel value of 'r' in the image. In our study, it plays a vital role due to the object color and darker background in the image. It helps to extract information from dark areas of the image. We generated color negative and gray negative images for our dataset.

Rotation ($\pm 10°$, $\pm 90°$): The propeller position will always be in random and we need different position images. So that, 90° rotation of clockwise and anti-clockwise,

and ±10 degree are used to generate different angles of images. This is a very common approach to enlarge the data volume and also helps to increase the model performance.



(a) color negative (b) gray negative (c) rotation 90°

(d) rotation 10° (e) shear (f) flip

(g) gray scale (h) blur (i) noise

Figure 4.5: Data augmentation techniques

Shear (10°): As camera is mounted in the garage, sometimes there could be displaced the camera position. In that case, it cannot be able to inspect the propeller. That's why, the shear from 0 to 10 degrees is applied horizontally and vertically.

Flip: Flipping involves moving the pixels while keeping the image's information unchanged. We used both horizontal and vertical flipping technique for our images. It helps to increase the data volume which is beneficial during training for getting better accuracy of the model.

Grayscale: We have converted RGB to gray images which reduces the dimension and computational cost is very low. It has a great impact in real time applications by performing fast processing. Furthermore, it helps to detect features and edges within a very short time as well as makes it easy to remove noise from the image.

Blur: We have added random Gaussian blur and increased up to 3% pixels value in each image that makes the blur image. Gaussian blur can assist minimize noise in an image by smoothing out minor variations in pixel intensity. When dealing with images that may have a lot of noise and were taken in low lighting, this can be extremely beneficial for model training.

Noise: This technique is applied to expand the dataset and helps to reduce the overfitting issue. We therefore employed a Gaussian noise known as salt and pepper noise, which manifests itself as sporadic black and white pixels scattered throughout the image. In order to enhance the image in the dataset, we incorporated 3% noise in each image.

In the above figure 4.5 represented the applied data augmentation techniques to increase the drone propeller dataset volume for getting the best result after training the model and their advantages are also discussed.

4.1.4 Duplicate Data Detector

Duplicate image detection is the method of recognizing and eliminating identical or nearly equivalent images from a collection of images or dataset. The purpose of duplicate image identification is to remove duplicated images that consume storage space, mislead the model, and slow down the processing period. Duplicate images create two major problems in machine learning algorithms. It produces the overfitting issues to the model which means it bias the model to learn a specific pattern from duplicate images. Another problem is about the less capability of generalization for the new input images. The trained model predicts wrong classification most of the time.

There are several methods for detecting duplicate images. The most widely used method is hashing, which creates a distinct digital signature or hash for each image and compares the hash values of each image to identify duplicates. In addition, the visual content of images is compared using attributes like color, texture, and shape in content-based approaches. Another approach is called metadata-based algorithms which compare image metadata such as file name, date, and size. Nowadays, to increase the precision of duplicate image detection, there are hybrid methods which integrate two or more approaches, such as content-based and hashing method.

We have not been able to achieve good results by applying all the above methods for detecting duplicate images in our research project. So that we used Mean Square

Error [96] algorithm to detect the duplicates. The main idea of this algorithm is to sum up the squared differences between two images by setting a certain threshold value. The lower the value of error, the images will be more similar. First, translate the image tensors to the numeric data. Then calculate the MSE value of each tensor and set a maximum threshold value for comparing with MSE. In our case, we have taken 200 for threshold. If MSE value is less than threshold value, that means one of the images are duplicates.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{4.1}$$

Here in this equation, n is the number of data points, Y_i is the observed values, and $\hat{Y}_i$ is the predicted values.

The below steps are as followed to detect duplicate images in dataset:

- Compute the tensor or image matrix.
- Calculate the sum of squared value for both tensors and extract difference.
- Set a maximum threshold value to check the similarity of the images.
- If difference is smaller than threshold value, the images are duplicated.

4.1.5 Data Consistency

The dataset is categorized by healthy and broken data. It is a common scenario having duplicate data in the dataset after applying the data augmentation technique. So that, duplicate data detector is used to remove that data. After deleting images from each classification folder (Healthy, Broken), the data is not equivalent in all folders. It is a big issue while training the model and makes it overfitting. The model will be biased and it cannot predict accurately. So that, we compared both folders and found out the difference between these two folders. Then we deleted a certain amount of random images from the large folder images to make consistency between the folders. We got 608 broken and 608 healthy original images. After applying augmentation, we obtained a total of 11404 images and then deleted duplicate images using the duplicate detector. Afterwards, we created a balance between folders by deleting random images. Finally, we prepared **dataset 1 which contains 7869 images**, consisting of original and augmented images. And **dataset 2 has 9350 images**, consisting of dataset 1 images, colour negative (1215) and grey negative (1209) transformed from the original images. Similarly, we used the duplicate detector and made a balance between folders.

4.1.6 Data Annotation

We used Roboflow[1] for annotating our dataset. This online platform is very friendly and provides the geometrical transformation of data augmentation with a certain amount of data. Here, we have set the value 0 for broken images and value 1 for healthy images. We have used rectangular bounding box for annotating object in the image.

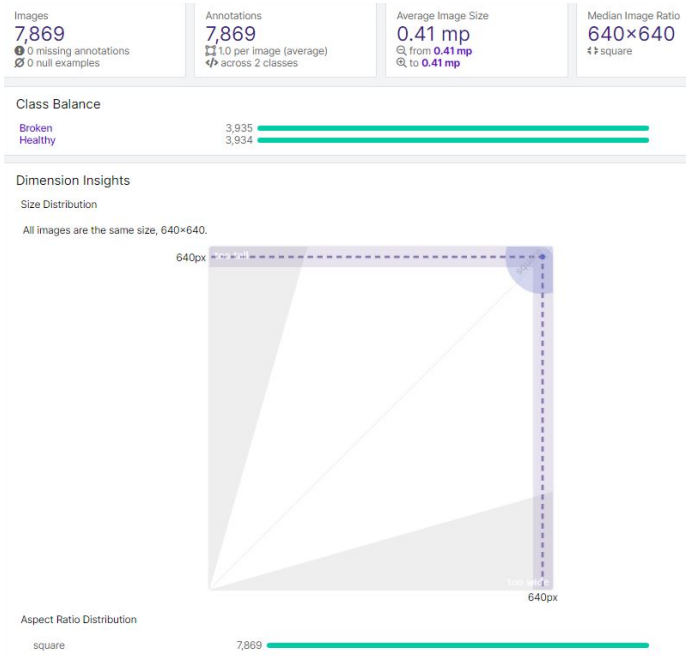


Figure 4.6: Data annotation health condition using Roboflow for dataset 1.

There is an option available in the Roboflow data annotation dashboard to check the health condition about the dataset. It generates the heatmap for individual labels data and also creates the combined one. In figure 4.6 demonstrated the health condition of dataset 1. Here, it converts all the images in 640x640px with the aspect ratio of square size as per model requirements. Both healthy and broken classes of data are equal and a total of 7869 images are annotated and average image size is 0.41mp. The same procedure has been followed for dataset 2 as well[**?**].

4.2 Selected Classifiers and Detectors

This section will discuss selected classifiers and detectors architectures that are used in this thesis for benchmarking performance. Although in the previous chapter, the backbone networks for the classification and detection have already been discussed, here the selected vintage architecture such as VGG family, ResNet family, and YOLO will be discussed in detail.

[1] `https://roboflow.com/annotate`

4.2.1 VGG Family

In 2014, Simonyan and Zisserman [24] presented a deep convolutional neural network that both extended AlexNet's [69] convolutional layer and enhanced the network's feature representation. They referred to it as VGGNet. VGG-16 and VGG-19 are the VGG family's core network architectures. VGGNet is one of the most well-known backbone or feature extractor network to date. It is utilized extensively in computer science and computer vision tasks like image classification and object detection.

Architecture

The VGG architecture starts with a series of convolutional layers, then moves to the pooling layers, and finishes with a several fully connected layers. The VGG network's depth, which is far deeper than the prior state-of-the-art networks at the time. It has up to 19 layers of trainable parameters. This level of granularity helps the network to learn more complicated features, resulting in greater accuracy.

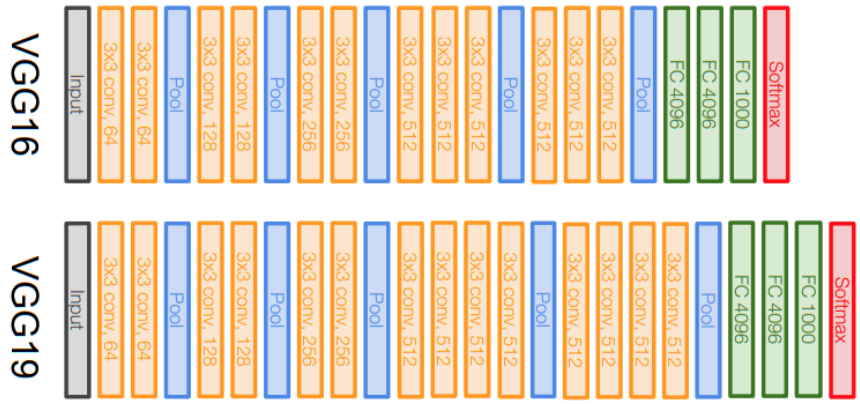


Figure 4.7: The architecture of VGGNet family [24].

Figure 4.7 depicted that VGG-16 network consist of 16 layers that contains 13 convolutional layers where 3x3 convolutional kernel in each layer with stride and padding value of 1, 5 max pooling layers with 2x2 size of kernel with stride value 2, and 3 fully connected (FC) layers. Moreover, it has 138 million of params. On the other hand, VGG-19 has only 3 more additional convolutional layers with 144 million of params. All layers' feature learning is improved by the consistency of the filters, which makes computation easier and more effective. Furthermore, the VGG architecture pioneered the concept of "repeated convolution," which involves stacking many convolutional layers one after the other before pooling [97].

The effective 5x5 receptive field consists of a stack of two 3x3 convolutional layers without a pooling layer. As a result, increasing the size of the stack through three

convolutional layers, it also increases the receptive field. Moreover, non-linearity makes the decision functions more complex but still learn less parameter. This can be interpreted as regularizing 7x7 filters and causing them to decompose into 3x3 filters with non-linearity injected in between layers [24].

4.2.2 ResNet Family

ResNet is an abbreviation for "Residual Network", a deep neural network architecture that was developed by Kaiming He et al. in 2015 [25]. It was developed to deal with the vanishing or exploding gradients problems, which can emerge during training of extremely deep neural networks. Basically, this vanishing gradient problems occur when gradient is back-propagated in its earlier layers which results in a very small gradient.

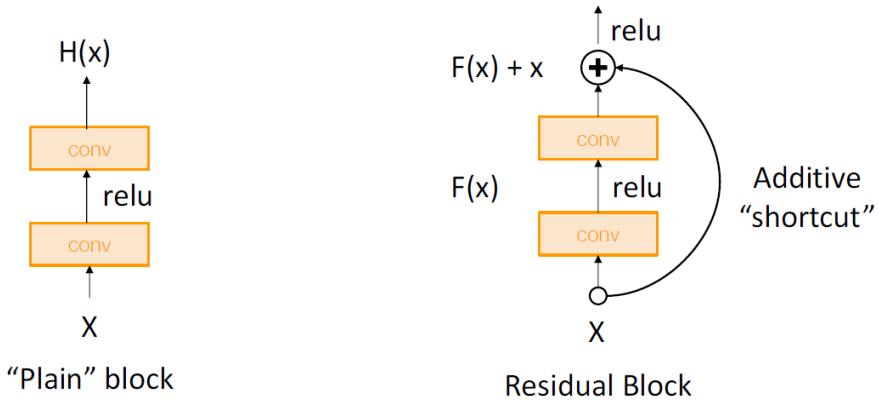


Figure 4.8: Difference between Plain and Residual block [25].

The above figure 4.8 describes the fundamental concept underlying ResNet is to offer shortcut connections, sometimes referred to as skip connections, that enable data to skip one or more network layers. To accomplish this, a residual block that adds the input to the block's output is added to the network's several layers before skipping connection. In this manner, the network can use the shortcut connection to skip a layer only if it realizes that it will have no impact on the output.

Architecture

When the convergence is commenced in deeper networks, it reveals the degradation problem. As the depth of the network increases, the accuracy of the model saturates and decreases rapidly. Saturating results are not only due to overfitting issues, but also directing to extreme training errors. The solution of degradation problem is resolved through a deep residual learning framework. The architecture specifically allowed these layers to suit a residual mapping rather than assuming that each few stacked layers would immediately fit a desired underlying mapping[25].

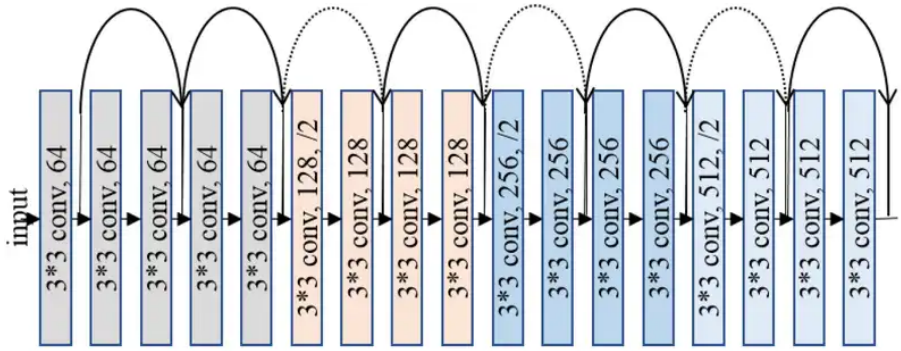


Figure 4.9: The architecture of ResNet Network [25].

In the above figure 4.9 illustrated the architecture of Resnet-18 which takes image as input, then send it to the convolutional layer of 3x3 kernels and get the output of 64 channels. The curved arrows represent the Residual Block which consists of two convolutional layers. The batch normalization and ReLU activation function are attached into these layers. The connection between layers is called shortcut connections which convert the network into residual version. The identity can be applied directly in the network while the dimension of input and output are equal. The expansion of dimension is introduced with the dotted arrows. During this expansion time, the connection of the layers use identity mapping with no extra parameter or projection shortcut can be used for dimension matching. The last layer decides to give the final output of taking image input.

In Resnet families, there are also ResNet-34, ResNet-50, ResNet-101, ResNet-152 available. The architectural design for ResNet families is given below.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 4.10: The architectural design for ResNet Families [25].

In figure 4.10 depicts that the architecture of different layers Resnet was built

for ImageNet classification challenges. Residual building blocks are shown for each convolutional layer with their number of block stacks. After taking input image, the first layer is convolved with 7x7 size of 64 kernels and stride value is 2. Then the max pooling layer is used 3x3 kernels with the same stride. From the third layer to 5th layer is performed the down sampling operation. The final layer, which is fully connected, uses the Softmax function for predicting the class of the image.

4.2.3 YOLO Family

YOLO (You Only Look Once) [26] is the state-of-the-art and most promising real-time single or one-stage object detector. A single network is utilized throughout the entire detection process, which makes YOLO distinctive compared to other networks. When compared to other detectors, this one is among the best. The original YOLO was the first object detection network to recognize class labels and drawing boxes in a single end-to-end distinguishing network. It is extremely fast due to the consideration of frame detection as a regression problem. It has achieved not only 45 frames per second (fps) but also the fast version is greater than 150 fps. Even, the latency of the processed real-time video frame is less than 25 milliseconds. Furthermore, mAP is gained more than twice from other networks. During the training and testing time, it implicitly encodes contextual information about the class and their appearance. It produces less than half of the background error which means it makes more localization errors but predicts fewer false positives in the background. It also learns highly generalizable representation.

Architecture

Since YOLO as a one-stage algorithm used a single network to generate bounding box coordinates and probabilities of classes instead of the anchor mechanism proposed by Redmon et al[26]. It takes an input image and passes it through a series of convolutional layers in the backbone. After that model fed those backbone-extracted features through the neck. Then the neck features through to the heads, which predicts object, class and box regression accordingly.

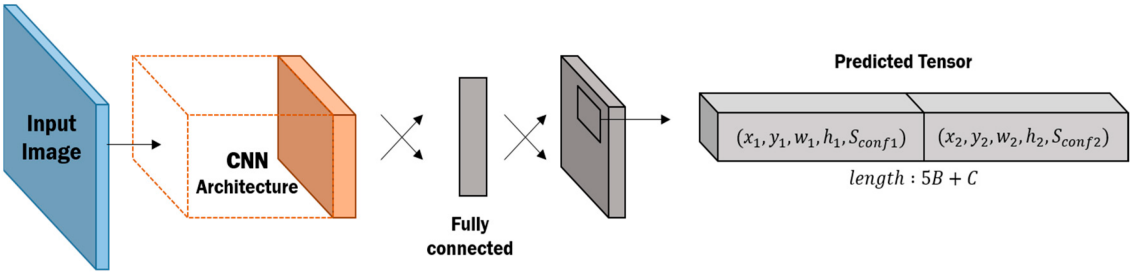


Figure 4.11: The architectural design for YOLO [26].

According to figure 4.11, YOLO's network architecture is modified from GoogleNet, with a total of 24 convolutional layers and 2 fully connected (FC) layers. Each layer

has 3x3 conv layers with a 1x1 reduction layer. 2 FC layers are added after the last 4 convolution layers to train the network. The final layer is responsible for predicting class probabilities and bounding box coordinates. Finally, Non-Max Suppression (NMS) is used to acquire a single optimal detection for an object.

With YOLO's unified architecture, object identification is reframed as a regression problem that is solved simultaneously by predicting multiple bounding boxes and probabilities of class for those boxes. The following steps are followed in the model:

- Input image is divided into an SxS grid.
- The bounding boxes (B) and confidence scores for each box are predicted in each grid cell.
  Confidence = Pr(Object) * $\text{IoU}_{truth}^{pred}$
- There will be 5 predictions in each bounding box:
  x, y, w, h, and confidence.
- The conditional class probabilities (C) will be predicted in each grid cell:
  $\Pr(\textbf{Class}_i|Object)$.
- Multiply the class probabilities and confidence.
- The predicted image tensor will be: S x S x (B * 5 + C).

This study aims to develop a reliable solution for drone propeller fault inspection based on the YOLOv5 or YOLOV7 architecture since the accuracy, real-time action, and lightweight model aspects are vital for the accuracy and efficiency of drone propeller fault inspection.

YOLOV5

YOLOV5[20] is the fifth version of the YOLO family and incorporates the top-notch algorithm optimization approach for cross-stage partial networks, data augmentation (mosaic), bounding box anchoring, and more. This network's benefits include lightweight, high detection accuracy, and faster inference speed that reaches 140 frames per second for detection.

Architecture

In the following figure 4.12, the YOLO's network architecture is comprised of three components: backbone, neck, and head. The input image is fed to the backbone network for feature extraction. After that, the output of the backbone network is fed to the PANet into the neck for doing feature fusion, And the final output is predicted in the head layer[95].

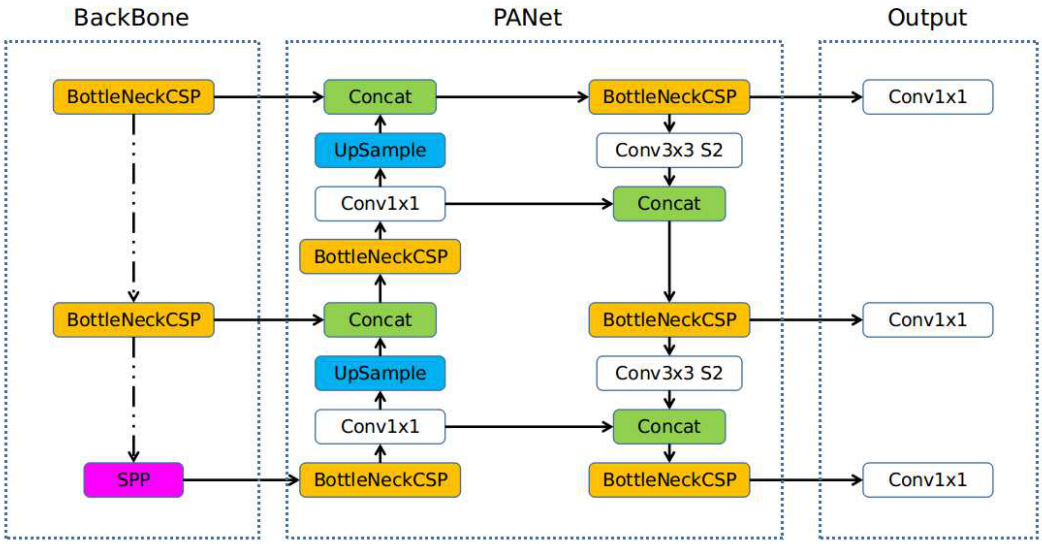


Figure 4.12: An overview of YOLOV5 architecture [20].

Backbone: The backbone network leverages CSPDarkNet-53, a cross-stage partial network [75] which helps to suppress the vanishing gradient problem by using residual and dense block, and spatial pyramid pooling (SPP) to extract feature maps of various sizes from the input tensor of an image via several convolution and pooling. BottleneckCSP is applied to decrease the number of calculations and speed up inference, while a three-scale feature map is generated by SPP that accomplishes several scales of feature extraction for the identical feature map which helps to enhance the detection accuracy [95].

Neck: Path aggregation network (PANet) is used in the neck for increasing the information flow. It adopts a new feature pyramid network (FPN) structure that includes top-down and bottom-up feature maps for improving low-level features propagation in the model [95]. The ultimate motivation of this network is to enhance the accuracy of localization in the lower layers.

Head: In the head, it uses the same YOLOV3 head which generates three different feature map output that predicts the confidence score, bounding box location (x, y, height, width), and the classes of objects[95].

Activation Function: The activation function Sigmoid Linear Unit (SiLU) and Sigmoid are used in this network model. Furthermore, SiLU is also known as swish and is applied with the operation of convolution between the hidden layers. On the other hand, Sigmoid is used in the final output layer[95].

Loss Function: This network uses Binary Cross Entropy (BCE) with logit loss for computing the class loss as well as abjectness loss. the location loss is calculated by the complete intersection over union (CIoU)[95]. The following equation is given for calculating the final loss:

$$LOSS = \lambda_1 L_{cls} + \lambda_2 L_{obj} + \lambda_3 L_{loc} \tag{4.2}$$

YOLOV7

YOLOV7 [23] was presented by Wang et al. in 2022. It introduces various architectural innovations that increase speed and accuracy than previous models. Like Scaled YOLOv4, YOLOv7 backbones don't rely on ImageNet pre-trained backbones. Rather, the models are trained solely on the COCO dataset.

Architecture

Yolov7 architecture is designed by using Extended Efficient Layer Aggregation Network (E-ELAN). It plays the leading role as a computational block in the backbone network. The design is developed by giving focus on several factors such as the cost of memory access, channel ratios of input and output, element-wise operation, and gradient path that increase both speed and accuracy in the model. The following figure 4.13 describes the architectural overview of the Yolov7 model.

Extended Efficient Layer Aggregation: The prime consideration of efficient architectural design is to optimize parameters, computations, and computational density. The effect of I/O channel ratio, branch architectures and elementwise inference speed calculation is analyzed in the VoVNet. The shortest longest gradient path is monitored for better learning and converging successfully by E-ELAN that is extended from ELAN architecture.

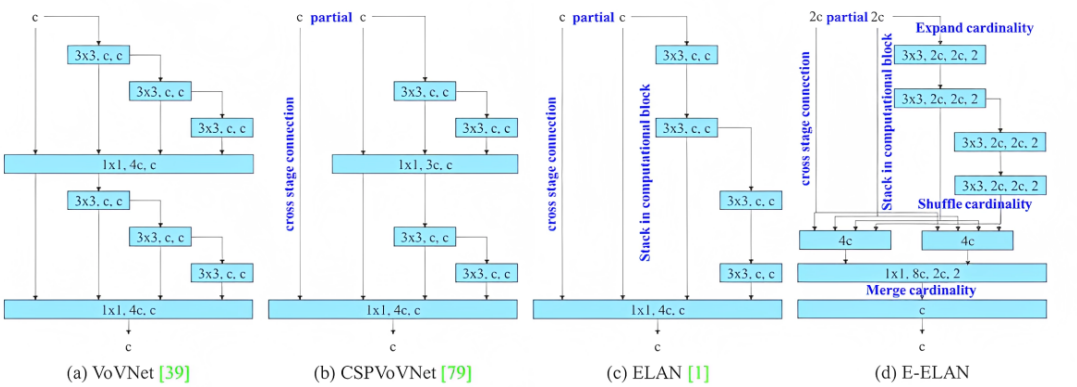


Figure 4.13: An overview of YOLOV7 architecture [23].

It also shuffles and merges cardinality to enhance learning capability without demolishing the original gradient path. The architecture of computational blocks has

changed but the transition layer is remaining the same. Group convolution with the same group parameters is applied to enlarge the computational block and channel. After that, both are concatenated. As a result, the feature map that makes merge cardinality will be exact in the original architecture. Furthermore, Several computational block groups can be led by E-ELAN architecture to learn more varied features.

Model scaling: In general, object detection models figure out the network's depth, width, and resolution when training the model to detect objects. In yolov7, it measures both network's width and depth while layers are concatenated together. This strategy makes an optimal solution for different sizes of scaling.

Re-parameterization: This strategy is applied to obtain the weighted average of model weight to make a more powerful model which is stronger than ordinary patterns. By using the propagated gradient flow path to look for which modules utilize the re-parameterization strategy.

Auxiliary head coarse-to-fine: In YOLOv7 framework, they have used multiple heads. One is called the Lead head, which gives the final output. The other is the Auxiliary head which assists in the middle layers training and the weight of this head is updated by applying the assistant loss technique. It boosts the model for deep supervision and better learning. Lead head and Label assigner are close couple concepts. The label assigner predicts the network result and grounds truth together and then soft labels are assigned. The soft and coarse labels are created by the label assigner the contrary, unlike hard ones.

4.3 Active Deep Learning Approach

The major advantage of an active deep learning approach is that models can be trained using fewer data and achieve robust performance. This approach reduces the cost and saves time for doing data annotation by selecting only essential features related to images. To the best of our extensive research, this active deep learning technology has not yet been used in any drone propeller fault inspection task using deep learning. Though, it has shown superior performance in other sectors such as burn mark detection[TUC3], insulator detection[TUC4], and traffic sign recognition [TUC5]. The approach is modified according to the requirements of our model, as shown in figure 4.14. The following steps of the suggested active deep learning approach are given below:

1. A training pool is created using the entire dataset by splitting the 80% training data and 20% testing data respectively.

2. Then 15% of the total random images are selected from the training pool for the first iteration process.

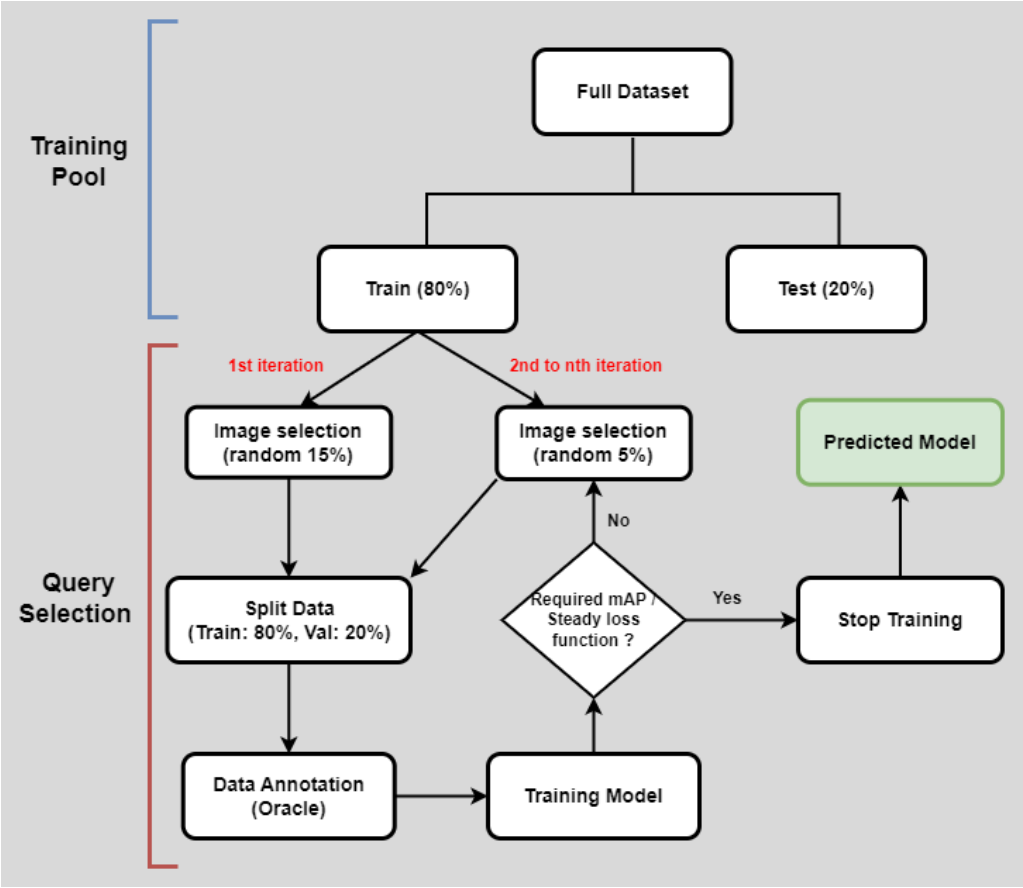


Figure 4.14: Proposed active deep learning approach.

3. After that, this query selected random data is split into training 80%, and validation 20% datasets respectively.
4. The selected dataset is annotated with the help of an expert oracle.
5. Then the proposed model is trained using this created dataset.
6. After training, the required mAP or the loss function stability is checked.
7. 5% of the images are randomly pooled from the training pool for the next iteration due to less required mAP.
8. Until the expected accuracy of the model is reached, steps 3 to 7 will be repeated continuously.
9. The final model will be selected by comparing the highest mAP for inference.

4.4 Software Deployment

One of the most crucial steps in the software development process is software deployment. Application, module, update, and patch delivery from developers to consumers is accomplished through deployment. The techniques employed by developers to develop, evaluate, and deploy new code will have an impact on both the speed and quality of each change that is made to a product in response to adjustments in consumer preferences or requirements.

Nowadays, the most familiar word "DevOps" has arisen when software deployment is required after the development process. DevOps is a software development methodology and set of best practices that aim to reduce delivery times for new software updates while keeping high quality. It typically comprises a framework called **CI/CD pipeline** for software deployment. Continuous Integration (CI), where new code is regularly integrated into a repository by working teams. On the other hand, Continuous Deployment (CD) denotes a software release strategy where new code is automatically released into the production environment and users can interact with it after passing automated tests.

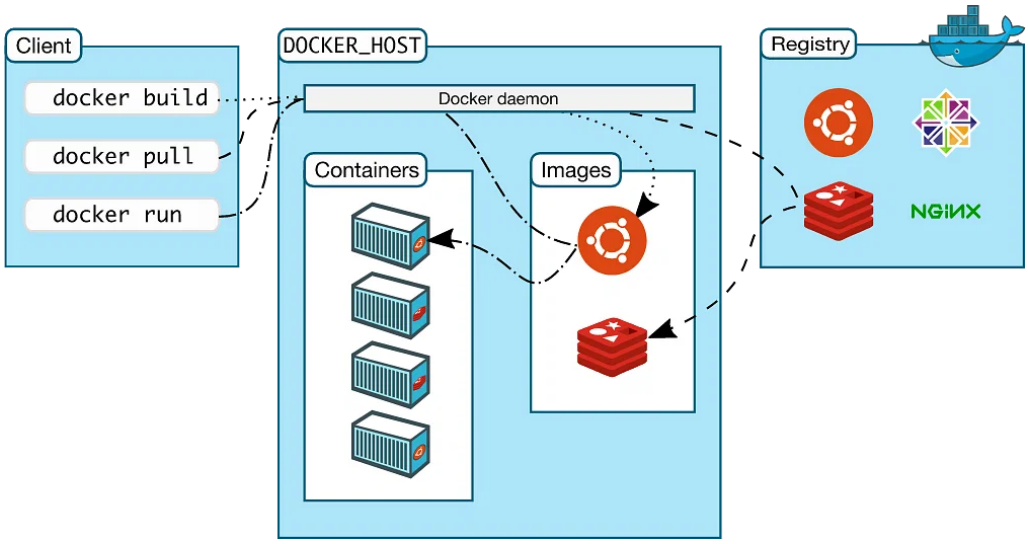


Figure 4.15: Docker architecture [27].

In this research project, we have used **Docker**[2]. It is a software framework that enables rapid application development, testing and deployment. After completing the development and automated testing phase, the software is packaged by docker containers, each of which contains the runtime, libraries, and system tools. It is installed in the local system and allowed to write and edit code inside the container. It is easy to deploy the application in a test environment and attach it to the CI/CD

[2] `https://docs.docker.com/get-started/`

workflow. If any bug needs to be fixed in the container, it is possible to resolve it. Then the container has to be redeployed for validation. Figure 4.15 illustrated the architectural overview of the docker.

The client-server architecture is followed by docker. A containerization platform for executing and deploying programs is provided by the docker architecture, which is made up of several essential key components.

- Docker Daemon: The containers, images, networks, and storage volumes are all managed by the Docker daemon, which runs in the background. The command request is sent from the Docker CLI to the daemon and responds to them by controlling the lifecycle of the containers.
- Docker CLI: a command-line interface (CLI) is used to communicate with the daemon. To create, manage, and deploy Docker images and containers, it provides an easy-to-use interface.
- Docker Image: The instructions for building a container are included in a read-only template known as a Docker image. To run the program in a containerized environment, it includes the required application code, libraries, and system utilities.
- Docker Container: A runnable instance of a docker image is known as a docker container. Each container has its file system, network connections, and process space. Each container is entirely separate from the host system and other containers.
- Docker Registry: Docker images can be distributed and kept in a registry, which acts as a central repository. The most common public registry is Docker Hub, although corporations can also build up their private registries for internal usage.
- Docker Networking: Containers use docker networking to communicate between themselves and with the host system. It supports several networking protocols such as bridge networks, host networks, overlay networks, and MACVLAN networks by utilizing a variety of network drivers.
- Docker Storage: For preserving data in containers, Docker provides several kinds of storage solutions. This includes network storage volumes, local storage volumes, and external storage solutions such as Ceph, GlusterFS, and NFS.

In general, the Docker architecture is designed to offer a flexible and scalable platform for building, deploying, and maintaining containerized applications. Docker is becoming a powerful component for contemporary software development and deployment since it provides a standardized method for packaging and running programs in containers.

4.5 Chapter Summary

This chapter introduces the concept that was formed as a result of the literature review from the previous chapter. Additionally, it also aims to overcome the stated issues that were noticed in chapter one. This chapter begins with the methodology of the proposed method.

Preparing the dataset is the most important stage for building a robust deep neural network. The deep learning model's accuracy and consistency are significantly influenced by how precisely the data is utilized. As a result, the data preprocessing step is broken down into several subtasks: data collection and selection, augmentation, duplicate data detection and data balancing.

The drone propeller dataset was formed using several propellers from TUC IFC Lab. The image resolution is set by a square aspect ratio of 720x720. Data augmentation techniques are applied to increase the data volume. Then detected duplicate data is deleted and the categorical data is manually balanced. Next, the dataset is split into the training pool. Finally, the dataset is annotated by oracle. The essential preprocessing stage has improved the model's accuracy and reliability.

The model is trained by following two strategies. One is passive learning which takes the entire annotated dataset to train the model. The other is active learning which takes a certain number of percentages data from the training pool and splits it into the train, Val, and test data. Then the data is annotated. After training the model, the required mAP is checked by applying an iterative process.

The selected object classifiers and detectors are used to train the model. VGGNet, ResNet, and YOLOv5 classification models are trained with passive learning strategy and benchmarking their accuracy. On the other hand, YOLOV5 and YOLOV7 both models are trained for real-time detection by applying an active learning strategy and benchmarking their performance.

The approach we recommend for active deep learning is outlined next. The active learning approach increases training data across successive iterations rather than using the complete dataset for a single training. Although this requires additional training time, it alleviates the stress of annotation tasks for oracles (humans). As a result, by using an active deep learning technique, more satisfactory accuracy can be obtained with fewer data.

In the final section, the model deployment is completed by using docker. It creates a package of models called containers where package-related libraries, utilities, and runtime are available. We used docker to deploy our best model and built a docker image inside the container which can take input from any host machine and give output after processing and send to the host machine.

5 Implementation

The implementation of the proposed drone propeller fault inspection architecture and functional approach is described in this chapter. It includes the required system requirements and deep neural model training. In addition, dataset preparation using an active deep learning framework and the model configuration are also described. Finally, the iterative process of dataset splitting is illustrated by the diagram.

5.1 System Requirements

The proposed method is implemented by applying several state-of-the-art solutions. They are divided into hardware and software components. Different computing platforms utilized for implementation and deployment are considered hardware. On the other hand, programming language and their libraries are acknowledged as software.

5.1.1 Hardware

This research study appliances two different hardware computing resources to implement the inspection model. One is local computing and the other is cloud computing. The assignment of various computational tasks among these two resources is depicted in the below figure 5.1.

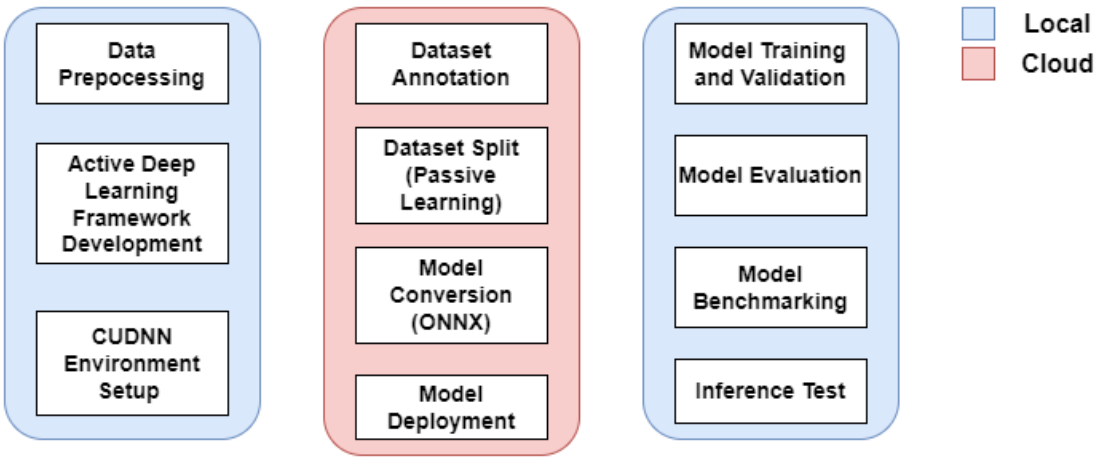


Figure 5.1: Distribution of work between hardware computing platforms.

Local Computing Platform

Two local computing platforms have been used to develop a propeller fault inspection system. Several tasks such as data preprocessing, active deep learning framework implementation and model benchmarking have been performed without a GPU-accelerated computing platform. Moreover, other tasks related to model training and evaluation have been performed using fast computational or GPU-accelerated computing platforms. Furthermore, inference speed using both local computing platforms has also been evaluated.

- Machine 1: An MSI Katana GF76 12UE laptop with 12th Gen Intel® Core™ i7-12700H (20 CPUs) with base frequency 2.7 Ghz, NVIDIA® GeForce RTX™ 3060 Laptop GPU 6GB GDDR6 Up to 1475MHz Boost Clock 105W Maximum Graphics Power with Dynamic Boost and 16GB RAM has been used as a local computing machine. Additionally, Microsoft Windows 10 Home 64-bit OS has been used.
- Machine 2: Another computing platform without GPU, Dell Inspiron 5559 with 6th Gen Intel® Core™ i5-6200U CPU @ 2.30GHz (4 CPUs), up to 2.4 GHz, and 12GB RAM has also been used. Microsoft Windows 10 Pro 64-bit has been used.

Cloud Computing Platform

We used Roboflow for data annotation and data splitting tasks for the passive learning approach. After finishing model training using local machine 1, we transferred the model into the cloud computing platform for converting the model into open neural network exchange (ONNX) format. We had to do this for avoiding several conflicting library versions which directly impact CUDA. We used Google colaboratory[1] (Colab). The colab has provided 12 hours of free GPU accelerated session, which was sufficient for this model conversion task but not for our model training. Python Jupyter Notebook and Google drive for getting storage are integrated with colab.

The colab setup includes a 2.30 GHz of Intel® Xeon® CPU with 2 cores, Haswell CPU family, and 12GB RAM with 25GB disk space. It has also an available Nvidia Tesla T4 or K80 GPU with 16G GDDR6. The GPU has 320 Turing Tensor Cores and 2,560 NVIDIA CUDA® Cores. The GPU has a clock speed of 0.82 to 1.59 GHz and a throughput of more than 320 GB/s. 4.1 to 8.1 TFLOPS are used to measure computing performance. The disk space is 358 GB.

[1] `https://colab.research.google.com/`

5.1.2 Software

Programming languages Python 3.10, Opencv 4.6, Tensorflow 2.10.0, tensorflow-gpu 2.10.1, torch 1.13.1, torchvision 0.13.1 are used in local computing machine 1 for developing the inspection system. Additionally, NVIDIA CUDA version 11.6 has been installed to obtain GPU service.

In machine 2, python 3.9 and Opencv 4.6 have been installed for checking inference performance without GPU. Other dependencies like scikit-learn, numpy, matplotlib, pandas, and so on are installed for both machines. PyCharm Community have been used as a code editor IDE.

Python

Propeller fault inspection application is developed using python which is a very popular, high-level language and easy to use. It is known for its simplicity, readability, and flexibility, making it a versatile language for a wide range of applications. Python has a large and active community of developers who have created a vast ecosystem of libraries and tools to extend its functionality. We have used python versions 3.9 and 3.10 on two local machines. Besides, built-in and third-party libraries have also been used.

OpenCV

The OpenCV software library was originally developed in C++ to support a variety of algorithms and tools for image and video processing, object detection and classification, and other computer vision and machine learning tasks. Additionally, it is also available for the Python version. We have pre-processed the image and video from the camera using the OpenCV library following our model's specifications.

Tensorflow

A free and end-to-end open-source framework called Tensorflow was created by Google Brain for applications such as machine learning and artificial intelligence for a variety of tasks such as image classification, natural language processing, and speech recognition. It provides a set of tools and APIs for building and deploying machine learning models, as well as support for distributed computing to scale up training and inference. We have used Tensorflow 2.10.0 and GPU-accelerated version 2.10.1.

PyTorch

PyTorch is an open-source machine learning framework that is primarily used for building deep neural networks. It is known for its dynamic computational graph feature that allows for more flexibility and easier debugging. PyTorch provides a

wide range of tools and APIs for building and training machine learning models, as well as support for GPU acceleration and distributed computing. We have used PyTorch-based detectors to benchmark the performance of the models.

PyCharm

PyCharm is an integrated development environment (IDE) for Python programming language developed by JetBrains. It provides a wide range of features to support the development of Python applications, including code completion, code inspection, debugging, testing, and version control integration. PyCharm is designed to boost developer productivity by providing a streamlined workflow for coding, debugging, and deploying Python applications. It also supports various frameworks and libraries such as NumPy, matplotlib, Scikit-learn, OpenCV, Tensorflow, and more.

5.2 Model Training

This section describes all the steps required to train our proposed model, from the beginning of the experimental setup to model deployment. Environment setup, dataset preparation strategies, model configuration, model training, and model deployment steps are involved in the implementation of this study. A brief explanation of each step is provided below.

5.2.1 Environment Setup

We used both offline and online computing platforms to train a robust model for our application. In the Offline, the virtual environment is set up for training the heavy and deep models in GPU-accelerated machines. We have also used the colab setup to convert the trained model which is online.

Offline - Virtual Environment Setup

A virtual environment is a self-contained Python environment that enables developers to install and handle project-specific dependencies without interfering with other projects or the system-wide Python installation on their local machine. It offers an isolated environment with its own Python engine and package dependencies, enabling devs to quickly transition between various projects with varying dependency requirements. We used the Anaconda command prompt (cmd) to create a virtual environment in our local machine.

After creating the environment, it must have to be activated. Then the selected folder has to be opened from the selected drive and run the python script. Before running the model script, make sure that GPU is activated. The below figure 5.2 illustrated a code snippet for checking GPU activation in a virtual environment.

```
In [1]: import tensorflow as tf
        from tensorflow import keras

In [2]: print("Tensorflow version: ", tf.__version__)
        #print(tf.version.VERSION)
        print("Num GPUs available: ",
        len(tf.config.experimental.list_physical_devices('GPU')))

        Tensorflow version:  2.10.1
        Num GPUs available:  1

In [ ]: print(tf.test.is_built_with_cuda())
        print(tf.test.is_gpu_available())

In [3]: import torch
        print(torch.cuda.is_available())
        print(torch.cuda.device_count())
        print(torch.cuda.current_device())
        print(torch.cuda.device(0))
        print(torch.cuda.get_device_name(0))

        True
        1
        0
        <torch.cuda.device object at 0x000002C5D7D8F730>
        NVIDIA GeForce RTX 3060 Laptop GPU
```

Figure 5.2: GPU activation check in virtual environment.

Online - Google Colab setup

We have trained all of our network in GPU accelerated local machine but used Google Colab to convert the trained weight file due to some library's versioning conflicts. For running google colab, below steps have to be followed:

- GPU runtime has to be connected to run the model and for that, go to "*Runtime* → *Change runtime type* → *GPU*".
- Google drive has to be mounted for data storage. Otherwise, data will be deleted after ending the limited session.
- We have used YOLO in our project. So, YOLO has to be cloned from the GitHub repository according to their version and uploaded to the trained model's weight file in the result folder.
- Dependencies have to be installed by navigating the requirements file in the YOLOV5 folder and have to use 'Pip' for executing the command.
- Finally, the conversion-related command has to be run by using 'Pip'.

5.2.2 Dataset Preparation

Data preprocessing steps include from the beginning of data collection to data annotation. We have used our local computing system for preparing the dataset except for data annotation. As we have to train an image-based deep learning model, it is very essential to check the duplicate data which will bias the model.

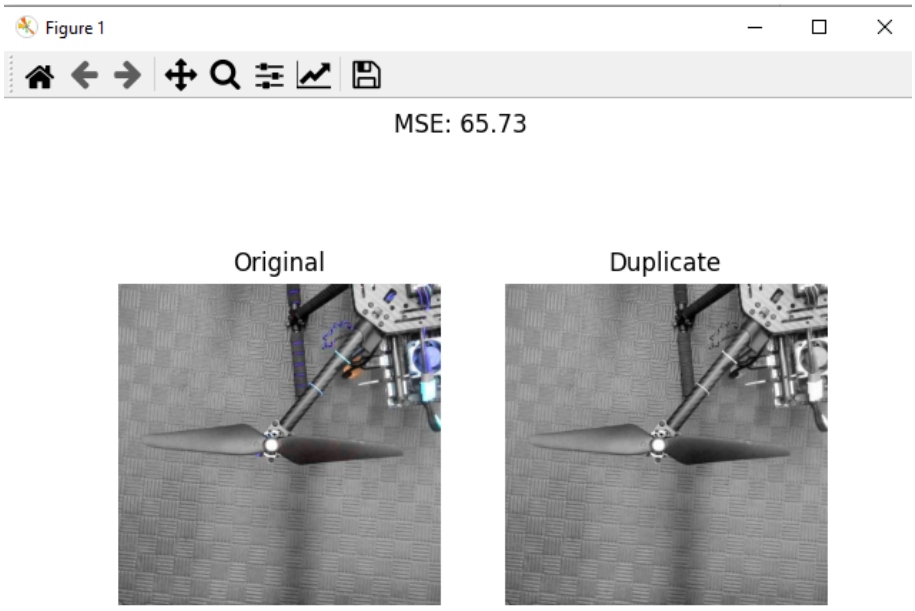


Figure 5.3: Duplicate image detection using MSE algorithm.

In the previous chapter, we have already discussed our algorithmic solution for duplicate image detection, where we use the MSE algorithm to compare two images by checking the threshold value. In our case, we have set a max threshold value of 470. If the MSE is less than the threshold value, the images are duplicated. Figure 5.3 depicted the duplicate image detection with an MSE value of 65.73 which is a less than max threshold value.

After deleting all duplicated images, we have made consistency in each class folder and deleted some random data from the larger class folder. The data annotation is done by using Roboflow. we have split data into 70% training, 20% valid, and 10% test data for passive learning. But in active learning, at first, we have to split the full dataset into train 80% and test 20% accordingly.

Mainly, active learning is an iterative model training process as previously described in figure 4.14. For example: a total of 7869 data are available in dataset. After splitting data, 6295 (80%) random data is selected for training pool and 1574 (20%) data is taken for testing data. According to the algorithm in below table 5.1, 15% of the data was taken from the training pool for the first iteration and

divided again between train (80%) and validation (20%) data to form the dataset for training the model. After training the model mAP has to be checked. If mAP is less than our expected result, we need more iteration. Afterwards, the model was trained sequentially by adding 5% more data to the previous dataset for the next iteration. This iteration was continued until the expected map was obtained. In this iteration, we have got a total of 20% training data for our model. This iteration process will be repeated until reaching the mAP goal. The training and validation data have been increased in each iteration process. We used only 60% data for obtaining our expected mAP result through the iterative process.

Iteration	**New data**	**Training (80%)**	**Validation (20%)**	**Total**	**(%)**
1	942	754	188	942	15%
2	266	968	240	1208	20%
3	254	1172	290	1462	25%
4	240	1364	338	1364	30%
5	230	1548	384	1932	35%
6	216	1722	426	2148	40%
7	206	1888	466	2354	45%
8	196	2046	504	2550	50%
9	190	2198	542	2740	55%
10	178	2340	578	2918	**60%**

Table 5.1: Dataset creation for model training using active learning.

5.2.3 Model Configuration

Deep learning model configuration involves optimizing the architecture, hyperparameters, and training parameters of the network for a specific reason. The architectural specification is defined by the number of layers with their types, activation functions, input, and output dimensions of the network. Moreover, to control the entire training process and even for tackling the overfitting issues, sometimes it is needed to hyperparameter tuning such as batch size, learning rate, momentum, regularization, and so on. Furthermore, the number of epochs, criteria for early stopping, network training duration and when to stop the training process are included in the training parameters configuration. Configuring these parameters requires careful experimentation and tuning to find the optimal configuration for a given task.

In this study, we used three classification networks VGGNet, ResNet, Yolov5-Classifier and two object detectors YoloV5 and V7. We modified the configuration file as required to train the model. It is needed to separate the images according to the class folder and provided the configuration file for training the classification model. On the other hand, the labelled data is required for the object detector model. We configured the location of the train, valid, and test data. We used a GPU-accelerated local machine to train the model.

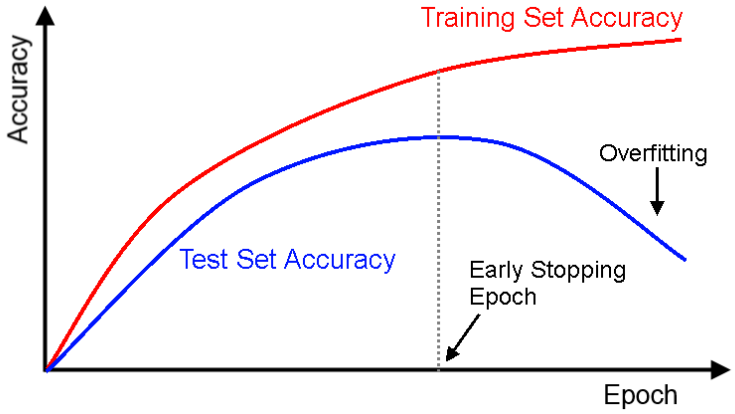


Figure 5.4: The point of early stopping [28].

A model may occasionally overfit on a small dataset. In other words, it remembers the training dataset, therefore it performs well in validation but poorly in testing. To address these problems, an early stopping point is used. The early stopping point is when the model does not overfit. Figure 5.4 shows the early stopping epoch point and the accuracy that should be chosen to avoid overfitting.

The image resolution of 224x224 is used for both VGGNet, ResNet network. We used batch-size 32, 150 of epochs, ReLu for activation function, softmax for final layer activation, dropuot layer 30%, learning rate $1e_{-5}$, momentum 0.9, SGD optimizer, binary-cross entropy for loss function, and pretrained weight Imagenetfor both classification model. On the other side, we used the default hyperparameter and default for both the yolov5m and v7 models. In addition, we set batch size 24 with 4 workers, and 300 epochs for the Yolov5 model containing 212 layers. For yolov7 with 314 layers, we set batch size 5 with 4 workers, and also 300 epochs.

5.2.4 Model Training and Conversion

We have created two datasets for training our models. The first dataset contains 7869 images and the second dataset contains 9350 images. The image size 224x224 px is fed to the VGGNet, and ResNet models. On the other hand, the image size of 640x640 is fixed for the YoloV5 and V7 model. In the passive learning strategy, we

used the entire dataset for both classification and detection models. After getting a result, later we decided to use only the first dataset for the iterative training process by using an active deep learning framework. we used this framework only for the detection model. A total of 10 iterations were applied for getting our expected result and used only (29.5%) 2340 images from the whole dataset. The dataset is created by randomly selecting from the training pool for each iteration. Finally, the model evaluation and inference graphs were saved after each iteration.

```
[ ]  %pip install torch==1.11.0 torchvision==0.12.0 torchaudio==0.11.0 torchtext==0.12.0
     %pip install onnx==1.12.0
     %pip show torch
     %pip show torchvision
```

Figure 5.5: Onnx conversion sloution for YOLOV5 and V7 model.

After obtaining the best model, we converted this model into ONNX format to check the inference using the OpenCV Dnn module. We converted the Pytorch weight file through Colab due to its versioning complication which made complication in GPU-accelerated local machine. Figure 5.5 illustrated the solution of mandatory version for torch and onnx.

5.3 Model Deploy

We deployed our model using docker. We have installed Docker windows on our local machine. Then we have to go to the directory of the project file and open windows PowerShell from the file option. Next, We created a docker image by following this "*Docker build -t (docker image name) -f (Dockerfile location) .*" command. Then we built the container and mount the input folder with the docker image and followed the "*Docker run -it -v (input path) : (docker images path) image name*" command. We uploaded our image to the DockerHub registry and maintained the version controlling policy. To do that, we registered in the DockerHub. Then we followed the command "*Docker login*" and it is needed for the first time. Next, we have to set the image name using the tag by following "*Docker tag (image name: image version) (your user-id/new image name: version)*". Finally, we have pushed the image in DockerHub using the "*Docker push username/name: version*" command. This image can be downloaded in any container. Afterwards, we have given input from the local host machine. After generating the output through the model into the container, we sent the output files to the local machine by copying them from the container output folder.

5.4 Chapter Summary

This chapter is covered the entire development scheme for implementation with a detailed discussion. This includes hardware specifications and software requirements in the first section. We used high computing power laptops with NVIDIA GPU RTX 3060 and google colab for the model training and converting purposes. In addition, Python, OpenCV, TensorFlow, and PyTorch are used for inspection.

The next section is described the entire process of model Training and started with data collection. After collecting and selecting image data, the MSE algorithm is used to delete the duplicate images from the database and made data consistent according to their class folders. Two datasets are created for model training as well as evaluation and split the dataset between 70% of the training, 20% of the validation, and 10% of the testing. In an active learning strategy, a training pool is created and then randomly queries selected images are applied to iteratively create a dataset for model training.

One of the most important tasks is to configure the model before the training stage. Image classification networks: VGGNet and ResNet have used the same model configuration. While YOLOv5 and v7 are followed by their default hyperparameter values for the model configuration. This step is tested very sincerely to overcome the overfitting problem.

The learning strategies for model training are also briefly discussed in this section. We followed passive learning for all classification and detection models using two sets of datasets. After getting the result, we decided to use only one dataset for testing the deep active learning strategy. We iterated 10 times for making a dataset to train the model. Finally, we got our expected result using 29.5% of the data from the selected dataset. After that, we converted the model using collab due to avoid the library version conflicting issues as we had to train several models in our local computing machine.

Last but not least, the final section is described the obtained best model deployment process. Here, we used Docker and DockerHub registry. We created a container after installing docker windows. Then we wrote a docker file for creating a docker image in the container. After that, we uploaded this image to DockerHub which is similar to Github. We tested the images by giving input from the local host machine and copied the processed result from the container to the host machine. Overall, from the beginning of installing the docker software to the end of deploying in the registry is discussed by following the given commands in this chapter.

6 Results and Evaluation

This chapter describes the evaluation and results of our trained deep neural classifier and detector models. At the beginning of this chapter, the evaluation matrix is briefly discussed. Afterwards, trained models result is described by following two learning strategies: passive learning and active learning. We evaluated VGG-19, ResNet-50, YOLOv5-classifier as well as YOLOv5m and YOLOv7 detection models. Finally, the inference and speed rate is evaluated based on deep learning framework formats.

6.1 Evaluation Metrices

After training the model, it is mandatory to evaluate the model's performance. It is done by applying some mathematical and statistical formulas, known as evaluation matrices. We used Accuracy, Confusion matrix, Precision, Recall, F1 score, and ROC-AUC curve for the classification model as well as Confidence score, Intersection over Union (IoU), and mean average precision (mAP) for the detection-based model. A brief description of these evaluation matrices is given below.

6.1.1 Accuracy

Accuracy quantifies the prediction by statistical metrics of the classifier that represents the level of precision or correctness. In other words, accuracy assesses how successfully a classification model or system identifies or predicts the desired variable.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{6.1}$$

Accuracy perfectly works for a well-balanced target class but is not suggested for the unbalanced data of classes. In this case, the model will be biased.

6.1.2 Confusion Matrix

A confusion matrix specifies a table that compares the true values with the predicted values for a given set of data to assess the efficiency of a classification model. The rows in the matrix represents the actual values and columns for the predicted values. The number of data points classified into each category of the model is represented by the number of cells in the matrix.

		Actual Class		
		Healthy	**Broken**	Total
Predicted Class	**Healthy**	TP	FP	$TP + FP$
	Broken	FN	TN	$FN + TN$
	Total	$TP + FN$	$FP + TN$	N

Table 6.1: Confusion matrix table for our model.

Table 6.1 presents the confusion matix cell by true positive (TP), false positive (FP), false negative (FN), and true negative (TN). TP is the number of instances in which the model correctly identified the positive class where FP is opposite and wrongly identified the positive class. FN is represented by the model that wrongly predicts the negative class. TN represents the number of instances where the model correctly predicts the negative class.

A confusion matrix provides a variety of performance indicators such as precision, recall, and F1 score, which can be used to evaluate the effectiveness of a classification model.

- **Precision:** The precision of the model is the level to which it correctly identified the target values. It is the proportion of the number of true positives and predicted positives.

$$Precision = \frac{TP}{TP + FP} \tag{6.2}$$

- **Recall:** Recall is defined as the true positive cases that can predict correctly in our model. It is a reliable metric where FN gets more focus than FP.

$$Recall = \frac{TP}{TP + FN} \tag{6.3}$$

- **F1 score:** The F1 score provides an integrated understanding of the metrics for Precision and Recall known as harmonic mean. It reaches its highest when Precision and Recall both are equal.

$$F1 = 2 \cdot \frac{Precision \times Recall}{Precision + Recall} \tag{6.4}$$

- **ROC-AUC Curve:** The effectiveness of a binary classifier is visualized by the ROC-AUC (Receiver Operating Characteristic - Area Under Curve) curve. The true positive rate (TPR) and false positive rate (FPR) are plotted at different categorization levels. The percentage of truly positive instances properly identified is known as the TPR. The FPR is the percentage of true negative cases that are falsely classified as positive cases by model calculation.

6.1.3 Confidence score

The confidence score is usually a numeric value of probability between 0 and 1. Here, the value of 1 indicates the model confirmation about its prediction and 0 means confusion or a lack of understanding about the prediction. Different levels of confidence are indicated by intermediate values.

6.1.4 Intersection over Union (IoU)

IoU (Intersection over Union) is a popular evaluation metric in object detection and image segmentation applications. The discrepancy between the predicted bounding box (B_{pd}) and the ground truth annotation (B_{gt}) is measured by IoU. It is represented by the intersection of the predicted and ground truth regions and the union of the two regions. It can be written mathematically as follows:

$$IoU = \frac{\text{area of intersection } (B_{pd} \cap B_{gt})}{\text{area of union } (B_{pd} \cup B_{gt})} \tag{6.5}$$

The confidence score calculates the probability of an object being located in an anchor box. It is generally predicted using a classifier. The metrics of confidence score and IoU are both utilized to determine the true positive or false positive value of a detected object. The IOU values are assigned between 0 to 1.

6.1.5 Average Precision (AP)

The precision-recall curves are often used to measure detection, although average precision (AP) provides numerical numbers and is easier to compare performance with other models. Depending on the precision-recall curve, AP summarizes the average precision and cumulative recall for every threshold value. AP is computed for individual objects. The equation are as follows:

$$AP@IoU = \int_0^1 p(r)dr$$

6.1.6 Mean Average Precision (mAP)

For individual classes, the AP is estimated separately. Moreover, the number of AP values is equal to the number of classes. In addition, AP is expanded as the mean Average Precision (mAP) and is calculated by averaging these values to acquire the entire model precision. Hence, it can be defined as:

$$mAP@IoU = \frac{1}{N} \sum_{i=0}^{N} AP_i \; for N \; classes$$

6.2 Model Evaluation

In this section, the model evaluation results are utterly described. We trained our model using passive and active learning strategies.

6.2.1 Passive learning evaluation

We used two well-balanced datasets to train the classifier network and objcet detectors. The first dataset contains 7869 images and the second dataset contains 9350 images. We split our dataset according to 70% training, 20% validation, and 10% test data.

Classifiers Result

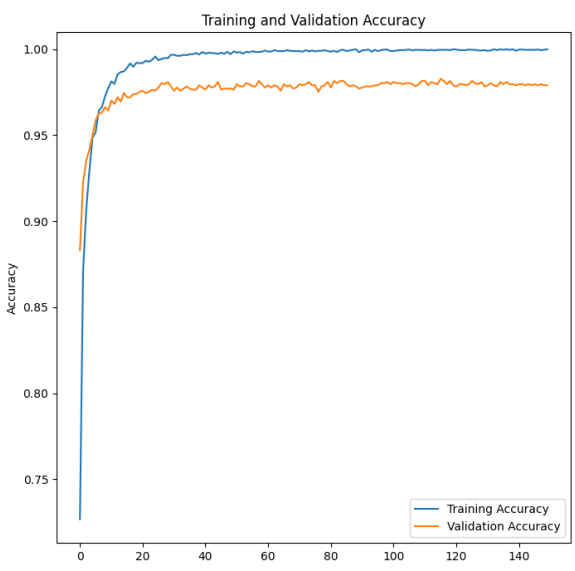
We trained the VGG-19, ResNet-50, and YoloV5m classification models by using their default image resolution. The experimented results are given below. We have tunned the hyperparameters and parameters for VGG-19 and ResNet-50. We set the batch size of 32, and 150 epochs for training parameters. In addition, We used the Relu activation function, Softmax for the final activation layer, Dropout 30%, learning rate $1e^{-5}$, momentum 0.9, Stochastic Gradient Descent (SGD), binary cross entropy for calculating loss function and pre-trained weight ImageNet as hyperparameters in the model.

VGG-19

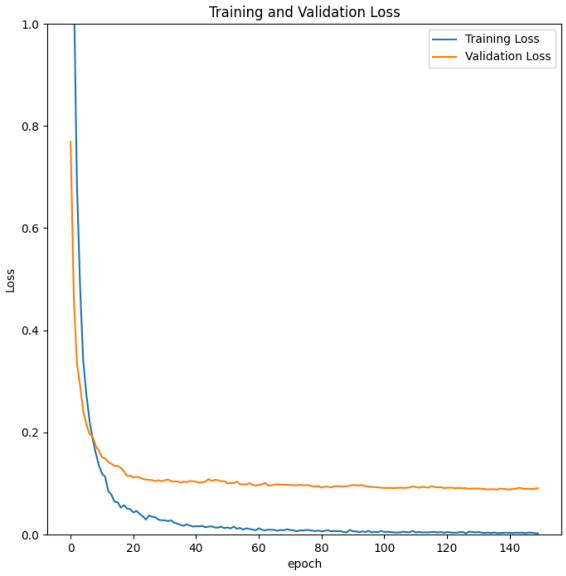
The below figure 6.1 is illustrated our model training and validation accuracy as well as loss calculation. The training accuracy is nearly reached 0.98 where validation is accuracy nearly 0.97 in figure 6.1(a). On the other hand, the training loss has reached close to 0.0 and the validation loss is close to 0.1 in figure 6.1(b). The model is trained perfectly but the accuracy is not satisfied due to the gap between training and validation accurarcy.

According to figure 6.1(c), we got a slightly bigger difference between the training and validation accuracy for dataset 2. Besides the loss difference of training and validation in figure 6.1(d) is also shown bigger than the training and validation loss of dataset 1. For this reason, we got relatively low accuracy after training this model using dataset 2.

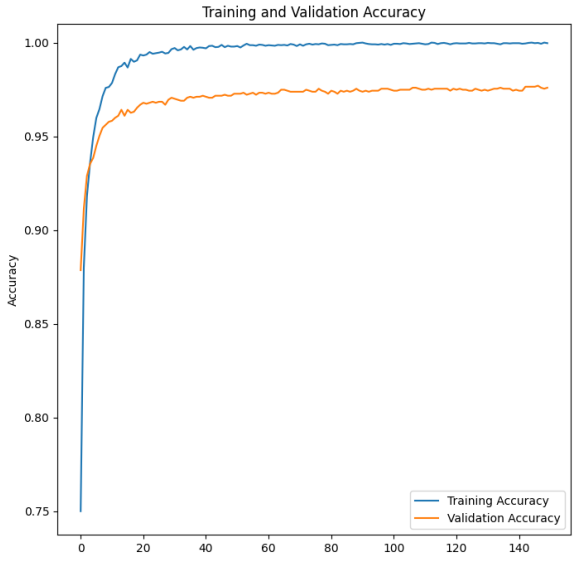
We checked the validation of our model using 1574 images from dataset 1. In figure 6.2 depicts the confusion matrix of VGG-19 where the true negative represents the broken and true positive as healthy predicted images. Here, we can see the confusion matrix result for dataset 1 in the figure 6.2(a). The model predicts 784 images as true negative (TN), 13 images are false negative (FN), 757 images are true positive (TP), and 20 images are false positive (FP). The model predicts almost



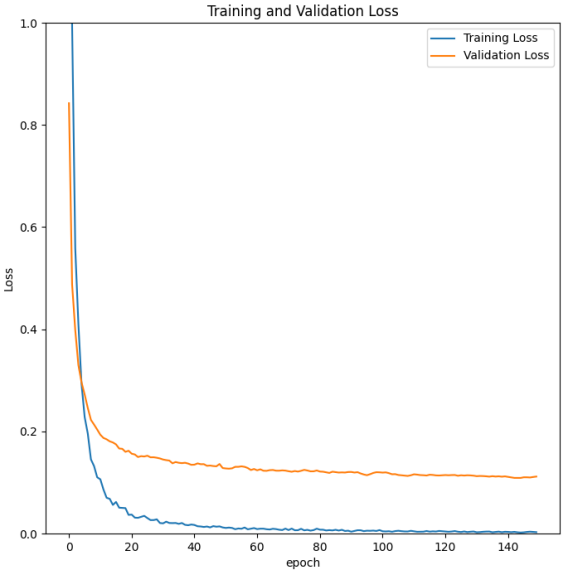
(a) Dataset 1: Accuracy calculation



(b) Dataset 1: Loss calculation



(c) Dataset 2: Accuracy calculation

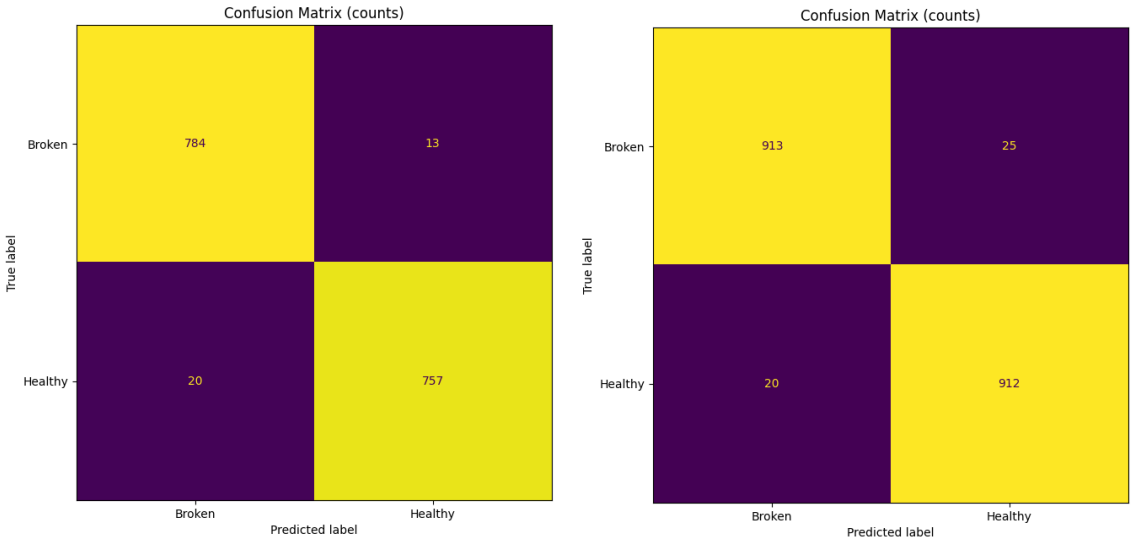


(d) Dataset 2: Loss calculation

Figure 6.1: VGG-19 network's accuracy and loss calculation for dataset 1 and dataset 2.

16% false positive, and 26% false negative images which are not acceptable for our proposed solution. Finally, we have got an accuracy of 97.9% using confusion matrix where 98% precision for both broken and healthy classes, recall 98% for broken and 97% for healthy and f1 score 98% is equivalent for both classes.

The figure 6.2(b) depicts that the model has predicted 913 images for true negative



(a) Dataset 1: Confusion matrix

(b) Dataset 2: Confusion matrix

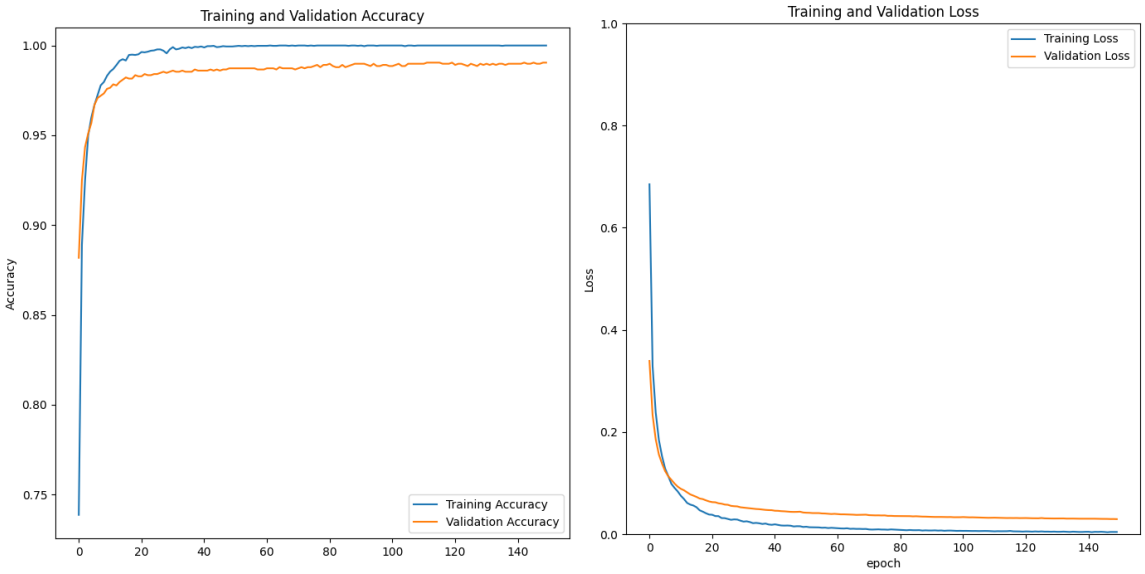
Figure 6.2: Confusion matrix of VGG-19 for dataset 1 and dataset 2.

(TN), 25 images for false negative (FN), 912 images for true positive (TP), and 20 images for false positive (FP). The model is predicted 16% for false positive but 27% for false negative. After calculating the confusion matrix, we got 97.59% accuracy from dataset 2 where 98% and 97% precision rate for broken and healthy classes, 97% and 98% recall for broken and healthy and f1-score 98% for both respectively.

ResNet-50

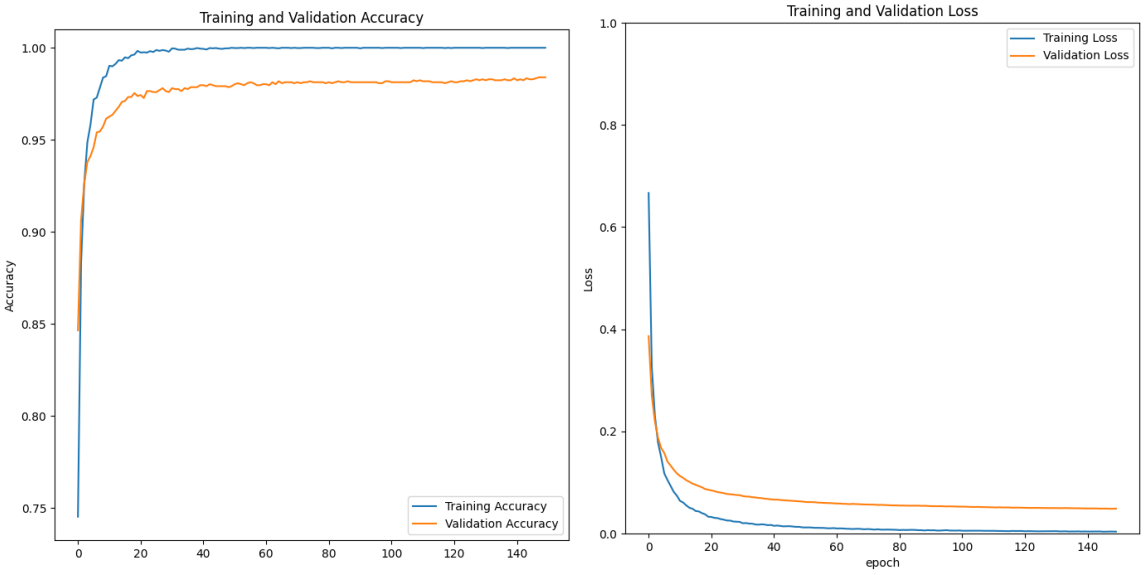
We have taken ResNet-50 from ResNet famlily and trained the model using our propeller two datasets. Figure 6.3 is illustrated the accuracy and loss of training and validation for ResNet-50. The result is comparatively better than VGG-19. There is not so much difference between the accuracy and loss curves. Figure 6.3(a) represents the performance of training and validation accuracy and figure 6.3(b) depicts the loss of training and validation for dataset 1 which is shown less than VGG-19 results. On the contrary, the second model which is trained by using dataset 2 has shown in figure 6.3(c) for accuracy calculation, and figure 6.3(d) for loss calculation This second model is less accurate than the first model.

In figure 6.4 depicts the confusion matrix of ResNet-50 where the true negative represents the broken and true positive as healthy predicted images. The first model's matrix created by using dataset 1 is shown in figure 6.4(a). It predicted 789 images as true negative (TN), 8 images are false negative (FN), 770 images are true positive (TP), and 7 images are false positive (FP). We can see that the model predicted almost 10% false positive, and 9% false negative images. After calculat-



(a) Dataset 1: Accuracy calculation

(b) Dataset 1: Loss calculation



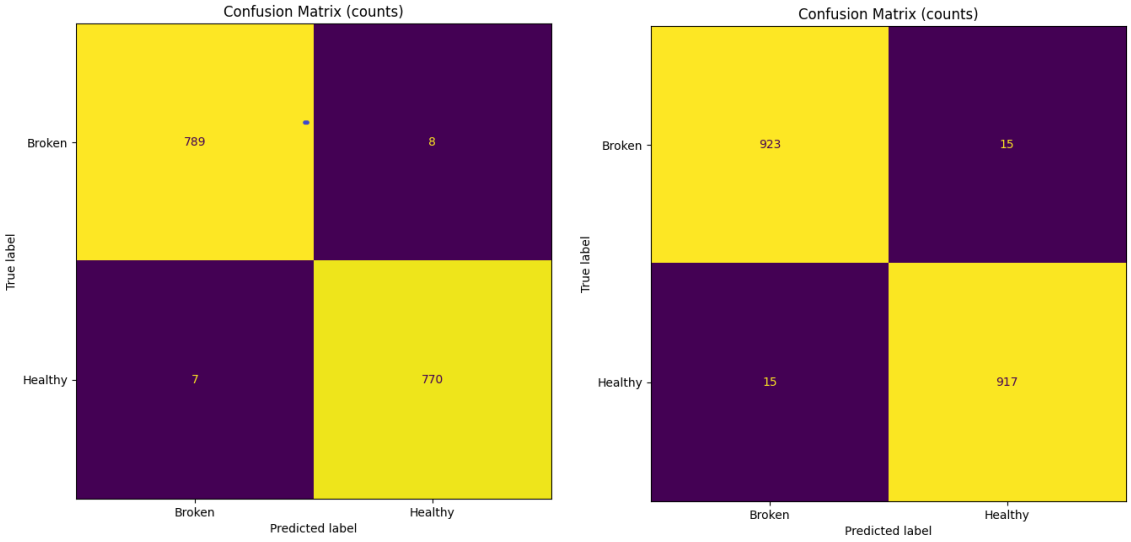
(c) Dataset 2: Accuracy calculation

(d) Dataset 2: Loss calculation

Figure 6.3: ResNet-50 network's accuracy and loss calculation for dataset 1 and dataset 2.

ing the confusion matrix, we have got the accuracy of 99.04% for dataset 1 where precision, recall and f1 score is 99% equivalent for both broken and healthy classes.

The second model's confusion matrix which is created by using dataset 2 is shown in figure 6.4(b). It identifies 923 images as true negative (TN), 15 images are FN, 917 images are TP, and 7 images are FP. After calculating the confusion matrix, we



(a) Dataset 1: Confusion matrix

(b) Dataset 2: Confusion matrix

Figure 6.4: Confusion matrix of ResNet-50 for dataset 1 and dataset 2.

got an accuracy of 98.39% for dataset 2 where precision, recall and f1 score is 98% equivalent for both broken and healthy classes. The performance comparison result is as follows:

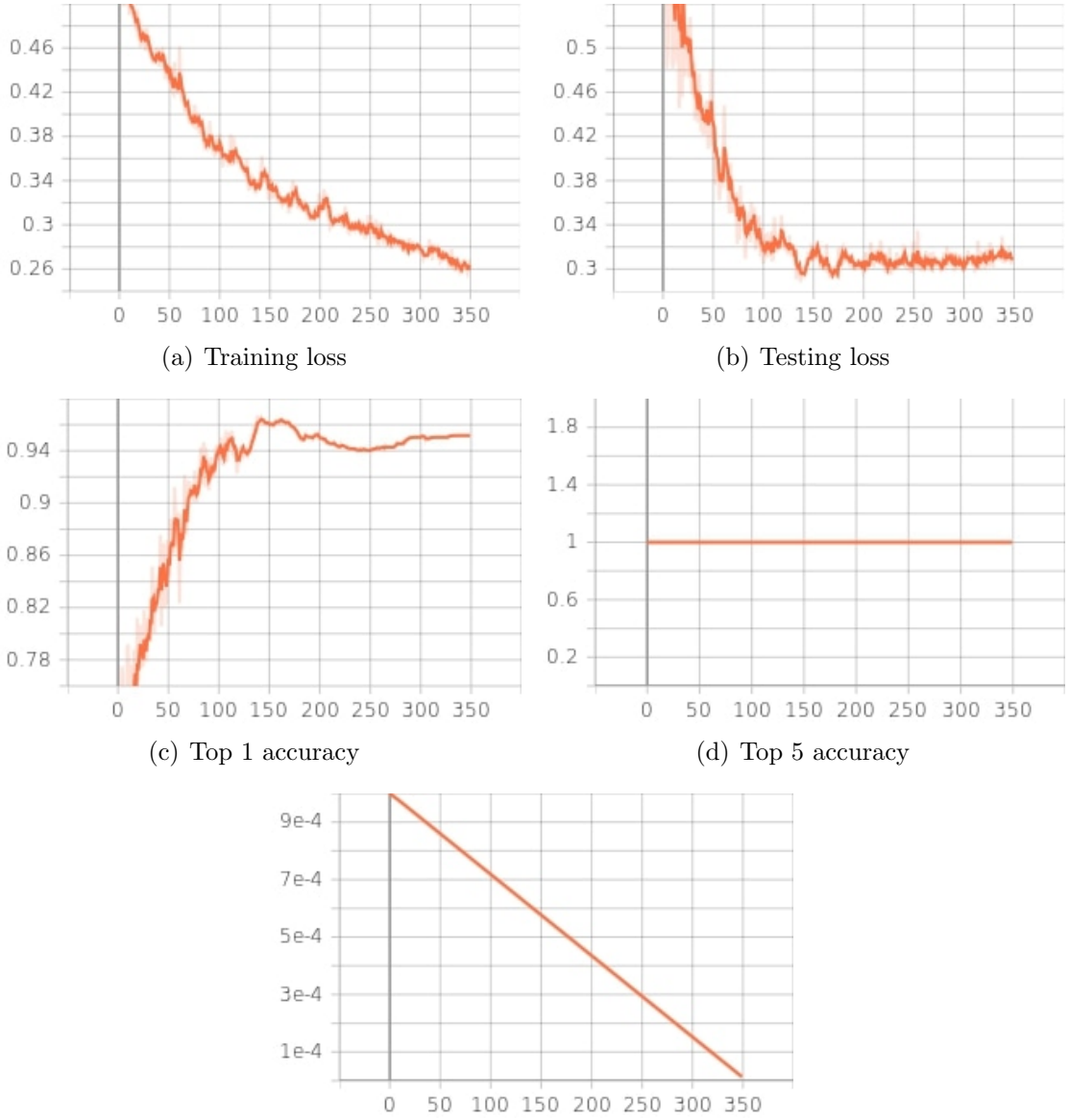
Dataset	Classifier	Classes	**Precision**	**Recall**	**F1**	**Accuracy**
Dataset 1	VGG-19	Broken	0.98	0.98	0.98	97.9%
		Healthy	0.98	0.97	0.97	
	ResNet-50	Broken	0.99	0.99	0.99	**99.04%**
		Healthy	0.98	0.97	0.97	
Dataset 2	VGG-19	Broken	0.98	0.97	0.98	97.59%
		Healthy	0.97	0.98	0.98	
	ResNet-50	Broken	0.98	0.98	0.98	**98.39%**
		Healthy	0.98	0.98	0.98	

Table 6.2: The performance comparison between VGG-19 and ResNet-50.

Table 6.2 represents the comparison results between VGG-19 and ResNet-50 using two datasets. ResNet-50 has achieved the better accuracy using dataset 1.

YoloV5m-Classifier

We also trained the Yolov5 medium classification model using both datasets. The experimental results are given below:



(a) Training loss

(b) Testing loss

(c) Top 1 accuracy

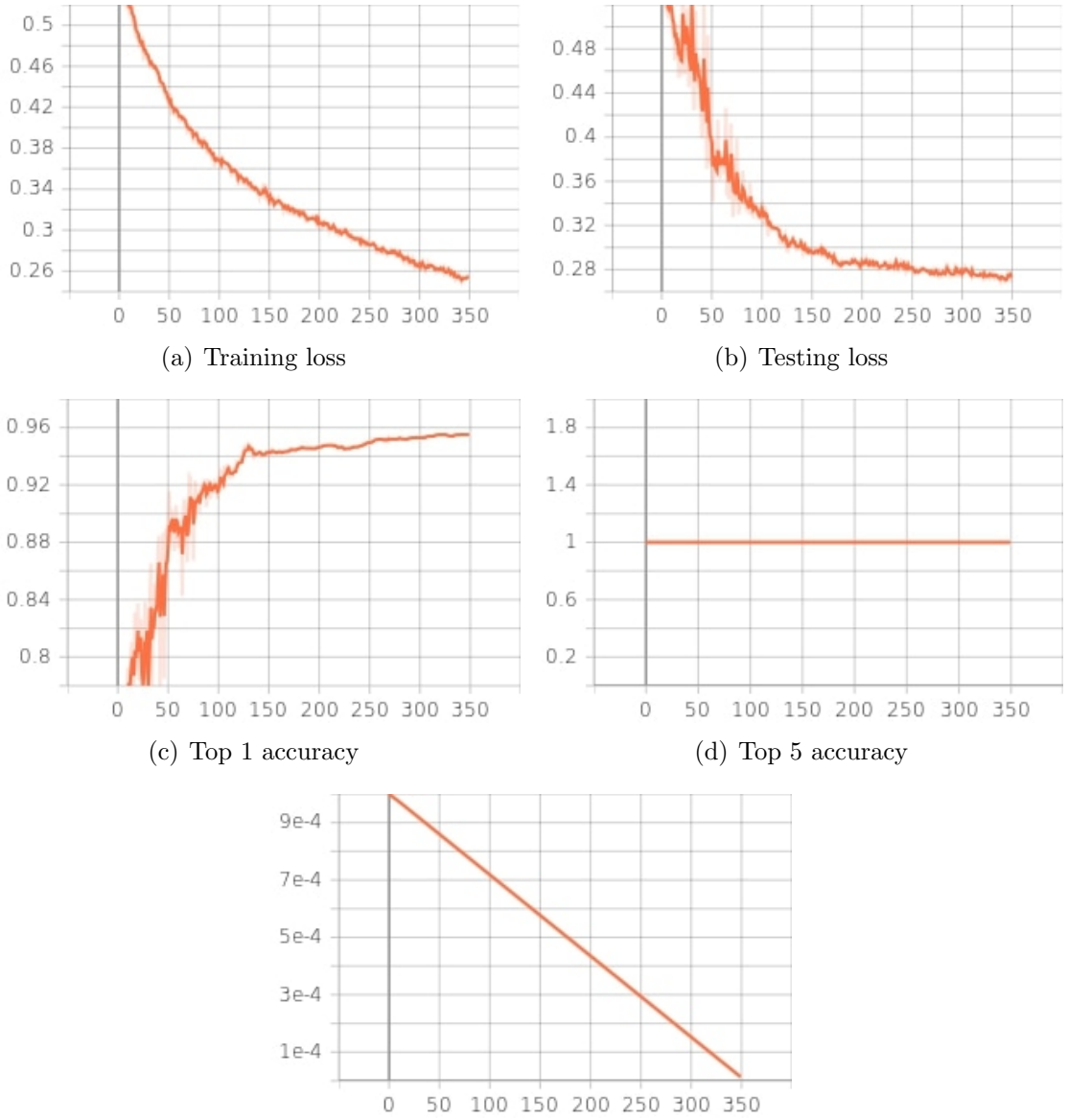
(d) Top 5 accuracy

(e) Learning rate

Figure 6.5: Yolov5m classification result for dataset 1.

Figure 6.5 depicts the result of the Yolov5m classification for dataset 1. We have given 350 epochs with default hyperparameters settings. The training loss decreased gradually which is shown in figure 6.5(a). Besides the test loss is also reduced as shown in figure 6.5(b). Furthermore, In figure 6.5(c), we achieved the overall Top 1 accuracy above 96.7% where broken 93.7% and healthy 99.7%. Moreover, We

achieved 1 for overall Top 5 accuracy where both broken and healthy achieved 1 as shown in figure 6.5(e). In addition, the learning rate is also decreased respectively.



(a) Training loss

(b) Testing loss

(c) Top 1 accuracy

(d) Top 5 accuracy

(e) Learning rate

Figure 6.6: Yolov5m classification result for dataset 2.

Figure 6.6 depicts the result of the Yolov5m classification for dataset 2. The training loss and testing loss are achieved 0.25, and 0.27 respectively. The Top 1 accuracy is nearly 96.0% and top 5 is 100% where the learning rate is decreased gradually. We found less accuracy in comparison to the result of Dataset 1. A comparison of the two dataset validation results is shown in the table below.

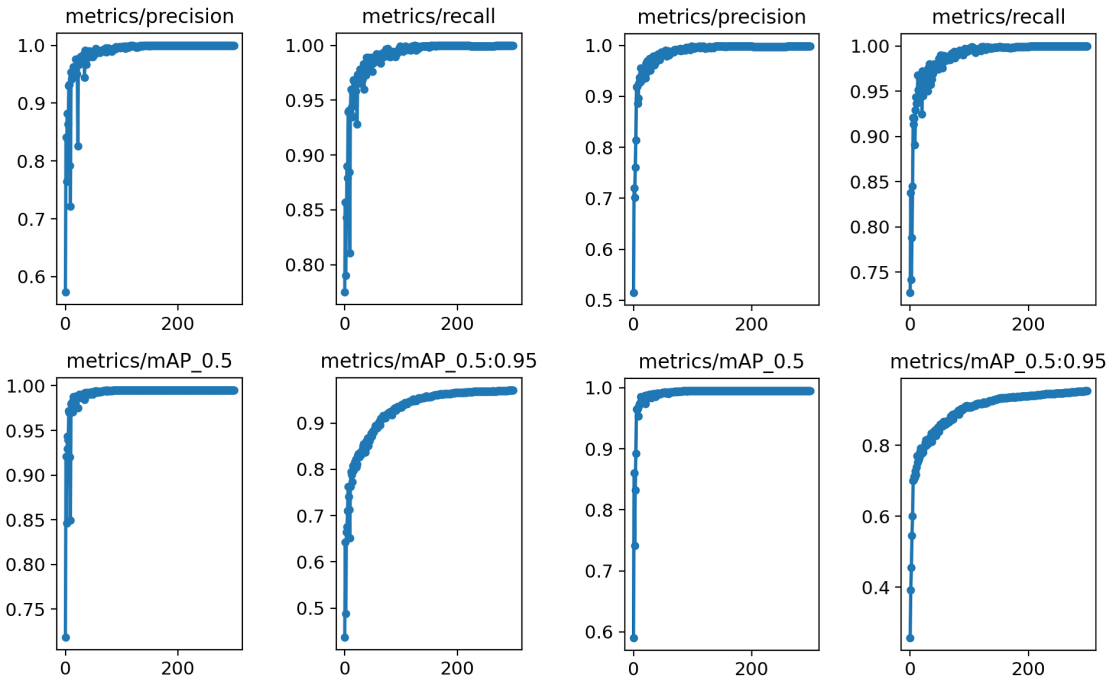
Dataset	Classifiers	**Top-1 Accuracy**	**Top-5 Accuracy**
Dataset 1	YoloV5m-cls	**96.7%**	100%
Dataset 2	YoloV5m-cls	95.5%	100%

Table 6.3: The performance comparison of YoloV5m (medium) classifier.

YOLOV5 and YOLOV7 Detectors Result

In the previous section, we achieved a tremendous result from the YoloV5m classifier. For that reason, we also wanted to experiment on YoloV5 and V7 object detectors. We have taken YoloV5 medium and small models for experimenting with our propeller datasets. A total of 7869 images for Dataset 1 and 9350 images for Dataset 2 were annotated. We have given 300 epochs for both models and default hyperparameter settings.

Experimental Results for Dataset 1:



(a) Yolov5m training result

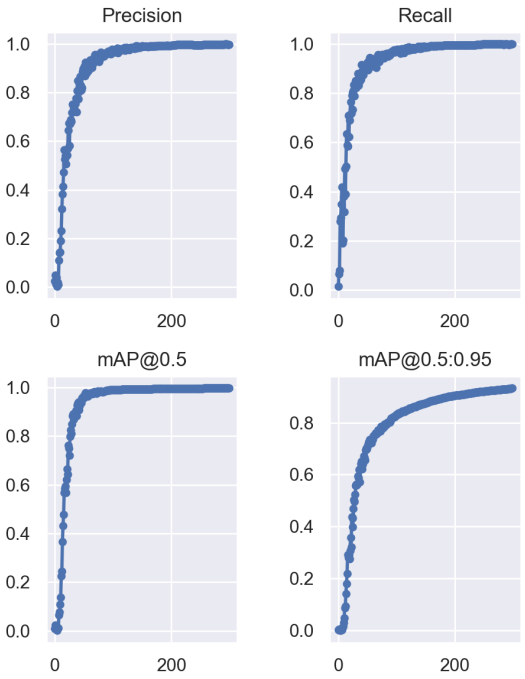
(b) Yolov5s training result

Figure 6.7: YoloV5 model training results for dataset 1 using passive learning.

In the following figure 6.7, a comparison result between YoloV5 medium and small models for dataset 1 is given above. Here, we can see that, figure 6.7(a) represents the training result using the YoloV5 medium model. The overall precision is

achieved at 99.9%, recall is 100% for both classes representing the harmonic mean. It achieved 99.5% of mAP@0.5 and 97.1% for mAP@0.5:0.95.The model validation has been done by giving confidence score 0.001 and IOU value from 45 to 95. The highest mAP@0.5 97.2% is achieved by using default iou 45. During the training period, background false negative (BFN) and background false positive (BFP) is calculated in confusion matrix. The BFN is 0. On the contrary, BFP is 0.67 for broken, and 0.33 for healthy.

The YoloV5 small model's training result is shown in figure 6.7(b). It achieved an overall 99.8% of precision, 100% of recall, 99.5% of mAP@0.5, and 95.3% of mAP@0.5:0.95 respectively. The model validation checked by giving confidence score 0.001 and IOU value from 0.45 to 0.95. The highest mAP@0.5 of 99.5% is achieved by using default iou 0.45. In the confusion matrix,the generated BFN is 0 and BFP is 1.0 only for broken.



(a) Yolov7 training result

Figure 6.8: YoloV7 training result for dataset 1 using passive learning.

The figure 6.8(a) illustrates the training result of yolov7. It achieved an overall 99.7% precision, 99.9% recall, 99.7% of mAP@0.5, and 93.3% of mAP@0.5:0.95. We evaluated the validation of these three models by using unseen test data. The model is achieved BFN value of 0, and BFP value of .57 for broken and 0.43 of healthy class. According to the validation test, the Yolov5 model gave the best detection

result with the highest accuracy compared to YoloV7. But the inference was very fast in the Yolov7 model.

In figure 6.9, the performance of yolov5m, yolov5s, and yolov7 is shown using dataset 1. The precision and recall of each model are almost equivalent and very high. The mAP@0.5 has fluctuated for all models with a tiny ratio of percentage. YoloV5m has achieved the highest 97.1% of mAP@0.5:0.95 where the training box loss has reached below 0.01. Furthermore, the training object loss is below 0.005 and the training and validation both class loss is nearly 0.000. Moreover, the validation box loss has reached below 0.005 with the validation object loss below 0.0010.

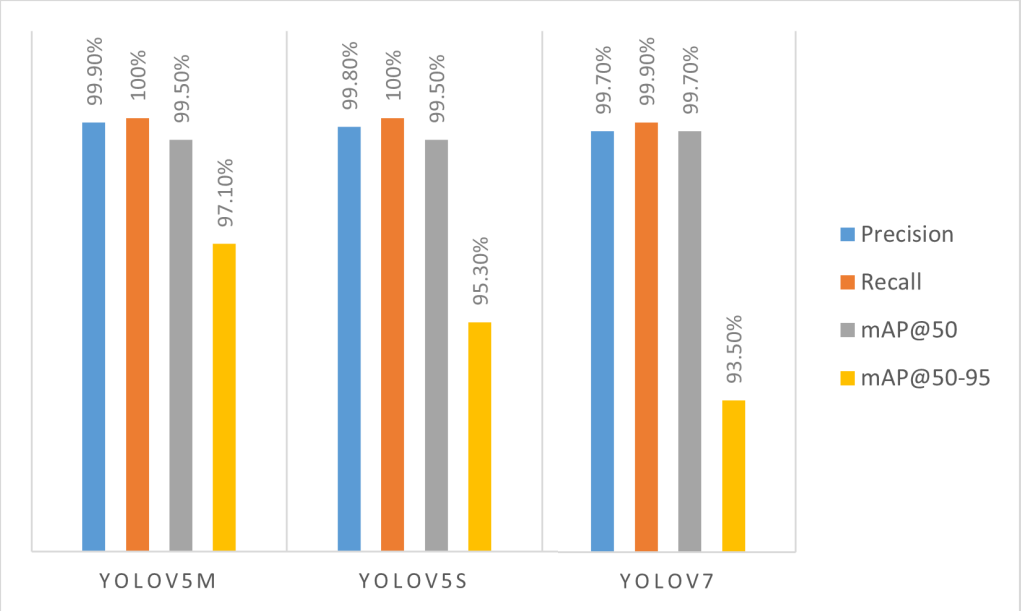
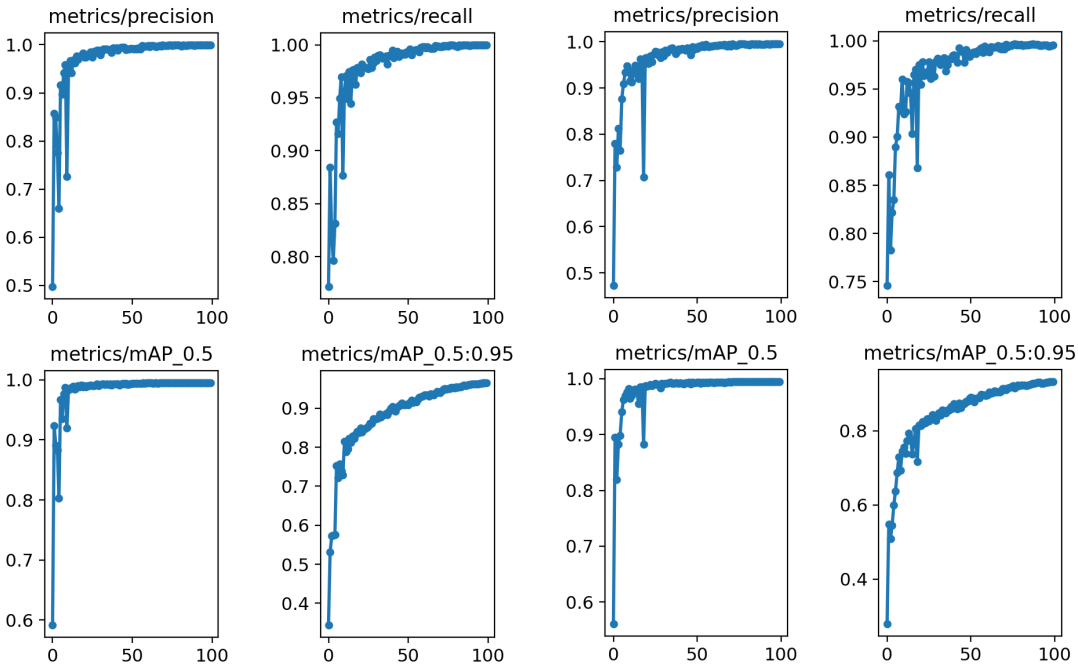


Figure 6.9: The performance comparison between YOLO's using dataset 1.

Experimental Results for Dataset 2:

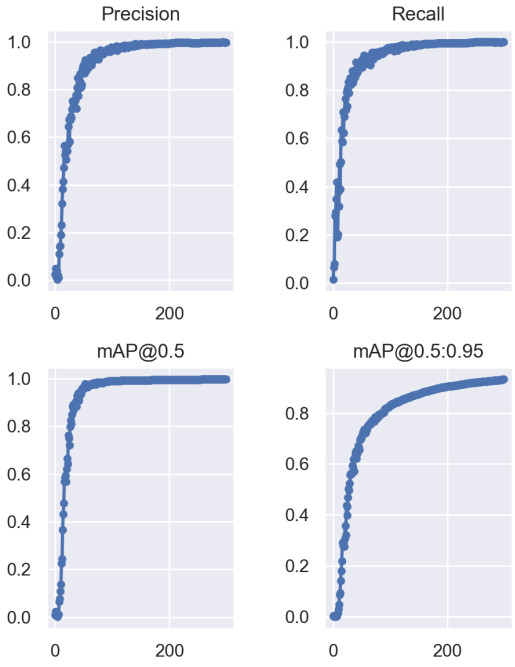
We also trained these three models using our second dataset as shown in figure 6.10. We trained YoloV5 model using 100 epochs. After training Yolov5m, we obtained both precision and recall 100%. That means There have no false positives and no false negatives in our model. All positive and negative prediction was taken as correct prediction. It also achieved 99.5% of mAP@0.5 and 96.5% of mAP@0.5:0.95 as shown in figure 6.10(a). According to the validation test, it achieved the similar results. The generated confusion matrix during the training, the BFN result is 0, but the BFP is 0.33 for broken, and 0.67 for healthy.

In addition, we have got the precision 99.6%, recall 99.5%, mAP@0.5 99.5%, and mAP@0.5:0.95 93.3% from YoloV5s model as shown in figure 6.10(b). After eva-



(a) Yolov5m training result

(b) Yolov5s training result



(c) Yolov7 training result

Figure 6.10: YoloV5, V7 model training results for dataset 2 using passive learning.

luting the validation result, we have got the similar result by using the confidence score 0.001 and changing the iou value from 0.45 to 0.95. It is slightly differ only the mAP@0.5:95 values. Moreover, BFN is 0 and BFP is 0.64 for broken class and 0.34 for healthy class. Both medium and small YoloV5 models detect few images background with the interest of objects.This is happening due to the background and lighting condition of images.

Figure 6.10(c) depicts the training results of YoloV7 and model was trained using 200 epochs with default hyperparameters. According to the figure, 99.1% precision, 99.2% recall, 99.5% mAP@0.5 and 89.5% mAP@0.5:0.95 is obtained. We have also evaluted the validation test for this model and achieved the equivalent result for iou 0.45 to 0.95 with the confidence value of 0.001. From the confusion matrix, it achieved 0.98 of true neagtives, 0.01 of false positives, 0.99 of true positives. Besides this, BFP is 0.45 for broken, 0.55 for healthy and BFN is always 0 in this model.

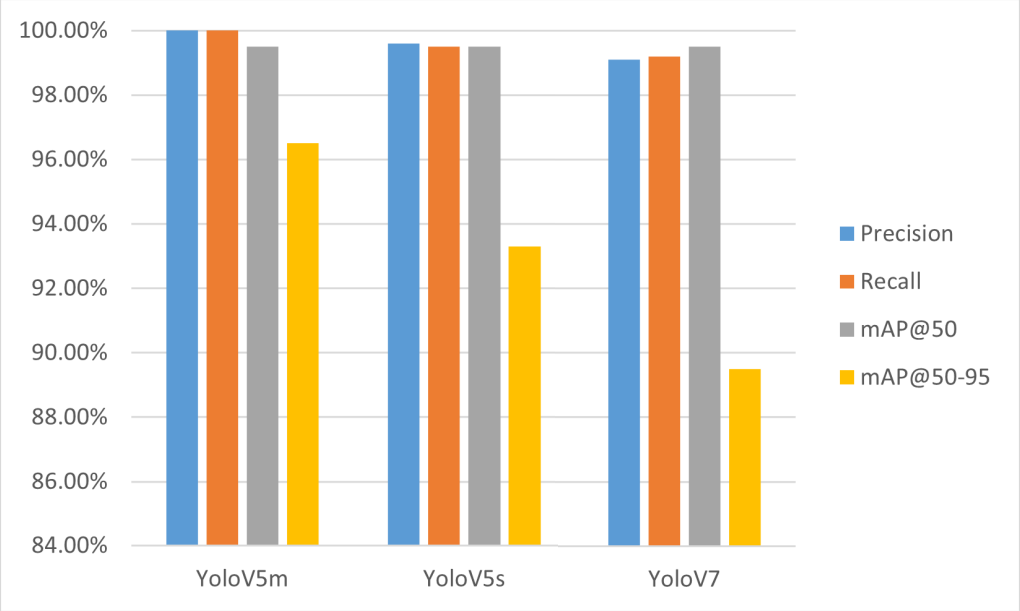


Figure 6.11: The performance comparison between YOLO's using dataset 2.

Figure 7.1 is illustrated the performance comparison between YoloV5m, Yolov5s, and Yolov7 model. These models are trained using dataset 2. Here Yolov5m is comparatively better than other models. It has achieved highest mAP@0.5:0.95 of 96.5% with precision and recall. In our experiment, the Yolov7 model achieved the lowest mAP for both dataset but the inference is faster than Yolov5.

After training these three models using different datasets, we have seen that the background of images are made impact on the performance. Each model has got the background false positive (BFP) for both classes. That means, it tries to identify

the background objects that are not a part of either class but are detected as one. It is happening due to the gray background images with black objects. The gray absorbs the black in different lighting conditions. On the other hand, it predicts almost 1.0 for both true negative as broken, and true positive as healthy. A performance comparison table for all trained models using both datasets is given below.

Dataset	Model	**Precision**	**Recall**	**mAP@0.5**	**mAP@0.5:0.95**
Dataset 1	**YoloV5m**	**99.9%**	**100%**	**99.5%**	**97.1%**
	YoloV5s	99.8%	100%	99.5%	95.3%
	YoloV7	99.7%	99.9%	99.7%	93.5%
Dataset 2	YoloV5m	100%	100%	99.5%	**96.5%**
	YoloV5s	99.6%	99.5%	99.5%	93.3%
	YoloV7	99.1%	99.2%	99.5%	89.5%

Table 6.4: The performance comparison between YoloV5 and YoloV7 using passive learning.

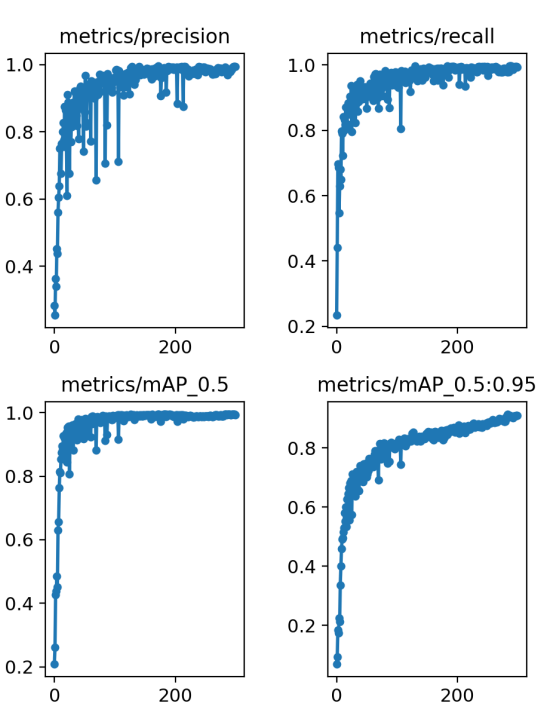
Finally, we have got the best model of YOLOV5m using dataset 1 as shown in table 6.4. It achieved the highest detection accuracy of the mAP at 97.1%.

6.2.2 Active learning evaluation

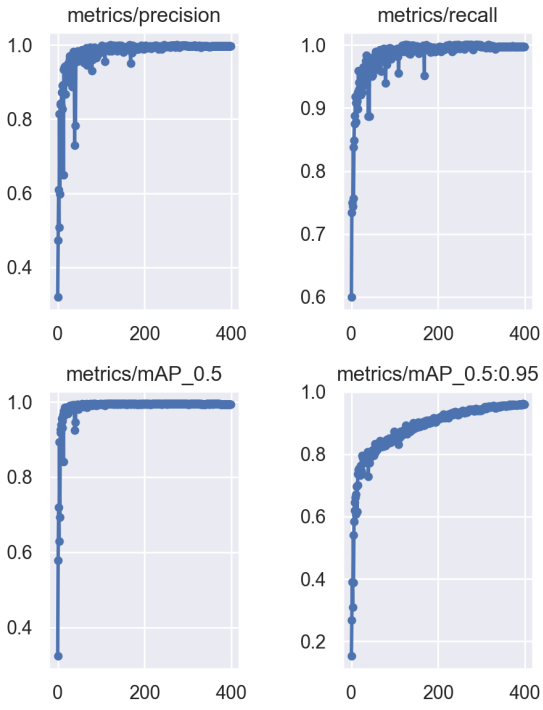
We applied deep active learning strategies (figure 4.2) to train the Yolov5m and YoloV7 models using dataset 1 based on the performance of passive learning. Mainly, it is an iterative model training process as previously described in figure 4.14. A tabular form of dataset generation for each iteration in the iterative process is already discussed in table 5.1. The training and validation classification loss, precision, recall, mAP@0.5, and mAP@0.5:0.95 curves for the 1st, 5th, 9th, and 10th iterations are illustrated below figures.

YoloV5m:

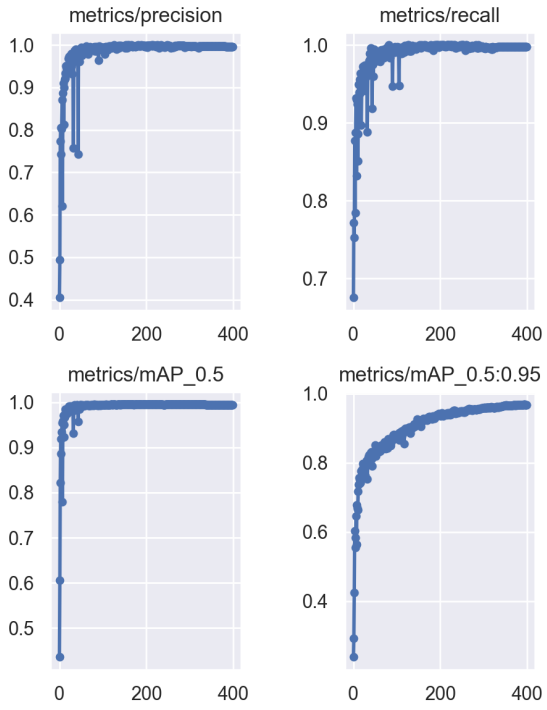
In the following figure 6.12, the first iteration (figure 6.12(a)) has been done by 754 training data. After evaluating the model, it achieved precision at 0.991, recall at 0.979, mAP@0.5 at 0.994, and mAP@0.5:0.95 at 0.913 where the training and validation class loss was a little bit high.



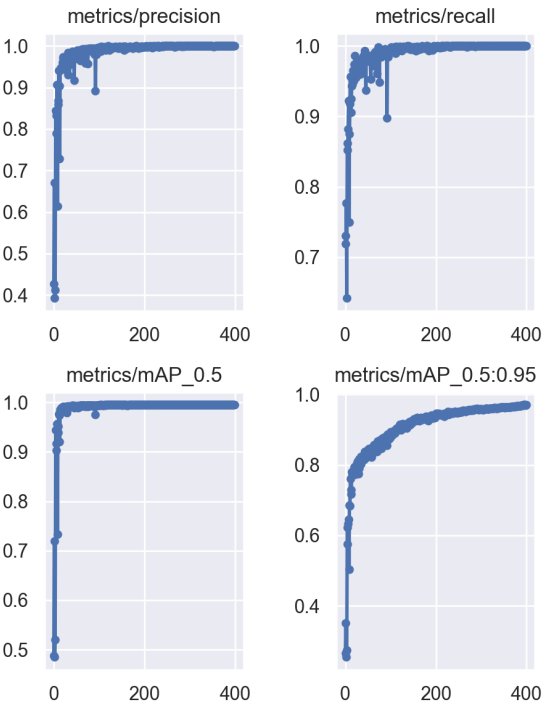
(a) 1st iteration



(b) 5th iteration



(c) 9th iteration



(d) 10th iteration

Figure 6.12: Iterative results of YoloV5 model training.

In the confusion matrix, it predicted FN at 0.02, FP at 0, and BFP is 0.50 for both classes. The 5th iteration (figure 6.12(b)) has been done by 1548 taining data and achieved better results. We got precision at 0.997, recall at 0.997, mAP@0.5 at 0.995, and mAP@0.5:0.95 at 0.961 where both class loss graphs showed the steady result. The false negative predicted 0.01 where background false positive(BFP) and background false negative(BFN) are 0 in the confusion matrix. Similarly, after training the 9th iteration (figure 6.12(c)) using 2198 training data, we got precision at 0.996, recall at 0.998, mAP@0.5 at 0.994, and mAP@0.5:0.95 at 0.968 with less training and validation class loss. Moreover, the false positive and false negative got 0 but BFP obtained 1 for the broken class in the confusion matrix. Finally, the 10th iteration (figure 6.12(a)) was done by 2340 training data. After training the model, we got precision at 1.0, recall at 1.0, mAP@0.5 at 0.995, and mAP@0.5:0.95 at 0.97 where both the training and validation class loss was nearly 0. We obtained BFP 1.0 for only healthy class where both false positive and false negative is 0 as well as both true positive and true negative is 1.0.

Iteration	**Query data (%)**	**Training data**	**Precision**	**Recall**	**mAP @0.5**	**mAP @0.5:0.95**
1	15%	754	99.10%	97.90%	99.40%	91.30%
2	20%	968	99.00%	98.80%	99.40%	93.30%
3	25%	1172	99.60%	100%	99.50%	94.60%
4	30%	1364	99.60%	99.40%	99.50%	95.30%
5	35%	1548	99.70%	99.70%	99.50%	96.10%
6	40%	1722	99.50%	99.50%	99.30%	95.60%
7	45%	1888	99.80%	99.80%	99.50%	96.90%
8	50%	2046	99.80%	99.80%	99.50%	96.70%
9	55%	2198	99.60%	99.80%	99.40%	96.80%
10	**60%**	2340	100%	100%	99.50%	**97.00%**

Table 6.5: YOLOV5 training results using active learning strategy.

Here, the following table 6.5 is representing the final result of YoloV5m with expected mAP result using active learning strategy. The main goal of this research thesis is to achieve the highest mAP value by training model with less data. We have got our expected result at iteration 10 where it has taken 60% of data for training

the model and obtained expected mAP of 97.00%.

YoloV7:

The iterative process is also applied to the YoloV7 model. In the following figure 6.13, the first iteration (figure 6.13(a)) has been done by the same 754 training data. After training the model, it achieved precision at 0.885, recall at 0.941, mAP@0.5 at 0.96, and mAP@0.5:0.95 at 0.697. The training and validation classification loss is too high. The training box and objectness loss graph are not consistence. Besides, the validation bounding box and objectness loss are also shown inconsistency. In the confusion matrix, it predicted true negative at 0.91, false negative at 0.11, false positive at 0.05, and true positive at 0.88. In addition, we got background false negative 0.05, 0.01 for broken, and healthy respectively and also background false positive(BFP) is 0.58, 0.42 for broken and healthy classes.

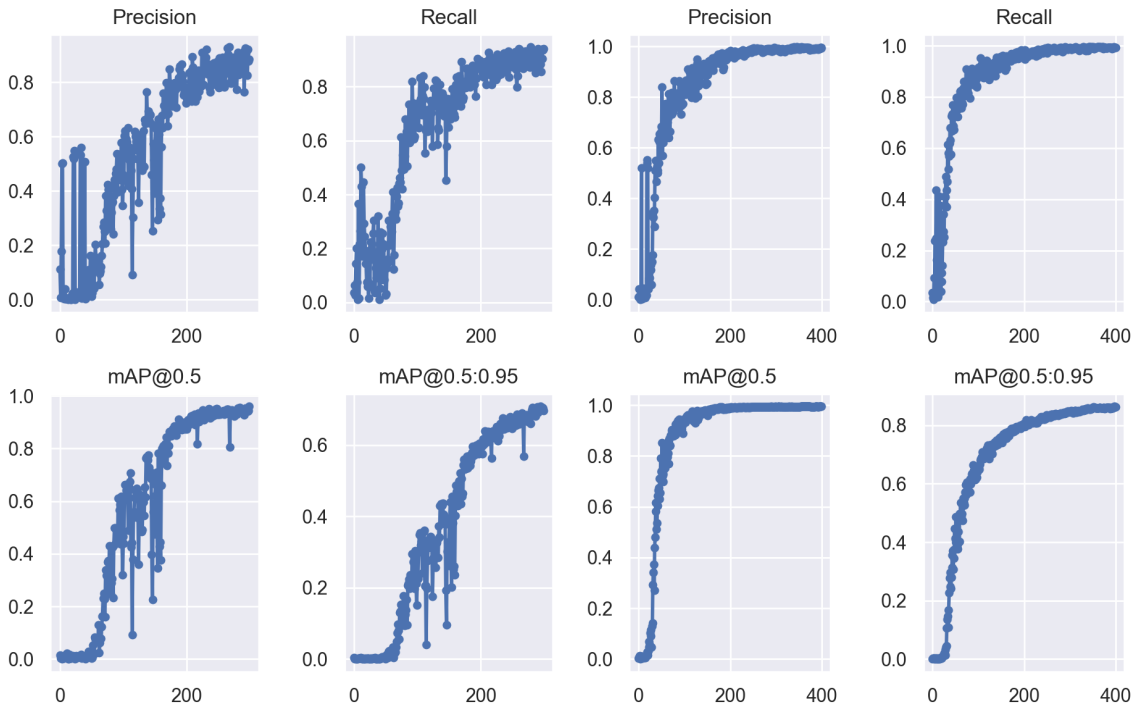
The 5th iteration (figure 6.13(b)) has been done by 1548 training data and achieved better results. We got precision at 0.994, recall at 0.992, mAP@0.5 at 0.995, and mAP@0.5:0.95 at 0.864. The training and validation classifications loss are decreased than the previous iterative results, as well as bounding box and objectness loss, are decreased. The FN and FP predicted both 0.01, where BFP got 0.62 and 0.37 for broken, healthy classes and BFN got 0 in the confusion matrix.

Similarly, after training the 9th iteration (figure 6.12(c)) using 2189 training data. We got precision at 0.996, recall at 0.997, mAP@0.5 at 0.996, and mAP@0.5:0.95 at 0.907 after model training. Furthermore, the classification loss for training and validation are remaining the same. Besides the bounding box loss is reached below 0.01. the objectness loss is 0.004 for training, and below 0.002 for validation objectness loss. Moreover, the FP predicted 0.01, FN got 0, TP predicted 1.0, and TN is 0.99 The BFP obtained 0.67 for the broken class and 0.33 for the healthy class and BFN predicted 0 in the confusion matrix.

Finally, the 10th iteration (figure 6.13(d)) was done by 2340 training data. We got precision at 0.991, recall at 0.989, mAP@0.5 at 0.996, and mAP@0.5:0.95 at 0.864. The training and validation classifications loss are increased than the previous 9th iterative results. The bounding box loss is also increased for both training and validation, as well as the objectness loss is increased slightly. The FN predicted 0, FP predicted 0.01, 0.98 for TN and TP predicted 1.0. The BFP got 0.50 for both broken and healthy classes in the confusion matrix. The expected mAP of this iteration is less than the previous iteration.

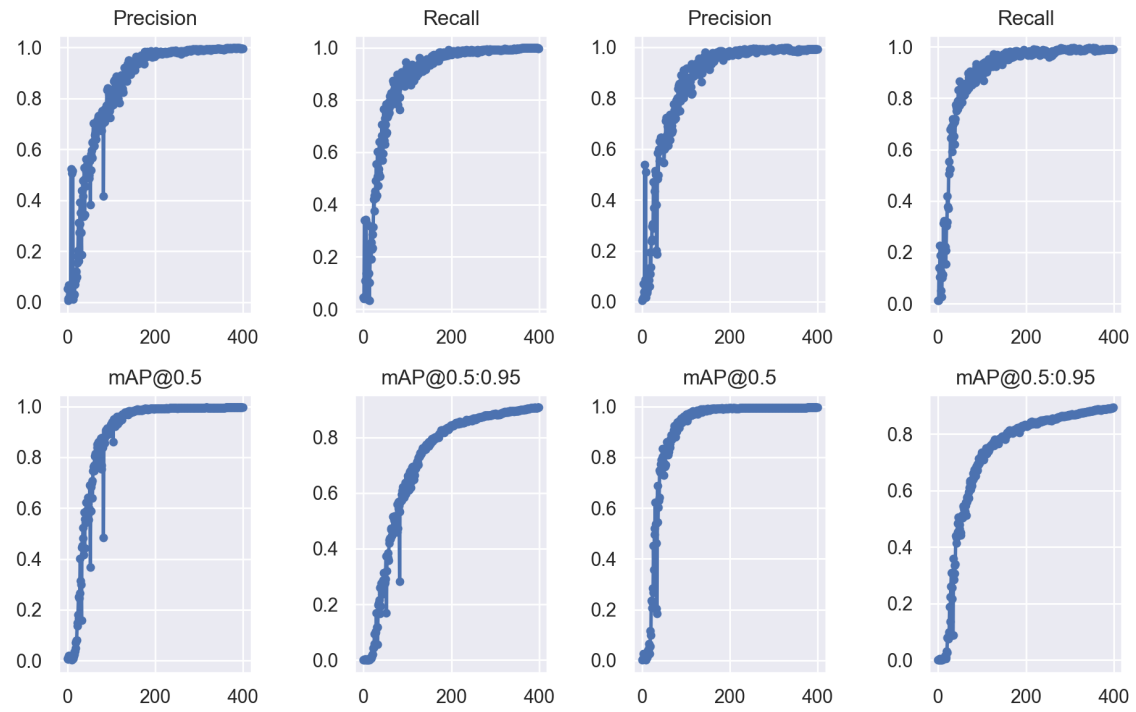
After the 10th iteration, we stopped the iterative process. We evaluated the training result and compared it with the previous one. It is shown that the result is decreasing. Finally, we have got the 9th mAP result is better than others.

The below table 6.5 is representing the final result of YoloV7 with expected mAP



(a) 1st iteration

(b) 5th iteration



(c) 9th iteration

(d) 10th iteration

Figure 6.13: Iterative results of YoloV7 model training.

using active learning strategy. The aim of this research study is to achieve the highest mAP value by training the model with less data. We have got our expected result at iteration 9 where it has taken 55% of data for training the model and obtained an expected mAP of 90.7%.

Iteration	Query data (%)	Training data	Precision	Recall	mAP @0.5	mAP @0.5:0.95
1	15%	754	90.80%	88.30%	95%	70.70%
2	20%	968	80.20%	86.00%	91.00%	61.50%
3	25%	1172	97.60%	97.90%	99.30%	82%
4	30%	1364	99.40%	98.70%	99.60%	84.50%
5	35%	1548	99.40%	99.20%	99.50%	86.40%
6	40%	1722	99.50%	99.10%	99.60%	86.40%
7	45%	1888	99.30%	99.40%	99.60%	86.10%
8	50%	2046	99.60%	99.40%	99.50%	87.40%
9	**55%**	2198	99.60%	99.70%	99.60%	**90.70%**
10	60%	2340	99.10%	98.90%	99.60%	89.40%

Table 6.6: YOLOV7 training results using active learning strategy.

6.3 Performance Analysis

In the passive learning strategy, we found that VGG-19 has achieved 97.9% accuracy for healthy (H) and broken (B) classes while ResNet-50 achieved 99.04% in dataset 1. Similarly, VGG-19 and Resnet-50 have achieved 97.59%, and 98.39% respectively in dataset 2. After testing the unseen data, we can see both VGG-19 and ResNet-50 can classify the images with a maximum accuracy of 73.11%. Classify results are illustrated in Figure 6.14. We have tested for both datasets. According to the result, dataset 1 obtained higher accuracy than dataset 2 and ResNet-50 model can classify images relatively better than VGG-19. But this accuracy is not enough sufficient for our application due to the high priority of safety issues.

We also tested the YoloV5m classifier. We obtained 96.7% top 1 accuracy, and 100% top 5 accuracy on dataset 1. Similarly in dataset 2, we got 95.5% top 1 accuracy, and 100% top 5 accuracy respectively. It has been able to classify broken

(a) VGG-19, H: 58.07% (b) VGG-19, B: 72.93% (c) VGG-19, B: 73.11%

(d) ResNet-50, H: 73.11% (e) ResNet-50, B: 73.11% (f) ResNet-50, B: 73.11%

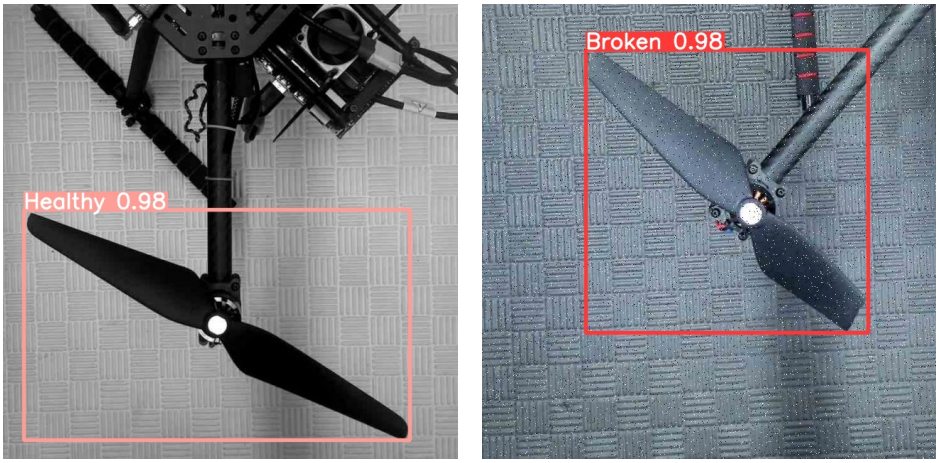
Figure 6.14: Results of (a-c) VGG-19 and (d-f) ResNet-50 classifiers.

and healthy images with a maximum accuracy rate of 95%. Sample results are illustrated in Figure 6.15. This model also achieved the highest accuracy from dataset 1 rather than dataset 2.

(a) YoloV5m-cls, H: 95% (b) YoloV5m-cls, B: 95% (c) YoloV5m-cls, B: 94%

Figure 6.15: Results of YoloV5m (medium) classifier.

We tested the YoloV5 medium, small and YoloV7 detector model using both dataset. According to dataset 1 training result, We got mAP@.5 99.5%, mAP@.5:.95 97.1% for Yolov5m, mAP@.5 99.5% and mAP.5:.95 95.3% for Yolov5s. Also, mAP.5 99.7% and mAP@.5:.95 93.5% is obtained on YoloV7. On the other hand, mAP.5 99.5% and mAP@.5:.95 96.5% reached for YoloV5m, mAP@.5 99.5% and mAP@.5:.95 93.3% for Yolov5s, mAP@.5 99.5% and mAP@.5:.95 89.5% for YoloV7 models using dataset 2.

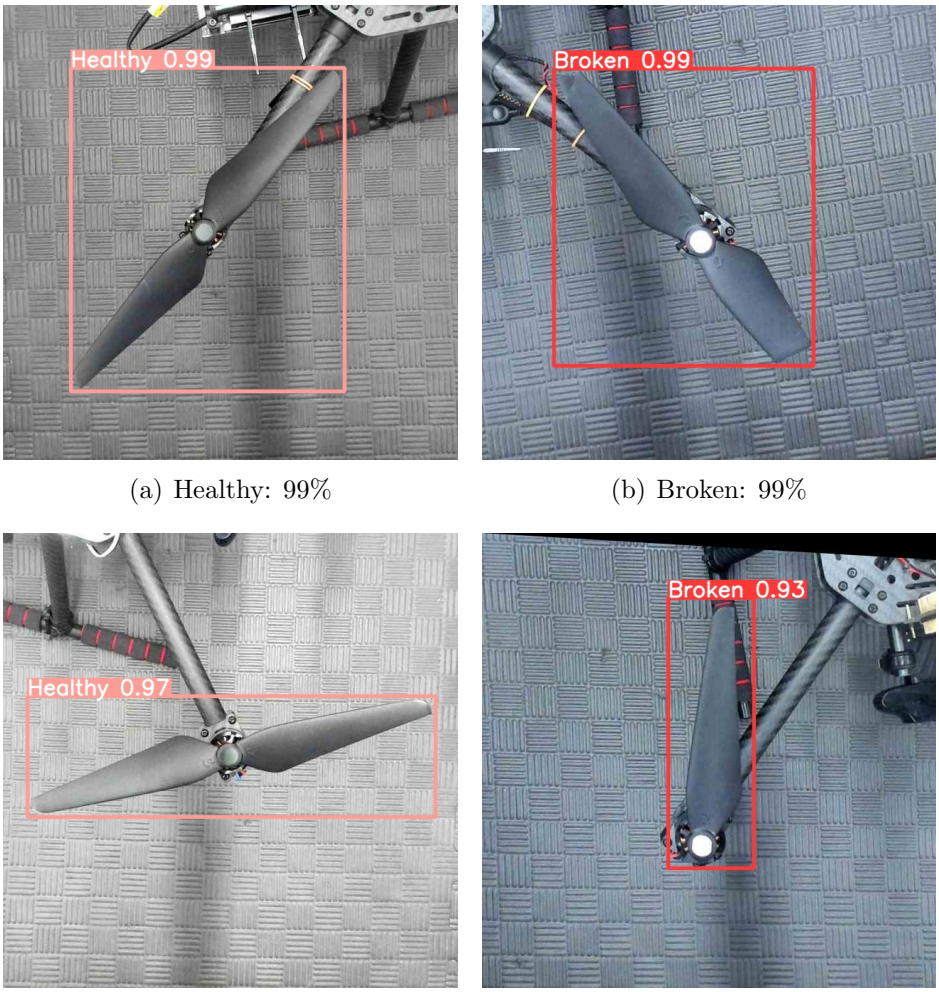


(a) Healthy: 98% (b) Broken: 98%

Figure 6.16: Results of YoloV5m object detector using passive learning.

After model testing, YoloV5m achieved the highest detection accuracy of 98% among all trained models in both dataset. The results are illustrted in Figure 6.16. According to the test results of dataset 1 and dataset 2, this model can accurately detect propeller faults in dataset 1 compared to dataset 2 with the highest accuracy.

In the active learning strategy, we have taken YoloV5m and YoloV7 model.We applied the iteration process in YoloV5 first. We have tested a total of 10 iterations and taken only 15% data from the training pool for the first iteration to train the model. The model achieved only 91.3% of mAP@.5:.95. The first iteration may not yield the expected results that were previously envisioned. From the second iteration, 5% more data was added to the dataset from the training pool and trained the model. We have applied this process iteratively. In the 5th iteration, we achieved better detection results than before. The mAP increased to 95.3%. Almost the desired result has been reached in the 9th iteration. A total of 2740 training data is used to train the model and achieved mAP 96.8%. In the 10th iteration, the model has achieved our expected mAP 97% for the detection task. A detection result is given in Figure 6.17. The detection accuracy is much better than the previous iteration results and used only 60% data from the training pool.



(a) Healthy: 99%

(b) Broken: 99%

(c) Healthy: 97%

(d) Broken: 93%

Figure 6.17: Detection results of YoloV5m at the 10th iteration using active learning.

We also repeated the iteration process 10 times to train the YoloV7 model. It achieved mAP@.5:.95 70.7% with a high bounding box and classification loss in the first iteration. The difference between precision and recall is also high. In the 5th iteration, mAP increased to 86.40%, precision to 99.4%, and recall to 99.2%. Afterwards, we got the highest mAP 90.7% for the detection task in the 5th iteration. Again in the 10th iteration, the model achieved mAP 89.4% which is lower than the result of the previous iteration. Then, we stopped the iteration process.

Finally, we found the best model with the mAP 97.00% for YoloV5m at the 10th iteration and mAP 90.70% for YoloV7 at the 9th iteration by applying the active learning approach. Based on these results, it is proved that the YoloV5m is more reliable and stable than YoloV7 in this research study for drone propeller inspection.

6.4 Inference

FPS

Frames Per Second (FPS) is a key indicator for assessing the real-time performance of computer vision applications, such as object detection, tracking, and so on. It refers to the number of images or frames that a model can process in a second. FPS is determined using the formula FPS = number of frames / time duration. After evaluating detection-based model, it is mandatory to test the real-time inference and speed in different environment with different framework. For that reason, we converted our model into ONNX format to compare the FPS between OpenCV DNN and Pytorch using our local computing machines (see 5.1.1). The results are given below:

Model	**Processor**	**GPU Access** (NVIDIA RTX 3060)	**Data**	**Framework**	**FPS**
YoloV5m	Core-i5 6th Gen	No	image	OpenCV	1.19 - 1.82
			video		0.61 - 1.41
YoloV7			image		2.34 - 3.80
			video		0.22 - 0.77
YoloV5m	Core-i7 12th Gen	No	image	OpenCV	2.43 - 2.77
			video		3.06 - 4.32
YoloV7			image		5.00 - 6.35
			video		1.65 - 2.49
YoloV5m	Core-i5 6th Gen	No	video	Pytorch	0.21 - 1.96
YoloV7					0.54 - 1.68
YoloV5m	**Core-i7 12th Gen**	**Yes**	video	**Pytorch**	**29 - 40**
YoloV7					31 - 36

Table 6.7: The FPS comparison for YoloV5m and YoloV7 detectors.

According to the table 6.7, we found YoloV5m detector model achieved the highest FPS using Pytorch framework. It can process upto 40 frames per second for our drone propeller inspection. On the other hand, YoloV7 are capable to process 30 to 37 frames per second in our GPU accelerated local computing machine.

6.5 Chapter Summary

In this chapter, we have described the model evaluation and result after completing the training process. The accuracies of various trained models following passive learning and active learning strategies are calculated and described in detail. Moreover, the model's inference and speed comparison is also mentioned.

At first, we described the evaluation matrix such as accuracy, confusion matrix, and confidence score that are used to obtain accuracy for the classification models. Also intersection over union (IoU), average precision (AP), and mean average precision (mAP) calculations that are used for the detection-based models are described.

Afterwards, we compared the performance of different classifiers and detectors using passive and active learning approaches. In passive learning, we found that ResNet-50 achieved the highest 99.04% accuracy compared to VGG-19. But after testing the unseen data, the model can classify the image with 73.11% accuracy. For that, we tested our dataset using the YoloV5m classifier. We got 96.7% Top 1 accuracy It can classify images with 95% accuracy which is a very impressive result. Moreover, we also trained the YoloV5m, YoloV5s and YoloV7 detector models using our two datasets. Yolov5m achieved the highest mAP@.5:.95 at 97.1%. YoloV5m achieved the highest detection accuracy of 98% among all trained models.

Afterwards, we used the active learning strategy for the YoloV5m and YoloV7 detectors model. We applied an iterative process and trained the models using the distinct datasets generated from the training pool dataset in each iteration. As a result, we found the best model with mAP@.5:.95 of 97.00% for YoloV5m at the 10th iteration and mAP@.5:.95 of 90.70% for YoloV7 at the 9th iteration. We found in the results that the highest detection accuracy of 99% is achieved by training the YoloV5m model using 60% of the data.

Finally, we tested the inferences and speed of our best models using our local computing machines. We converted our models into ONNX format to compare the FPS between OpenCV DNN and Pytorch. We found that YoloV5m can process up to 40 frames per second using the Pytorch framework in GPU-accelerated machines while YoloV7 is capable of up to 37 frames per second.

7 Conclusion

This research study is involved to create a robust and reliable solution for drone propeller inspection using deep learning. Several state-of-the-art classifiers and detectors performance are compared. The primary concern of this study was to implement a deep learning-based model that would obtain the highest accuracy during the inspection time. In addition, the model would be able to perform in real time as well.

7.1 Summary of The Thesis

As a part of the "Rescue Fly" operation, this research study was involved to find the best image-based solution using state-of-the-art deep learning technology. The basic convolutional neural network architecture has been described to form the technical background knowledge. Besides, the idea of the state-of-the-art solution of object classifiers and detectors with their architectural forms are also discussed. The primary focus of this thesis project is the passive and active learning strategies for model training also composed in several chapters.

The drone propeller inspection system has been developed from scratch. The proposed methodology is illustrated in Chapter 4 (see figure 4.1, 4.2). The datasets were generated following several data preprocessing steps discussed in section 4.1. The proposed active deep learning approach is elaborated in section 4.3. The several selected classifiers and detector architectures are described in section 4.2. The system requirements and the training process for the model are described in sections 5.1 and 5.1 respectively. Afterwards, VGG-19, ResNet-50, and Yolov5m classifier training results are compared using the passive learning strategy and a detailed discussion is given in section 6.2.1. In addition, the Yolov5m and YoloV7 detector models are also trained and compared in this section. Furthermore, the active deep learning approach is used to obtain the highest accuracy from the detector models using fewer data and their training results are described in section 6.2.2. Then the trained model's performances using test data are shown in section 6.3. The FPS comparison of models in real-time using different frameworks in two local computing resources is described in section 6.4. Finally, the best model deployment process is briefly described in section 5.3. **As per our experiment**, we found that YoloV5m achieved the highest accuracy for both classification and detection tasks. YoloV5m classifier has obtained **96.7% Top-1 accuracy** for classification. On the other hand, the YoloV5m detector model acquired the highest **mAP 97.1%** for detection using the

passive learning strategy. Moreover, this model achieved **97% mAP using only 60% of the data** following the active learning strategy. Also, it achieved **40 FPS** real-time inference in our GPU-accelerated local machine.

An overview of thesis summary is described below:

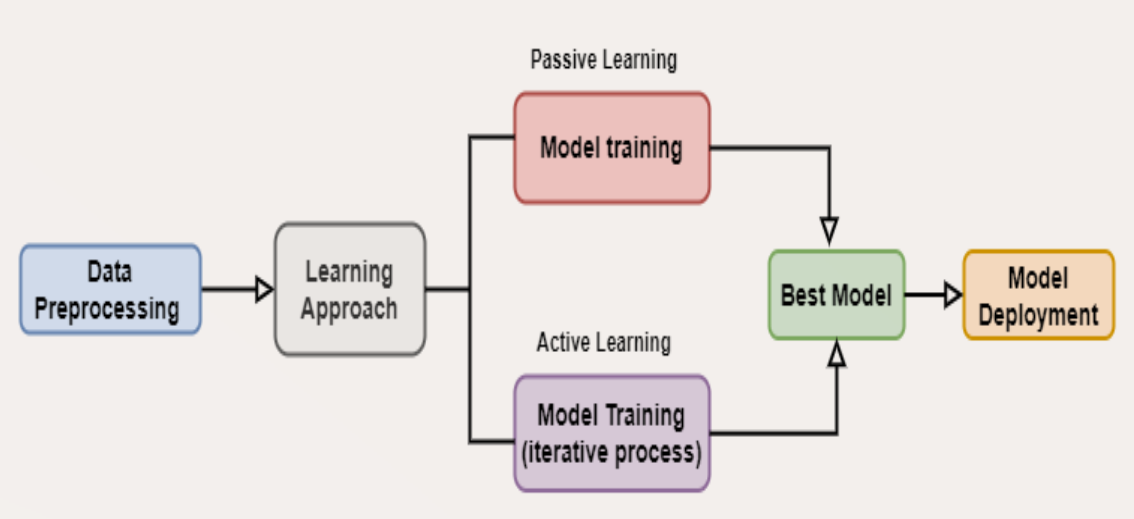


Figure 7.1: An overview of the thesis workflow

7.2 Future Work

Though the suggested approach has positive results, nevertheless there is always a possibility of enhancement. Based on the results achieved and the goals of this research study, the following advancements are recommended for future directions:

- The main focus of this research study was to develop an automated drone propeller inspection. The models achieved remarkable accuracy and inference in real-time using a limited dataset. Nevertheless, the performance of the model can be enhanced by adding more diversity of images to a well-balanced dataset.
- The low quality of the background image creates more background false positive and negative results during the training and impacts accuracy. To improve the model performance, attention can be paid more to examining the effects of background images as well as lighting conditions.
- It is possible to use different backbone network in YoloV5. After training customized model, performance can be compared with the real YoloV5 model. Besides, the frames per second (FPS) can be increased by converting the model into different types of deep learning frameworks.

Bibliography

[1] J.-y. Lee, W.-t. Lee, S.-h. Ko, and H.-s. Oh, "Fault classification and diagnosis of uav motor based on estimated nonlinear parameter of steady-state model," *Int. J. Mech. Eng. Robot. Res*, vol. 10, pp. 22–31, 2020.

[2] G. Iannace, G. Ciaburro, and A. Trematerra, "Fault diagnosis for uav blades using artificial neural network," *Robotics*, vol. 8, no. 3, p. 59, 2019.

[3] M. H. M. Ghazali and W. Rahiman, "An investigation of the reliability of different types of sensors in the real-time vibration-based anomaly inspection in drone," *Sensors*, vol. 22, no. 16, p. 6015, 2022.

[4] S. Albelwi and A. Mahmood, "A framework for designing the architectures of deep convolutional neural networks," *Entropy*, vol. 19, no. 6, p. 242, 2017.

[5] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, "Convolutional neural networks: an overview and application in radiology," *Insights into imaging*, vol. 9, pp. 611–629, 2018.

[6] Dharmaraj, "Zero-padding in convolutional neural networks," https://medium.com/@draj0718/zero-padding-in-convolutional-neural-networks-bf1410438e99, Sep. 2021, accessed: 2023-2-26.

[7] S. Khan, H. Rahmani, S. A. A. Shah, and M. Bennamoun, "A guide to convolutional neural networks for computer vision," *Synthesis Lectures on Computer Vision*, vol. 8, no. 1, pp. 1–207, 2018.

[8] "CS231n Convolutional Neural Networks for Visual Recognition." [Online]. Available: https://cs231n.github.io/convolutional-networks/#pool

[9] S. Saha, "A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way," Dec. 2018. [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[10] P. Gavali and J. S. Banu, "Deep convolutional neural network for image classification on cuda platform," in *Deep learning and parallel computing environment for bioengineering systems*. Elsevier, 2019, pp. 99–122.

[11] https://machinelearningmastery.com/object-recognition-with-deep-learning/, accessed: 2023-3-2.

[12] M. Carranza-García, J. Torres-Mateo, P. Lara-Benítez, and J. García-Gutiérrez, "On the Performance of One-Stage and Two-Stage Object Detectors in Autonomous Vehicles Using Camera Data," *Remote Sensing*, vol. 13, no. 1, p. 89, Jan. 2021, number: 1 Publisher: Multidisciplinary Digital Publishing Institute. [Online]. Available: https://www.mdpi.com/2072-4292/13/1/89

[13] https://www.datanami.com/2022/01/11/big-growth-forecasted-for-big-data/, accessed: 2023-3-9.

[14] B. Settles, "Active learning literature survey," 2009.

[15] J. Wang, Y. Ma, L. Zhang, R. X. Gao, and D. Wu, "Deep learning for smart manufacturing: Methods and applications," *Journal of manufacturing systems*, vol. 48, pp. 144–156, 2018.

[16] Y. Xiao, Z. Tian, J. Yu, Y. Zhang, S. Liu, S. Du, and X. Lan, "A review of object detection based on deep learning," *Multimedia Tools and Applications*, vol. 79, pp. 23 729–23 791, 2020.

[17] Z. Zou, K. Chen, Z. Shi, Y. Guo, and J. Ye, "Object detection in 20 years: A survey," *Proceedings of the IEEE*, 2023.

[18] T. Hoeser and C. Kuenzer, "Object detection and image segmentation with deep learning on earth observation data: A review-part i: Evolution and recent trends," *Remote Sensing*, vol. 12, no. 10, p. 1667, 2020.

[19] H. Lin and J. Yang, "Ensemble cross-stage partial attention network for image classification," *IET Image Processing*, vol. 16, no. 1, pp. 102–112, 2022.

[20] J. Glenn, "ultralytics /yolov5," https://github.com/ultralytics/yolov5, Aug. 21 2020, accessed: 2022-20-10.

[21] L. Liu, W. Ouyang, X. Wang, P. Fieguth, J. Chen, X. Liu, and M. Pietikäinen, "Deep learning for generic object detection: A survey," *International journal of computer vision*, vol. 128, pp. 261–318, 2020.

[22] M. Ivanov, "The evolution of the YOLO neural networks family from v1 to v7," https://medium.com/deelvin-machine-learning/the-evolution-of-the-yolo-neural-networks-family-from-v1-to-v7-96d0687b4dce, Oct. 2022, accessed: 2023-3-27.

[23] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," *arXiv preprint arXiv:2207.02696*, 2022.

[24] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[25] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[26] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[27] Simplilearn, "Docker architecture: Understanding how docker works with examples," https://www.simplilearn.com/tutorials/docker-tutorial/docker-architecture, May 2021, accessed: 2023-3-30.

[28] C. V. Nicholson and A. Gibson, "Early stopping," https://mgubaidullin.github.io/deeplearning4j-docs/earlystopping, accessed: 2023-4-2.

[29] M. Honarmand and H. Shahriari, "Geological mapping using drone-based photogrammetry: An application for exploration of vein-type cu mineralization," *Minerals*, vol. 11, no. 6, p. 585, 2021.

[30] N. A. Khan, N. Jhanjhi, S. N. Brohi, R. S. A. Usmani, and A. Nayyar, "Smart traffic monitoring system using unmanned aerial vehicles (uavs)," *Computer Communications*, vol. 157, pp. 434–443, 2020.

[31] J. N. McRae, C. J. Gay, B. M. Nielsen, and A. P. Hunt, "Using an unmanned aircraft system (drone) to conduct a complex high altitude search and rescue operation: a case study," *Wilderness & environmental medicine*, vol. 30, no. 3, pp. 287–290, 2019.

[32] C. Van Tilburg, "First report of using portable unmanned aircraft systems (drones) for search and rescue," *Wilderness & environmental medicine*, vol. 28, no. 2, pp. 116–118, 2017.

[33] H. Shakhatreh, A. H. Sawalmeh, A. Al-Fuqaha, Z. Dou, E. Almaita, I. Khalil, N. S. Othman, A. Khreishah, and M. Guizani, "Unmanned Aerial Vehicles (UAVs): A Survey on Civil Applications and Key Research Challenges," *IEEE Access*, vol. 7, pp. 48 572–48 634, 2019, conference Name: IEEE Access.

[34] S. Hayat, E. Yanmaz, and R. Muzaffar, "Survey on Unmanned Aerial Vehicle Networks for Civil Applications: A Communications Viewpoint," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 4, pp. 2624–2661, 2016. [Online]. Available: http://ieeexplore.ieee.org/document/7463007/

[35] N. Hossein Motlagh, T. Taleb, and O. Arouk, "Low-Altitude Unmanned Aerial Vehicles-Based Internet of Things Services: Comprehensive Survey and Future Perspectives," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 899–922, Dec. 2016, conference Name: IEEE Internet of Things Journal.

[36] R. L. Medeiros, A. C. Lima Filho, J. G. G. Ramos, T. P. Nascimento, and A. V. Brito, "A novel approach for speed and failure detection in brushless dc motors based on chaos," *IEEE Transactions on Industrial Electronics*, vol. 66, no. 11, pp. 8751–8759, 2018.

[37] A. Altinors, F. Yol, and O. Yaman, "A sound based method for fault detection with statistical feature extraction in uav motors," *Applied Acoustics*, vol. 183, p. 108325, 2021.

[38] J. de Jesus Rangel-Magdaleno, J. Ureña-Ureña, A. Hernández, and C. Perez-Rubio, "Detection of unbalanced blade on uav by means of audio signal," in *2018 IEEE International Autumn Meeting on Power, Electronics and Computing (ROPEC)*. IEEE, 2018, pp. 1–5.

[39] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *2017 international conference on engineering and technology (ICET)*. Ieee, 2017, pp. 1–6.

[40] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi, "A survey of the recent architectures of deep convolutional neural networks," *Artificial intelligence review*, vol. 53, pp. 5455–5516, 2020.

[41] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," *arXiv preprint arXiv:1511.08458*, 2015.

[42] "What are convolutional neural networks?" https://www.ibm.com/topics/convolutional-neural-networks, accessed: 2023-2-27.

[43] D. Unzueta, "Convolutional layers vs fully connected layers," https://towardsdatascience.com/convolutional-layers-vs-fully-connected-layers-364f05ab460b, Nov. 2021, accessed: 2023-2-27.

[44] Y. Liu, P. Sun, N. Wergeles, and Y. Shang, "A survey and performance evaluation of deep learning methods for small object detection," *Expert Systems with Applications*, vol. 172, p. 114602, 2021.

[45] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, 2015.

[46] S. Zhang, G. He, H.-B. Chen, N. Jing, and Q. Wang, "Scale adaptive proposal network for object detection in remote sensing images," *IEEE Geoscience and Remote Sensing Letters*, vol. 16, no. 6, pp. 864–868, 2019.

[47] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2980–2988.

[48] L. Du, R. Zhang, and X. Wang, "Overview of two-stage object detection algorithms," in *Journal of Physics: Conference Series*, vol. 1544, no. 1. IOP Publishing, 2020, p. 012033.

[49] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[50] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[51] P. Ren, Y. Xiao, X. Chang, P.-Y. Huang, Z. Li, B. B. Gupta, X. Chen, and X. Wang, "A survey of deep active learning," *ACM computing surveys (CSUR)*, vol. 54, no. 9, pp. 1–40, 2021.

[52] J. Sivic and A. Zisserman, "Video google: A text retrieval approach to object matching in videos," in *Computer Vision, IEEE International Conference on*, vol. 3. IEEE Computer Society, 2003, pp. 1470–1470.

[53] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, pp. 273–297, 1995.

[54] P. Cunningham and S. J. Delany, "k-nearest neighbour classifiers-a tutorial," *ACM computing surveys (CSUR)*, vol. 54, no. 6, pp. 1–25, 2021.

[55] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, pp. 91–110, 2004.

[56] N. Kambhatla and T. K. Leen, "Dimension reduction by local principal component analysis," *Neural computation*, vol. 9, no. 7, pp. 1493–1516, 1997.

[57] P. Felzenszwalb, D. McAllester, and D. Ramanan, "A discriminatively trained, multiscale, deformable part model," in *2008 IEEE conference on computer vision and pattern recognition*. Ieee, 2008, pp. 1–8.

[58] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *Computer Vision–ECCV 2006: 9th European Conference on Computer Vision, Graz, Austria, May 7-13, 2006. Proceedings, Part I 9*. Springer, 2006, pp. 430–443.

[59] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, vol. 1. Ieee, 2005, pp. 886–893.

[60] R. Lienhart and J. Maydt, "An extended set of haar-like features for rapid object detection," in *Proceedings. international conference on image processing*, vol. 1. IEEE, 2002, pp. I–I.

[61] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001*, vol. 1. Ieee, 2001, pp. I–I.

[62] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, no. 9, pp. 1627–1645, 2009.

[63] A. Vedaldi, V. Gulshan, M. Varma, and A. Zisserman, "Multiple kernels for object detection," in *2009 IEEE 12th international conference on computer vision*. IEEE, 2009, pp. 606–613.

[64] J. R. Uijlings, K. E. Van De Sande, T. Gevers, and A. W. Smeulders, "Selective search for object recognition," *International journal of computer vision*, vol. 104, pp. 154–171, 2013.

[65] P. Viola and M. J. Jones, "Robust real-time face detection," *International journal of computer vision*, vol. 57, pp. 137–154, 2004.

[66] P. F. Felzenszwalb, R. B. Girshick, and D. McAllester, "Cascade object detection with deformable part models," in *2010 IEEE Computer society conference on computer vision and pattern recognition*. Ieee, 2010, pp. 2241–2248.

[67] R. Girshick, P. Felzenszwalb, and D. McAllester, "Object detection with grammar models," *Advances in neural information processing systems*, vol. 24, 2011.

[68] R. B. Girshick, *From rigid templates to grammars: Object detection with structured models*. The University of Chicago, 2012.

[69] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems*, vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://papers.nips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html

[70] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*. Springer, 2014, pp. 818–833.

[71] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[72] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*. PMLR, 2015, pp. 448–456.

[73] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[74] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*, 2020.

[75] C.-Y. Wang, H.-Y. M. Liao, Y.-H. Wu, P.-Y. Chen, J.-W. Hsieh, and I.-H. Yeh, "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*, 2020, pp. 390–391.

[76] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 31, no. 1, 2017.

[77] J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," 2017, pp. 7263–7271. [Online]. Available: https://openaccess.thecvf.com/content_cvpr_2017/html/Redmon_YOLO9000_Better_Faster_CVPR_2017_paper.html

[78] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 37, no. 9, pp. 1904–1916, 2015.

[79] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

[80] J. Dai, Y. Li, K. He, and J. Sun, "R-fcn: Object detection via region-based fully convolutional networks," *Advances in neural information processing systems*, vol. 29, 2016.

[81] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.

[82] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*. Springer, 2016, pp. 21–37.

[83] J. Redmon and A. Farhadi, "Yolo9000: better, faster, stronger," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7263–7271.

[84] A. Farhadi and J. Redmon, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[85] M. Tan, R. Pang, and Q. V. Le, "Efficientdet: Scalable and efficient object detection," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 10 781–10 790.

[86] Y. Liu, Z. Wang, X. Wu, F. Fang, and A. S. Saqlain, "Cloud-edge-end cooperative detection of wind turbine blade surface damage based on lightweight deep learning network," *IEEE Internet Computing*, 2022.

[87] M. Crous, B. O. K. Intelligentie, and A. Visser, "Combining weakly and strongly supervised segmentation methods for wind turbine damage annotation," *Computer Science*, 2018.

[88] A. Shihavuddin, X. Chen, V. Fedorov, A. Nymark Christensen, N. Andre Brogaard Riis, K. Branner, A. Bjorholm Dahl, and R. Reinhold Paulsen, "Wind turbine surface damage detection by deep learning aided drone inspection analysis," *Energies*, vol. 12, no. 4, p. 676, 2019.

[89] J. Zhang, G. Cosma, and J. Watkins, "Image enhanced mask r-cnn: A deep learning pipeline with new evaluation measures for wind turbine blade defect detection and classification," *Journal of Imaging*, vol. 7, no. 3, p. 46, 2021.

[90] D. Sarkar and S. K. Gunturi, "Wind turbine blade structural state evaluation by hybrid object detector relying on deep learning models," *Journal of Ambient Intelligence and Humanized Computing*, vol. 12, pp. 8535–8548, 2021.

[91] A. Foster, O. Best, M. Gianni, A. Khan, K. Collins, and S. Sharma, "Drone footage wind turbine surface damage detection," in *2022 IEEE 14th Image, Video, and Multidimensional Signal Processing Workshop (IVMSP)*. IEEE, 2022, pp. 1–5.

[92] A. Shihavuddin, M. R. A. Rashid, M. H. Maruf, M. A. Hasan, M. A. ul Haq, R. H. Ashique, and A. Al Mansur, "Image based surface damage detection of renewable energy installations using a unified deep learning approach," *Energy Reports*, vol. 7, pp. 4566–4576, 2021.

[93] D. Liao, Z. Cui, X. Zhang, J. Li, W. Li, Z. Zhu, and N. Wu, "Surface defect detection and classification of si3n4 turbine blades based on convolutional neural network and yolov5," *Advances in Mechanical Engineering*, vol. 14, no. 2, p. 16878132221081580, 2022.

[94] R. Zhang and C. Wen, "Sod-yolo: A small target defect detection algorithm for wind turbine blades based on improved yolov5," *Advanced Theory and Simulations*, vol. 5, no. 7, p. 2100631, 2022.

[95] U. Nepal and H. Eslamiat, "Comparing yolov3, yolov4 and yolov5 for autonomous landing spot detection in faulty uavs," *Sensors*, vol. 22, no. 2, p. 464, 2022.

[96] D. Wallach and B. Goffinet, "Mean squared error of prediction as a criterion for evaluating and comparing system models," *Ecological modelling*, vol. 44, no. 3-4, pp. 299–306, 1989.

[97] O. Elharrouss, Y. Akbari, N. Almaadeed, and S. Al-Maadeed, "Backbones-review: Feature extraction networks for deep learning and deep reinforcement learning approaches," *arXiv preprint arXiv:2206.08016*, 2022.

References form Professorship of Computer Engineering

[TUC1] C. Sunduijav, W. Hardt, and Z. Bayasgalan, "Image Processing of Insulator and Vibration Damper by YOLO Algorithm," in *2021 XV International Scientific-Technical Conference on Actual Problems Of Electronic Instrument Engineering (APEIE)*, Nov. 2021, pp. 375–379, iSSN: 2473-8573.

[TUC2] B. Battseren, M. S. Harras, and S. ., "Deep-learning-based insulator detector for edge computing platforms," 11 2021.

[TUC3] A. J. Chaudhry, "Burn-Mark Detection Based on Active Deep Learning," 2021.

[TUC4] B. Battseren and M. S. Harras, "Deep-Learning-Based Insulator Detector for Edge Computing Platforms," in *International Symposium on Computer Science, Computer Engineering and Educational Technology (ISCSET-2021)*, p. 21.

[TUC5] S. Saleh, S. A. Khwandah, A. Heller, A. Mumtaz, and W. Hardt, "Traffic signs recognition and distance estimation using a monocular camera," in *6th International Conference Actual Problems of System and Software Engineering.[online] Moscow: IEEE*, 2019, pp. 407–418.

Chemnitzer Informatik-Berichte

In der Reihe der Chemnitzer Informatik-Berichte sind folgende Berichte erschienen:

CSR-20-01 Danny Kowerko, Chemnitzer Linux-Tage 2019 - LocalizeIT Workshop, Januar 2020, Chemnitz

CSR-20-02 Robert Manthey, Tom Kretzschmar, Falk Schmidsberger, Hussein Hussein, René Erler, Tobias Schlosser, Frederik Beuth, Marcel Heinz, Thomas Kronfeld, Maximilian Eibl, Marc Ritter, Danny Kowerko, Schlussbericht zum InnoProfile-Transfer Begleitprojekt localizeI, Januar 2020, Chemnitz

CSR-20-03 Jörn Roth, Reda Harradi und Wolfram Hardt, Indoor Lokalisierung auf Basis von Ultra Wideband Modulen zur Emulation von GPS Positionen, Februar 2020, Chemnitz

CSR-20-04 Christian Graf, Reda Harradi, René Schmidt, Wolfram Hardt, Automatisierte Kameraausrichtung für Micro Air Vehicle basierte Inspektion, März 2020, Chemnitz

CSR-20-05 Julius Lochbaum, René Bergelt, Time Pech, Wolfram Hardt, Erzeugung von Testdaten für automatisiertes Fahren auf Basis eines Open Source Fahrsimulators, März 2020, Chemnitz

CSR-20-06 Narankhuu Natsagdorj, Uranchimeg Tudevdagva, Jiantao Zhou, Logical Structure of Structure Oriented Evaluation for E-Learning, April 2020, Chemnitz

CSR-20-07 Batbayar Battseren, Reda Harradi, Fatih Kilic, Wolfram Hardt, Automated Power Line Inspection, September 2020, Chemnitz

CSR-21-01 Marco Stephan, Batbayar Battseren, Wolfram Hardt, UAV Flight using a Monocular Camera, März 2021, Chemnitz

CSR-21-02 Hasan Aljzaere, Owes Khan, Wolfram Hardt, Adaptive User Interface for Automotive Demonstrator, Juli 2021, Chemnitz

CSR-21-03 Chibundu Ogbonnia, René Bergelt, Wolfram Hardt, Embedded System Optimization of Radar Post-processing in an ARM CPU Core, Dezember 2021, Chemnitz

CSR-21-04 Julius Lochbaum, René Bergelt, Wolfram Hardt, Entwicklung und Bewertung von Algorithmen zur Umfeldmodellierung mithilfe von Radarsensoren im Automotive Umfeld, Dezember 2021, Chemnitz

Chemnitzer Informatik-Berichte

CSR-22-01 Henrik Zant, Reda Harradi, Wolfram Hardt, Expert System-based Embedded Software Module and Ruleset for Adaptive Flight Missions, September 2022, Chemnitz

CSR-23-01 Stephan Lede, René Schmidt, Wolfram Hardt, Analyse des Ressourcenverbrauchs von Deep Learning Methoden zur Einschlagslokalisierung auf eingebetteten Systemen, Januar 2023, Chemnitz

CSR-23-02 André Böhle, René Schmidt, Wolfram Hardt, Schnittstelle zur Datenakquise von Daten des Lernmanagementsystems unter Berücksichtigung bestehender Datenschutzrichtlinien, Januar 2023, Chemnitz

CSR-23-03 Falk Zaumseil, Sabrina, Bräuer, Thomas L. Milani, Guido Brunnett, Gender Dissimilarities in Body Gait Kinematics at Different Speeds, März 2023, Chemnitz

CSR-23-04 Tom Uhlmann, Sabrina Bräuer, Falk Zaumseil, Guido Brunnett, A Novel Inexpensive Camera-based Photoelectric Barrier System for Accurate Flying Sprint Time Measurement, März 2023, Chemnitz

CSR-23-05 Samer Salamah, Guido Brunnett, Sabrina Bräuer, Tom Uhlmann, Oliver Rehren, Katharina Jahn, Thomas L. Milani, Güunter Daniel Rey, NaturalWalk: An Anatomy-based Synthesizer for Human Walking Motions, März 2023, Chemnitz

CSR-24-01 Seyhmus Akaslan, Ariane Heller, Wolfram Hardt, Hardware-Supported Test Environment Analysis for CAN Message Communication, Juni 2024, Chemnitz

CSR-24-02 S. M. Rizwanur Rahman, Wolfram Hardt, Image Classification for Drone Propeller Inspection using Deep Learning, August 2024, Chemnitz

Chemnitzer Informatik-Berichte

ISSN 0947-5125

Herausgeber: Fakultät für Informatik, TU Chemnitz
Straße der Nationen 62, D-09111 Chemnitz