

Object Detection Using Computer Vision

Jiramate Rujikornhirun¹

Received: May 25, 2025 – Revised: August 17, 2025 – Accepted: August 21, 2025

Abstract

Object detection has become an important technology in artificial intelligence and computer vision. Many industries now use this technology, including self-driving cars, medical image analysis, security systems, and manufacturing. This paper presents a detailed study of object detection methods through three main parts. The first part covers the basic theories and concepts of object detection. We explain the differences between image classification and object detection, and describe the main parts of detection systems. We also discuss how to measure performance using average precision (AP) and mean average precision (mAP). The section includes a review of how detection models have developed over time, from older methods to modern one-stage and two-stage approaches. We provide practical examples using Faster R-CNN. The second part focuses on the YOLO (You Only Look Once) algorithm, which represents single-stage detection methods. YOLO is popular because it works fast while keeping good accuracy. We explain how YOLO works, including how it divides images into grids and predicts bounding boxes and object types at the same time. We show detailed code examples for detecting objects in still images using YOLOv5. The last part extends to video object detection, which brings new challenges for real-time processing. We discuss various techniques to improve performance, such as model quantization, pruning, knowledge distillation, and combining object tracking with detection using DeepSORT. We demonstrate practical applications by showing how to detect and track cars in traffic videos. The demonstrations illustrate how pretrained object detection models, particularly Faster R-CNN and YOLOv5, can be applied to still images and video frames for instructional purposes. The article does not aim to provide a systematic benchmark comparison, but rather to present practical implementation examples for readers who are beginning to use object detection techniques.

Keywords: Object Detection, Computer Vision, Deep Learning

¹ Corresponding Author,

Bodindecha (Sing Singhaseni) School, Bangkok, 10310, Thailand

Email: jiramate.r@bodind.ac.th

การตรวจจับวัตถุโดยใช้คอมพิวเตอร์วิทัศน์

จิระเมศร์ รุจิกรหิรัญย์¹

รับต้นฉบับ : 25 พฤษภาคม 2568 – รับแก้ไข : 17 สิงหาคม 2568 – ตอบรับตีพิมพ์ : 21 สิงหาคม 2568

บทคัดย่อ

ในปัจจุบัน การพัฒนาเทคโนโลยีปัญญาประดิษฐ์และคอมพิวเตอร์วิทัศน์มีการเติบโตอย่างก้าวกระโดด การตรวจจับวัตถุ (object detection) ถือเป็นหนึ่งในเทคโนโลยีสำคัญที่ได้รับความสนใจอย่างต่อเนื่องจากนักวิจัยและนักพัฒนา หลายอุตสาหกรรมนำเทคโนโลยีนี้ไปประยุกต์ใช้ เช่น ระบบยานยนต์อัตโนมัติ การวิเคราะห์ภาพทางการแพทย์ ระบบเฝ้าระวังความปลอดภัย และการประมวลผลภาพในอุตสาหกรรม บทความนี้มุ่งนำเสนอการศึกษาเกี่ยวกับวิธีการตรวจจับวัตถุใน 3 ส่วนหลัก ส่วนแรกนำเสนอแนวคิดพื้นฐานและทฤษฎีที่เกี่ยวข้องกับการตรวจจับวัตถุ โดยอธิบายความแตกต่างระหว่างการจำแนกภาพ (image classification) และการตรวจจับวัตถุ รวมถึงองค์ประกอบหลักของระบบตรวจจับวัตถุ นอกจากนี้ ยังกล่าวถึงวิธีการประเมินประสิทธิภาพของโมเดลด้วยค่า average precision (AP) และ mean average precision (mAP) รวมทั้งทบทวนพัฒนาการของโมเดลตรวจจับวัตถุจากวิธีการดั้งเดิมไปสู่แนวทางสมัยใหม่แบบ one-stage และ two-stage พร้อมยกตัวอย่างการใช้งาน Faster R-CNN ส่วนที่สองมุ่งเน้นการศึกษาอัลกอริทึม YOLO (You Only Look Once) ซึ่งเป็นตัวแทนของวิธีการตรวจจับวัตถุแบบ single-stage โดย YOLO ได้รับความนิยมเนื่องจากสามารถทำงานได้รวดเร็วและยังคงรักษาความแม่นยำไว้ในระดับที่ดี บทความอธิบายหลักการทำงานของ YOLO รวมถึงการแบ่งภาพออกเป็นกริด และการทำนายกรอบขอบเขตกับประเภทของวัตถุไปพร้อมกัน พร้อมนำเสนอตัวอย่างโค้ดสำหรับการตรวจจับวัตถุในภาพนิ่งด้วย YOLOv5 ส่วนสุดท้ายขยายไปสู่การตรวจจับวัตถุในวิดีโอ ซึ่งมีความท้าทายเพิ่มเติมด้านการประมวลผลแบบเรียลไทม์ โดยกล่าวถึงเทคนิคต่าง ๆ เพื่อเพิ่มประสิทธิภาพของโมเดล เช่น model quantization, pruning, knowledge distillation และการผสมผสานการติดตามวัตถุเข้ากับการตรวจจับวัตถุโดยใช้ DeepSORT บทความยังสาธิตการประยุกต์ใช้โดยแสดงตัวอย่างการตรวจจับและติดตามรถยนต์ในวิดีโอจราจร การสาธิตเหล่านี้แสดงให้เห็นว่าโมเดลตรวจจับวัตถุที่ผ่านการฝึกฝนล่วงหน้า โดยเฉพาะ Faster R-CNN และ YOLOv5 สามารถนำไปประยุกต์ใช้กับภาพนิ่งและเฟรมวิดีโอเพื่อวัตถุประสงค์เชิงการรับรู้ได้ ทั้งนี้ บทความไม่ได้มุ่งเปรียบเทียบประสิทธิภาพของโมเดลอย่างเป็นระบบ แต่มุ่งนำเสนอตัวอย่างการใช้งานเชิงปฏิบัติสำหรับผู้อ่านที่เริ่มต้นศึกษาการตรวจจับวัตถุ

คำสำคัญ : การตรวจจับวัตถุ, คอมพิวเตอร์วิทัศน์, การเรียนรู้เชิงลึก

¹ ผู้รับผิดชอบบทความหลัก,
โรงเรียนดินทรเดชา (สิงห์ สิงหเสนี) กรุงเทพฯ 10310
jjramate.r@bodin.ac.th

ในปัจจุบันการพัฒนาเทคโนโลยีปัญญาประดิษฐ์และคอมพิวเตอร์วิทัศน์มีการเติบโตอย่างรวดเร็ว การกระโดด การตรวจจับวัตถุ (object detection) ถือเป็นหนึ่งในกระบวนการสำคัญที่ได้รับ ความสนใจอย่างต่อเนื่องจากนักวิจัยและนักพัฒนา ปัจจัยหลักมาจากศักยภาพในการประยุกต์ใช้ ที่หลากหลาย ไม่ว่าจะเป็นระบบยานยนต์อัตโนมัติ การวิเคราะห์ภาพทางการแพทย์ ระบบการเฝ้า ระวังความปลอดภัย และการประมวลผลภาพในอุตสาหกรรม ซึ่งการทำงานดังกล่าวอาศัย ความสามารถในการระบุและกำหนดตำแหน่งของวัตถุในภาพหรือวิดีโอได้อย่างถูกต้องและ แม่นยำ

การตรวจจับวัตถุเป็นงานที่ซับซ้อนกว่าการจำแนกภาพ (image classification) นอกจาก การระบุประเภทของวัตถุแล้ว ยังจำเป็นต้องระบุพิกัดตำแหน่งที่ถูกต้องในรูปแบบของกรอบ ขอบเขต (bounding box) ร่วมกับการแสดงค่าความน่าจะเป็น (probability score) ที่สะท้อน ความเชื่อมั่นในการระบุชนิดของวัตถุนั้น ๆ กระบวนการดังกล่าวอาศัยกรอบการทำงานของ โครงข่ายประสาทเทียมแบบคอนโวลูชัน (Convolutional Neural Networks: CNN) ที่ถูกพัฒนา เพื่อให้อาจสามารถประมวลผลข้อมูลเชิงพื้นที่จากภาพได้อย่างมีประสิทธิภาพ

บทความฉบับนี้มุ่งนำเสนอขั้นตอนและวิธีการการตรวจจับวัตถุใน 3 หัวข้อหลักที่มีความ สอดคล้องและต่อเนื่องกัน โดยเริ่มจากการวิเคราะห์แนวคิดพื้นฐานและทฤษฎีที่เกี่ยวข้องกับการ ตรวจจับวัตถุ การประเมินประสิทธิภาพของโมเดลตรวจจับวัตถุด้วยเมตริกซ์ต่าง ๆ รวมถึงการ ทบทวนวรรณกรรมที่เกี่ยวข้องกับอัลกอริทึมตรวจจับวัตถุที่ได้รับการพัฒนาในช่วงทศวรรษที่ผ่าน มา หัวข้อที่สองจะมุ่งเน้นการศึกษาโมเดล YOLO (You Only Look Once) ซึ่งเป็นอัลกอริทึมแบบ Single-Stage Detection ที่ได้รับการยอมรับในวงกว้างเนื่องจากสามารถทำงานได้อย่างรวดเร็ว โดยยังคงรักษาความแม่นยำไว้ในระดับที่น่าพอใจ บทความจะนำเสนอสถาปัตยกรรมของ YOLO รวมถึงกระบวนการติดตั้ง การปรับพารามิเตอร์ที่เหมาะสม และการวิเคราะห์ผลลัพธ์จากการ ประมวลผลภาพนิ่ง โดยจะมีการสาธิตการประมวลผลผ่านตัวอย่างโค้ดที่มีการอธิบายอย่าง ละเอียด และในหัวข้อสุดท้ายจะขยายขอบเขตไปสู่การตรวจจับวัตถุในภาพเคลื่อนไหว ซึ่งนำเสนอ ความท้าทายเพิ่มเติมในด้านประสิทธิภาพและการประมวลผลแบบเรียลไทม์ เนื่องจากต้อง ดำเนินการตรวจจับในหลายเฟรมต่อเนื่องกัน โดยมีจุดมุ่งหมายให้ผู้อ่านมีความเข้าใจพื้นฐาน เกี่ยวกับการตรวจจับวัตถุโดยใช้คอมพิวเตอร์วิทัศน์ หรือสามารถปรับประยุกต์ใช้โค้ดในการ ตรวจจับวัตถุพื้นฐานได้

ความรู้พื้นฐานที่สำคัญของผู้อ่าน

ผู้อ่านบทความเกี่ยวกับการตรวจจับวัตถุฉบับนี้ควรมีความรู้พื้นฐานในประเด็นต่อไปนี้เพื่อ ความเข้าใจที่ดียิ่งขึ้น 1) ความรู้พื้นฐานการเขียนโปรแกรมด้วยภาษา Python เนื่องจากเป็นภาษา หลักที่ใช้ในการพัฒนาโมเดลการมองเห็นของคอมพิวเตอร์ รวมถึงไลบรารีสำคัญเช่น OpenCV, PyTorch และ TensorFlow 2) ความรู้พื้นฐานด้านการประมวลผลภาพ (image processing) เนื่องจากการตรวจจับวัตถุจำเป็นต้องเริ่มต้นจากการประมวลผลภาพเป็นพื้นฐานก่อน 3) แนวคิด พื้นฐานของการเรียนรู้ของเครื่อง เนื่องจากโมเดลตรวจจับวัตถุทั้งหมดอาศัยหลักการของการ เรียนรู้ของเครื่อง การเข้าใจแนวคิดพื้นฐานจะช่วยให้เข้าใจกระบวนการฝึกฝนและปรับปรุงโมเดล และ 4) แนวคิดพื้นฐานของโครงข่ายประสาทเทียมแบบคอนโวลูชัน เนื่องจากเป็นหัวใจสำคัญของ โมเดลตรวจจับวัตถุรวมถึงสถาปัตยกรรมของ YOLO

- บทความฉบับนี้แบ่งเนื้อหาออกเป็น 3 หัวข้อหลัก ได้แก่ 1) พื้นฐานการตรวจจับวัตถุ 2) การตรวจจับวัตถุในภาพนิ่งด้วย YOLO (object detection in static images using YOLO) 3) การตรวจจับวัตถุในวิดีโอด้วย YOLO (real-time object detection in video with YOLO)

ส่วนที่ 1 พื้นฐานการตรวจจับวัตถุ

ในส่วนที่ 1 จะเป็นการอธิบายพื้นฐานความรู้เกี่ยวกับการตรวจจับวัตถุโดยแบ่งออกเป็น 5 หัวข้อ ได้แก่ ความแตกต่างระหว่างการจำแนกวัตถุและการตรวจจับวัตถุ องค์ประกอบพื้นฐานของการตรวจจับวัตถุ วิธีการประเมินประสิทธิภาพของโมเดลการตรวจจับวัตถุ วิวัฒนาการของโมเดลตรวจจับวัตถุ และตัวอย่างการตรวจจับวัตถุขั้นพื้นฐาน โดยมีรายละเอียดดังนี้

1. ความแตกต่างระหว่างการจำแนกวัตถุและการตรวจจับวัตถุ

การจำแนกวัตถุ (image classification) และการตรวจจับวัตถุ (object detection) เป็นงานพื้นฐานสองประเภทในคอมพิวเตอร์วิทัศน์ที่มีวัตถุประสงค์และความซับซ้อนแตกต่างกัน การจำแนกวัตถุมีเป้าหมายในการระบุภาพที่กำหนดให้มีวัตถุประเภทใดอยู่ โดยผลลัพธ์จะเป็นป้ายกำกับ (label) เดียวหรือหลายป้ายกำกับที่บ่งบอกถึงประเภทของวัตถุที่พบในภาพ เช่น แมว สุนัข หรือรถยนต์ โดยไม่ต้องระบุตำแหน่งที่แน่นอนของวัตถุเหล่านั้น แต่การตรวจจับวัตถุมีความซับซ้อนมากกว่า เนื่องจากไม่เพียงแต่ต้องระบุประเภทของวัตถุ แต่ยังต้องกำหนดตำแหน่งของวัตถุในภาพด้วย ผลลัพธ์ของการตรวจจับวัตถุจะประกอบด้วยกล่องขอบเขต (bounding box) ที่ระบุพิกัด (x, y, width, height) ของวัตถุแต่ละชิ้น พร้อมกับป้ายกำกับและคะแนนความเชื่อมั่น (confidence score) สำหรับแต่ละวัตถุ (Redmon et al., 2016)

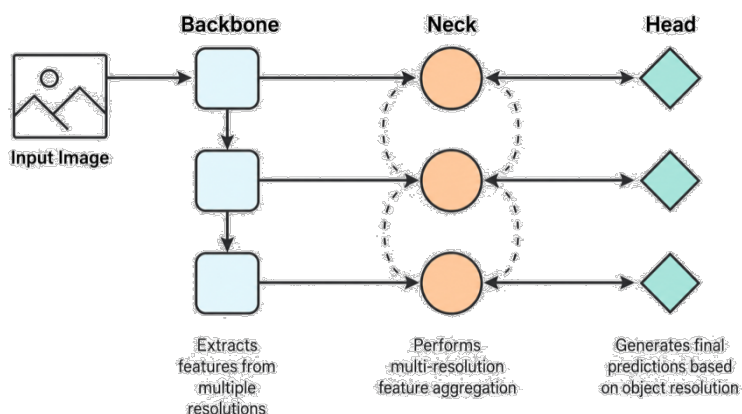
ความแตกต่างสำคัญอีกประการหนึ่ง คือ การตรวจจับวัตถุสามารถค้นหวัตถุหลายชิ้นในภาพเดียวได้ ในขณะที่การจำแนกวัตถุแบบดั้งเดิมมักจะให้ผลลัพธ์เป็นคลาสเดียวสำหรับทั้งภาพ นอกจากนี้ การตรวจจับวัตถุยังต้องจัดการกับปัญหาการซ้อนทับของวัตถุ (occlusion) และวัตถุที่มีขนาดแตกต่างกันในภาพเดียว ทำให้เป็นงานที่ท้าทายกว่าการจำแนกวัตถุทั่วไป (Liu et al., 2020)

2. องค์ประกอบพื้นฐานของการตรวจจับวัตถุ

การตรวจจับวัตถุประกอบด้วยองค์ประกอบหลัก 4 ส่วน ได้แก่ 1) Backbone Network 2) Neck 3) Detection Head และ 4) Loss Function แต่ละส่วนมีบทบาทสำคัญในการทำงานของระบบ ดังนี้

ภาพ 1

องค์ประกอบหลักของการตรวจจับวัตถุ



ที่มา : Park et al. (2024)

1) Backbone Network ทำหน้าที่เป็นโครงสร้างพื้นฐานในการสกัดคุณลักษณะเด่น (feature extraction) จากภาพต้นฉบับ โดยทั่วไปจะใช้โครงข่ายประสาทเทียมแบบ Convolutional Neural Network (CNN) เช่น ResNet VGG หรือ EfficientNet ที่ได้รับการฝึกฝนล่วงหน้าด้วยชุดข้อมูล ImageNet เพื่อเรียนรู้การแทนค่าลักษณะเด่นที่มีประสิทธิภาพ (He et al., 2016)

2) Neck เป็นส่วนที่เชื่อมต่อระหว่าง backbone และ detection head โดยมีหน้าที่ในการรวมและปรับปรุงลักษณะเด่นจากหลายระดับ (multi-scale features) เพื่อให้สามารถตรวจจับวัตถุที่มีขนาดแตกต่างกันได้อย่างมีประสิทธิภาพ เช่น Feature Pyramid Network (FPN) และ Path Aggregation Network (PANet)

3) Detection Head เป็นส่วนสุดท้ายที่ทำการทำนายผลลัพธ์ โดยจะแบ่งออกเป็นสองส่วนหลัก คือ classification head ที่ทำนายประเภทของวัตถุ และ regression head ที่ทำนายตำแหน่งและขนาดของกล่องขอบเขต หลายโมเดลยังมี objectness head เพิ่มเติมเพื่อประเมินความน่าจะเป็นที่จุดนั้นจะมีวัตถุอยู่

4) Loss Function ใช้สำหรับการฝึกฝนโมเดล โดยจะรวมการสูญเสียจากการจำแนกประเภท (classification loss) และการถดถอยของตำแหน่ง (localization loss) เข้าด้วยกัน Loss Function ที่นิยมใช้ เช่น Cross-entropy loss สำหรับการจำแนกประเภท และ Smooth L1 loss หรือ IoU loss สำหรับการถดถอย (Lin, Dollár, et al., 2017)

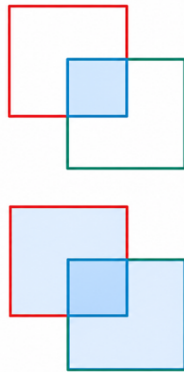
3. วิธีการประเมินประสิทธิภาพของโมเดลการตรวจจับวัตถุ

การประเมินประสิทธิภาพของโมเดลการตรวจจับวัตถุใช้เมตริกซ์หลัก คือ Average Precision (AP) และ mean Average Precision (mAP) ซึ่งเป็นมาตรฐานสากลที่ได้รับการยอมรับในวงการคอมพิวเตอร์วิทัศน์ เมตริกซ์เหล่านี้คำนึงถึงทั้งความแม่นยำในการจำแนกประเภทและความแม่นยำในการระบุตำแหน่งของวัตถุ

Intersection over Union (IoU) เป็นพื้นฐานสำคัญในการคำนวณ AP โดยใช้วัดความซ้อนทับระหว่างกล่องขอบเขตที่ทำนายได้กับกล่องขอบเขตที่ถูกต้อง (ground truth) การทำนายจะถือว่าถูกต้อง (True Positive) เมื่อ IoU มีค่ามากกว่าเกณฑ์ที่กำหนด โดยทั่วไปใช้ 0.5 หรือช่วง 0.5 ถึง 0.95

ภาพ 2

ที่มาและความหมายของ Intersection over Union (IoU)

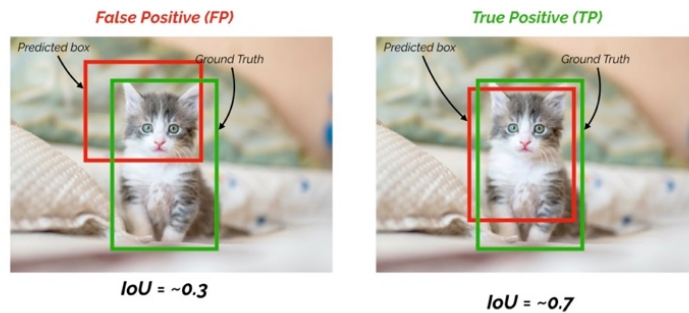
$$\text{Intersection over Union (IoU)} = \frac{\text{Area of Overlap}}{\text{Area of Union}} =$$


ที่มา : Rosebrock (2016)

ภาพ 3

ตัวอย่างการคำนวณ Intersection over Union (IoU)

If IoU threshold = 0.5

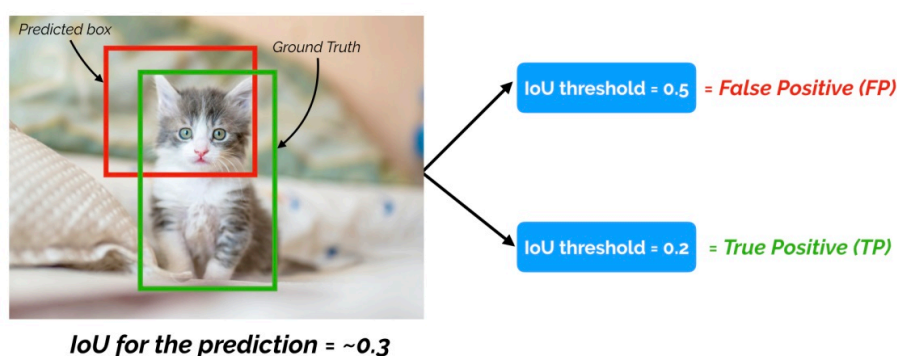


ที่มา : Shah (2022)

Precision คือ สัดส่วนของการทำนายที่ถูกต้องจากการทำนายทั้งหมด ในขณะที่ Recall คือ สัดส่วนของวัตถุจริงที่ถูกตรวจพบได้ การคำนวณ AP จะใช้เส้นโค้ง Precision-Recall โดยการหาพื้นที่ใต้เส้นโค้ง (Area Under Curve) สำหรับแต่ละคลาส และ mAP คือค่าเฉลี่ยของ AP ทุกคลาส โดยมีลักษณะคล้ายคลึงกับการคำนวณในการเรียนรู้ของเครื่อง นอกจากนี้เมทริกซ์เพิ่มเติมที่สำคัญ ได้แก่ AP ที่ IoU threshold = 0.5 หรือ AP ที่ IoU threshold = 0.75 และ AP ที่เฉลี่ยจาก IoU threshold 0.5 ถึง 0.95 โดยเว้นระยะ 0.05 นอกจากนี้ยังมีการแบ่งตามขนาดของวัตถุเป็น AP small AP medium และ AP large เพื่อประเมินประสิทธิภาพในการตรวจจับวัตถุขนาดต่าง ๆ (Lin et al., 2014)

ภาพ 4

ตัวอย่างการพิจารณา AP ที่ IoU threshold ที่แตกต่างกันในแต่ละภาพ



ที่มา : Yohanandan (2020)

4. วิวัฒนาการของโมเดลตรวจจับวัตถุ

วิวัฒนาการของโมเดลตรวจจับวัตถุสามารถแบ่งออกเป็นสามยุคหลัก ได้แก่ ยุคดั้งเดิม (Traditional Methods) ยุค Two-Stage Detectors และ ยุค One-Stage Detectors โดยแต่ละยุคมีรายละเอียดดังนี้

ยุคดั้งเดิมหรือยุคก่อน Deep Learning ใช้วิธีการสกัดลักษณะเด่นด้วยมือ เช่น SIFT HOG และ Haar features ร่วมกับ classifier แบบดั้งเดิม เช่น SVM ตัวอย่างที่มีชื่อเสียง คือ Viola-Jones face detector ซึ่งใช้ Haar features และ AdaBoost แม้จะมีประสิทธิภาพจำกัด แต่วิธีการเหล่านี้เป็นรากฐานที่สำคัญสำหรับงานตรวจจับวัตถุ

ยุค Two-Stage Detectors เริ่มต้นด้วย R-CNN ในปี 2014 ที่นำเสนอแนวคิดการใช้ CNN สำหรับการตรวจจับวัตถุ โดยแบ่งกระบวนการเป็นสองขั้นตอน คือ การสร้าง region proposals และการจำแนกประเภท ต่อมา Fast R-CNN (2015) ปรับปรุงความเร็วด้วยการแชร์ computation และ Faster R-CNN (2015) เพิ่ม Region Proposal Network (RPN) เพื่อสร้าง proposals แบบ end-to-end ทำให้เป็นโมเดลที่สมบูรณ์และมีประสิทธิภาพสูง (Girshick et al., 2014; Ren et al., 2015)

ยุค One-Stage Detectors มุ่งเน้นความเร็วโดยรวมการตรวจจับเป็นขั้นตอนเดียว YOLO (You Only Look Once) เป็นผู้นำบุกเบิกที่แบ่งภาพเป็นกริดและทำนายกล่องขอบเขตโดยตรง ต่อมา SSD (Single Shot MultiBox Detector) ปรับปรุงด้วย multi-scale predictions และ RetinaNet แก้ปัญหา class imbalance ด้วย Focal Loss ทำให้ one-stage detectors มีประสิทธิภาพใกล้เคียง two-stage แต่เร็วกว่ามาก (Redmon et al., 2016; Liu et al., 2016; Lin, Goyal, et al., 2017) โดยในปัจจุบันได้มีการผสมผสานและการพัฒนาต่อยอด เช่น EfficientDet ที่เน้นประสิทธิภาพ DETR (Detection Transformer) ที่นำ Transformer มาประยุกต์ใช้ และ YOLO series ที่พัฒนาต่อเนื่องจนถึง YOLOv8 และ YOLOv11 การพัฒนาในปัจจุบันมุ่งเน้นการปรับสมดุลระหว่างความแม่นยำ ความเร็ว และประสิทธิภาพการใช้ทรัพยากร เพื่อตอบสนองความต้องการของการใช้งานจริงที่หลากหลาย (Carion et al., 2020; Tan et al., 2020)

5. ตัวอย่างการตรวจจับวัตถุขั้นพื้นฐาน

ตัวอย่างการตรวจจับวัตถุขั้นพื้นฐาน ผู้เขียนได้ยกตัวอย่างการทำงานของ Faster R-CNN ซึ่งเป็นอัลกอริทึมแบบ Region-based ตามข้อ 4.

1) โครงสร้างของ Faster R-CNN

Faster R-CNN เป็นหนึ่งในโมเดลตรวจจับวัตถุที่มีประสิทธิภาพและได้รับความนิยมมาก พัฒนาโดย Ren และคณะในปี 2015 โมเดลนี้ถือเป็นการพัฒนาต่อยอดจาก R-CNN และ Fast R-CNN โดยมีการปรับปรุงสำคัญคือการใช้ Region Proposal Network (RPN) ทำงานร่วมกับโครงข่ายสกัดคุณลักษณะแบบ CNN โมเดลนี้ประกอบด้วยองค์ประกอบหลัก 4 ส่วน

- Backbone Network: โครงข่าย CNN เช่น ResNet ที่ทำหน้าที่สกัดคุณลักษณะจากภาพ
- Region Proposal Network (RPN): เครือข่ายที่เสนอพื้นที่ที่น่าจะมีวัตถุอยู่
- RoI Pooling Layer: ชั้นที่ปรับขนาดของพื้นที่ที่เสนอให้มีขนาดเท่ากัน
- Classification & Bounding Box Regression Heads: ส่วนที่ทำการจำแนกประเภทวัตถุและปรับปรุงตำแหน่งของกรอบสี่เหลี่ยม

Faster R-CNN ยังคงใช้วิธีการแบบ two-stage detection โดยขั้นแรก คือ การระบุตำแหน่งที่น่าจะมีวัตถุ (region proposals) และขั้นที่สอง คือ การจำแนกประเภทวัตถุในแต่ละ

ตำแหน่งที่เสนอ ความแตกต่างจากรุ่นก่อนหน้าคือ RPN ทำงานได้เร็วกว่าและมีประสิทธิภาพมากกว่าทำให้สามารถตรวจจับวัตถุได้เร็วขึ้นอย่างมีนัยสำคัญ

2) ตัวอย่างโคดการตรวจจับวัตถุด้วยอัลกอริทึม Faster R-CNN

ขั้นตอนที่ 1: การติดตั้งไลบรารีที่จำเป็นสำหรับการตรวจจับวัตถุ

```
pip install torch torchvision matplotlib pillow opencv-python
```

คำสั่งนี้เป็นการติดตั้งแพ็คเกจ ซึ่งในแต่ละแพ็คเกจมีรายละเอียด ดังนี้

- แพ็คเกจ torch และ torchvision ซึ่งเป็นแพ็คเกจหลักของ PyTorch สำหรับการเรียนรู้เชิงลึกและคอมพิวเตอร์วิทัศน์

- แพ็คเกจ matplotlib สำหรับการแสดงผลภาพ

- แพ็คเกจ pillow สำหรับการประมวลผลภาพ

- แพ็คเกจ opencv-python สำหรับการจัดการภาพเพิ่มเติมและการแสดงผล

ขั้นตอนที่ 2: การนำเข้าไลบรารีที่จำเป็น

```
import torch
import torchvision.transforms as T
from torchvision.models.detection import
fasterrcnn_resnet50_fpn, FasterRCNN_ResNet50_FPN_Weights
import cv2
import matplotlib.pyplot as plt
from google.colab.patches import cv2_imshow
# เพื่อให้สามารถแสดงภาพใน Jupyter notebook แทนการเปิด window ใหม่
import numpy as np
from PIL import Image
```

ไลบรารีเหล่านี้มีหน้าที่แตกต่างกันไป โดยมีรายละเอียดดังนี้

- torch แพ็คเกจหลักของ PyTorch สำหรับการคำนวณเทนเซอร์

- torchvision.transforms สำหรับการประมวลผลภาพก่อนนำเข้าโมเดล

- torchvision.models.detection มีโมเดลตรวจจับวัตถุที่ผ่านการเทรนมาแล้ว

- cv2 OpenCV สำหรับการจัดการภาพและการแสดงผล

- matplotlib.pyplot สำหรับแสดงภาพและการแสดงผลต่าง ๆ

- numpy สำหรับการคำนวณเชิงตัวเลข

- PIL.Image สำหรับการโหลดและประมวลผลภาพ

ขั้นตอนที่ 3 การโหลดและตั้งค่าโมเดล Faster R-CNN

```
# โหลดโมเดล Faster R-CNN ที่เทรนบนชุดข้อมูล COCO
weights = FasterRCNN_ResNet50_FPN_Weights.COCO_V1
model = fasterrcnn_resnet50_fpn(weights=weights)
model.eval() # ตั้งค่าโมเดลให้อยู่ในโหมดการประเมินผล
```

ที่มา : โค้ดตัวอย่างในส่วนนี้ใช้โมเดล Faster R-CNN จาก Torchvision documentation (PyTorch, n.d.) และใช้ OpenCV สำหรับการประมวลผลและแสดงผลภาพ (OpenCV, n.d.)

รายละเอียดของขั้นตอนนี้ มีดังนี้

- โหลดโมเดล Faster R-CNN ที่มี ResNet50 เป็น backbone และใช้ Feature Pyramid Network (FPN)
- โมเดลได้รับการเทรนบนชุดข้อมูล COCO ซึ่งประกอบด้วยหมวดหมู่วัตถุทั่วไป 80 ประเภท
- `model.eval()` ทำให้มั่นใจว่าโมเดลทำงานในโหมดอนุมาน (inference mode) โดยปิดการทำงานของชั้น dropout และ batch normalization เพื่อให้การทำนายมีความสม่ำเสมอมากขึ้น

ชุดข้อมูล COCO (Common Objects in Context) เป็นชุดข้อมูลสำหรับการตรวจจับวัตถุขนาดใหญ่ที่มีหมวดหมู่วัตถุทั่วไป 80 ประเภท รวมถึงคน ยานพาหนะ สัตว์ และวัตถุในชีวิตประจำวัน การเทรนบนชุดข้อมูลที่หลากหลายนี้ช่วยให้โมเดลของเราสามารถจดจำวัตถุที่หลากหลายในสภาพแวดล้อมต่าง ๆ ได้

ขั้นตอนที่ 4 การโหลดและประมวลผลภาพก่อนนำเข้าโมเดล

ทำการเลือกรูปภาพที่ต้องการจะตรวจจับวัตถุ โดยควรเป็นภาพที่มีวัตถุอยู่ในกลุ่มประเภทที่ปรากฏในชุดข้อมูล COCO เนื่องจากโมเดล Faster R-CNN ที่โหลดมาในขั้นตอนที่ 3 เป็นโมเดลที่ผ่านการฝึกฝนล่วงหน้าบนชุดข้อมูล COCO หากภาพที่ใช้มีวัตถุอยู่ในประเภทที่โมเดลได้รับการฝึกฝนจากชุดข้อมูล COCO โมเดลจะสามารถตรวจจับวัตถุนั้นได้ อย่างไรก็ตาม หากวัตถุอยู่นอกเหนือประเภทที่โมเดลเรียนรู้มา ผลการตรวจจับอาจไม่ถูกต้องหรือไม่สามารถตรวจจับได้

ผู้เขียนได้อัปโหลดภาพ 5 (ถ่ายด้วยตนเอง) สำหรับใช้ในการตรวจจับวัตถุซึ่งเป็นภาพแมวโดยสิ่งสำคัญที่ผู้อ่านต้องระมัดระวังในการใช้งาน คือ ประเภทของไฟล์ภาพที่นำมาควรเป็น .jpg และสามารถคัดลอก path ของภาพนี้เพื่อมาใส่แทน /path/ ในโค้ดด้านล่าง

ภาพ 5

ภาพสำหรับตรวจจับวัตถุ



```
# โหลดภาพ

image_path = "/path/.jpg" # เปลี่ยนเป็นไฟล์ภาพของคุณ
image = Image.open(image_path).convert("RGB")
# กำหนดการแปลงภาพ
transform = T.Compose([
    T.ToTensor()]) # แปลงภาพเป็นเทนเซอร์ PyTorch
# ใช้การแปลงภาพ
image_tensor = transform(image).unsqueeze(0) # เพิ่มมิติของ batch
```

รายละเอียดของขั้นตอนนี้ มีดังนี้

- การโหลดภาพจากเส้นทางที่ระบุ (path)
- การแปลงเป็นรูปแบบ RGB
- การใช้การแปลงภาพเพื่อแปลงภาพให้อยู่ในรูปแบบที่โมเดลคาดหวัง
- การเพิ่มมิติของ batch เนื่องจากโมเดลคาดหวังอินพุตในรูปแบบ batch ของภาพ

การดำเนินการ `unsqueeze(0)` เพิ่มมิติเบตซ์ให้กับเทนเซอร์ของเรา เนื่องจากโมเดล PyTorch คาดหวังอินพุตในรูปแบบ `[batch_size, channels, height, width]`

ขั้นตอนที่ 5 การทำการตรวจจับวัตถุ

```
with torch.no_grad(): # ปิดการคำนวณเกรเดียนต์ (การอนุมานเร็วขึ้น)
    predictions = model(image_tensor)
```

รายละเอียดของขั้นตอนนี้ มีดังนี้

- ใช้ `torch.no_grad()` เพื่อปิดการติดตามเกรเดียนต์ ซึ่งช่วยลดการใช้หน่วยความจำและเพิ่มความเร็วในการอนุมาน
- ส่งภาพที่ผ่านการประมวลผลล่วงหน้าผ่านโมเดล
- เก็บผลการตรวจจับไว้ในตัวแปร `predictions`

เอาต์พุตของโมเดลประกอบด้วยกรอบสี่เหลี่ยม (bounding boxes) ป้ายกำกับคลาส (class labels) และค่าความเชื่อมั่น (confidence scores) สำหรับวัตถุที่ตรวจพบ

ขั้นตอนที่ 6 การประมวลผลผลลัพธ์การตรวจจับ

```
# ดึงผลการทำนาย

boxes = predictions[0]['boxes'].detach().numpy() # กรอบสี่เหลี่ยม
labels = predictions[0]['labels'].detach().numpy() # ป้ายกำกับ
คลาสน์
scores = predictions[0]['scores'].detach().numpy() # ค่าความเชื่อมั่น
```

```
# กำหนดค่าความเชื่อมั่นขั้นต่ำ
threshold = 0.8
filtered_boxes = []
filtered_labels = []
filtered_scores = []
for i in range(len(scores)):
    if scores[i] > threshold:
        filtered_boxes.append(boxes[i])
        filtered_labels.append(labels[i])
        filtered_scores.append(scores[i])
```

รายละเอียดของขั้นตอนนี้ มีดังนี้

- ดึงกรอบสี่เหลี่ยม ป้ายกำกับคลาส และค่าความเชื่อมั่นจากการทำนาย
- แปลงเทนเซอร์ PyTorch เป็นอาร์เรย์ NumPy เพื่อการประมวลผลที่ง่ายขึ้น
- กำหนดค่าความเชื่อมั่นขั้นต่ำ (0.8) เพื่กรองการตรวจจับที่มีความเชื่อมั่นต่ำออก
- วนลูปผ่านการตรวจจับทั้งหมดและเก็บเฉพาะที่มีค่าสูงกว่าค่าขั้นต่ำที่กำหนด

ค่าความเชื่อมั่นขั้นต่ำเป็นพารามิเตอร์ที่สำคัญเนื่องจากเป็นเกณฑ์ของค่าความน่าจะเป็นที่จะตรวจจับแล้วถูกต้องเมื่อค่าสูงขึ้น หมายถึง สามารถตรวจจับวัตถุโดยมีความน่าจะเป็นที่จะถูกต้องมากขึ้น แต่ส่งผลทำให้ตรวจจับวัตถุได้น้อยลง ในขณะที่ค่าต่ำลงจะให้การตรวจจับวัตถุได้จำนวนมากขึ้นแต่มีความน่าจะเป็นที่จะตอบถุณอย

ขั้นตอนที่ 7 การวาดกรอบสี่เหลี่ยม และการแสดงผลการตรวจจับวัตถุ

```
# โหลดภาพด้วย OpenCV สำหรับการแสดงผล
image_cv = cv2.imread(image_path)
image_cv = cv2.cvtColor(image_cv, cv2.COLOR_BGR2RGB) # แปลงเป็น RGB
# กำหนดป้ายกำกับ COCO (20 คลาสแรกเพื่อความเรียบง่าย)
COCO_LABELS = [
    "__background__", "person", "bicycle", "car",
    "motorcycle", "airplane",
    "bus", "train", "truck", "boat", "traffic light", "fire
    hydrant", "stop sign",
    "parking meter", "bench", "bird", "cat", "dog", "horse",
    "sheep"]
# วาดกรอบสี่เหลี่ยม
for i, box in enumerate(filtered_boxes):
    x1, y1, x2, y2 = box
```

```

    label = COCO_LABELS[filtered_labels[i]] if
filtered_labels[i] < len(COCO_LABELS) else "Object"
    score = filtered_scores[i]
    # วาดกรอบสี่เหลี่ยม
    cv2.rectangle(image_cv, (int(x1), int(y1)), (int(x2),
int(y2)), (0, 255, 0), 2)
    # แสดงป้ายกำกับและค่าความเชื่อมั่น
    text = f"{label}: {score:.2f}"
    cv2.putText(image_cv, text, (int(x1), int(y1)-10),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
# ใช้ฟังก์ชันแสดงผลของ OpenCV ใน Colab
cv2.imshow(cv2.cvtColor(image_cv, cv2.COLOR_RGB2BGR))
# หรือใช้ Matplotlib เพื่อการแสดงผล
plt.close("all")
plt.figure(figsize=(10, 6))
plt.imshow(image_cv)
plt.axis("off")
plt.show()
plt.close()

```

รายละเอียดของขั้นตอนนี้มีดังนี้

- โหลดภาพต้นฉบับด้วย OpenCV
- กำหนดป้ายกำกับชุดข้อมูล COCO เพื่อแปลความหมายดัชนีคลาส
- สำหรับการตรวจจ็ับที่ผ่านการกรอง จะปรากฏกรอบสี่เหลี่ยมรอบวัตถุที่ตรวจพบ

ป้ายกำกับชนิดของวัตถุและค่าความเชื่อมั่นว่าเป็นวัตถุใดบริเวณไหนกรอบ ตามภาพ 6

ภาพ 6

ภาพที่ได้จากการตรวจจ็ับวัตถุ (แมว) ด้วย *Faster R-CNN*



ส่วนที่ 2 การตรวจจับวัตถุในภาพนิ่งด้วย YOLO

ในส่วนที่ 2 จะเป็นการอภิปรายประกอบการโค้ดตัวอย่างที่เกี่ยวข้องกับการตรวจจับวัตถุด้วยอัลกอริทึม YOLO โดยแบ่งออกเป็น 3 หัวข้อ ได้แก่ สถาปัตยกรรมและหลักการทำงานของ YOLO ตัวอย่างโค้ดการตรวจจับวัตถุในภาพนิ่งด้วย YOLO และตัวอย่างการตรวจจับวัตถุในภาพที่มีความซับซ้อน โดยมีรายละเอียด ดังนี้

1. สถาปัตยกรรมและหลักการทำงานของ YOLO

YOLO (You Only Look Once) เป็นอัลกอริทึมการตรวจจับวัตถุที่ปฏิวัติวงการคอมพิวเตอร์วิทัศน์ด้วยแนวคิดการประมวลผลภาพเพียงครั้งเดียวเพื่อทำนายตำแหน่งและประเภทของวัตถุทั้งหมดในภาพพร้อมกัน แตกต่างจากวิธีการแบบดั้งเดิมที่ต้องทำการประมวลผลหลายครั้งหรือใช้วิธี sliding window ที่ใช้เวลานาน หลักการทำงานของ YOLO เริ่มต้นจากการแบ่งภาพอินพุตเป็นกริด $S \times S$ (โดยทั่วไปใช้ 7×7 สำหรับ YOLO เวอร์ชันแรก) แต่ละเซลล์ในกริดจะทำนายวัตถุที่มีจุดศูนย์กลางอยู่ภายในเซลล์นั้น การทำนายในแต่ละเซลล์จะประกอบด้วยสองส่วนหลัก คือ การทำนายกล่องขอบเขต (bounding box predictions) และการจำแนกประเภท (class predictions)

สำหรับกล่องขอบเขต แต่ละเซลล์จะทำนาย B กล่อง (โดยทั่วไปเป็น 2 กล่อง) โดยแต่ละกล่องจะมีค่า 5 ตัว ได้แก่ x, y, w, h และ confidence score โดยที่ค่า x และ y แทนพิกัดจุดศูนย์กลางของกล่องขอบเขตเทียบกับเซลล์ที่ทำนาย มีค่าระหว่าง 0 ถึง 1 ค่า w และ h แทนความกว้างและความสูงของกล่องเทียบกับขนาดของภาพทั้งหมด และ confidence score แทนความมั่นใจที่กล่องนั้นมีวัตถุอยู่และความแม่นยำของกล่องขอบเขต คำนวณเป็น $\text{Pr}(\text{Object}) \times \text{IoU}$ ระหว่างกล่องที่ทำนายกับ ground truth

สำหรับการจำแนกประเภท แต่ละเซลล์จะทำนายความน่าจะเป็นของแต่ละคลาส C คลาสโดยไม่ขึ้นอยู่กับการทำนายกล่องขอบเขต การทำนายสุดท้ายจะได้จากการคูณ confidence score ของกล่องกับความน่าจะเป็นของคลาส ทำให้ได้ค่า class-specific confidence scores สำหรับแต่ละกล่อง

สถาปัตยกรรมของ YOLO เวอร์ชันแรกใช้โครงข่ายประสาทเทียมแบบ CNN ที่ได้แรงบันดาลใจจาก GoogLeNet ประกอบด้วย 24 convolutional layers และ 2 fully connected layers การออกแบบเน้นความเรียบง่ายและประสิทธิภาพ โดยใช้ convolutional layers สำหรับการสกัดลักษณะเด่นและ fully connected layers สำหรับการทำนายพิกัดและความน่าจะเป็น

ภาพ 7

สถาปัตยกรรมของ YOLO เวอร์ชันแรก

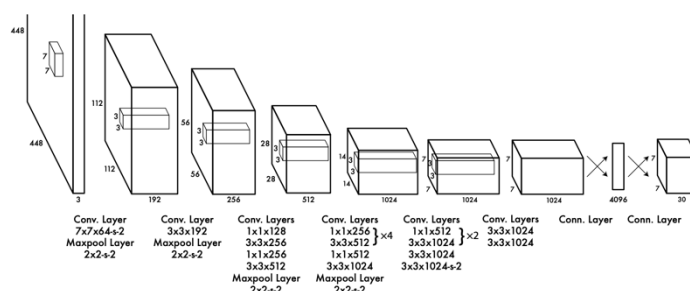


Figure 3: The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224 input image) and then double the resolution for detection.

ที่มา : Redmon et al. (2016)

ข้อดีหลักของ YOLO คือความเร็วสูงมาก เนื่องจากการประมวลผลแบบ single forward pass และสามารถทำงานแบบเรียลไทม์ได้ นอกจากนี้ YOLO ยังมองภาพในมุมมองกว้าง (global context) ทำให้เกิดข้อผิดพลาดจากการตีความผิด (background errors) น้อยกว่าวิธีการอื่น อย่างไรก็ตาม YOLO มีข้อจำกัดในการตรวจจับวัตถุขนาดเล็กและวัตถุที่อยู่ใกล้กันมาก เนื่องจากข้อจำกัดของระบบกริดและจำนวนกล่องขอบเขตที่แต่ละเซลล์สามารถทำนายได้ (Redmon et al., 2016)

2. การตรวจจับวัตถุโดยอัลกอริทึม YOLO

การตรวจจับวัตถุของ YOLO เริ่มต้นจากการปรับขนาดภาพที่นำเข้าให้มีขนาดคงที่ โดยทั่วไปเป็น 448×448 พิกเซล สำหรับ YOLO เวอร์ชันแรก การปรับขนาดนี้ช่วยให้การประมวลผลมีความสม่ำเสมอและสามารถใช้ประโยชน์จากการประมวลผลแบบ batch ได้อย่างมีประสิทธิภาพ หลังจากนั้นภาพจะถูกส่งผ่านโครงข่าย CNN เพื่อสกัดลักษณะเด่นและทำการทำนาย

ขั้นตอนการกรองเริ่มต้นด้วยการคำนวณ class-specific confidence scores สำหรับแต่ละกล่องขอบเขตโดยการคูณ confidence score ของกล่องกับ class probability จากนั้นจะกำหนดเกณฑ์ความมั่นใจขั้นต่ำ (confidence threshold) เพื่อกรองกล่องที่มีความมั่นใจต่ำออกไป โดยทั่วไปจะใช้ค่าประมาณ 0.2-0.5 ขึ้นอยู่กับการใช้งาน หลังจากการกรองด้วย confidence threshold แล้ว จะดำเนินการ Non-Maximum Suppression (NMS) เพื่อลดการตรวจจับซ้ำ (duplicate detections) ของวัตถุชิ้นเดียวกัน กระบวนการ NMS จะเรียงลำดับกล่องขอบเขตตาม confidence score จากมากไปน้อย จากนั้นจะตรวจสอบการทับซ้อนระหว่างกล่องที่มี confidence สูงสุดกับกล่องอื่น ๆ หากค่า IoU มากกว่าเกณฑ์ที่กำหนด (โดยทั่วไปเป็น 0.5) กล่องที่มี confidence ต่ำกว่าจะถูกลบออก

ข้อสำคัญของการทำงานของ YOLO คือ การจัดการกับวัตถุที่มีขนาดและอัตราส่วนต่างกัน เนื่องจากแต่ละเซลล์มีขนาดคงที่ การตรวจจับวัตถุขนาดใหญ่ที่ครอบคลุมหลายเซลล์อาจเกิดปัญหาได้ ดังนั้น YOLO จึงใช้เทคนิค anchor boxes ในเวอร์ชันที่ใหม่กว่า เพื่อช่วยในการทำนายวัตถุที่มีรูปร่างและขนาดหลากหลายมากขึ้น

การฝึกฝน YOLO ใช้ loss function ที่ซับซ้อนซึ่งรวม localization loss confidence loss และ classification loss เข้าด้วยกัน โดยมีการปรับน้ำหนักที่แตกต่างกันสำหรับแต่ละส่วน เพื่อให้การเรียนรู้มีความสมดุลและมีประสิทธิภาพสูงสุด การใช้เทคนิค data augmentation เช่น random scaling translation และ HSV adjustment ช่วยเพิ่มความหลากหลายของข้อมูลฝึกฝน และลดการ overfitting การพัฒนาต่อมาของ YOLO รุ่นใหม่ๆ เช่น YOLOv2, YOLOv3 และ YOLOv4 ได้นำเสนอการปรับปรุงหลายประการ รวมถึงการใช้ anchor boxes feature pyramid networks และเทคนิคการฝึกฝนที่ทันสมัยกว่า ทำให้ประสิทธิภาพและความแม่นยำดีขึ้นอย่างมีนัยสำคัญ (Bochkovskiy et al., 2020; Redmon & Farhadi, 2017; Redmon & Farhadi, 2018)

3. ตัวอย่างโคดการตรวจจับวัตถุในภาพนิ่งด้วย YOLO

การตรวจจับวัตถุ (Object Detection) เป็นเทคโนโลยีที่สำคัญในด้านคอมพิวเตอร์วิทัศน์ ซึ่ง YOLOv5 (You Only Look Once version 5) เป็นหนึ่งในอัลกอริทึมที่ได้รับความนิยมสูง

เนื่องจากมีความเร็วและความแม่นยำที่ดี ในบทความนี้จะแสดงการใช้งาน YOLOv5 สำหรับตรวจจับวัตถุในรูปภาพเดี่ยว

ขั้นตอนที่ 1 ติดตั้ง Dependencies

การติดตั้งไลบรารีที่จำเป็น

```
!pip install torch torchvision opencv-python matplotlib
```

รายละเอียดของขั้นตอนนี้ มีดังนี้

- torch และ torchvision เฟรมเวิร์ก PyTorch สำหรับ Deep Learning
- opencv-python ไลบรารีสำหรับการประมวลผลภาพ
- matplotlib ไลบรารีสำหรับการแสดงผลกราฟและภาพ

ขั้นตอนที่ 2 การนำเข้าไลบรารีและปิดคำเตือนที่ไม่จำเป็น

```
import torch
import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab.patches import cv2_imshow
# แสดงภาพใน Jupyter notebook แทนการเปิด window ใหม่
from IPython.display import display
from PIL import Image
import warnings
warnings.simplefilter("ignore", category=FutureWarning)
```

ไลบรารีเหล่านี้มีหน้าที่แตกต่างกันไป โดยมีรายละเอียดดังนี้

- torch สำหรับโหลดและใช้งานโมเดล YOLO
- cv2 สำหรับการประมวลผลภาพ อ่านหรือเขียนไฟล์ รวมทั้งวาดกล่องขอบเขต
- numpy: สำหรับการจัดการข้อมูลแบบ array
- matplotlib.pyplot: สำหรับการแสดงผลภาพ
- warnings.simplefilter(): ปิดการแสดงคำเตือนที่ไม่จำเป็น

ขั้นตอนที่ 3 โหลดโมเดล YOLOv5 ที่ฝึกฝนแล้ว

```
model = torch.hub.load('ultralytics/yolov5', 'yolov5s',
pretrained=True, trust_repo=True)
model.eval()
```

ที่มา : โค้ดตัวอย่างนี้อ้างอิงแนวทางการโหลด YOLOv5 ผ่าน PyTorch Hub ของ Ultralytics (Ultralytics, n.d.)

รายละเอียดของขั้นตอนนี้ มีดังนี้

- torch.hub.load() โหลดโมเดลจาก PyTorch Hub
- 'ultralytics/yolov5' repository ของ YOLOv5 บน GitHub
- 'yolov5s' โมเดลขนาดเล็กที่เร็วและเหมาะสำหรับการทดสอบ
- pretrained=True ใช้น้ำหนักที่ฝึกฝนบนชุดข้อมูล COCO แล้ว
- trust_repo=True อนุญาตให้ดาวน์โหลดจาก repository
- model.eval() ตั้งโมเดลเป็นโหมด evaluation

ขั้นตอนที่ 4 โหลดและประมวลผลภาพ

```
image_path = "/content/3EE13CC4-3C57-4878-9B1D-1F813E958938.jpg"
image_cv = cv2.imread(image_path)
image_rgb = cv2.cvtColor(image_cv, cv2.COLOR_BGR2RGB) # แปลงเป็น RGB
# ตรวจสอบให้ OpenCV สามารถแก้ไขภาพได้
image_rgb = np.ascontiguousarray(image_rgb)
```

รายละเอียดของขั้นตอนนี้ มีดังนี้

- cv2.imread() สำหรับอ่านไฟล์ภาพในรูปแบบ BGR (Blue-Green-Red)
- cv2.cvtColor() ใช้แปลงจาก BGR เป็น RGB เพราะ matplotlib ใช้ RGB
- np.ascontiguousarray() สำหรับจัดการหน่วยความจำให้ข้อมูลอยู่ติดกันเพื่อประสิทธิภาพ

ขั้นตอนที่ 5 ทำการตรวจจับวัตถุ

```
# แปลงภาพเป็น tensor และทำ normalization
results = model(image_rgb)
# แยกผลการทำนาย
detections = results.xyxy[0].cpu().numpy() # Bounding boxes, confidence, class
```

รายละเอียดของขั้นตอนนี้ มีดังนี้

- model(image_rgb) สำหรับส่งภาพเข้าโมเดลเพื่อตรวจจับวัตถุ
- results.xyxy[0] ได้ผลลัพธ์ในรูปแบบ [x1, y1, x2, y2, confidence, class]
- .cpu().numpy() แปลงจาก GPU tensor เป็น numpy array

ขั้นตอนที่ 6 ตรวจสอบผลการตรวจจับ

```
print(f"Number of detections: {len(detections)}")
print(detections)
```

ผลลัพธ์ที่ได้ ดังนี้

```
Number of detections: 1
[[ 670.44      1048.9      1943.5      1934.8      0.61422      15]]
```

คำอธิบายผลลัพธ์ที่ได้จากขั้นตอนนี้

- ตรวจพบวัตถุ 1 ชิ้น
- พิกัด bounding box: x1=670.44, y1=1048.9, x2=1943.5, y2=1934.8
- ความเชื่อมั่น: 0.61422 (61.4%)
- Class ID: 15 (ในชุดข้อมูล COCO)

ขั้นตอนที่ 7 วิเคราะห์รายละเอียดการตรวจจับ

```
# แสดงขนาดภาพและรายละเอียดการตรวจจับ

print(f"Image shape: {image_rgb.shape}")

for detection in detections:

    x1, y1, x2, y2, confidence, class_id = detection

    print(f"Box: ({x1}, {y1}), ({x2}, {y2}), Conf: {confidence:.2f}, Class: {class_id}")
```

รายละเอียดของขั้นตอนนี้มีดังนี้

- ภาพมีขนาด 4032×3024 pixels (ความละเอียดสูง)
- วัตถุที่ตรวจพบอยู่ในพิกัด (670, 1048) ถึง (1943, 1934)
- Class ID 15 สอดคล้องกับคลาส "cat" ตามลำดับคลาสของโมเดล YOLOv5 ที่ใช้งาน

ขั้นตอนที่ 8 การแปลง class ID เป็นชื่อคลาสและวาดกล่องขอบเขต

```
# กำหนด COCO labels (20 คลาสแรก)

COCO_LABELS = [

    "__background__", "person", "bicycle", "car",
    "motorcycle", "airplane",

    "bus", "train", "truck", "boat", "traffic light", "fire
    hydrant", "stop sign",

    "parking meter", "bench", "bird", "cat", "dog", "horse",
    "sheep"]

# รับขนาดภาพ

h, w, _ = image_rgb.shape

# วาดกล่องขอบเขต

for detection in detections:

    x1, y1, x2, y2, confidence, class_id = detection
```

```

class_id = int(class_id)

# กรองเฉพาะ 20 คลาสแรก

if confidence > 0.3 and 0 <= class_id < len(COCO_LABELS):
    label = f"{model.names[int(class_id)]}: {confidence:.2f}"

    # แปลงพิกัดเป็นจำนวนเต็ม

    x1, y1, x2, y2 = map(int, [x1, y1, x2, y2])

    x1, y1, x2, y2 = max(0, x1), max(0, y1), min(w-1, x2),
min(h-1, y2)

    # วาดกล่องขอบเขต

    cv2.rectangle(image_rgb, (x1, y1), (x2, y2), (0, 255, 0), 3)

    # วาด label

    cv2.putText(image_rgb, label, (x1, max(y1 - 10, 10)),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

```

รายละเอียดของขั้นตอนนี้มีดังนี้

- COCO_LABELS รายชื่อคลาสในชุดข้อมูล COCO (person, car, dog เป็นต้น)
- กรองเฉพาะวัตถุที่มี confidence > 0.3
- cv2.rectangle() วาดสี่เหลี่ยมสีเขียว ความหนา 3 pixels
- cv2.putText() เขียนข้อความ label บนกล่องขอบเขต
- max(), min() ตรวจสอบไม่ให้พิกัดเกินขอบเขตภาพ

ขั้นตอนที่ 9 บันทึกและแสดงผลลัพธ์

```

cv2.imwrite("output.jpg", cv2.cvtColor(image_rgb,
cv2.COLOR_RGB2BGR))

print("Saved output image as output.jpg")

# ใช้ OpenCV's Colab display function

cv2_imshow(cv2.cvtColor(image_rgb, cv2.COLOR_RGB2BGR))

# แสดงผลด้วย matplotlib

plt.close('all')

plt.figure(figsize=(10, 6))

plt.imshow(image_rgb)

plt.axis('off')

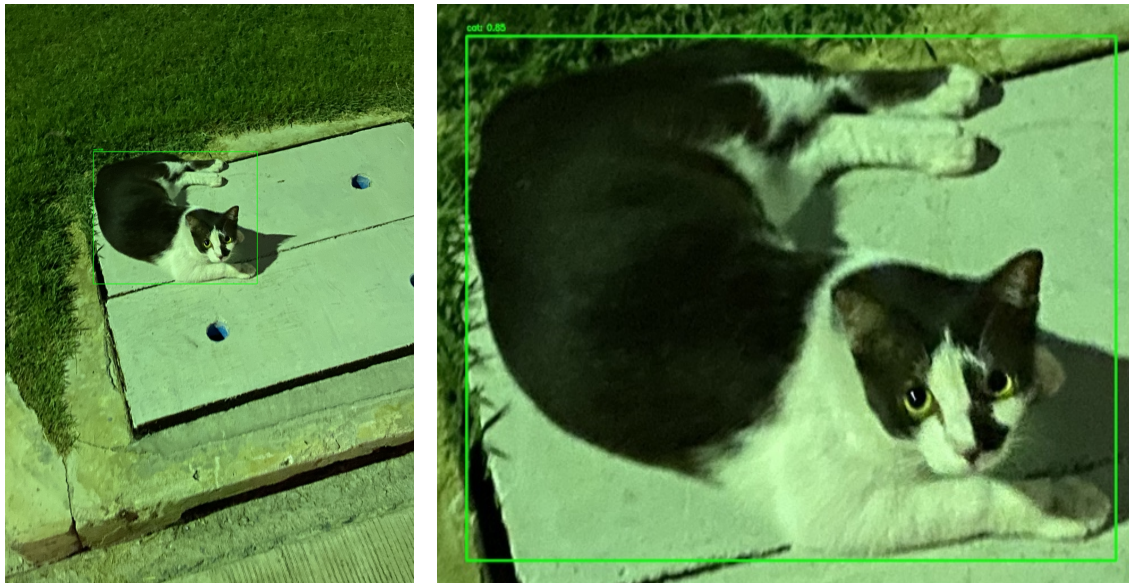
plt.show()

plt.close()

```

ภาพ 8

ภาพที่ได้จากการตรวจจับวัตถุ (แมว) ด้วย YOLO



4. ตัวอย่างการตรวจจับวัตถุในภาพที่มีความซับซ้อน

- การตรวจจับวัตถุในภาพที่มีความแออัด
- การตรวจจับวัตถุขนาดเล็ก
- การตรวจจับในสภาพแสงที่ไม่เอื้ออำนวย

ส่วนที่ 3 การตรวจจับวัตถุในวิดีโอด้วย YOLO

ในส่วนที่ 3 จะเป็นการอภิปรายประกอบการยกตัวอย่างกรณีศึกษาในการตรวจจับวัตถุในวิดีโอด้วย YOLO โดยแบ่งออกเป็น 4 หัวข้อได้แก่ การประยุกต์ใช้ YOLO กับวิดีโอ เทคนิคการเพิ่มประสิทธิภาพสำหรับการตรวจจับแบบเรียลไทม์ การติดตามวัตถุรวมกับการตรวจจับ และตัวอย่างโค้ดสำหรับการใช้งานในสถานการณ์จริง มีรายละเอียด ดังนี้

1. การประยุกต์ใช้ YOLO กับวิดีโอ

การนำ YOLO มาประยุกต์ใช้กับวิดีโอเป็นการขยายขอบเขตจากการตรวจจับในภาพนิ่งสู่การประมวลผลต่อเนื่องในมิติเวลา ซึ่งมีความท้าทายและความซับซ้อนเพิ่มขึ้นอย่างมาก การตรวจจับวัตถุในวิดีโอไม่เพียงแต่ต้องการความแม่นยำในการระบุวัตถุ แต่ยังต้องคำนึงถึงความเร็วในการประมวลผลเพื่อให้สามารถทำงานแบบเรียลไทม์ได้ ความแตกต่างระหว่างการตรวจจับในภาพนิ่งและวิดีโออยู่ที่ความต่อเนื่องของข้อมูล ในวิดีโอเฟรมที่อยู่ติดกันมักจะมีเนื้อหาคล้ายคลึงกัน ซึ่งสามารถนำมาใช้ประโยชน์ในการเพิ่มประสิทธิภาพและความเสถียรของการตรวจจับ การเปลี่ยนแปลงของตำแหน่งวัตถุระหว่างเฟรมมักจะเป็นไปอย่างค่อยเป็นค่อยไป ทำให้สามารถคาดการณ์ตำแหน่งของวัตถุในเฟรมถัดไปได้ระดับหนึ่ง

วิธีการพื้นฐานในการประยุกต์ YOLO กับวิดีโอ คือ การนำโมเดลที่ฝึกฝนแล้วมาประมวลผลแต่ละเฟรมแยกต่างหาก โดยถือว่าแต่ละเฟรมเป็นภาพนิ่งอิสระ วิธีนี้เรียกว่า "frame-

by-frame detection" ซึ่งสามารถดำเนินการได้ง่ายและให้ผลลัพธ์ที่ดีในหลายกรณี อย่างไรก็ตามวิธีนี้อาจทำให้เกิดปัญหาการกระพริบ (flickering) ของกล่องขอบเขตและการขาดความต่อเนื่องในการตรวจจับ

ข้อดีของการใช้ YOLO กับวิดิทัศน์ประกอบด้วยความเร็วสูงที่เหมาะสมกับการประมวลผลเรียลไทม์ ความสามารถในการตรวจจับวัตถุหลายชนิดพร้อมกัน และการใช้ทรัพยากรที่มีประสิทธิภาพ ในทางตรงข้าม ข้อจำกัดสำคัญได้แก่การขาดความต่อเนื่องในการตรวจจับระหว่างเฟรม ความยากในการจัดการกับวัตถุที่เคลื่อนไหวเร็ว และปัญหาการตรวจจับวัตถุที่ถูกบดบังชั่วคราว (temporary occlusion)

การใช้งานจริงของ YOLO ในวิดิทัศน์พบในหลายสาขา เช่น ระบบรักษาความปลอดภัย การควบคุมการจราจร การวิเคราะห์ความแออัดของพื้นที่ และการติดตามวัตถุในกีฬา แต่การใช้งานมีข้อกำหนดเฉพาะเกี่ยวกับความเร็ว ความแม่นยำ และประเภทของวัตถุที่ต้องการตรวจจับ (Wang et al., 2020; Wojke et al., 2017)

2. เทคนิคการเพิ่มประสิทธิภาพสำหรับการตรวจจับแบบเรียลไทม์

การเพิ่มประสิทธิภาพสำหรับการตรวจจับแบบเรียลไทม์เป็นปัจจัยสำคัญที่กำหนดความสำเร็จของการนำ YOLO ไปใช้งานจริง โดยเฉพาะในแอปพลิเคชันที่ต้องการการตอบสนองทันที เทคนิคเหล่านี้สามารถแบ่งออกเป็นหลายประเภทตามจุดที่ใช้ในขั้นตอนการประมวลผล (Howard et al., 2017; Sandler et al., 2018) โดยแต่ละเทคนิคมีรายละเอียดดังนี้

การปรับปรุงโมเดลและสถาปัตยกรรม เป็นแนวทางหลักในการเพิ่มความเร็ว การใช้ lightweight backbone networks เช่น MobileNet, EfficientNet หรือ GhostNet แทนโครงข่ายขนาดใหญ่อย่าง ResNet สามารถลดจำนวนพารามิเตอร์และการคำนวณได้อย่างมาก หรือการใช้ depthwise separable convolutions แทน standard convolutions ช่วยลดจำนวน FLOPs (Floating Point Operations) โดยรักษาประสิทธิภาพการตรวจจับไว้ในระดับที่ยอมรับได้

เทคนิค Model Quantization เป็นวิธีการลดขนาดโมเดลและเพิ่มความเร็วโดยการลดความละเอียดของน้ำหนัก (weights) จาก 32-bit floating point เป็น 8-bit integer หรือแม้แต่ 1-bit binary การ quantization สามารถทำได้ทั้งหลังการฝึกฝน และระหว่างการฝึกฝน ซึ่งวิธีหลังมักให้ผลลัพธ์ที่ดีกว่า

การใช้ Model Pruning เป็นเทคนิคการลดขนาดโมเดลโดยการตัดการเชื่อมต่อหรือ neurons ที่มีผลกระทบน้อยต่อประสิทธิภาพออกไป สามารถแบ่งเป็น structured pruning ที่ตัดทั้ง channel หรือ layer และ unstructured pruning ที่ตัดน้ำหนักแต่ละค่าที่มีค่าน้อย การ pruning สามารถลดขนาดโมเดลได้ถึง 90% โดยคงประสิทธิภาพไว้ในระดับที่ยอมรับได้

เทคนิค Knowledge Distillation ใช้โมเดลขนาดใหญ่ที่มีประสิทธิภาพสูง (teacher model) เพื่อสอนโมเดลขนาดเล็ก (student model) ให้มีประสิทธิภาพใกล้เคียงกัน โดยการให้ student model เรียนรู้จากทั้ง ground truth labels และ soft predictions จาก teacher model วิธีนี้ช่วยให้ได้โมเดลขนาดเล็กที่มีประสิทธิภาพสูงกว่าการฝึกฝนแบบปกติ

เทคนิค Multi-threading และ Asynchronous Processing ช่วยให้สามารถประมวลผลหลายเฟรมพร้อมกันหรือทำงานหลายขั้นตอนแบบขนาน การใช้ frame skipping ในกรณี

ประสิทธิภาพสำคัญกว่าความละเอียดของการตรวจจับและการใช้ adaptive resolution ที่ปรับขนาดภาพตามความซับซ้อนของเนื้อหา

การจัดการหน่วยความจำ เป็นปัจจัยสำคัญในระบบเรียลไทม์ การใช้ memory pooling เพื่อลดการ allocation/deallocation และการใช้ in-place operations ช่วยลดการใช้หน่วยความจำและเพิ่มความเร็ว การใช้ mixed precision training และ inference ยังช่วยลดการใช้หน่วยความจำและเพิ่มความเร็วโดยไม่กระทบต่อประสิทธิภาพมากนัก

3. การติดตามวัตถุรวมกับการตรวจจับ

การรวมการติดตามวัตถุ (Object Tracking) เข้ากับการตรวจจับเป็นขั้นตอนที่สำคัญในการพัฒนาระบบวิเคราะห์วิดีโอที่สมบูรณ์ การตรวจจับเพียงอย่างเดียวสามารถระบุว่ามีวัตถุอะไรอยู่ในแต่ละเฟรม แต่ไม่สามารถเชื่อมโยงว่าวัตถุชิ้นเดียวกันในเฟรมที่ต่างกันเป็นวัตถุเดียวกัน การติดตามช่วยให้ระบบสามารถติดตามเส้นทางการเคลื่อนไหวของวัตถุแต่ละชิ้นได้อย่างต่อเนื่อง แนวทาง Detection-based Tracking เป็นวิธีการหลักที่นิยมใช้กับ YOLO โดยใช้ผลลัพธ์จากการตรวจจับในแต่ละเฟรมเป็นพื้นฐานสำหรับการติดตาม กระบวนการจะเริ่มจากการตรวจจับวัตถุในเฟรมปัจจุบัน จากนั้นจึงเชื่อมโยงกับวัตถุที่ติดตามอยู่จากเฟรมก่อนหน้า การเชื่อมโยงนี้อาศัยหลายปัจจัย เช่น ตำแหน่ง ขนาด รูปร่าง และลักษณะเด่นของวัตถุ

Simple Online and Realtime Tracking (SORT) เป็นอัลกอริทึมติดตามที่เรียบง่ายแต่มีประสิทธิภาพสูง ใช้ Kalman filter เพื่อคาดการณ์ตำแหน่งของวัตถุในเฟรมถัดไป และใช้ Hungarian algorithm สำหรับการจับคู่ระหว่างการตรวจจับใหม่กับ tracks ที่มีอยู่ SORT มีความเร็วสูงและเหมาะสมกับการใช้งานเรียลไทม์ แต่มีข้อจำกัดในการจัดการกับการบดบังและการเปลี่ยนรูปร่างของวัตถุ

DeepSORT เป็นการพัฒนาดียอดจาก SORT โดยเพิ่ม deep association metric ที่ใช้ deep learning เพื่อสกัดคุณลักษณะเด่นของวัตถุ (appearance features) การเพิ่มข้อมูลลักษณะภายนอกนี้ช่วยปรับปรุงประสิทธิภาพการติดตามในสถานการณ์ที่ซับซ้อน เช่น การบดบังชั่วคราว หรือการเปลี่ยนแปลงของท่าทางวัตถุ DeepSORT ใช้ CNN ที่ฝึกฝนเพื่อสกัดลักษณะเด่นจากภาพของวัตถุ และใช้ cosine distance เพื่อเปรียบเทียบความคล้ายคลึง

ความท้าทายในการติดตาม ประกอบด้วย การจัดการกับ occlusions ที่วัตถุถูกบดบังโดยวัตถุอื่น การเปลี่ยนแปลงของลักษณะภายนอกของวัตถุเนื่องจากการเปลี่ยนมุมมองหรือแสง การเข้าและออกของวัตถุจากขอบเขตของภาพ และการจัดการกับวัตถุที่มีการเคลื่อนไหวแบบไม่สม่ำเสมอหรือเปลี่ยนทิศทางอย่างกะทันหัน

การประยุกต์ใช้ในโลกจริงของระบบติดตามรวมกับการตรวจจับพบในระบบรักษาความปลอดภัย การวิเคราะห์การจราจร การนับจำนวนคน การวิเคราะห์พฤติกรรมลูกค้าในร้านค้า และการวิเคราะห์การเคลื่อนไหวในกีฬา แต่ละการใช้งานต้องการการปรับแต่งพารามิเตอร์และอาจต้องการเทคนิคเฉพาะสำหรับสภาพแวดล้อมนั้นๆ (Bewley et al., 2016; Wojke et al., 2017; Zhang et al., 2022)

4. ตัวอย่างโค้ดสำหรับการใช้งานในสถานการณ์จริง

การนำ YOLO และ DeepSORT มาใช้ในการตรวจจับและติดตามวัตถุในวิดีโอเป็นกระบวนการที่ต้องอาศัยความเข้าใจรายละเอียดของแต่ละขั้นตอน ตัวอย่างต่อไปนี้อธิบายการพัฒนากระบวนการตรวจจับและติดตามรถยนต์ในวิดีโอที่คนจราจรโดยใช้ YOLOv5 และ DeepSORT ทั้งนี้ ตัวอย่างนี้ใช้ไลบรารี deep_sort_realtime ซึ่งเป็น implementation สำหรับการใช้งาน DeepSORT แบบเรียลไทม์ (levan92, n.d.)

ขั้นตอนที่ 1 การติดตั้งและนำเข้าไลบรารี

```
!pip install ultralytics deep_sort_realtime opencv-python
import cv2
import torch
from google.colab.patches import cv2_imshow
# เพื่อให้สามารถแสดงภาพใน Jupyter notebook แทนการเปิด window ใหม่
from deep_sort_realtime.deepsort_tracker import DeepSort
import warnings
warnings.simplefilter(action="ignore", category=FutureWarning)
```

รายละเอียดของขั้นตอนนี้ มีดังนี้

ขั้นตอนแรกเป็นการเตรียมสภาพแวดล้อมสำหรับการทำงาน โดยใช้ pip install เพื่อติดตั้งไลบรารีหลัก ได้แก่ ultralytics สำหรับ YOLO deep_sort_realtime สำหรับการติดตาม และ opencv-python สำหรับการประมวลผลวิดีโอ การนำเข้าไลบรารีจะช่วยให้เราสามารถใช้งานฟังก์ชันต่าง ๆ ได้ โดย warnings.simplefilter ใช้เพื่อซ่อนคำเตือนที่ไม่จำเป็น

ขั้นตอนที่ 2 การโหลดโมเดลและเตรียมการติดตาม

```
# Load YOLO model
model = torch.hub.load("ultralytics/yolov5", "yolov5s",
pretrained=True)
deep_sort = DeepSort(max_age=5)
```

รายละเอียดของขั้นตอนนี้ มีดังนี้

การโหลดโมเดล YOLOv5 ทำผ่าน torch.hub.load() โดยใช้โมเดล "yolov5s" ซึ่งเป็นเวอร์ชันที่มีขนาดเล็กและเร็ว เหมาะสำหรับการใช้งานเรียลไทม์ พารามิเตอร์ pretrained=True ทำให้โมเดลถูกโหลดพร้อมกับน้ำหนักที่ฝึกฝนแล้วบนชุดข้อมูล COCO DeepSort จะถูกสร้างด้วย max_age=5 หมายความว่าหากวัตถุไม่ถูกตรวจจับเป็นเวลา 5 เฟรมติดต่อกัน การติดตามจะหยุดลง

ขั้นตอนที่ 3 การเปิดและตั้งค่าวิดีโอ

```
# Load Video

video_path = "/content/vecteezy_car-and-truck-traffic-on-the-
highway-in-europe-poland_7957364.mp4"

cap = cv2.VideoCapture(video_path)

if not cap.isOpened():
    print("Error: Video file could not be opened.")
    exit()

frame_limit = 20 # Modify as needed
frame_count = 0
```

รายละเอียดของขั้นตอนนี้ มีดังนี้

การใช้ `cv2.VideoCapture()` เพื่อเปิดไฟล์วิดีโอ จากนั้นตรวจสอบว่าไฟล์สามารถเปิดได้หรือไม่ด้วย `cap.isOpened()` หากไม่สามารถเปิดได้ โปรแกรมจะแสดงข้อความแสดงข้อผิดพลาดและหยุดทำงาน `frame_limit` ถูกตั้งไว้ที่ 20 เพื่อจำกัดจำนวนเฟรมที่จะประมวลผลซึ่งช่วยประหยัดเวลาในการทดสอบ

ขั้นตอนที่ 4 การประมวลผลวิดีโอแบบวนซ้ำ

```
while cap.isOpened():
    if frame_count >= frame_limit:
        print("Frame limit reached.")
        break

    ret, frame = cap.read()
    if not ret:
        print("End of video or error reading frame.")
        break
```

รายละเอียดของขั้นตอนนี้ มีดังนี้

วนลูป `while cap.isOpened()` จะทำงานต่อไปจนกว่าวิดีโอจะจบหรือเกิดข้อผิดพลาด `cap.read()` อ่านเฟรมถัดไปและคืนค่า `ret` (boolean บอกรัฐความสำเร็จ) และ `frame` (ข้อมูลภาพ) หาก `ret` เป็น False แสดงว่าวิดีโอจบแล้วหรือเกิดข้อผิดพลาด การตรวจสอบ `frame_limit` ช่วยควบคุมจำนวนเฟรมที่ประมวลผล

ขั้นตอนที่ 5 การตรวจสอบความถูกต้องของเฟรม

```
# Ensure the frame is properly loaded
if frame is None or frame.size == 0:
    print(f"Frame {frame_count}: No valid frame detected.")
    continue
```

รายละเอียดของขั้นตอนนี้ มีดังนี้

การตรวจสอบว่าเฟรมที่อ่านมา มีข้อมูลจริงหรือไม่ โดยตรวจสอบว่า frame ไม่เป็น None และมีขนาดมากกว่า 0 หากเฟรมไม่ถูกต้องจะข้ามไปยังเฟรมถัดไปด้วยคำสั่ง continue

ขั้นตอนที่ 6 การตรวจจับวัตถุด้วย YOLO

```
# Run YOLO detection
results = model(frame)
processed_detections = []
if results and len(results.xyxy[0]) > 0: # Ensure detections exist
    for i in range(len(results.xyxy[0])):
        x1, y1, x2, y2 = results.xyxy[0][i][:4].tolist() # Bounding box
        score = float(results.xyxy[0][i][4]) # Confidence score
        class_id = int(results.xyxy[0][i][5]) # Class ID
```

รายละเอียดของขั้นตอนนี้ มีดังนี้

การเรียกใช้โมเดล YOLO ด้วย model(frame) จะคืนค่าผลลัพธ์การตรวจจับ results.xyxy[0] ประกอบด้วยข้อมูลของแต่ละวัตถุที่ตรวจพบ สำหรับแต่ละการตรวจจับ จะมีข้อมูล 6 ค่า คือ พิกัด x1, y1, x2, y2 ของกล่องขอบเขต คะแนนความเชื่อมั่น (score), และ ID ของคลาส (class_id)

ขั้นตอนที่ 7 การกรองและประมวลผลการตรวจจับ

```
# Debugging: Print detected class names
detected_class = model.names[class_id] if class_id in model.names else "Unknown"
print(f"Frame {frame_count}: Detected {detected_class} with confidence {score:.2f}")

# Only keep cars (Class ID for car in COCO dataset is 2)
if class_id == 2 and score > 0.3:
    processed_detections.append([(x1, y1, x2 - x1, y2 - y1), score, class_id])
```

รายละเอียดของขั้นตอนนี้ มีดังนี้

การแปลง class_id เป็นชื่อคลาสที่เข้าใจได้ด้วย model.names[class_id] เพื่อการ debugging จากนั้นกรองเฉพาะรถยนต์ (class_id == 2 ในชุดข้อมูล COCO) ที่มีคะแนนความเชื่อมั่นมากกว่า 0.3 การแปลงรูปแบบพิกัดจาก (x1, y1, x2, y2) เป็น (x, y, width, height) เพื่อให้เข้ากับข้อกำหนดของ DeepSORT

ขั้นตอนที่ 8 การอัปเดตการติดตาม

```
if not processed_detections:
    print(f"Frame {frame_count}: No cars detected.")
    continue
# Update tracking
tracked_objects =
deep_sort.update_tracks(processed_detections, frame=frame)
```

รายละเอียดของขั้นตอนนี้ มีดังนี้

หากไม่มีรถยนต์ถูกตรวจจับในเฟรมนั้น จะข้ามไปยังเฟรมถัดไป หากมีการตรวจจับ จะส่งข้อมูลไปยัง DeepSORT ด้วย update_tracks() ซึ่งจะคืนค่ารายการของวัตถุที่ถูกติดตาม แต่ละวัตถุจะมี track_id ที่ไม่ซ้ำกัน

ขั้นตอนที่ 9 การวาดผลลัพธ์และแสดงผล

```
# Draw bounding boxes and IDs
for obj in tracked_objects:
    x1, y1, x2, y2 = obj.to_tlbr()
    track_id = obj.track_id
    cv2.rectangle(frame, (int(x1), int(y1)), (int(x2),
int(y2)), (0, 255, 0), 2)
    cv2.putText(frame, f'ID {track_id}', (int(x1), int(y1) -
10),
    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

# Display frame in Colab
cv2.imshow(frame)
frame_count += 1
```

รายละเอียดของขั้นตอนนี้ มีดังนี้

สำหรับแต่ละวัตถุที่ถูกติดตาม จะใช้ `obj.to_tlbr()` เพื่อได้พิกัดของกล่องขอบเขตในรูปแบบ (top-left, bottom-right) จากนั้นวาดรูปสี่เหลี่ยมด้วย `cv2.rectangle()` และใส่ข้อความ `track_id` ด้วย `cv2.putText()` การใช้ `cv2.imshow()` แทน `cv2.imshow()` เป็นเพราะการทำงานใน Google Colab

```
ขั้นตอนที่ 10 การปิดและทำความสะอาด
cap.release()

cv2.destroyAllWindows()

print("Processing completed.")
```

ผลลัพธ์ที่ได้

จากการรันโค้ดจะได้ตัวอย่างผลลัพธ์ในเฟรมที่ 1 และ 4 ดังนี้

ภาพ 9

ภาพผลลัพธ์ในเฟรมที่ 1 ของการนำ YOLO และ DeepSORT มาใช้งานจริงในการตรวจจับรถยนต์



ที่มา: ประมวลผลจากวิดีโอที่ stock video ของ Sokolan (n.d.) บน Vecteezy โดยใช้เพื่อการสาธิตเชิงวิชาการตามเงื่อนไขของแหล่งที่มา

รายละเอียดของภาพ 9 ที่ได้จากการตรวจจับรถยนต์เป็นดังนี้

```
Frame 1: Detected car with confidence 0.91
Frame 1: Detected truck with confidence 0.89
Frame 1: Detected car with confidence 0.88
Frame 1: Detected truck with confidence 0.87
Frame 1: Detected car with confidence 0.76
Frame 1: Detected car with confidence 0.75
Frame 1: Detected truck with confidence 0.65
```

Frame 1: Detected car with confidence 0.64
Frame 1: Detected car with confidence 0.48
Frame 1: Detected truck with confidence 0.46
Frame 1: Detected truck with confidence 0.35
Frame 1: Detected car with confidence 0.31
Frame 1: Detected stop sign with confidence 0.31
Frame 1: Detected truck with confidence 0.31

ภาพ 10

ภาพผลลัพธ์ในเฟรมที่ 4 ของการนำ YOLO และ DeepSORT มาใช้งานจริงในการตรวจจับรถยนต์



ที่มา: ประมวลผลจากวิดีโอ stock video ของ Sokolan (n.d.) บน Vecteezy โดยใช้เพื่อการสาธิตเชิงวิชาการตามเงื่อนไขของแหล่งที่มา

รายละเอียดของภาพ 10 ที่ได้จากการตรวจจับรถยนต์เป็นดังนี้

Frame 4: Detected car with confidence 0.91
Frame 4: Detected truck with confidence 0.89
Frame 4: Detected car with confidence 0.88
Frame 4: Detected truck with confidence 0.87
Frame 4: Detected car with confidence 0.86
Frame 4: Detected car with confidence 0.73
Frame 4: Detected truck with confidence 0.69
Frame 4: Detected truck with confidence 0.58
Frame 4: Detected car with confidence 0.47
Frame 4: Detected car with confidence 0.44
Frame 4: Detected truck with confidence 0.43
Frame 4: Detected car with confidence 0.39

สรุป

ผลการสาธิตแสดงให้เห็นว่า YOLOv5 ร่วมกับ DeepSORT สามารถใช้เป็นตัวช่วยอย่างเบื้องต้นสำหรับการตรวจจับและติดตามรถยนต์ในวิดีโอได้ อย่างไรก็ตาม บทความนี้ไม่ได้มุ่งประเมินประสิทธิภาพของระบบอย่างเป็นทางการ จึงควรตีความผลลัพธ์ในฐานะตัวอย่างการประยุกต์ใช้เชิงปฏิบัติ

บทความนี้เสนอแนวคิดหลักและแนวทางปฏิบัติในการตรวจจับวัตถุโดยใช้เทคโนโลยีคอมพิวเตอร์วิทัศน์และการเรียนรู้เชิงลึกผ่านตัวอย่างการใช้งานโมเดล Faster R-CNN และ YOLO ในบริบทของภาพนิ่งและวิดีโอ จุดเน้นคือการสร้างความเข้าใจเกี่ยวกับพื้นฐาน แนวคิดหลัก โครงสร้างของโมเดล และแนวทางการนำไปใช้จริง โดยเฉพาะการใช้ YOLOv5 ร่วมกับเทคนิคการติดตามวัตถุ เช่น DeepSORT เพื่อเพิ่มความต่อเนื่องและความแม่นยำในการวิเคราะห์วัตถุในวิดีโอแบบเรียลไทม์

ผู้เขียนหวังว่าบทความนี้จะช่วยให้ผู้อ่านสามารถ 1) แยกความแตกต่างระหว่างการจำแนกภาพและการตรวจจับวัตถุ 2) เข้าใจองค์ประกอบหลักของระบบตรวจจับวัตถุ รวมถึงแนวทางการประเมินประสิทธิภาพของโมเดล 3) มองเห็นความเปลี่ยนแปลงของโมเดลตรวจจับวัตถุจากยุคก่อน deep learning มาจนถึง one-stage และ two-stage detectors ที่ใช้งานกันในปัจจุบัน 4) ประยุกต์ใช้โค้ดเบื้องต้นเพื่อนำไปปรับใช้กับข้อมูลจริง ทั้งในภาพนิ่งและวิดีโอ 5) เลือกใช้เทคนิคปรับแต่ง เช่น pruning, quantization หรือการรวมโมเดลตรวจจับและติดตาม ให้เหมาะสมกับข้อจำกัดของอุปกรณ์หรือข้อกำหนดของงาน

แม้เทคโนโลยีจะก้าวหน้าขึ้นมากในช่วงไม่กี่ปีที่ผ่านมา แต่งานตรวจจับวัตถุยังมีความท้าทายหลายด้าน โดยเฉพาะการประมวลผลแบบเรียลไทม์ในสภาพแวดล้อมที่ซับซ้อน หลากหลาย และเปลี่ยนแปลงตลอดเวลา เช่น วัตถุที่ซ้อนทับกัน วัตถุขนาดเล็ก หรือการเปลี่ยนแสงและมุมมองอย่างรวดเร็ว งานวิจัยในปัจจุบันจึงมุ่งไปที่การพัฒนาโมเดลที่มีประสิทธิภาพมากขึ้น ใช้ทรัพยากรน้อยลง และสามารถปรับใช้ได้หลากหลายสถานการณ์

ท้ายที่สุด ผู้เขียนหวังว่าผู้อ่านจะสามารถนำความรู้จากบทความนี้ไปใช้เป็นฐานในการเรียนรู้เพิ่มเติม ทดลองปรับใช้ หรือพัฒนาต่อยอดระบบตรวจจับวัตถุให้เหมาะสมกับปัญหาและข้อจำกัดของตนเองได้อย่างมั่นใจและมีทิศทาง

บทบาทของผู้เขียน (author contributions) จิระเมศร์ รุจิกรศิริณย์ เป็นผู้ประพันธ์อันดับแรก (first author) และผู้รับผิดชอบบทความหลัก (corresponding author)

คำชี้แจงของผู้เขียน (author declarations) บทความนี้มีการใช้โมเดลที่ผ่านการฝึกฝนล่วงหน้า ไลบรารีสาธารณะ โค้ดตัวอย่าง และสื่อประกอบจากแหล่งภายนอกเพื่อการสาธิตเชิงการเรียนรู้ ผู้เขียนได้ตรวจสอบและระบุแหล่งที่มาของโค้ด ไลบรารี ภาพ วิดีโอ และผลลัพธ์ที่เกี่ยวข้องอย่างเหมาะสม ทั้งนี้ ความถูกต้องของการอ้างอิง แหล่งที่มา สิทธิการใช้สื่อ และความสามารถในการค้นหาของโค้ดเป็นความรับผิดชอบของผู้เขียน

ผลประโยชน์ทับซ้อน (competing interests) ผู้เขียนขอยืนยันว่าการจัดทำบทความฉบับนี้ ปราศจากผลประโยชน์ทับซ้อนทั้งในเชิงวิชาการ การเงิน หรือความสัมพันธ์ส่วนบุคคลใด ๆ

การเปิดเผยข้อมูล (data availability) ไฟล์ตัวอย่าง โค้ดประกอบ และเอกสารเพิ่มเติมสามารถขอรับได้จากผู้รับผิดชอบบทความหลักผ่านทางอีเมลที่ระบุไว้

รายการอ้างอิง

- Bewley, A., Ge, Z., Ott, L., Ramos, F., & Upcroft, B. (2016). Simple online and realtime tracking. In *2016 IEEE International Conference on Image Processing (ICIP)* (pp. 3464–3468). IEEE. <https://doi.org/10.1109/ICIP.2016.7533003>
- Bochkovskiy, A., Wang, C.-Y., & Liao, H.-Y. M. (2020). YOLOv4: Optimal speed and accuracy of object detection. arXiv. <https://arxiv.org/abs/2004.10934>
- Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., & Zagoruyko, S. (2020). End-to-end object detection with transformers. In A. Vedaldi, H. Bischof, T. Brox, & J.-M. Frahm (Eds.), *Computer vision – ECCV 2020* (Lecture Notes in Computer Science, Vol. 12346, pp. 213–229). Springer. https://doi.org/10.1007/978-3-030-58452-8_13
- Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 580–587). IEEE. <https://doi.org/10.1109/CVPR.2014.81>
- He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 770–778). IEEE. <https://doi.org/10.1109/CVPR.2016.90>
- Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H. (2017). MobileNets: Efficient convolutional neural networks for mobile vision applications. arXiv. <https://arxiv.org/abs/1704.04861>
- levan92. (n.d.). *deep_sort_realtime* [Computer software]. GitHub. Retrieved June 16, 2026, from https://github.com/levan92/deep_sort_realtime
- Lin, T.-Y., Dollár, P., Girshick, R., He, K., Hariharan, B., & Belongie, S. (2017). Feature pyramid networks for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 2117–2125). IEEE.
- Lin, T.-Y., Goyal, P., Girshick, R., He, K., & Dollár, P. (2017). Focal loss for dense object detection. In *Proceedings of the IEEE International Conference on Computer Vision* (pp. 2980–2988). IEEE. <https://doi.org/10.1109/ICCV.2017.324>
- Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., & Zitnick, C. L. (2014). Microsoft COCO: Common objects in context. In D. Fleet, T. Pajdla, B. Schiele, & T. Tuytelaars (Eds.), *Computer vision – ECCV 2014* (Lecture Notes in Computer Science, Vol. 8693, pp. 740–755). Springer. https://doi.org/10.1007/978-3-319-10602-1_48
- Liu, L., Ouyang, W., Wang, X., Fieguth, P., Chen, J., Liu, X., & Pietikäinen, M. (2020). Deep learning for generic object detection: A survey. *International Journal of Computer Vision*, 128(2), 261–318. <https://doi.org/10.1007/s11263-019-01247-4>
- Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., & Berg, A. C. (2016). SSD: Single shot multibox detector. In B. Leibe, J. Matas, N. Sebe, & M. Welling (Eds.), *Computer vision – ECCV 2016* (Lecture Notes in Computer Science, Vol. 9905, pp. 21–37). Springer. https://doi.org/10.1007/978-3-319-46448-0_2

- OpenCV. (n.d.). *OpenCV documentation*. Retrieved June 16, 2026, from <https://docs.opencv.org/4.x/>
- Park, J., Lee, J., & Jeong, J. (2024). LyFormer based object detection in reel package X-ray images of semiconductor component. *Journal of King Saud University – Computer and Information Sciences*, 36(1), Article 101859. <https://doi.org/10.1016/j.jksuci.2023.101859>
- PyTorch. (n.d.). *fasterrcnn_resnet50_fpn*. Torchvision documentation. Retrieved June 16, 2026, from https://docs.pytorch.org/vision/stable/models/generated/torchvision.models.detection.fasterrcnn_resnet50_fpn.html
- Redmon, J., & Farhadi, A. (2017). YOLO9000: Better, faster, stronger. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 7263–7271). IEEE. <https://doi.org/10.1109/CVPR.2017.690>
- Redmon, J., & Farhadi, A. (2018). YOLOv3: An incremental improvement. arXiv. <https://arxiv.org/abs/1804.02767>
- Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 779–788). IEEE. <https://doi.org/10.1109/CVPR.2016.91>
- Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards real-time object detection with region proposal networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 28, pp. 91–99). Curran Associates, Inc.
- Rosebrock, A. (2016). *Intersection over Union (IoU) for object detection*. PyImageSearch. <https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/>
- Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018). MobileNetV2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 4510–4520). IEEE. <https://doi.org/10.1109/CVPR.2018.00474>
- Shah, D. (2022, March 7). *Mean average precision (mAP) explained: Everything you need to know*. V7 Labs. <https://www.v7labs.com/blog/mean-average-precision>
- Sokolan, R. (n.d.). *Car and truck traffic on the highway in Europe, Poland – Summer day* [Stock video]. Vecteezy. Retrieved June 16, 2026, from <https://www.vecteezy.com/video/7957364-car-and-truck-traffic-on-the-highway-in-europe-poland-summer-day>
- Tan, M., Pang, R., & Le, Q. V. (2020). EfficientDet: Scalable and efficient object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 10781–10790). IEEE. <https://doi.org/10.1109/CVPR42600.2020.01079>

- Ultralytics. (n.d.). *Loading YOLOv5 from PyTorch Hub*. Ultralytics Docs. Retrieved June 16, 2026, from https://docs.ultralytics.com/yolov5/tutorials/pytorch_hub_model_loading
- Wang, C.-Y., Liao, H.-Y. M., Wu, Y.-H., Chen, P.-Y., Hsieh, J.-W., & Yeh, I.-H. (2020). CSPNet: A new backbone that can enhance learning capability of CNN. In *Proceedings of the 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* (pp. 1571–1580). IEEE Computer Society. <https://doi.org/10.1109/CVPRW50498.2020.00203>
- Wojke, N., Bewley, A., & Paulus, D. (2017). Simple online and realtime tracking with a deep association metric. In *2017 IEEE International Conference on Image Processing (ICIP)* (pp. 3645–3649). IEEE. <https://doi.org/10.1109/ICIP.2017.8296962>
- Yohanandan, S. (2020, June 9). *mAP (mean average precision) might confuse you!* Medium. <https://medium.com/data-science/map-mean-average-precision-might-confuse-you-5956f1bfa9e2>
- Zhang, Y., Sun, P., Jiang, Y., Yu, D., Weng, F., Yuan, Z., Luo, P., Liu, W., & Wang, X. (2022). ByteTrack: Multi-object tracking by associating every detection box. In S. Avidan, G. Brostow, M. Cissé, G. M. Farinella, & T. Hassner (Eds.), *Computer vision – ECCV 2022* (Lecture Notes in Computer Science, Vol. 13682, pp. 1–21). Springer. https://doi.org/10.1007/978-3-031-20047-2_1