Scene Understanding for Billiard Ball Striking System

Boning Yang

A project report submitted to the Auckland University of Technology

in partial fulfillment of the requirements for the degree of

Master of Computer and Information Sciences (MCIS)

2023

School of Engineering, Computer & Mathematical Sciences

Abstract

In billiards competition and training, the stability of a player's shot is a critical factor influencing match outcomes and individual skill development. This study employs the YOLOv8 (You Only Look Once version 8) algorithm to perform deep learning and recognition on billiards scenarios, aiming to real-time assess the shot stability of players or beginners. The project focuses on two key aspects: The movement of the cue tip and the "bridge hand" (i.e., the point of contact between the player's hand and the cue stick) during the shot. By continuously monitoring the position and motion of these two elements, the model can accurately predict shot stability and provide corresponding training or adjustment suggestions. Preliminary experimental results indicate that our method can effectively identify unstable shooting behaviors and contribute to enhancing players' overall technical skills.

In this project, 472 images were manually annotated from 163 minutes of video footage. The annotated data, initially in JSON format, were converted to YOLO format and fed into the official YOLOv8 model for training. This yielded a customized model tailored for detecting the cue tip and the player's bridge hand. Detected targets were then normalized to identify key points. The stability of the player's shot was assessed by calculating the slope of the line connecting points from consecutive frames.

Keywords: YOLOv8, Deep learning, OpenCV, Annotation, Real-time assessment

Table of Contents

List of Figures

List of Tables

Attestation of Authorship

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person (except where explicitly defined in the acknowledgments), nor material which to a substantial extent has been submitted for the award of any other degree or diploma of a university or other institution of higher learning.

Signature: Boning Yang   Date: 5 Oct 2023

Acknowledgment

First of all, I would like to extend my heartfelt thanks to my parents for their financial support. Their selfless and generous sponsorship enabled me to pursue and complete my Master’s study at the Auckland University of Technology (AUT), New Zealand.

I also wish to express my profound gratitude to my primary supervisor, Dr. Wei Qi Yan. Throughout this project, he not only endowed me with professional knowledge and attentive guidance but also generously provided the necessary hardware support, including a camera and tripod. I firmly believe that without Dr. Yan’s unwavering supervision and guidance, the completion of my studies would have been immensely challenging.

Boning Yang

Auckland, New Zealand

October 2023

Chapter 1
Introduction

Chapter1 comprises 5 sections: 1st section introduces the background and motivation, 2nd section presents the research question, 3rd section details the contributions, the 4th outlines the objectives, and the last describes the structure of this report.

1.1 Background and Motivation

In recent years, despite the continuous development of billiard techniques and training methods, for many players and coaches, accurately assessing and improving a player's shot stability remains a significant challenge. Every stroke,slight change in angle and force, can lead to substantial positional variations, potentially altering the entire trajectory of a match. Billiards, a sport that balances skill and strategy, is beloved by enthusiasts. A successful billiard shot depends not only on strategy and visual judgment but even more so on the stability of the player's stroke, which directly influences the ball's path and the final outcome of the game.

Traditionally, training heavily relies on the experience and observational skills of coaches. However, this method cannot escape its inherent subjectivity, especially when trying to capture subtle changes in a player's movements. For individual players, self-assessing the shot stability often proves challenging, frequently requiring the onsite guidance of professional coaches or advanced players. And while these traditional methods are rich in experience, they rarely provide players with precise, quantifiable feedback.

With the rapid advancement of modern technology, especially computer vision and deep learning, tremendous potential and value have been demonstrated across various fields. This opens up new possibilities in the realm of sports. Using these cutting-edge technologies to develop a real-time shot stability system can offer billiard players instant, objective, and accurate feedback. Not only can beginners grasp basic techniques faster, but advanced players can also fine-tune their skills, enhancing their competitive level. Introducing these technologies into billiards training signifies not just a significant technological leap but also a revolutionary innovation in billiards education and training methods, achieving a perfect fusion of technology and sports.

1.2 Research Questions

As discussed earlier, this report focuses on employing deep learning, computer vision, and YOLOv8 techniques to develop a real-time system for detecting batting stability. In order to accomplish this system, the following 6 aspects are considered in this paper:

(1) *Feasibility of YOLOv8: Can the YOLOv8 deep learning architecture be effectively adapted to detect and track the subtle movements of a billiards player's shooting action in real-time, especially focusing on the tip contact point and the player's grip (known as the "bridge")?*

(2) *Precision and Recall: How does the YOLOv8-based model's accuracy, in terms of precision and recall, compare to manual annotations or other leading object detection methods when identifying and tracking key components in a live billiard game?*

(3) *Latency Issues: What is the system's latency? Given the real-time requirements, how quickly can the YOLOv8 model process and return feedback about a player's shot stability after a stroke is executed?*

(4) *Integration with Training Regimes: How can insights derived from the YOLOv8-powered real-time shot stability system be effectively integrated into training regimes to provide actionable recommendations for players to improve their stroke consistency and accuracy?*

(5) *Adaptability: How adaptable is the system to different playing environments, lighting conditions, or variations in player technique? Can it be generalized to work across different levels of players, from novices to professionals?*

(6) *User Feedback: How do players and coaches perceive the feedback provided by the YOLOv8-based shot stability system? Is the system's feedback considered valuable and actionable in a real-world training or competitive context?*

These research questions aim to explore the potential of integrating YOLOv8 into the realm of sports training, specifically focusing on the development of a real-time shot stability system for billiards. Addressing these questions could pave the way for a more technologically advanced and data-driven approach to billiards training.

1.3 Contributions

The primary contribution of this paper is the development of a feasible scheme for custom training deep learning models, which is further validated through the completion of a billiard shot stability assessment system. It includes the following four aspects:

(1) Custom Dataset Creation: This research pioneered the construction of a dedicated dataset specifically tailored for the stability of billiard shots. This dataset encompasses shots taken in various settings, lighting conditions, and players' technical skills, providing a comprehensive sample base for subsequent model training.

(2) Customized YOLOv8 Model: By training on our custom dataset, we successfully developed a tailor-made YOLOv8 model. This model has been specially optimized for billiards scenarios, demonstrating higher precision and speed in real-time detection of shooting actions.

(3) Real-time Shot Stability System: Leveraging the customized YOLOv8 model, we constructed a real-time shot stability system. This system offers instantaneous, objective, and precise feedback on shots to both coaches and players, facilitating more effective guidance during training and matches.

(4) Practical Implementation: We tested the system in actual billiards training and competitive environments, validating its value and potential in real-world applications.

1.4 Objectives of This Report

The primary objective of this report is to elucidate the efforts made in developing a real-time shot stability system for billiards, leveraging a customized YOLOv8 model. In fulfilling this aim, the report sets forth the following specific objectives:

To outline the underlying motivation and significance of creating a dedicated system for assessing the stability of billiard shots, emphasizing the existing gaps and challenges in traditional training methods.

To detail the process of constructing our custom dataset tailored for billiard shots, discussing its diversity in settings, lighting conditions, and player techniques, and how it stands to enhance the model training.

To delve into the intricacies of customizing the YOLOv8 model, illustrating the specific optimizations made to suit billiards scenarios and to improve its real-time detection capabilities.

To provide an in-depth overview of the real-time shot stability system, explaining its functionalities, benefits, and the manner in which it offers immediate, objective feedback to users.

To highlight the system's application in real-world billiards training and competition, sharing insights on its effectiveness, potential benefits.

1.5 Structure of This Report

The main structure of the article is as follows:

- In Chapter 2, we delve into a comprehensive literature review, with a particular emphasis on the YOLO model utilized in this project. We will draw comparisons between the YOLO model and other prevalent MOT (Multiple Object Tracking) algorithms, elucidating our rationale behind choosing YOLO for this project. Additionally, we will introduce and discuss common parameters intrinsic to the YOLO network.
- In Chapter 3, we delve into methods for training a bespoke object detection system and strategies to refine and boost its efficacy.
- In Chapter 4, we will show some of the issues encountered during model training and analyze them. Following this, we will implement the trained model, harnessing a range of techniques from OpenCV to attain the intended features.
- In Chapter 5, through the model's application, our goal is to present a Demo consistent with the project's specifications. After finalizing the Demo, we will review the project's progression, recap the techniques employed, underscore any challenges faced, and sketch out potential avenues for continued growth.

Chapter 2
Literature Review

In Chapter 2, we will discuss YOLO in comparison with other Real-time Multiple Object Tracking algorithms and explain our rationale for choosing YOLO along with its key parameters.

2.1 Why YOLO?

Drawing from information on the "paperswithcode" website pertaining to "Real-Time Object Detection on COCO," the term "on COCO" indicates that these algorithms underwent training and evaluation using the MS COCO dataset. MS COCO is an extensive dataset designed for tasks like object detection, segmentation, key-point detection, and image captioning, comprising 328K images. Within this scope, the YOLO algorithm showcases notable efficiency.

The data in the table below comes from the website "paperswithcode". The algorithms are ranked based on Box AP, revealing the best-performing real-time object detection methods. The details are presented in the following table.

Table 2.1: Real-time target detection ranking based on MSCOCO dataset.

Rank	Model	Box AP	FPS	Paper	Year
1	YOLOv6-L6	57.2	46	*YOLOv6 v3.0: A Full-Scale Reloading*	2023
2	PRB-FPN6-MSP	57.2	27	*Parallel Residual Bi-Fusion Feature Pyramid Network for Accurate Single-Shot Object Detection*	2020
3	YOLOv7-E6E	56.8	36	*YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors*	2022
4	YOLOv7-D6	56.6	44		2022
5	YOLOv7-E6	56	56		2022
6	YOLOv7-W6	54.9	84		2022
7	PP-YOLOE+_X	54.7	45	*PP-YOLOE: An evolved version of YOLO*	2022
8	PP-YOLOE+_L	54.0	78		2022
9	PRB-FPN-MSP	53.3	94	*Parallel Residual Bi-Fusion Feature Pyramid Network for Accurate Single-Shot Object Detection*	2020
10	Gold-YOLO-L	53.28	116	*Gold-YOLO: Efficient Object Detector via Gather-and-Distribute Mechanism*	2023

From Table 2.1, it is evident that among the top 10 real-time object detection rankings, 8 are based on the YOLO algorithm. This clearly demonstrates the significant advantage of YOLO in real-time object detection.

2.2 Introduction to Several Top-Ranked Real-Time Object Detection Models

YOLOv6-L6 introduces enhancements to the original YOLOv6, including network design modifications, anchor-assisted training, and self-distillation. In terms of design, it employs bidirectional connections for the neck of object detector, catering better to varying object scales, ensuring accurate detection across different resolutions. Anchor-assisted training is another pivotal improvement, stabilizing the training process and bolstering performance, as empirical evidence shows superior results using anchors over non-anchor methods on YOLOv6. Lastly, self-distillation leverages knowledge transfer between models of varied sizes, refining the model's generalization and accuracy. Collectively, these advancements significantly boost YOLOv6's accuracy and real-time capabilities.(Li et al., 2023)

PRB-FPN, which stands for Parallel Residual Bi-Fusion Feature Pyramid Network, is a design architecture for object detection. It is tailored to pinpoint both minute and sizable objects with precision and efficiency. This architecture brings forth multiple advancements to tackle the constraints found in current feature pyramid configurations. Its hallmark lies in the concurrent merging of contextual features sourced from neighboring layers.

Contrary to conventional feature pyramids that merge expansive feature maps, leading to a surge in memory use, PRB-FPN employs a bi-fusion technique that adeptly integrates deep and surface-level feature layers. The network is structured around three pivotal components:

1. The Bi-Fusion Module (BFM), which promotes the concurrent amalgamation of features from neighboring layers, bolstering detection precision across a spectrum of object dimensions.

2. The Concatenation and Re-organization (CORE) Module establishes a bottom-up feature fusion route and refines these features to sustain a comprehensive context.

3. The Residual CORE design intertwines ResNet-driven residuals with the CORE, facilitating adaptable training and ensuring harmony with a range of backbone structures, ultimately enhancing detection capabilities.

In essence, the PRB-FPN network design excels in precise and streamlined object detection, adept at identifying a diverse array of object sizes and categories. It stands out with top-tier results on the MS COCO datasets.(P.-Y. Chen et al., 2021)

YOLOv7 has made notable advancements in real-time object detection, outpacing existing detectors in terms of both speed and precision. According to the presented data, three primary models were designed based on various GPU settings: YOLOv7-tiny, YOLOv7, and YOLOv7-W6, which demonstrated remarkable improvements in parameters, computation, and accuracy over baseline models. By incorporating innovative architectural and training optimization methods such as VoVNet, PRN, CSPNet, and the new "bag-of-freebies" strategy, YOLOv7 ensures enhanced information fusion and multi-scale feature extraction. Notably, the efficient ELAN structure serves as the foundational architecture, further amplifying model performance. In essence, by leveraging pioneering techniques and components like VoVNet, PRN, CSPNet, and ELAN, YOLOv7 has set new benchmarks in real-time object detection, exhibiting substantial advancements in both architecture and training optimization.(C.-Y. Wang et al., 2023)

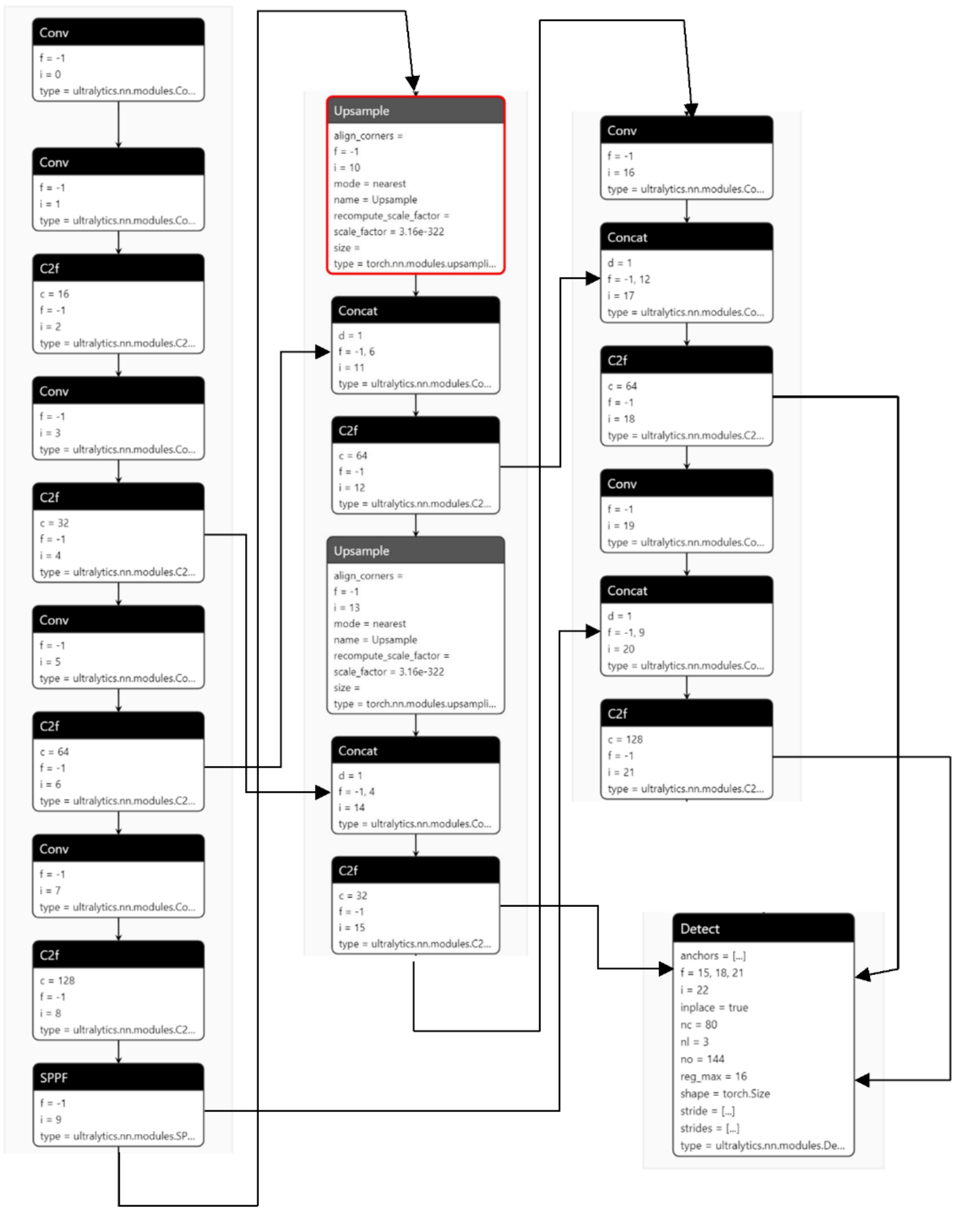
PP-YOLOE, an adaptation of the PP-YOLOv2 model, incorporates multiple refinements to the foundational YOLO architecture. These include the adoption of an anchor-free paradigm, eliminating the reliance on anchor boxes for detection. It integrates a more robust backbone and neck structure, featuring components like CSPRepResStage and ET-head, enhancing feature extraction. Furthermore, it introduces a dynamic label assignment algorithm (TAL) for more precise target label allocation. The enhancements in PP-YOLOE translate to significant performance gains.

On the COCO test-dev dataset, PP-YOLOE-l achieved a mean average precision (mAP) of 51.4%, marking a 1.9% improvement from PP-YOLOv2. In terms of inference speed on Tesla V100, it reached 78.1 FPS, a 13.35% speed boost compared to PP-YOLOv2. When juxtaposed with YOLOX, PP-YOLOE-l saw a 1.3% rise in mAP and a 24.96% acceleration in speed. Hence, PP-YOLOE, through its architectural and algorithmic enhancements, strikes an optimal balance between detection accuracy and speed.(S. Xu et al., n.d.)

Gold-YOLO, an advanced adaptation of the YOLO series models such as YOLOv1-v3, introduces the Gather-and-Distribute (GD) mechanism to address previous information fusion challenges inherent in the series. Through the employment of convolution and self-attention operations, it elevates multi-scale feature fusion. Uniquely, Gold-YOLO incorporates the unsupervised pre-training MAE-style method. As a result, upon evaluation on a Tesla T4 GPU using TensorRT, Gold-YOLO-N outperforms earlier versions such as YOLOv8-N, YOLOv6-3.0-N, and YOLOv7-Tiny, leading by 2.6%, 2.4%, and 6.6% respectively. Additionally, Gold-YOLO-S and Gold-YOLO-M showcase considerable improvements in AP and speed against counterparts, including YOLOX-S, PPYOLOE-S, and YOLOv6-3.0-M.(C. Wang et al., n.d.)

The YOLO series, comprising iterations like YOLOv6-L6, YOLOv7, PP-YOLOE, and Gold-YOLO, underscores the adaptability and superiority of YOLO in the realm of real-time object detection. With each successive version, YOLO has demonstrated remarkable enhancements, be it through the bidirectional connections in YOLOv6-L6, the architectural and training optimizations in YOLOv7, or the innovative Gather-and-Distribute mechanism in Gold-YOLO. The continual refinements across the series, paired with their proven efficacy on benchmarks, highlight YOLO's adaptability. Its flexible design and consistent performance make it an ideal choice for diverse real-time object detection tasks, substantiating its preeminence in the domain.

2.3 Characterization and Parameters of YOLOv8



Conv
f = -1
i = 0
type = ultralytics.nn.modules.Co...
Conv
f = -1
i = 1
type = ultralytics.nn.modules.Co...
C2f
c = 16
f = -1
i = 2
type = ultralytics.nn.modules.C2...
Conv
f = -1
i = 3
type = ultralytics.nn.modules.Co...
C2f
c = 32
f = -1
i = 4
type = ultralytics.nn.modules.C2...
Conv
f = -1
i = 5
type = ultralytics.nn.modules.Co...
C2f
c = 64
f = -1
i = 6
type = ultralytics.nn.modules.C2...
Conv
f = -1
i = 7
type = ultralytics.nn.modules.Co...
C2f
c = 128
f = -1
i = 8
type = ultralytics.nn.modules.C2...
SPPF
f = -1
i = 9
type = ultralytics.nn.modules.SP...
Upsample
align_corners =
f = -1
i = 10
mode = nearest
name = Upsample
recompute_scale_factor =
scale_factor = 3.16e-322
size =
type = torch.nn.modules.upsampli...
Concat
d = 1
f = -1, 6
i = 11
type = ultralytics.nn.modules.Co...
C2f
c = 64
f = -1
i = 12
type = ultralytics.nn.modules.C2...
Upsample
align_corners =
f = -1
i = 13
mode = nearest
name = Upsample
recompute_scale_factor =
scale_factor = 3.16e-322
size =
type = torch.nn.modules.upsampli...
Concat
d = 1
f = -1, 4
i = 14
type = ultralytics.nn.modules.Co...
C2f
c = 32
f = -1
i = 15
type = ultralytics.nn.modules.C2...
Conv
f = -1
i = 16
type = ultralytics.nn.modules.Co...
Concat
d = 1
f = -1, 12
i = 17
type = ultralytics.nn.modules.Co...
C2f
c = 64
f = -1
i = 18
type = ultralytics.nn.modules.C2...
Conv
f = -1
i = 19
type = ultralytics.nn.modules.Co...
Concat
d = 1
f = -1, 9
i = 20
type = ultralytics.nn.modules.Co...
C2f
c = 128
f = -1
i = 21
type = ultralytics.nn.modules.C2...
Detect
anchors = [...]
f = 15, 18, 21
i = 22
inplace = true
nc = 80
nl = 3
no = 144
reg_max = 16
shape = torch.Size
stride = [...]
strides = [...]
type = ultralytics.nn.modules.De...

Figure 2.1: YOLOv8 Model Structure

Figure 2.1 shows the network structure of the YOLOv8 model in .pt format opened using the Netron software. By referring to the official diagram provided by the open-source community Open-mmlab, it can be understood that modules 0-9 are the Backbone part of YOLO, 10-21 are the Neck part, and 22 is the Head part.

YOLO, an acronym for “You Only Look Once”, is a deep learning framework for object detection. It primarily comprises three segments: the Backbone, Neck, and Head. The Backbone, typically a deep convolutional neural network such as VGG, ResNet, or DarkNet, is tasked with deriving basic spatial and contextual features from unprocessed images.(Ayob et al., 2021; Demetriou et al., 2023; Sujatha et al., 2023; Zhou et al., 2019) The Neck component, structures such as FPN or PANet, is added post-Backbone and is aimed at integrating and enhancing features of varying depths and resolutions to capture multi-scale information of objects. The Head, on the other hand, directly predicts the object’s class, location, and size based on the features obtained from the Neck. In essence, the entire YOLO framework first extracts image features through the Backbone, enhances and integrates them via the Neck, and subsequently outputs the final object detection results through the Head.(Huang et al., 2023; Vasanthi & Mohan, 2023; C. Wang et al., n.d.; Z. Zhang et al., 2021)

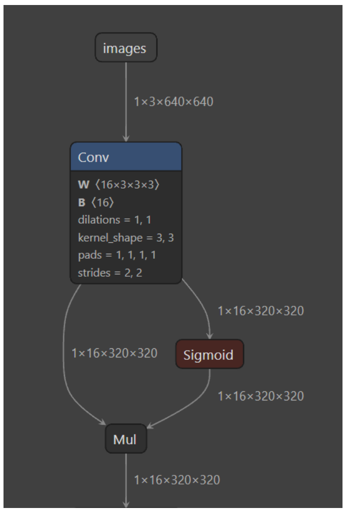


Figure 2.2 : ConvModule of YOLOv8

From Figure 2.2, we can clearly see that 1×3×640×640 describes a four-dimensional tensor, where the four dimensions are as follows:

“1” denotes the batch size, signifying a single image in the batch.

“3” symbolizes the channel count, representing the trio of color channels: red, green, and blue..

The initial 640 signifies the image's height, while the subsequent 640 indicates its width.

In the first line of convolution parameters **W<16×3×3×3>**:

"16" refers to the count of output channels, indicating that this convolutional layer will generate 16 distinct feature maps.

"3" denotes the number of input channels, commonly associated with the three-color channels in an RGB image. "3×3" represents the dimensions of the convolutional kernel, with both its height and width being 3.(Lecun et al., 1998)

Therefore, **W<16×3×3×3>** describes a four-dimensional weight tensor. Each output channel has a 3×3 convolutional kernel spanning 3 channels.

In the second line with **B<16>**:

"16" indicates that this value corresponds to the number of output channels.

The subsequent four lines delineate essential characteristics of the 2D convolution process:

dilations = 1, 1

This means there are no extra gaps between the elements in the convolutional kernel, i.e., a standard convolution operation is used. If the dilations value is greater than 1, there will be extra "gaps" between the kernel's elements, known as "dilated convolution" or "atrous convolution".(Yu & Koltun, 2015)

kernel_shape = 3, 3:

This defines the size of the convolutional kernel. Specifically, both the height and width of the kernel are 3 pixels.

pads = 1, 1, 1, 1:

This is the padding value, used to add extra pixels around the input feature map. Here, the padding size is 1 pixel. Typically, the first two values in the padding list represent the top and bottom padding in the height direction, while the last two values represent the left and right padding in the width direction. In this case, 1 pixel of padding is added to all four sides.(Lecun et al., 1998)

strides = 2, 2:

This specifies the stride of the convolutional kernel as it traverses the input feature map.

With a stride of 2, the kernel moves 2 pixels both horizontally and vertically for each step. Typically, this action will halve the size of the resulting output feature map compared to its input.

Thus, after passing through the convolution, the output tensor is observed to be 1×16×320×320.

Sigmoid (often called the logistic function) and ReLU are indeed two different activation functions used in neural networks.

The sigmoid function squeezes its output values into a range between 0 and 1.(Ramachandran et al., 2017) Its expression is:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.1}$$

The term “SiLU” is not the traditional term used for Sigmoid; instead, SiLU often refers to a newer activation function which is the product of the input and its sigmoid, defined as $x \times \sigma(x)$. It is also sometimes called the Swish activation function.

ReLU (Rectified Linear Unit):

Due to its straightforward nature and effectiveness, ReLU stands out as a widely used activation function in deep learning. It introduces non-linearity into the model without requiring expensive computations. (Krizhevsky et al., 2017)The ReLU function is mathematically represented as:

$$f(x) = max(0, x) \tag{2.2}$$

Exactly. If the input is positive, the function outputs the same positive value. However, for any negative or zero input, the function outputs 0.

The main difference between Sigmoid and ReLU is in their shapes and value ranges. Sigmoid outputs values between 0 and 1 that is S-shaped. In contrast, ReLU outputs values between 0 and infinity and looks like a ramp function. Every activation function offers its unique strengths and limitations. The selection often hinges on the particular challenge at hand and the neural network architecture in place.(Banerjee et al., 2019;

Mastromichalakis, 2021; Sharma, 2022)

After that we see Mul, which in this convolution acts as a BatchNorm2d. In deep learning, BatchNorm2d or Batch Normalization is a commonly used regularization technique designed to accelerate network training and enhance its performance. The fundamental idea behind batch normalization is to normalize activations on each mini batch of data to have a mean of 0 and a variance of 1. Then, two learnable parameters (i.e., a scale and an offset parameter) are applied to scale and shift the normalized values.(Ioffe & Szegedy, 2015)

The computational formula for BatchNorm2d is:

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} \tag{2.3}$$

$$y_i = \gamma \hat{x}_i + \beta \tag{2.4}$$

where:

x_i represents the input value.

μ and σ^2 are the mean and variance of the mini-batch, respectively.

ϵ is a small number to ensure the denominator isn't zero.

γ is the scale parameter.

β is the offset parameter.

$\hat{x}_i$ is the normalized value.

y_i is the output value.

In eq.(2.4), the part related to the Mul operation is the step multiplying by the scale parameter γ. This essentially scales the normalized values, allowing the network to learn the best scale suitable for the task. Introducing this operation enhances the network's adaptability and allows Batch Normalization to both regularize and maintain the network's representational capacity.(Kozlov et al., 2022; X. Xu et al., 2020; Y. Zhang & Freris, 2023)

2.4 Convolutional Neural Networks

Since 1995, CNNs(Convolutional Neural Networks) have been employed in digital image processing. Convolutional kernels are typically come in sizes like 3×3, 5×5, 7×7, and 9×9, among others. The convolution operation produces receptive fields that form the feature maps in a convolutional neural network. Each receptive field maps to a particular section of the source image. (H. Chen & Ji, 2022; Jia et al., 2022; Trung et al., 2022) Convolution is a mathematical process that combines two functions to yield a third, providing insight into how one function alters or influences the other's shape. Convolutional Neural Networks are prevalent in areas like image processing, voice recognition, and sequential data analysis.(W. Q. Yan, 2021)

Three equations are used below to demonstrate the **convolution operation**, **average pooling** and **maximum pooling**.

First, take $\mathbf{F} = \left(f_{x,y}^{(k)}\right)_{m\times m}$ at level k, $p^{(k)}, q^{(k)}, r^{(k)}, s^{(k)} \in \mathcal{R}$, g( ) is a nonlinear function, an operation of convolution is:

$$f_{x,y}^{(k+1)} = g\left(p^{(k)} \cdot f_{x,y}^{(k)} + q^{(k)} \cdot f_{x+1,y}^{(k)} + r^{(k)} \cdot f_{x,y+1}^{(k)} + s^{(k)} \cdot f_{x+1,y+1}^{(k)}\right) \tag{2.5}$$

where $f_{x,y}^{(k)}$ denotes the position in the layer k feature mapping The value of x, y, x, y are two-dimensional coordinates that indicate a specific location or pixel. When convolution is performed, a new feature mapping $f^{(k+1)}$ is computed based on $f^{(k)}$.

$p^{(k)} \cdot f_{x,y}^{(k)}$ represents the multiplication of the pixel with the coefficient $p^{(k)}$.

$q^{(k)} \cdot f_{x+1,y}^{(k)}$ is the multiplication of the pixel below with the coefficient $q^{(k)}$

$r^{(k)} \cdot f_{x,y+1}^{(k)}$ is the multiplication of the pixel to the right with the coefficient $r^{(k)}$

$s^{(k)} \cdot f_{x+1,y+1}^{(k)}$ is the multiplication of the diagonal pixel with the coefficient $s^{(k)}$

All these values are summed up and passed through a nonlinear function g(·) to produce a pixel in the new feature map.

After that, there is the **average pooling** process, expressed in the following equation:

$$\bar{f}_{x,y}^{(k+1)} = \frac{1}{4}\left(p^{(k)} \cdot f_{x,y}^{(k)} + q^{(k)} \cdot f_{x+1,y}^{(k)} + r^{(k)} \cdot f_{x,y+1}^{(k)} + s^{(k)} \cdot f_{x+1,y+1}^{(k)}\right) \qquad (2.6)$$

This describes how to take a 2x2 block from the feature map and calculate the average of these four pixels. The four coordinates (x, y), $(\ x+1, y)$, $(\ x, y+1)$, $(\ x+1, y+1)$ describe a 2x2 area. This is commonly used to halve the size of the feature map.

Max pooling is also a dimensionality reduction method, but it selects the maximum value from a 2×2 pixel block as the pixel value for the new feature map.

$$f_{max}^{(k+1)} = max\left(p^{(k)} \cdot f_{x,y}^{(k)}, q^{(k)} \cdot f_{x+1,y}^{(k)}, r^{(k)} \cdot f_{x,y+1}^{(k)}, s^{(k)} \cdot f_{x+1,y+1}^{(k)}\right) \qquad (2.7)$$

Similar to average pooling, but instead of calculating the average of the pixels, it selects the maximum value among these four pixels. During max pooling, a max filter, denoted as max(·), is utilized on distinct, non-overlapping sections of the initial representation.

2.5 Key Evaluation Metrics and Parameters

YOLO, an acronym for "You Only Look Once", is a distinguished object detection approach that examines an entire image in just one forward operation, enabling instantaneous detection. When evaluating the effectiveness of YOLO or similar object detection methods, key metrics such as Precision, Recall, mAP (mean Average Precision), and IoU (Intersection over Union) are considered. These metrics help in assessing the accuracy and robustness of the model. Here are some common evaluation metrics:

$$\text{Precision } = \frac{\text{True Positives(TP)}}{\text{True Positives(TP) } + \text{ False Positives(FP)}} \qquad (2.8)$$

If the model predicts the presence of 100 objects and 90 of them are correct, the precision is $\frac{90}{100} = 0.9$ or 90%.

From all the actual positive samples, the proportion that was correctly detected.

$$\text{Recall} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Negatives (FN)}} \tag{2.9}$$

If there are 100 actual objects and your model detected only 80 of them, the recall is $\frac{80}{100} = 0.8$ or 80%.

The F1 score is a balance between precision and recall, and is particularly useful when the distribution of categories is uneven.

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.10}$$

That definition corresponds to the IoU (Intersection over Union). It's a measure used in object detection to quantify the overlap between two bounding boxes.(Everingham et al., 2010)

$$IoU = \frac{\text{Area of Overlap (Intersection)}}{\text{Area of Union (Union)}} \tag{2.11}$$

The evaluation metrics mentioned above serve as a means to quantify the performance and accuracy of models in object detection tasks. These metrics provide a way to gauge the model’s performance and accuracy when tweaking the model or comparing different models.

Chapter 3
Methodology

During this chapter, we will delve into the iterative process and optimization techniques employed to realize our final model objectives.

3.1 Creation of the Dataset

Data collection equipment is our personal laptop and Ultra HD Webcam (provided by the school)

The resolution (3840×2160), FPS (29), total video files collected 78.9GB, duration 162 minutes.

Raw Data Information

To achieve a comprehensive analysis, we embarked on a data collection journey focused on billiards. The footage was captured from a total of 12 distinct angles. The cumulative duration of these videos is 162 minutes. With a recording pace of 29 frames every second, this amounts to an impressive 282,480 frames in total.

In the data preprocessing phase, we took a strategic approach. Instead of using every frame, we extracted images from the video at consistent intervals - every 80 frames. This method yielded 3,531 images. By doing so, we maintained the chronological integrity of the events while also reducing redundancy in the data. Each of these image samples boasts a resolution of 3840x2160 pixels.

To efficiently transform the video content into image data, we employed a Python script. This script was tailored to segment the video into images at our predefined intervals.

Sample Dehomogenization

In object detection tasks, diversity in the data is crucial for both the training and generalization of the model. Although we extracted 3,531 images from the raw footage, many of these images displayed evident similarities. When confronted with such a highly homogenized dataset, there’s a notable risk: the model might overfit to these repeated or similar samples, compromising its predictive ability on new data.

To address this and optimize our dataset, we decided to employ the Structural Similarity Index (SSIM) algorithm. SSIM is a classic method used to quantify the visual similarity between two images. We used SSIM to compare each image to its neighboring images and efficiently removed those samples that bore a high resemblance to others based on a predetermined threshold.

This step was undertaken with a dual intention. While shrinking the dataset size was one facet, the more crucial objective was to enrich the model's training experience with a diverse array of samples. Ensuring variety in the data helps in fostering a model that's more robust and adaptable to various scenarios. This can be viewed as an indirect data augmentation technique, with the central aim being to bolster the model's generalization capabilities.

After extracting all the video frames as images, we trimmed the images, deleting those with excessively high similarity. We used SSIM (Structural Similarity Index Measure) for this purpose.(Fuentes-Hurtado et al., 2022)

SSIM mainly focuses on three key features of an image: Luminance, Contrast, and Structure.(Z. Wang et al., 2004)

Luminance is measured by average gray level, obtained by averaging the values of all pixels.

$$\mu_x = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{3.1.1}$$

The luminance contrast function $l(x,y)$ is a function expressed in terms of μ_x and μ_y

$$l(x,y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} \tag{3.1.2}$$

Here, C_1 is a constant, $C_1 = (K_1L)^2$, where L is the number of gray levels in the image. For an 8-bit grayscale image, L= 255, K_1 << 1.

Using the standard deviation of the image as the measure.

$$\sigma_x = \left(\frac{1}{N-1}\sum_{i=1}^{N} (x_i - \mu_x)^2\right)^{\frac{1}{2}} \tag{3.1.3}$$

The contrast comparison function A is a function of 1 and 2, expressed as:

$$C(x,y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2} \tag{3.1.4}$$

Constant $C_2 = (K_2L)^2$, and $K_2 \ll 1$.

The structural comparison compares $\frac{(x-\mu_x)}{\sigma_x}$ and $\frac{(y-\mu_y)}{\sigma_y}$ after normalization, which can be measured by the correlation coefficient.

$$S(x,y) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3} \tag{3.1.5}$$

$$\sigma_{xy} = \frac{1}{N-1}\sum_{i=1}^{N} (x_i - \mu_x)(y_i - \mu_y) \tag{3.1.6}$$

where, $C_3 = \frac{C_2}{2}$

The constants C_1, C_2, and C_3 are set to prevent instability in the function's value when it approaches 0.

In this way, the general equation for SSIM can be written as:

$$\text{SSIM}\,(x,y) = [I(x,y)]^{\alpha}[c(x,y)]^{\beta}[s(x,y)]^{\gamma} \tag{3.1.7}$$

Where α, β, and γ respectively represent the proportions of different features in the SSIM measurement. When all are 1 and $C_3 = \frac{C_2}{2}$, we can expand the general formula for SSIM as:

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{3.1.8}$$

After using the above easy methods, we ultimately extracted 472 images from the 78.9G of video data. Afterwards, we used the image annotation tool LabelMe to annotate the extracted images, resulting in annotation files in JSON format. Then, we need to convert the JSON annotation information into the YOLO format.

Once we have secured the image data alongside its corresponding annotations in the YOLO format, it's imperative to partition this dataset into two distinct sets: one for training and another for testing. Commonly, a split ratio of 80:20 is adhered to, allocating 80% of the data to training and the remaining 20% to testing. This ensures that the model is exposed to a vast majority of the data during training, while still reserving a significant portion for validation, helping in gauging its real-world performance and robustness.

Annotation & Target Detection Display

As shown in Figure 3.1, the model has four detection targets, namely, hand, cue, sp, and hp. "sp" stands for StandPoint, which is a support point where the player sets the cue stick when taking a shot. "hp" stands for HitPoint, also referred to as the Tip in billiards terms, which is the point where the cue stick makes contacts with the cue.

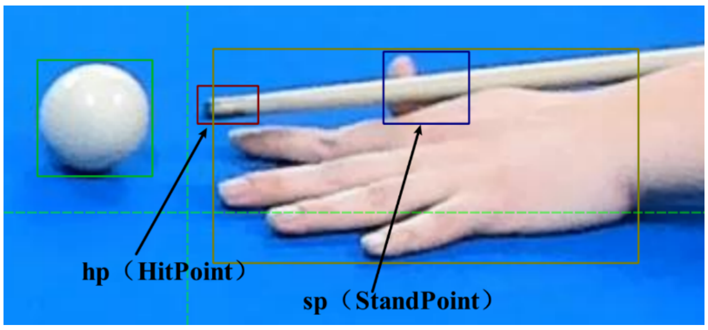


Figure3.1 Sample Annotation Explanation

3.2 Dataset Visualization

Figure 3.2: Sample of Dataset

In Figure 3.2, we see the data samples from the dataset. We can observe that the samples used for training are fed in full size (samples include the pool table, background, and characters), which implies that there are a lot of impurities or irrelevant information present in the samples.

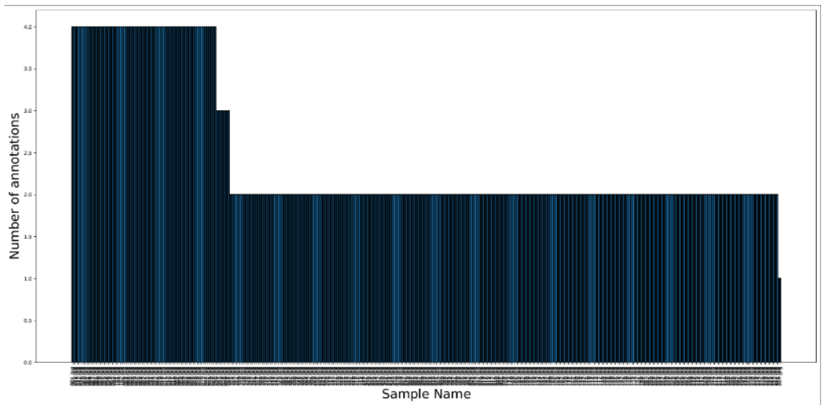
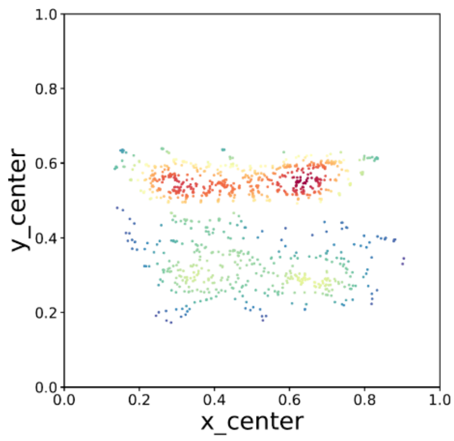
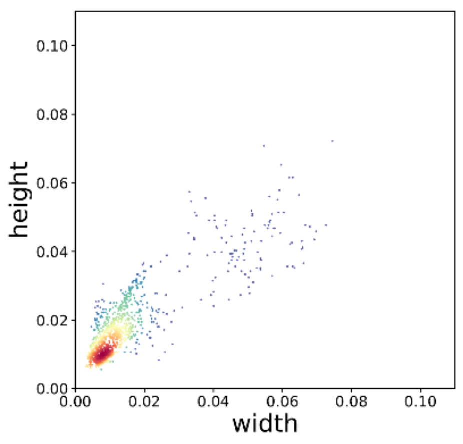


Figure 3.3: Number of annotations per sample

In the dataset, only rectangles were used to annotate the objects for detection. In all the samples, the number of annotated objects in each sample is 4, 3, and 2, respectively, as shown in Figure 3.3. This implies that the distribution of detection targets within the samples is uneven.



(a)



(b)

Figure3.4: Bounding Box Distribution and Size

In Figure 3.4(a), the distribution of the bounding boxes throughout the image is shown. It can be observed that most annotations are concentrated in the center of the sample. Meanwhile, in (b), it is evident that the area of the annotation boxes is minimal, with most of them occupying only about 1% of the sample.

3.3 Sample Augmentation

In the initial training without sample augmentation, the first version of the model achieved a mAP of only 0.7, which did not meet our expectations. Therefore, we proceeded with sample augmentation, increasing the dataset from 472 samples to 944. The original samples were rotated 15 degrees counterclockwise, and corresponding JSON files were also generated.

This operation can be described as:

Original sample set

$$S = \{s_1, s_2, \dots, s_{472}\} \tag{3.3.1}$$

rotation operation

$$s_i' = \mathrm{Rotate}(s_i, -15^\circ) \tag{3.3.2}$$

Sample after rotation

$$S' = \{s'_1, s'_2, \ldots, s'_{472}\} \tag{3.3.3}$$

Combined sample

$$S_{\text{total}} = S \cup S' \tag{3.3.4}$$

$$|S_{\text{total}}| = 944 \tag{3.3.5}$$

After expanding the dataset, the mAP of the retrained model increased to 0.89.

3.4 Model Training Parameter Optimization

Despite increasing the sample quantities, we still didn't achieve the desired accuracy. Consequently, during the model training process, we experimentally adjusted the image size parameter 'imgsz' to investigate its impact on model accuracy and training efficiency.(Hao et al., 2022; Liu et al., 2020; S. Zhang, 2022)

We gradually increased the value of imgsz, starting from 640, passing through 1280, and ultimately reaching 2560. With the increase in image size, the model's accuracy also saw significant improvements. Specifically, with imgsz at 640, the model's accuracy stood at 0.747. When the value was raised to 1280, the accuracy reached 0.91. And at imgsz of 2560, the model's accuracy further increased to 0.954.

While a larger imgsz value can enhance model accuracy, it also introduces a substantial computational cost. In our experimental setup, using the same hardware and dataset, when imgsz was increased from 640 to 1280, the model's training time surged from 0.546 hours to a staggering 17.406 hours.

Therefore, although adjusting the imgsz parameter can significantly boost the model's performance, it also implies a higher consumption of computational resources. In practical applications, there's a need to strike a balance between accuracy and computational efficiency, choosing an appropriate imgsz value.

3.5 Introduction to Transfer Learning

Transfer learning capitalizes on the concept of utilizing knowledge gained while solving one problem to assist in addressing a different but related problem. Instead of starting from scratch, a model that's already been trained on a particular task is fine-tuned to adapt to a new task. By reusing parameters, features, or samples from a previously trained model, the time and resources needed for training on the new task can be considerably reduced. The applicability of transfer learning largely depends on the similarity between the original and target tasks, as well as the characteristics of the domains in question. This approach is especially beneficial in scenarios where data is scarce or when training a comprehensive model from the ground up is computationally intensive. Some common methods include fine-tuning and feature extraction. In the process, various aspects such as samples, instances, parameters, and features can be transferred to enhance the performance on the new task.(Sarker, 2021; Tsinghua University et al., 2018; J. Yan & Wang, 2022) Transfer learning thrives on the premise of reusing knowledge from a source task to benefit a target task. However, its efficacy is contingent on the compatibility of labels between these tasks. If there's a stark divergence in labels, a straightforward transfer might falter. It's essential to ensure that the underlying patterns and features learned in the source task are relevant and adaptable to the target task. In cases where labels diverge significantly, techniques like domain adaptation or feature-level transfer might be more apt than directly leveraging pre-trained models. Despite these constraints, one of the strengths of transfer learning is its flexibility in using models across different domains and tasks. However, it's essential to note that distinct domains often lead to separate tasks, each with its own distribution. This can affect the performance when applying knowledge from one domain to another.(Brownlow et al., 2018; Sukhija et al., 2018) Different machine learning paradigms, including inductive learning, transductive learning, and unsupervised learning methods like clustering, can be applied within the framework of transfer learning. Each of these paradigms may have distinct domains, tasks, and algorithms when applied to transfer learning.(Hu et al., 2022; Kobylarz et al., 2020; Xiao et al., 2021; Yang et al., 2022)

In this report, we initialized the training of the YOLO object model using a pre-trained model from the large-scale MsCOCO dataset. Subsequently, we fine-tuned this model on our specific billiards dataset. The advantage of this approach is that it quickly adapts the model to the new dataset, saving training time and computational resources. Transfer learning is suitable when the target dataset and the pre-trained dataset have some degree of relevance.

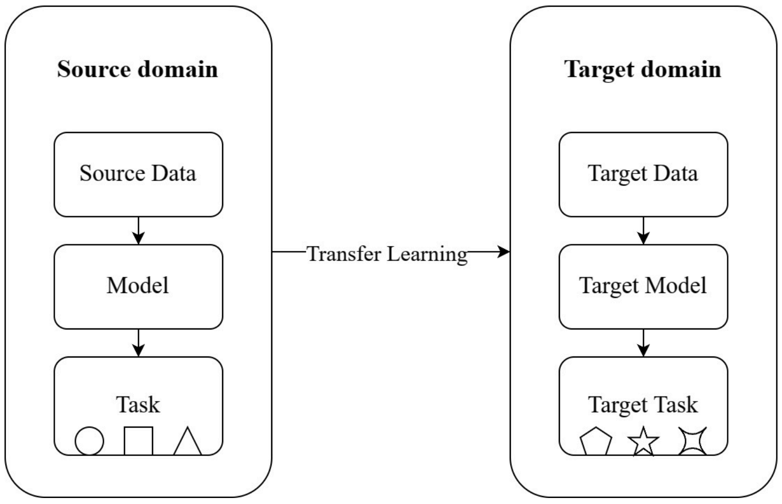


Figure 3.5: Transfer learning schematic diagram

Transfer learning typically focuses on a scenario with a source domain D_s and a target domain D_T. D_s and D_T can be expressed as follows:

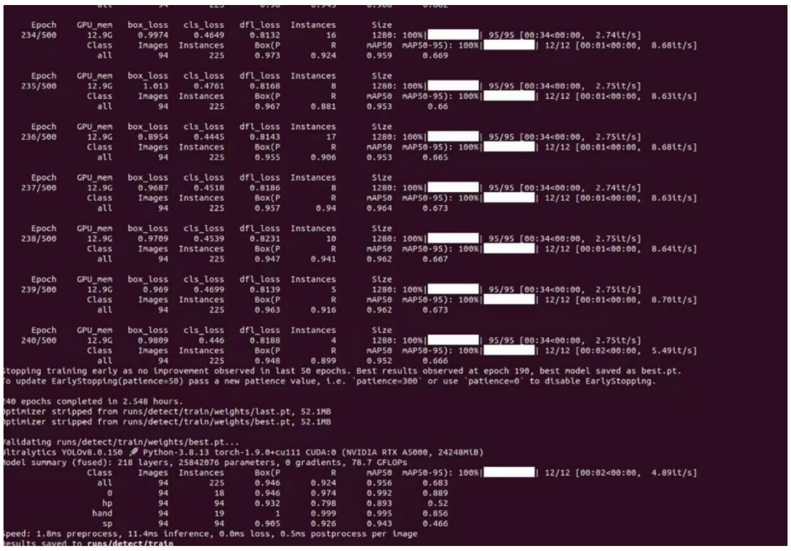
$$D_s = \{x_i, y_i\}_i^{N_s} \tag{3.1.9}$$

$$D_T = \{x_i, y_i\}_i^{N_T} \tag{3.1.10}$$

i: It is an index used to refer to a specific data sample and its corresponding label. In the source domain D_s , x_i , y_i respectively represent the ith data sample and its label.
N_s and N_T: These two symbols represent the total number of data samples in the source and target domains, respectively.

Given the source domain D_s and source task T_s , target domain D_T and target task T_T, transfer learning aims to harness knowledge from a source domain and task to enhance the learning of the prediction function in the desired target domain.

3.6 Model Evaluation



```
      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    234/500      12.9G     0.9974     0.4649     0.8132         16       1280: 100%|██████████| 95/95 [00:34<00:00,  2.74it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 12/12 [00:01<00:00,  8.68it/s]
                   all         94        225      0.973      0.924      0.959      0.669

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    235/500      12.9G      1.013     0.4761     0.8168          8       1280: 100%|██████████| 95/95 [00:34<00:00,  2.75it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 12/12 [00:01<00:00,  8.63it/s]
                   all         94        225      0.967      0.881      0.953       0.66

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    236/500      12.9G     0.8954     0.4445     0.8143         17       1280: 100%|██████████| 95/95 [00:34<00:00,  2.75it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 12/12 [00:01<00:00,  8.68it/s]
                   all         94        225      0.955      0.906      0.953      0.665

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    237/500      12.9G     0.9687     0.4518     0.8186          8       1280: 100%|██████████| 95/95 [00:34<00:00,  2.74it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 12/12 [00:01<00:00,  8.63it/s]
                   all         94        225      0.957       0.94      0.964      0.673

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    238/500      12.9G     0.9709     0.4539     0.8231         10       1280: 100%|██████████| 95/95 [00:34<00:00,  2.75it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 12/12 [00:01<00:00,  8.64it/s]
                   all         94        225      0.947      0.941      0.962      0.667

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    239/500      12.9G      0.969     0.4699     0.8139          5       1280: 100%|██████████| 95/95 [00:34<00:00,  2.75it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 12/12 [00:01<00:00,  8.70it/s]
                   all         94        225      0.963      0.916      0.962      0.673

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    240/500      12.9G     0.9809      0.446     0.8188          4       1280: 100%|██████████| 95/95 [00:34<00:00,  2.75it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 12/12 [00:02<00:00,  5.49it/s]
                   all         94        225      0.948      0.899      0.952      0.666
topping training early as no improvement observed in last 50 epochs. Best results observed at epoch 190, best model saved as best.pt.
o update EarlyStopping(patience=50) pass a new patience value, i.e. `patience=300` or use `patience=0` to disable EarlyStopping.

40 epochs completed in 2.548 hours.
ptimizer stripped from runs/detect/train/weights/last.pt, 52.1MB
ptimizer stripped from runs/detect/train/weights/best.pt, 52.1MB

alidating runs/detect/train/weights/best.pt...
ltralytics YOLOv8.0.150 🚀 Python-3.8.13 torch-1.9.0+cu111 CUDA:0 (NVIDIA RTX A5000, 24248MiB)
odel summary (fused): 218 layers, 25842076 parameters, 0 gradients, 78.7 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 12/12 [00:02<00:00,  4.89it/s]
                   all         94        225      0.946      0.924      0.956      0.683
                     0         94         18      0.946      0.974      0.992      0.889
                    hp         94         94      0.932      0.798      0.893       0.52
                  hand         94         19          1      0.999      0.995      0.856
                    sp         94         94      0.905      0.926      0.943      0.466
peed: 1.8ms preprocess, 11.4ms inference, 0.0ms loss, 0.5ms postprocess per image
esults saved to runs/detect/train
```

Figure 3.6: Model training results

As shown in Figure 3.6, the best prediction model was obtained after 240 training epochs on the NVIDIA RTX A5000. The training process lasted for 2.548 hours.

Table 3.1: Model training results

Class	Box(Precision)	Recall	mAP50
All	0.946	0.924	0.956
0	0.946	0.974	0.992
hp	0.932	0.798	0.893
hand	1	0.999	0.995
sp	0.905	0.926	0.943

As shown in Table 3.1, the overall accuracy of the model can reach 94.6%, which meets expectations.

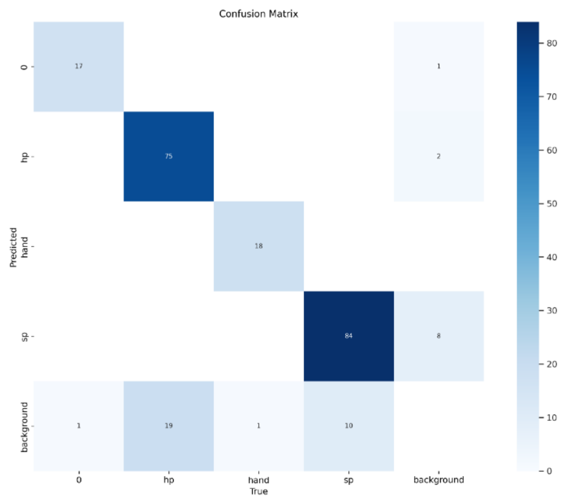
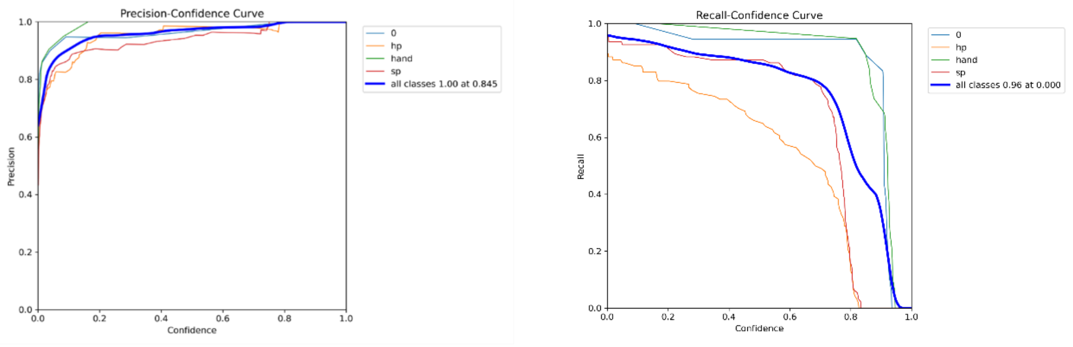


Figure 3.7: Confusion matrix after completion of training

Based on the data from Figure 3.7, the samples hp and samples sp are matched to each other and their true number is 94 samples for each class, in the figure we can learn that the accuracy of the hp class is not as expected and is able to recognize 75 samples out of 94 samples with a recognition rate of 79%. Each column indicates the count of instances for a given true class, whereas each row shows the count of instances predicted for that specific class. From this chart, we can determine which classes the model identifies well and which it struggles with, allowing for subsequent optimization of the dataset.(Dixit & Nain Chi, 2021; Jayaram et al., 2023; Umadevi T P & Murugan A, 2021)



(a) (b)

Figure 3.8: Precision-confidence Curve & Recall-Confidence Curve

The figure 3.8 (a) depicts the Precision-Confidence curve, indicating that all classes can be recognized when the confidence reaches 0.845, but there's also a possibility of missing some classes with low confidence. Meanwhile,(b) is the Recall-Confidence curve, showing that the detection of classes becomes more comprehensive as the confidence decreases.(Boyd et al., 2013; Oksuz et al., 2018)

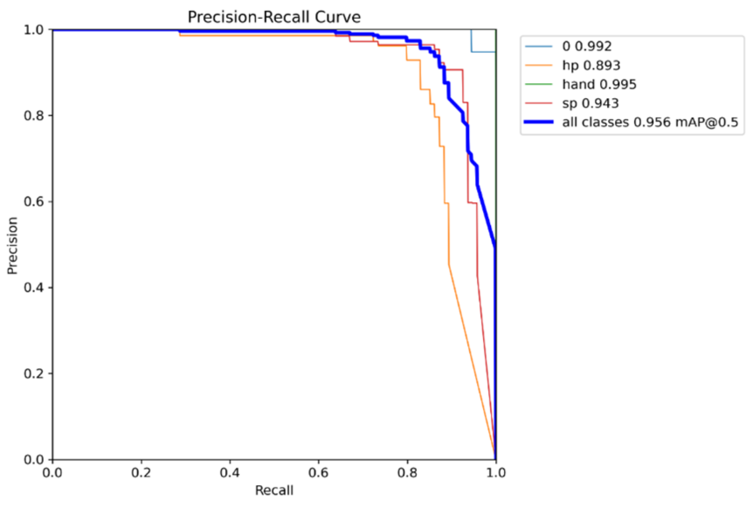


Figure 3.9: Precision-Recall Curve

In Figure 3.9, mAP refers to Mean Average Precision. It's evident that as precision increases, recall tends to decrease. However, we hope that our model can detect all classes as much as possible while maintaining high accuracy. Hence, our objective is to have our curve gravitate towards the (1,1) point, implying an aspiration for the area under the mAP curve to be near 1.(Altukroni et al., 2023; Çelik et al., 2023; Çelik & Çelik, 2023; Yen et al., 2021)

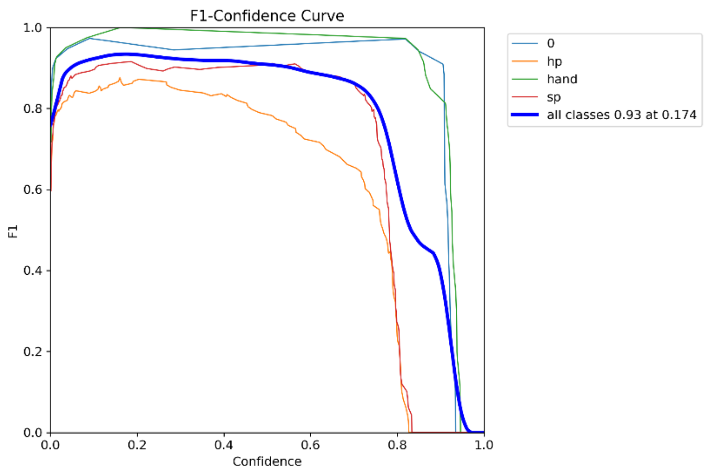


Figure 3.10: F1-Confidence Curve

The F1-score serves as a metric for classification tasks. Many machine learning competitions, especially those involving multi-class challenges, employ the F1-score for final evaluation. Representing the harmonic mean of precision and recall, its value ranges between 0 and 1, with 1 being the maximum. From the figure 3.10, we can observe that for all classes, the F1 score reaches 0.93 when the confidence level is at 0.174.(Jayapermana et al., 2022; Jayasundara et al., 2022; Thakur & M, 2022)

3.7 Experimental Flow Chart

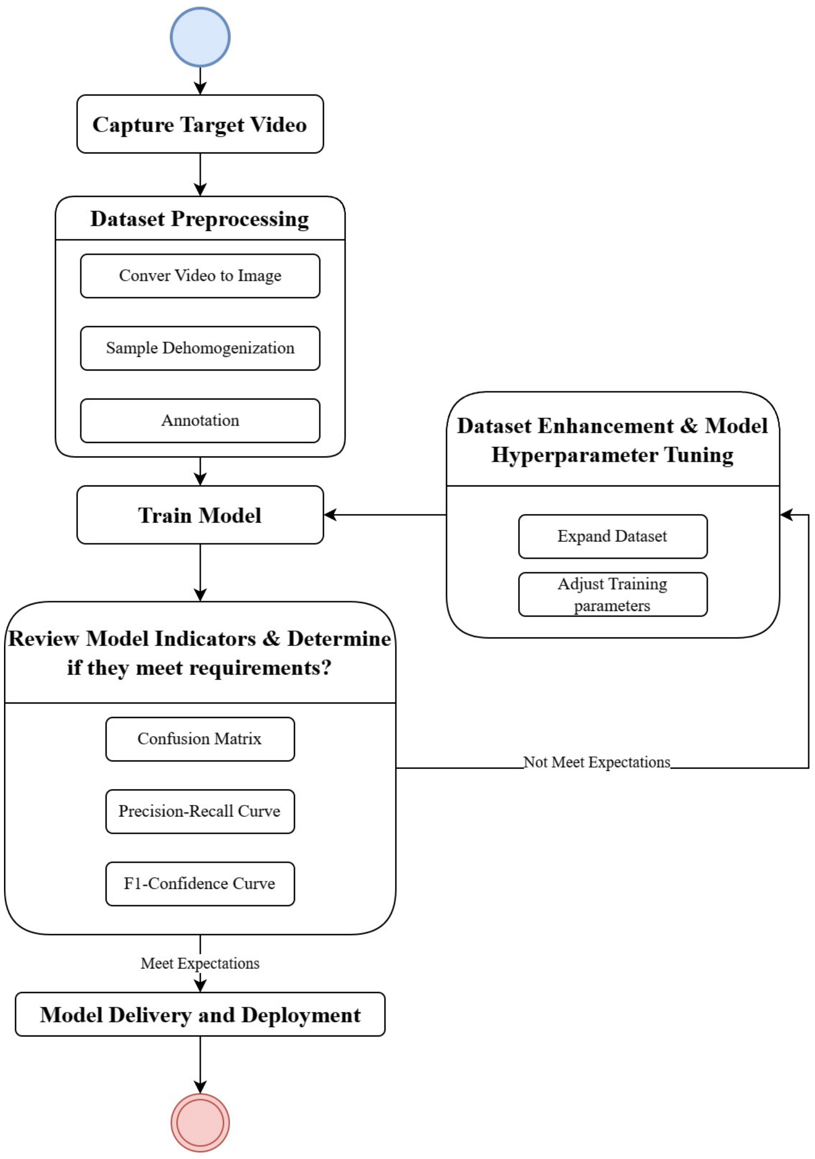


Figure 3.11: Model Iteration and Optimization Flowchart.

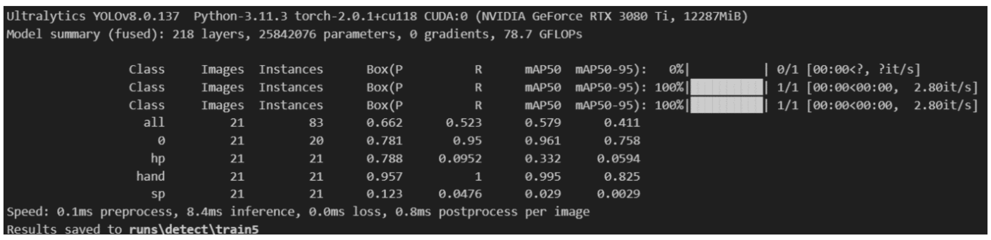
As shown in Figure 3.11, the entire experimental process is based on continuous training and evaluation. We expect the model's recognition accuracy for all classes to achieve over 0.95 mAP before proceeding with model deployment and real-time testing.

Chapter 4
Model Results Analysis and Deployment

The main content of this chapter is summarized the issues encountered during model training and analyze them. Subsequently, we will deploy the trained model and achieve our desired functionalities using various methods from OpenCV.

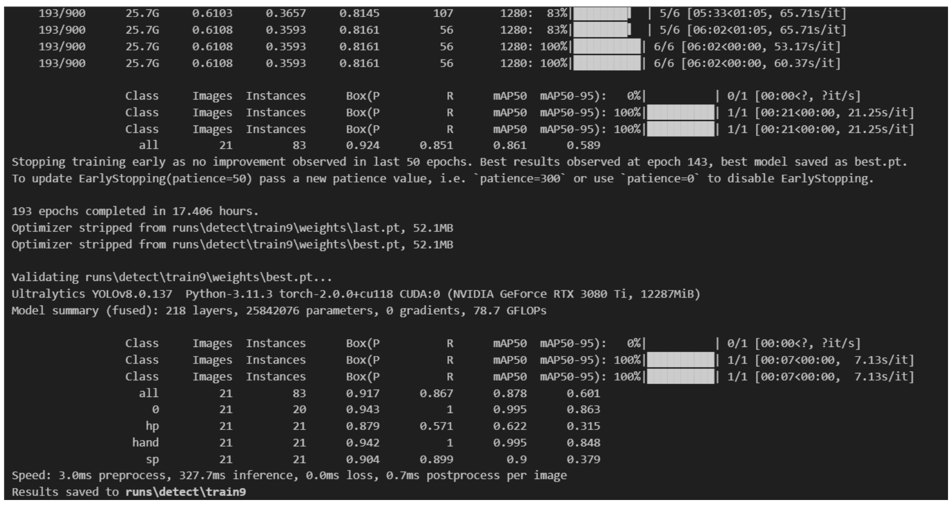
4.1 Issues During Model Training

Apart from the initial issues with setting up the training environment, our primary concern was the progress of the pre-trained model after undergoing transfer learning with a custom dataset. During the initial training, due to insufficient samples in the dataset and incorrect "imagesize" parameters, there were two unsatisfactory training outcomes. The detailed results are shown in the following figure:



```
Ultralytics YOLOv8.0.137  Python-3.11.3 torch-2.0.1+cu118 CUDA:0 (NVIDIA GeForce RTX 3080 Ti, 12287MiB)
Model summary (fused): 218 layers, 25842076 parameters, 0 gradients, 78.7 GFLOPs

                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95):   0%|          | 0/1 [00:00<?, ?it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 1/1 [00:00<00:00,  2.80it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 1/1 [00:00<00:00,  2.80it/s]
                   all         21         83      0.662      0.523      0.579      0.411
                     0         21         20      0.781       0.95      0.961      0.758
                    hp         21         21      0.788     0.0952      0.332     0.0594
                  hand         21         21      0.957          1      0.995      0.825
                    sp         21         21      0.123     0.0476      0.029     0.0029
Speed: 0.1ms preprocess, 8.4ms inference, 0.0ms loss, 0.8ms postprocess per image
Results saved to runs\detect\train5
```

Figure 4.1: Unsatisfactory model training result A (Model A)

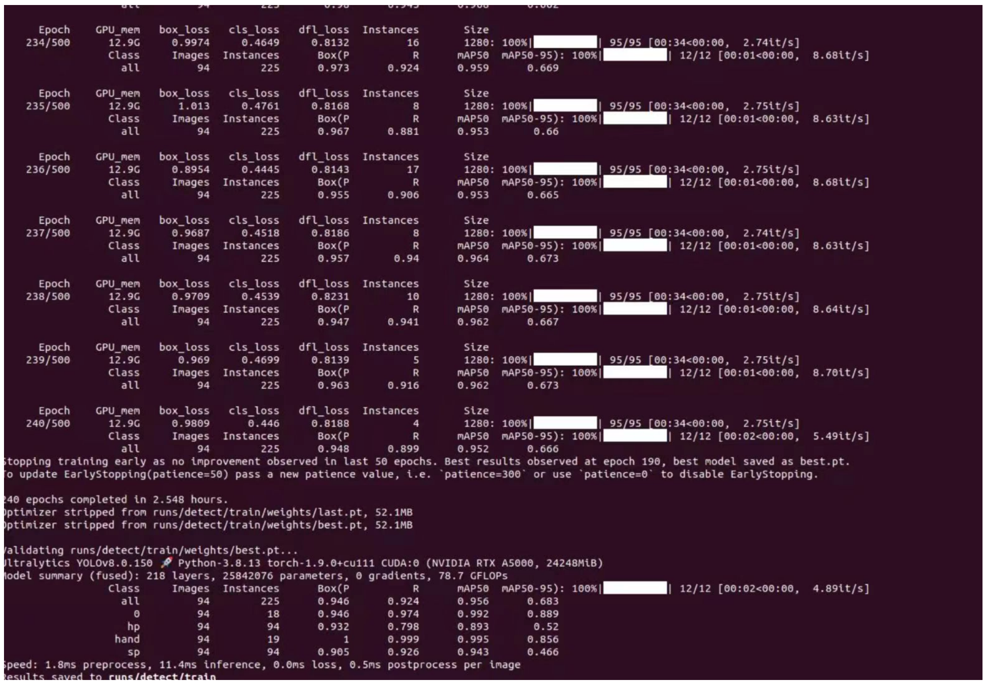


```
    193/900      25.7G     0.6103     0.3657     0.8145        107       1280:  83%|████████  | 5/6 [05:33<01:05, 65.71s/it]
    193/900      25.7G     0.6108     0.3593     0.8161         56       1280:  83%|████████  | 5/6 [06:02<01:05, 65.71s/it]
    193/900      25.7G     0.6108     0.3593     0.8161         56       1280: 100%|██████████| 6/6 [06:02<00:00, 53.17s/it]
    193/900      25.7G     0.6108     0.3593     0.8161         56       1280: 100%|██████████| 6/6 [06:02<00:00, 60.37s/it]

                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95):   0%|          | 0/1 [00:00<?, ?it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 1/1 [00:21<00:00, 21.25s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 1/1 [00:21<00:00, 21.25s/it]
                   all         21         83      0.924      0.851      0.861      0.589
Stopping training early as no improvement observed in last 50 epochs. Best results observed at epoch 143, best model saved as best.pt.
To update EarlyStopping(patience=50) pass a new patience value, i.e. `patience=300` or use `patience=0` to disable EarlyStopping.

193 epochs completed in 17.406 hours.
Optimizer stripped from runs\detect\train9\weights\last.pt, 52.1MB
Optimizer stripped from runs\detect\train9\weights\best.pt, 52.1MB

Validating runs\detect\train9\weights\best.pt...
Ultralytics YOLOv8.0.137  Python-3.11.3 torch-2.0.0+cu118 CUDA:0 (NVIDIA GeForce RTX 3080 Ti, 12287MiB)
Model summary (fused): 218 layers, 25842076 parameters, 0 gradients, 78.7 GFLOPs

                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95):   0%|          | 0/1 [00:00<?, ?it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 1/1 [00:07<00:00,  7.13s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 1/1 [00:07<00:00,  7.13s/it]
                   all         21         83      0.917      0.867      0.878      0.601
                     0         21         20      0.943          1      0.995      0.863
                    hp         21         21      0.879      0.571      0.622      0.315
                  hand         21         21      0.942          1      0.995      0.848
                    sp         21         21      0.904      0.899        0.9      0.379
Speed: 3.0ms preprocess, 327.7ms inference, 0.0ms loss, 0.7ms postprocess per image
Results saved to runs\detect\train9
```

Figure 4.2: Unsatisfactory model training result B (Model B)

Figure 4.1 and Figure 4.2 were trained using the same dataset. The difference during their training was that the 'imageSize' increased from 640 to 1280. This change improved the accuracy of Model B in Figure 22 from 0.662 to 0.917. Therefore, when using the YOLO pre-trained model for transfer learning, increasing the 'imageSize' parameter can enhance the accuracy.

However, in Model B, the accuracy for the "hp" class is still below 0.9, which is not up to expectations. Therefore, we adjusted the dataset in the following steps.



```
     Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
   234/500      12.9G     0.9974     0.4649     0.8132         16       1280: 100%|██████████| 95/95 [00:34<00:00,  2.74it/s]
                Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 12/12 [00:01<00:00,  8.68it/s]
                  all         94        225      0.973      0.924      0.959      0.669

     Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
   235/500      12.9G      1.013     0.4761     0.8168          8       1280: 100%|██████████| 95/95 [00:34<00:00,  2.75it/s]
                Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 12/12 [00:01<00:00,  8.63it/s]
                  all         94        225      0.967      0.881      0.953       0.66

     Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
   236/500      12.9G     0.8954     0.4445     0.8143         17       1280: 100%|██████████| 95/95 [00:34<00:00,  2.75it/s]
                Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 12/12 [00:01<00:00,  8.68it/s]
                  all         94        225      0.955      0.906      0.953      0.665

     Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
   237/500      12.9G     0.9687     0.4518     0.8186          8       1280: 100%|██████████| 95/95 [00:34<00:00,  2.74it/s]
                Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 12/12 [00:01<00:00,  8.63it/s]
                  all         94        225      0.957       0.94      0.964      0.673

     Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
   238/500      12.9G     0.9709     0.4539     0.8231         10       1280: 100%|██████████| 95/95 [00:34<00:00,  2.75it/s]
                Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 12/12 [00:01<00:00,  8.64it/s]
                  all         94        225      0.947      0.941      0.962      0.667

     Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
   239/500      12.9G      0.969     0.4699     0.8139          5       1280: 100%|██████████| 95/95 [00:34<00:00,  2.75it/s]
                Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 12/12 [00:01<00:00,  8.70it/s]
                  all         94        225      0.963      0.916      0.962      0.673

     Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
   240/500      12.9G     0.9809      0.446     0.8188          4       1280: 100%|██████████| 95/95 [00:34<00:00,  2.75it/s]
                Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 12/12 [00:02<00:00,  5.49it/s]
                  all         94        225      0.948      0.899      0.952      0.666
topping training early as no improvement observed in last 50 epochs. Best results observed at epoch 190, best model saved as best.pt.
o update EarlyStopping(patience=50) pass a new patience value, i.e. `patience=300` or use `patience=0` to disable EarlyStopping.

40 epochs completed in 2.548 hours.
ptimizer stripped from runs/detect/train/weights/last.pt, 52.1MB
ptimizer stripped from runs/detect/train/weights/best.pt, 52.1MB

alidating runs/detect/train/weights/best.pt...
ltralytics YOLOv8.0.150 🚀 Python-3.8.13 torch-1.9.0+cu111 CUDA:0 (NVIDIA RTX A5000, 24248MiB)
odel summary (fused): 218 layers, 25842076 parameters, 0 gradients, 78.7 GFLOPs
                Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 12/12 [00:02<00:00,  4.89it/s]
                  all         94        225      0.946      0.924      0.956      0.683
                    0         94         18      0.946      0.974      0.992      0.889
                   hp         94         94      0.932      0.798      0.893       0.52
                 hand         94         19          1      0.999      0.995      0.856
                   sp         94         94      0.905      0.926      0.943      0.466
peed: 1.8ms preprocess, 11.4ms inference, 0.0ms loss, 0.5ms postprocess per image
esults saved to runs/detect/train
```

Figure 4.3: Satisfactory results for Model C

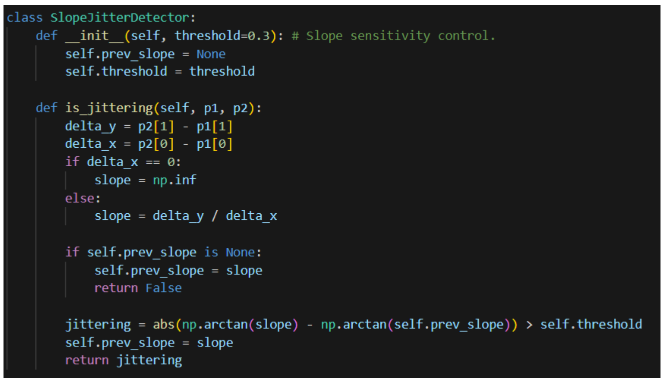
As shown in Figure 4.3, after expanding the sample size in the dataset to 472 images, we obtained a model with an accuracy 0.946. This model meets our expectations.

Figures 4.1, 4.2, and 4.3 represent the three iterations of the model, with each resulting model being labeled as Model A, Model B, and Model C, respectively. The result, as shown in Figure 4.3, met our expectations with an accuracy of 0.946 mAP; this is Model C. Both Model A and Model B utilized the same dataset. During the iterative training, the parameter 'imgsz' was adjusted, expanded from the original 640 to 1280. This change propelled Model A's mAP value from 0.662 to a significant 0.917. However, this was the mAP for all classes; the 'hp' class's detection accuracy was still below 0.9. Transitioning from Model B to Model C, we augmented the dataset, increasing from 172 images to 472 images. We finally elevated the mAP value for 'hp' to 0.893, which boosted the overall mAP value for all classes to 0.946.

4.2 Model Deployment

After obtaining a high-accuracy model, we normalized the prediction boxes for "hp" and "sp" to obtain points. These two points correspond to the contact point of the cue stick in a billiards game, also known as the ‘tip’, and the ‘bridge’, where the player supports the cue stick. By connecting these two points, we form a line. When the player strokes the ball, the stability of the slope of this line is monitored to determine whether the player’s shot is stable. At the same time, text will be displayed on the prediction screen and sound will be emitted to prompt the player whether their aiming and shooting are stable.

Figures 4.4 and 4.5 showcase the code for detecting slope jitter and the visualization of prediction outcomes, respectively.



```python
class SlopeJitterDetector:
    def __init__(self, threshold=0.3): # Slope sensitivity control.
        self.prev_slope = None
        self.threshold = threshold

    def is_jittering(self, p1, p2):
        delta_y = p2[1] - p1[1]
        delta_x = p2[0] - p1[0]
        if delta_x == 0:
            slope = np.inf
        else:
            slope = delta_y / delta_x

        if self.prev_slope is None:
            self.prev_slope = slope
            return False

        jittering = abs(np.arctan(slope) - np.arctan(self.prev_slope)) > self.threshold
        self.prev_slope = slope
        return jittering
```

Figure 4.4: Slope jitter detection

$$\text{Slope}(m) = \frac{\Delta y}{\Delta x} \tag{4.2.1}$$

Where,

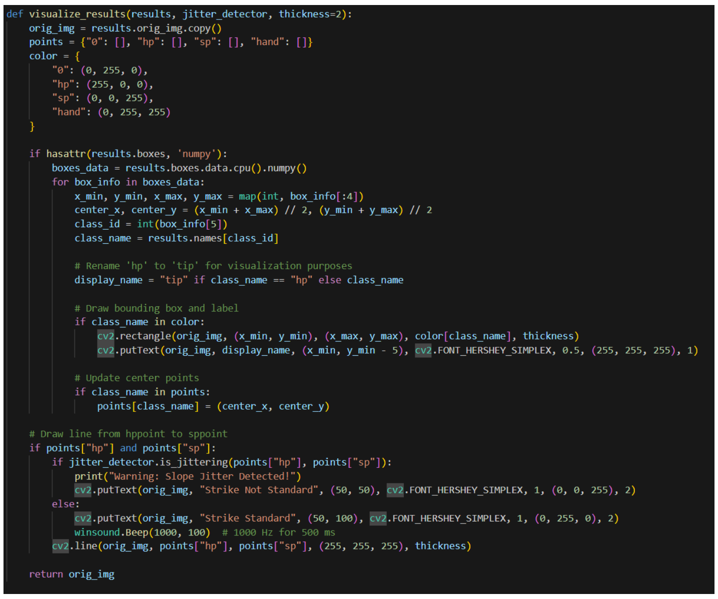
$$\Delta y = y_2 - y_1 \tag{4.2.2}$$

$$\Delta x = x_2 - x_1 \tag{4.2.3}$$

Here, $(x_1,\ y_1)$ and $(x_2,\ y_2)$ are the coordinates of two consecutive points.

$$\text{Jittering} \ = \left|\arctan(m) - \arctan\left(m_{\text{prev}}\right)\right| > \text{threshold} \tag{4.2.4}$$

Where m is the current calculated slope, and m_{prev} is the previously calculated slope. To detect significant variation between two consecutive slopes, we compute their difference (transforming the slope into angles using the arctangent function) and see if this difference surpasses a set threshold. If it exceeds the threshold, we consider jitter to be detected.



```python
def visualize_results(results, jitter_detector, thickness=2):
    orig_img = results.orig_img.copy()
    points = {"0": [], "hp": [], "sp": [], "hand": []}
    color = {
        "0": (0, 255, 0),
        "hp": (255, 0, 0),
        "sp": (0, 0, 255),
        "hand": (0, 255, 255)
    }

    if hasattr(results.boxes, 'numpy'):
        boxes_data = results.boxes.data.cpu().numpy()
        for box_info in boxes_data:
            x_min, y_min, x_max, y_max = map(int, box_info[:4])
            center_x, center_y = (x_min + x_max) // 2, (y_min + y_max) // 2
            class_id = int(box_info[5])
            class_name = results.names[class_id]

            # Rename 'hp' to 'tip' for visualization purposes
            display_name = "tip" if class_name == "hp" else class_name

            # Draw bounding box and label
            if class_name in color:
                cv2.rectangle(orig_img, (x_min, y_min), (x_max, y_max), color[class_name], thickness)
                cv2.putText(orig_img, display_name, (x_min, y_min - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 1)

            # Update center points
            if class_name in points:
                points[class_name] = (center_x, center_y)

    # Draw line from hppoint to sppoint
    if points["hp"] and points["sp"]:
        if jitter_detector.is_jittering(points["hp"], points["sp"]):
            print("Warning: Slope Jitter Detected!")
            cv2.putText(orig_img, "Strike Not Standard", (50, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
        else:
            cv2.putText(orig_img, "Strike Standard", (50, 100), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
            winsound.Beep(1000, 100)  # 1000 Hz for 500 ms
        cv2.line(orig_img, points["hp"], points["sp"], (255, 255, 255), thickness)

    return orig_img
```

Figure 4.5: visualization of prediction results

As shown in Figure 4.5, we crafted the *visualize_results* function to bring object detection outcomes to visual representation. In executing this, we employed dictionary structures, points and color, to respectively archive center point coordinates and colors tied to distinct categories. Our toolset from *OpenCV*, such as *cv2.rectangle*, *cv2.putText*, and *cv2.line*, facilitated the drawing of bounding boxes, the overlaying of text labels onto the image, and the creation of lines between two coordinates. Leveraging the provided *jitter_detector*, we're equipped to discern jitters between two successive points. A detection cues a warning display on the image; in its absence, an auditory signal is emitted, coupled with the display, "Strike Standard". We've incorporated checks for the boxes attribute within the results, verifying its transformation potential into a numpy array, and modulating the bounding box and label visual based on category identification. Enhancing visualization, category nomenclature was shifted from hp to tip, and standard strikes are audibly underscored with the *winsound.Beep* feedback.

Chapter 5
Demo & Conclusion and Future Work

By applying the model, we aim to achieve a Demo that aligns with the project's requirements. Once the Demo is ready, we'll reflect on the project's journey, summarize the methodologies used, highlighting any limitations encountered, and outlining next steps for further development.

5.1 Demos

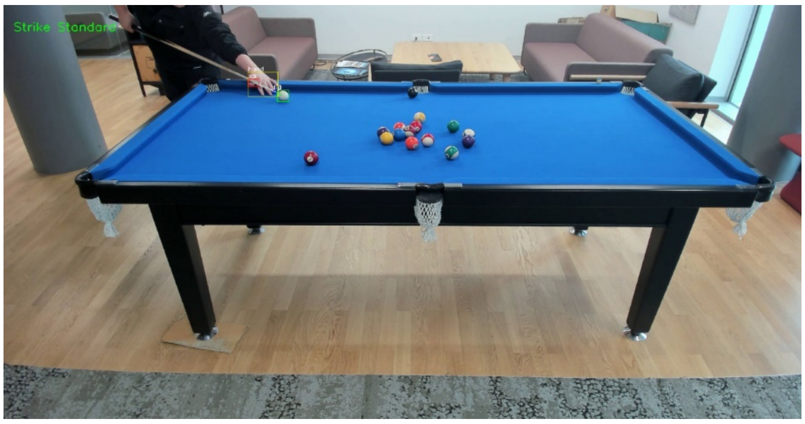


Figure 5.1: Shooting from the top left of the pool table

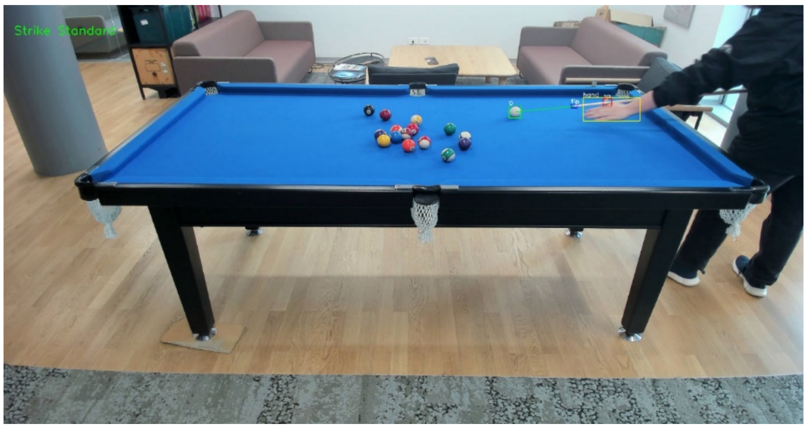


Figure 5.2: Shooting from the top right of the pool table.

Figures 5.1 and 5.2 clearly show us that the system can detect whether the cue stick is stable when the player shoots from different positions. There will be text prompts in the top left corner of the demo.

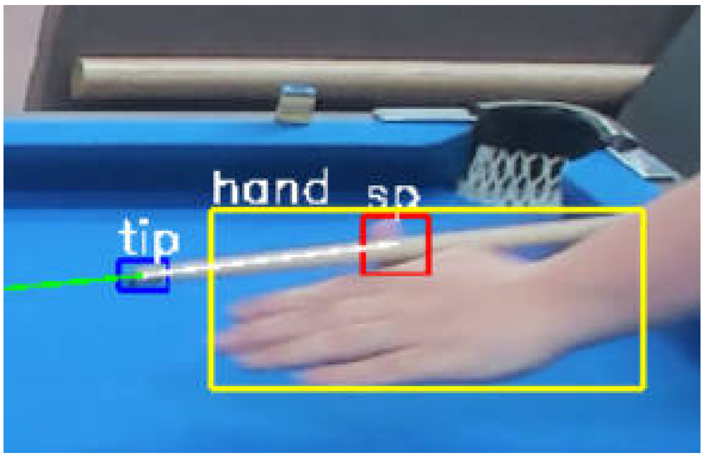


Figure 5.3: Details of the demo

As shown in Figure 5.3, the key targets the system uses to determine whether the player's stroke is stable are 'sp' and 'tip'. After normalizing these two detection target boxes, the system obtains the positions of the center points of the boxes and connects them with a line (this line is white in the Figure 5.3). The variation in the slope of this line serves as the basis for judging the stability of the player's stroke.

5.2 Discussions of Demos

In the pursuit of enhancing billiard training methodologies, our system, rooted in the capabilities of the yolov8m pretrained model, provides a revolutionary approach to evaluate the stability of players during their shots. This system is particularly beneficial for players and beginners during regular training sessions as it determines the steadiness of their strokes. The essence of our model revolves around the precise identification of two pivotal points: the sp, symbolizing the "snooker bridge" - a paramount region in a player's hand for efficiently gripping and supporting the cue stick, and the hp, indicating the point where the cue makes contact with the ball. The stability and relationship between these points are fundamental metrics, shedding light on a player's cue alignment proficiency and overall stability. Such factors significantly dictate the accuracy and consistency of shots in real-world gameplay or training.

Our innovative system offers the ability to promptly discern and evaluate the fluctuations and stability of these crucial coordinates, whether in a real-time setting or post-game video reviews. By offering both coaches and players this profound analytical tool, we are opening a window into the nuances of technical strengths and potential enhancements. Such an innovation, to a degree, can operate as a surrogate to traditional billiard coaches, focusing on rectifying players' striking patterns and movements, and thereby augmenting their competitive stature. We are optimistic that our solution will pave the way for players to refine their techniques, leading to heightened performance in the game.

5.3 Conclusion

The core of the entire project lies in the iterative training and optimization of the model. In terms of results, this development is undoubtedly successful. After only four rounds of model training and parameter tuning, we obtained a model with a precision of 0.946 mAP. The model can recognize from different angles the four positions when a billiard player strikes the ball: the hand, the cue ball, the hp(hitting point of the cue stick), and the sp(stand point of the cue stick). Among them, the sp is a relatively complex feature point, which is the contact point between the cue stick and the player's webbing. These findings prove that YOLOv8 can use minimal computational resources and develop a customized recognition model in a short period.

This project delved into a bespoke and confidential development process for YOLOv8, the newest convolutional neural network, underscoring its promising capabilities in the realm of sports.

5.4 Limitations of Outcomes

Due to the short development cycle, samples under different lighting conditions were not collected when building the dataset. As a result, the model's prediction accuracy might be unstable in various lighting environments. The system has only completed a demo so far and hasn't developed a full GUI. Therefore, general users would need to learn how to use this system.

5.5 Future Tasks

In subsequent development, we plan to incorporate the recognition of the entire billiard table surface and the balls. This will encompass a total of 22 classes, which will include 6 pockets and 16 balls.

The following figure 5.2 is a preliminary demo containing 22 classes. In the image, the balls were still not recognized. This is because the prediction accuracy of the model has not yet met expectations.

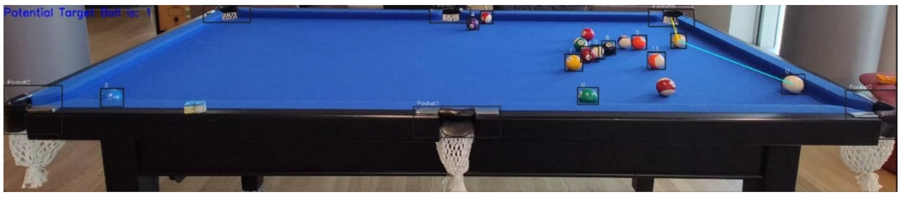


Figure 5.4: 22-class billiard table recognition demo

Afterwards, we aim to train a YOLO prediction model that meets expectations for the 22 classes. To achieve a precision of 0.95 mAP or above for a model with 22 categories, it's essential to make structural improvements to the base model. This necessitates a multitude of ablation experiments to identify components that are particularly sensitive to ball-related objectives. Subsequently, we will also employ the Taguchi method (often referred to as orthogonal array testing) to analyze the weight of the impact of each parameter tuning on model accuracy. This will further optimize the training speed of the model and reduce the time it takes for model training.

References

Altukroni, A., Alsaeedi, A., Gonzalez-Losada, C., Lee, J. H., Alabudh, M., Mirah, M., El-Amri, S., & Ezz El-Deen, O. (2023). Detection of the pathological exposure of pulp using an artificial intelligence tool: A multicentric study over periapical radiographs. *BMC Oral Health*, *23*(1), 553. https://doi.org/10.1186/s12903-023-03251-0

An, N., Yan, W. (2021) Multitarget tracking using Siamese neural networks. *ACM Transactions on Multimedia Computing, Communications and Applications.*

An, N. (2020) *Anomalies Detection and Tracking Using Siamese Neural Networks.* Master's Thesis. Auckland University of Technology, New Zealand.

Ayob, A. F., Khairuddin, K., Mustafah, Y. M., Salisa, A. R., & Kadir, K. (2021). Analysis of pruned neural networks (MobileNetV2-YOLO v2) for underwater object detection. In Z. Md Zain, H. Ahmad, D. Pebrianti, M. Mustafa, N. R. H. Abdullah,

R. Samad, & M. Mat Noh (Eds.), *Proceedings of the 11th National Technical Seminar on Unmanned System Technology 2019* (Vol. 666, pp. 87–98). Springer Nature Singapore. https://doi.org/10.1007/978-981-15-5281-6_7

Banerjee, C., Mukherjee, T., & Pasiliao, E. (2019). An empirical study on generalizations of the ReLU activation function. *ACM Southeast Conference*, 164–167. https://doi.org/10.1145/3299815.3314450

Boyd, K., Eng, K. H., & Page, C. D. (2013). Area under the precision-recall curve: point estimates and confidence intervals. In C. Salinesi, M. C. Norrie, & Ó. Pastor (Eds.), *Advanced Information Systems Engineering* (Vol. 7908, pp. 451–466). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-40994-3_29

Brownlow, J., Chu, C., Xu, G., Culbert, B., Fu, B., & Meng, Q. (2018). A multiple source based transfer learning framework for marketing campaigns. *International Joint Conference on Neural Networks (IJCNN)*, 1–8. https://doi.org/10.1109/IJCNN.2018.8489772

Cao, X. (2022) *Pose Estimation of Swimmers from Digital Images Using Deep Learning*. Master's Thesis, Auckland University of Technology.

Cao, X. and Yan, W. (2022) Pose estimation for swimmers in video surveillance. Multimedia Tools and Applications, Springer.

Çelik, B., & Çelik, M. E. (2023). Root dilaceration using deep learning: a diagnostic approach. *Applied* Sciences, *13*(14), 8260. https://doi.org/10.3390/app13148260

Çelik, B., Savaştaer, E. F., Kaya, H. I., & Çelik, M. E. (2023). The role of deep learning for periapical lesion detection on panoramic radiographs. *Dentomaxillofacial Radiology*, 20230118. https://doi.org/10.1259/dmfr.20230118

Chen, H., & Ji, Q. (2022). Convolutional neural network with attention mechanism for image-based smoke detection. *2022 4th International Conference on Advances in*

Computer Technology, Information Science and Communications (CTISC), 1–8. https://doi.org/10.1109/CTISC54888.2022.9849826

Chen, P.-Y., Chang, M.-C., Hsieh, J.-W., & Chen, Y.-S. (2021). Parallel residual bi-fusion feature pyramid network for accurate single-shot object detection. *IEEE Transactions on Image Processing*, *30*, 9099–9111. https://doi.org/10.1109/TIP.2021.3118953

Demetriou, D., Mavromatidis, P., Robert, P. M., Papadopoulos, H., Petrou, M. F., & Nicolaides, D. (2023). Real-time construction demolition waste detection using state-of-the-art deep learning methods*; Single – Stage vs Two-Stage Detectors* [Preprint]. SSRN. https://doi.org/10.2139/ssrn.4330569

Dixit, A., & Nain Chi, Y. (2021). Classification and recognition of urban tree defects in a small dataset using convolutional neural network, Resnet-50 Architecture, and Data Augmentation. *Journal of Forests*, *8*(1), 61–70. https://doi.org/10.18488/journal.101.2021.81.61.70

Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., & Zisserman, A. (2010). The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, *88*(2), 303–338. https://doi.org/10.1007/s11263-009-0275-4

Fuentes-Hurtado, F., Delaire, T., Levet, F., Sibarita, J.-B., & Viasnoff, V. (2022). MID3A: microscopy image denoising meets differentiable data augmentation. *2022 International Joint Conference on Neural Networks (IJCNN)*, 1–9. https://doi.org/10.1109/IJCNN55064.2022.9892954

Gowdra, N., Sinha, R., MacDonell, S., Yan, W. (2021) Maximum Categorical Cross Entropy (MCCE): A noise-robust alternative loss function to mitigate racial bias in Convolutional Neural Networks (CNNs) by reducing overfitting. *Pattern Recognition.*

Gowdra, N. (2021) *Entropy-Based Optimization Strategies for Convolutional Neural*

Networks. PhD Thesis, Auckland University of Technology, New Zealand.

Hao, Z., Post, C. J., Mikhailova, E. A., Lin, L., Liu, J., & Yu, K. (2022). How does sample labeling and distribution affect the accuracy and efficiency of a deep learning model for individual tree-crown detection and delineation. *Remote Sensing*, *14*(7), 1561. https://doi.org/10.3390/rs14071561

Herrera, A., Beck, A., Bell, D., Miller, P., Wu, Q., Yan, W. (2008) Behavior analysis and prediction in image sequences using rough sets. *International Machine Vision and Image Processing Conference* (pp.71-76)

Hu, Q., Si, X., Qin, A., Lv, Y., & Liu, M. (2022). Balanced adaptation regularization based transfer learning for unsupervised cross-domain fault diagnosis. *IEEE Sensors Journal*, *22*(12), 12139–*12151*. https://doi.org/10.1109/JSEN.2022.3174396

Huang, M., Liu, Z., Liu, T., & Wang, J. (2023). CCDS-YOLO: Multi-Category synthetic *aperture* radar image object detection model based on yolov5s. *Electronics*, *12*(16), 3497. https://doi.org/10.3390/electronics12163497

Ioffe, S., & Szegedy, C. (2015). Batch normalization: accelerating deep network training by *reducing* internal covariate shift. https://doi.org/10.48550/ARXIV.1502.03167

Jayapermana, R., Aradea, A., & Kurniati, N. I. (2022). Implementation of stacking ensemble classifier for multi-class classification of covid-19 vaccines topics on Twitter. *Scientific Journal of Informatics*, *9*(1), 8–15. https://doi.org/10.15294/sji.v9i1.31648

Jayaram, M., Kalpana, G., Borra, S. R., & Bhavani, B. D. (2023). A brief study on rice diseases recognition and image classification: Fusion deep belief network and S-particle swarm optimization algorithm. *International Journal of Electrical and Computer Engineering (IJECE)*, *13*(6), 6302. https://doi.org/10.11591/ijece.v13i6.pp6302-6311

Jayasundara, S., Indika, A., & Herath, D. (2022). Interpretable Student Performance Prediction Using Explainable Boosting Machine for Multi-Class Classification. *2022 2nd* International *Conference on Advanced Research in Computing (ICARC)*, 391–396. https://doi.org/10.1109/ICARC54489.2022.9753867

Jia, X., Peng, Y., Li, J., Xin, Y., Ge, B., & Liu, S. (2022). Pyramid dilated convolutional neural network for image denoising. *Journal of Electronic Imaging*, *31*(02). https://doi.org/10.1117/1.JEI.31.2.023024

Kieran, D., Yan, W. (2010) A framework for an event-driven video surveillance system. *Advanced Video and Signal Based Surveillance* (AVSS).

Kobylarz, J., Bird, J. J., Faria, D. R., Ribeiro, E. P., & Ekárt, A. (2020). Thumbs up, thumbs down: Non-verbal human-robot interaction through real-time EMG classification via inductive and supervised transductive transfer learning. *Journal of Ambient Intelligence And Humanized Computing*, *11*(12), 6021–6031. https://doi.org/10.1007/s12652-020-01852-z

Kozlov, D., Pavlov, S., Zuev, A., Bakulin, M., Krylova, M., & Kharchikov, I. (2022). Dual-valued Neural Networks. *IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, 1–8. https://doi.org/10.1109/AVSS56176.2022.9959227

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet classification with deep *convolutional* neural networks. *Communications of the ACM*, *60*(6), 84–90. https://doi.org/10.1145/3065386

Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324. https://doi.org/10.1109/5.726791

Li, F., Zhang, Y., Yan, W., Klette, R. (2016) Adaptive and compressive target tracking based on feature point matching. *International Conference on Pattern*

Recognition (ICPR), (pp.2734-2739).

Li, C., Li, L., Geng, Y., Jiang, H., Cheng, M., Zhang, B., Ke, Z., Xu, X., & Chu, X. (2023). *YOLOv6 v3.0: A Full-Scale Reloading*. https://doi.org/10.48550/ARXIV.2301.05586

Liang, C., Lu, J., Yan, W. (2022) Human action recognition from digital videos based on deep learning. *ACM ICCCV.*

Liu, C., Yan, W. (2020) Gait recognition using deep learning. *Handbook of Research on Multimedia Cyber Security* (pp.214-226)

Liu, P., Wang, X., Yin, L., & Liu, B. (2020). Flat random forest: A new ensemble learning method towards better training efficiency and adaptive model size to deep forest. *International Journal of Machine Learning and Cybernetics*, *11*(11), 2501–2513. https://doi.org/10.1007/s13042-020-01136-0

Liu, Z., Yan, W., Yang, B. (2018) Image denoising based on a CNN model. *International Conference on Control, Automation and Robotics.*

Lu, J. (2016) Empirical Approaches for Human Behavior Analytics. Master's Thesis. Auckland University of Technology, New Zealand.

Lu, J., Shen, J., Yan, W., Boris, B. (2017) An empirical study for human behaviors analysis. *International Journal of Digital Crime and Forensics* 9 (3), 11-17.

Lu, J., Nguyen, M., Yan, W. (2018) Pedestrian detection using deep learning. IEEE AVSS.

Lu, J., Nguyen, M., Yan, W. (2020) Human behavior recognition using deep learning. *International Conference on Image and Vision Computing New Zealand*.

Lu, J., Nguyen, M., Yan, W. (2020) Comparative evaluations of human behavior recognition using deep learning. *Handbook of Research on Multimedia Cyber Security*, 176-189.

Lu, J., Nguyen, M., Yan, W. (2021) Sign language recognition from digital videos using deep learning methods. *International Symposium on Geometry and Vision.*

Lu, J. (2021) *Deep Learning Methods for Human Behavior Recognition*. PhD Thesis. Auckland University of Technology, New Zealand.

Mastromichalakis, S. (2021). *SigmoReLU: An improvement activation function by combining sigmoid and ReLU*. ENGINEERING. https://doi.org/10.20944/preprints202106.0252.v1

Oksuz, K., Cam, B. C., Akbas, E., & Kalkan, S. (2018). Localization Recall Precision (LRP): A New Performance Metric for Object Detection. In V. Ferrari, M. Hebert, C. Sminchisescu, & Y. Weiss (Eds.), *Computer Vision – ECCV 2018* (Vol. 11211, pp. 521–537). Springer International Publishing. https://doi.org/10.1007/978-3-030-01234-2_31

Pan, C., Yan, W. (2018) A learning-based positive feedback in salient object detection. International Conference on Image and Vision Computing New Zealand.

Pan, C., Yan, W. (2020) Object detection based on saturation of visual perception. *Multimedia Tools and Applications*, 79 (27-28), 19925-19944.

Pan, C., Liu, J., Yan, W., Zhou, Y. (2021) Salient object detection based on visual perceptual saturation and two-stream hybrid networks. *IEEE Transactions on Image Processing.*

Ramachandran, P., Zoph, B., & Le, Q. V. (2017). *Searching for Activation Functions* (arXiv:1710.05941). arXiv. http://arxiv.org/abs/1710.05941

Sarker, I. H. (2021). Machine Learning: Algorithms, Real-World Applications and Research Directions. *SN Computer Science*, *2*(3), 160. https://doi.org/10.1007/s42979-021-00592-x

Sharma, O. (2022). Exploring the statistical properties and developing a non-linear activation function. *International Conference on Automation, Computing and Renewable Systems (ICACRS)*, 1370–1375. https://doi.org/10.1109/ICACRS55517.2022.10029124

Sujatha, K., Amrutha, K., & Veeranjaneyulu, N. (2023). Enhancing object detection with Mask R-CNN: a deep learning perspective. *International Conference on Network, Multimedia and Information Technology (NMITCON)*, 1–6. https://doi.org/10.1109/NMITCON58196.2023.10276033

Sukhija, S., Krishnan, N. C., & Kumar, D. (2018). Supervised heterogeneous transfer learning using random forests. *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data*, 157–166. https://doi.org/10.1145/3152494.3152510

Thakur, P. M., & M, T. (2022). Multi-class classification of twitter sentiments using frequency based, LSTM and BERT methods. *International Journal for Research in Applied Science and Engineering Technology*, *10*(4), 3393–3402. https://doi.org/10.22214/ijraset.2022.42085

Trung, N. T., Trinh, D.-H., Trung, N. L., & Luong, M. (2022). Low-dose CT image denoising using *deep* convolutional neural networks with extended receptive fields. *Signal, Image and Video Processing*, *16*(7), 1963–1971. https://doi.org/10.1007/s11760-022-02157-8

Chen, K., He, Z., Wang, S. X. (2018). Learning-based data analytics: Moving towards transparent *power* grids. *CSEE Journal of Power and Energy Systems*, *4*(1), 67–82. https://doi.org/10.17775/CSEEJPES.2017.01070

Umadevi T P & Murugan A. (2021). Enhanced handwritten document recognition using *confusion* matrix analysis. In D. J. Hemanth, M. Elhosney, T. N. Nguyen, & S. Lakshmanan (Eds.), *Advances in Parallel Computing*. IOS Press. https://doi.org/10.3233/APC210131

Vasanthi, P., & Mohan, L. (2023). Multi-Head-Self-Attention based YOLOv5X-transformer for *multi*-scale object detection. *Multimedia Tools and Applications*. https://doi.org/10.1007/s11042-023-15773-4

Wang, C.-Y., Bochkovskiy, A., & Liao, H.-Y. M. (2023). YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 7464–7475. https://doi.org/10.1109/CVPR52729.2023.00721

Wang, Z., Bovik, A. C., Sheikh, H. R., & Simoncelli, E. P. (2004). Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, *13*(4), *600*–612. https://doi.org/10.1109/TIP.2003.819861

Xiao, F., Pang, L., Lan, Y., Wang, Y., Shen, H., & Cheng, X. (2021). Transductive learning for unsupervised text style transfer. *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 2510–2521. https://doi.org/10.*18653*/v1/2021.emnlp-main.195

Xu, X., Chen, Q., Xie, L., & Su, H. (2020). Batch-Normalization-based soft filter pruning for deep *convolutional* neural networks. *International Conference on Control, Automation, Robotics and Vision (ICARCV)*, 951–956. https://doi.org/10.1109/ICARCV50220.2020.9305319

Yan, J., & Wang, X. (2022). Unsupervised and semi-supervised learning: The next frontier in machine *learning* for plant systems biology. *The Plant Journal*, *111*(6), 1527–1538. https://doi.org/10.1111/tpj.15905

Yan, W. Q. (*2019*). *Introduction to Intelligent Surveillance: Surveillance Data Capture, Transmission, and Analytics*. Springer International Publishing.

Yan, W. Q. (202). *Computational Methods for Deep Learning: Theoretic, Practice and Applications*. Springer International Publishing.

Yang, P., Chen, J., Wu, L., & Li, S. (2022). Fault identification of electric submersible pumps based on unsupervised and multi-source transfer learning integration. *Sustainability*, *14*(16), 9870. https://doi.org/10.3390/su14169870

Yen, S.-Y., Huang, H.-E., Lien, G.-S., Liu, C.-W., Chu, C.-F., Huang, W.-M., & Suk, F.-M. (2021). Automatic lumen detection and magnetic alignment control for magnetic-assisted capsule colonoscope system optimization. *Scientific Reports*, *11*(1), 6460. https://doi.org/10.1038/s41598-021-86101-9

Yu, F., & Koltun, V. (2015). *Multi-scale context aggregation by dilated convolutions*. https://doi.org/10.48550/ARXIV.1511.07122

Yu, Z. (2021) *Deep Learning Methods for Human Action Recognition*. Master's Thesis, Auckland University of Technology, New Zealand.

Yu, Z., Yan, W. (2020) Human action recognition using deep learning methods. *International Conference on Image and Vision Computing New Zealand.*

Zhang, S. (2022). Evaluation of the physical education teaching and training efficiency by the integration of ideological and political courses with lightweight deep learning. *Computational Intelligence and Neuroscience*, 1–10. https://doi.org/10.1155/2022/4670523

Zhang, Y., & Freris, N. M. (2023). Adaptive filter pruning via sensitivity feedback. *IEEE Transactions on Neural Networks and Learning Systems*, 1–13. https://doi.org/10.1109/TNNLS.2023.3246263

Zhang, Z., Lu, X., Cao, G., Yang, Y., Jiao, L., & Liu, F. (2021). ViT-YOLO: Transformer-based YOLO for object detection. *IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, 2799–2808.

Zhou, L., Wei, S., Cui, Z., & Ding, W. (2019). YOLO-RD: A lightweight object detection network for range doppler radar images. *IOP Conference Series: Materials*

Science and Engineering, *563*(4), 042027.

Zhou, H., Nguyen, M., Yan, W. (2023) Computational analysis of table tennis matches from real-time videos using deep learning. PSIVT 2023.