ประสิทธิภาพของระบบตรวจจับวัตถุเคลื่อนที่
กรณีศึกษา ฟาร์มนกแอ่นกินรัง

ชวิศ ภูริพัฒน์

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
บัณฑิตวิทยาลัย สาขาวิชาเทคโนโลยีสารสนเทศ
สถาบันเทคโนโลยีไทย-ญี่ปุ่น
ปีการศึกษา 2563

THE EFFICIENCY OF THE MOTION DETECTION SYSTEM
A CASE STUDY OF EDIBLE-NEST SWIFTLET FARM

Shawiss Puripat

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Science Program in Information Technology
Graduate School
Thai-Nichi Institute of Technology
Academic Year 2020

หัวข้อวิทยานิพนธ์	ประสิทธิภาพของระบบตรวจจับวัตถุเคลื่อนที่ กรณีศึกษา ฟาร์มนกแอ่นกินรัง
โดย	ชวิศ ภูริพัฒน์
สาขาวิชา	เทคโนโลยีสารสนเทศ
อาจารย์ที่ปรึกษาวิทยานิพนธ์	รองศาสตราจารย์ ดร. รัตติกร วรากูลศิริพันธุ์

บัณฑิตวิทยาลัย สถาบันเทคโนโลยีไทย-ญี่ปุ่น อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

.....คณบดีบัณฑิตวิทยาลัย

(รองศาสตราจารย์ ดร. พิชิต สุขเจริญพงษ์)

วันที่.....เดือน.....พ.ศ.....

คณะกรรมการสอบวิทยานิพนธ์

.....ประธานกรรมการ

(ดร. โสภณ มงคลลักษมี)

.....กรรมการ

(รองศาสตราจารย์ ดร. อรรณพ หมั่นสกุล)

.....กรรมการ

(ดร. สรมย์พร เจริญพิทย์)

.....อาจารย์ที่ปรึกษาวิทยานิพนธ์

(รองศาสตราจารย์ ดร. รัตติกร วรากูลศิริพันธุ์)

ชวิศ ภูริพัฒน์ : ประสิทธิภาพของระบบตรวจจับวัตถุเคลื่อนที่ กรณีศึกษา ฟาร์มนกแอ่นกินรัง. อาจารย์ที่ปรึกษา : รองศาสตราจารย์ ดร. รัตติกร วรากูลศิริพันธุ์, 71 หน้า.

การบริหารจัดการธุรกิจฟาร์มนกแอ่นกินรังให้ประสบความสำเร็จนั้น โดยปัจจัยที่นำมาพิจารณาที่สำคัญที่สุดคือจำนวนของรังนกภายในฟาร์ม ซึ่งนกแอ่นกินรังถือว่าเป็นสัตว์ป่าที่ไม่สามารถเพาะเลี้ยงได้ตามธรรมชาติ จึงเป็นผลทำให้เกิดการคิดค้นหาเทคนิควิธีการต่างๆ เพื่อจำลองอาคารฟาร์มให้มีสภาพแวดล้อมภายในให้คล้ายถ้ำในธรรมชาติมากที่สุด โดยพฤติกรรมเสียงในการจับคู่เพื่อดึงดูดให้นกแอ่นกินรังเข้ามาทำรังวางไข่ต่อไป ดังนั้น งานวิจัยชิ้นนี้ได้ทำการออกแบบระบบตรวจนับจำนวนนกแอ่นกินรังที่บินเข้าออกอาคารฟาร์ม เพื่อสามารถนำไปวิเคราะห์การตรวจสอบนกที่แม่นยำและปรับสภาพแวดล้อมภายในอาคารฟาร์มให้มีจำนวนนกแอ่นกินรังเข้ามาทำรังให้มากยิ่งขึ้น โดยผู้วิจัยได้ออกแบบอุปกรณ์และพัฒนาโปรแกรมของระบบตรวจนับนกแอ่นกินรังนี้ ภายใต้อัลกอริทึม YOLO v.5 ที่สามารถตรวจนับจำนวนนกแอ่นกินรังที่บินเข้าออกของตัวอาคารฟาร์มได้อย่างแม่นยำที่ความถูกต้องสูงถึง 92 เปอร์เซ็น และสามารถตอบโจทย์ความต้องการของเกษตรกรได้อย่างดี

บัณฑิตวิทยาลัย ลายมือชื่อนักศึกษา.....

สาขาวิชา เทคโนโลยีสารสนเทศ ลายมือชื่ออาจารย์ที่ปรึกษา.....

ปีการศึกษา 2563

SHAWISS PURIPAT : THE EFFICIENCY OF THE MOTION DETECTION SYSTEM A CASE STUDY OF EDIBLE-NEST SWIFTLET FARM. ADVISOR : ASSOC. PROF. DR. RUTTIKORN VARAKULSIRIPUNTH, 71 PP.

The successful management of the nesting bird business is the most important factor to be considered is the number of bird nests within the farm. The edible-nest swiftlet is considered a wild animal that can't be cultured naturally. as a result, the idea of searching for techniques and methods for simulate farm buildings to have an internal environment similar to a natural cave as possible. By sound behavior in the mating to attract the edible-nest swiftlet to lay eggs. Therefore, this research has designed a system for counting the number of edible-nest swiftlets that fly in and out of buildings. In order to be able to analyze the accurate bird inspection and adjust the environment inside the building to have the number of edible-nest swiftlets to build more nests. The researcher has designed a device and developed a program for this edible-nest swiftlet counting system. Using the YOLO v.5 algorithm that can accurately count the number of edible-nest swiftlets that fly in and out of the building, the accuracy is up to 92 percent and can meet the needs of farmers very well.

Graduate School Student's Signature.....

Field of Study Information Technology Advisor's Signature.....

Academic Year 2020

กิตติกรรมประกาศ

การทำวิทยานิพนธ์นี้จะสำเร็จลุล่วงไปได้ด้วยดีนั้น ผู้วิจัยขอกราบขอบพระคุณในความกรุณาของรองศาสตราจารย์ รัตติกร วรากูลศิริพันธุ์ และรองศาสตราจารย์ ดร. อรรณพ หมั่นสกุล ซึ่งได้สละเวลาในการให้คำแนะนำ คำปรึกษา การสนับสนุน รวมถึงตรวจสอบข้อบกพร่องทุกขั้นตอนและกำลังใจเต็มเปี่ยมตลอดระยะเวลาในการทำวิทยานิพนธ์ฉบับนี้

ทั้งนี้ผู้วิจัยขอกราบขอบพระคุณคณะกรรมการสอบทุกท่านได้แก่ ดร. โสภณ มงคลลักษมี, ดร. สรมย์พร เจริญพิทย์ และรองศาสตราจารย์ ดร. อรรณพ หมั่นสกุล ที่ได้กรุณาให้ข้อเสนอแนะที่เป็นประโยชน์และช่วยตรวจสอบชี้ข้อบกพร่องของวิทยานิพนธ์ฉบับนี้ด้วยความเอาใจใส่ จนทำให้วิทยานิพนธ์ฉบับนี้สำเร็จและมีความสมบูรณ์

เหนือสิ่งอื่นใดผู้วิจัยขอกราบขอบพระคุณบิดามารดาของผู้วิจัยที่คอยให้กำลังใจ และให้การสนับสนุนในทุกๆ ด้านอย่างที่สุดเสมอมา

ผู้วิจัยหวังเป็นอย่างยิ่งว่า วิทยานิพนธ์ฉบับนี้จะเกิดประโยชน์ต่อผู้ที่มีความสนใจ สามารถนำไปต่อยอด องค์ความรู้เพื่อประยุกต์ใช้งานกับระบบการแนะนำการศึกษาเพื่อให้เกิดคุณค่าและสามารถใช้งานได้จริง

ชวิศ ภูริพัฒน์

สารบัญ

หน้า

สารบัญ (ต่อ)

สารบัญตาราง

สารบัญรูป

สารบัญรูป(ต่อ)

บทที่ 1
บทนำ

1.1 สภาวะความเป็นมา แนวทางเหตุผล และปัญหา

การทำฟาร์มเลี้ยงนกแอ่นกินรังมีวัตถุประสงค์ของการทำฟาร์มนกแอ่นกินรัง คือการพยายามปรับสภาพฟาร์มให้เหมาะสมกับการทำรังวางไข่แบบธรรมชาติของนกแอ่นกินรังตามถิ่นธรรมชาติ รวมทั้งการปรับอุณหภูมิ ความชื้น การไหลเวียนของอากาศและกลิ่น ป้องกันศัตรูผู้ล่าเช่น เหยี่ยว นกเขา เสียงเรียกนกที่ใช้ดึงดูดนกให้เข้ามาทำรัง และอื่นๆ โดยปกติการสร้างฟาร์มนกแอ่นจะใช้เวลาประมาณ 2-3 ปี ในประเทศไทยมีนกแอ่นกินรัง 12 สายพันธุ์ รังนกที่มีมูลค่าคือรังของ นกแอ่นกินรัง (Edible-Nest Swiftlet) ซึ่งในประเทศไทยมีอยู่ด้วยกัน 3 ชนิด คือ

- นกแอ่นกินรัง (Collocalia fuciphaga, Gmelin)
- นกแอ่นกินรังตะโพกขาว (C. germanni, Oustalet)
- นกแอ่นหางสี่เหลี่ยมหรือนกแอ่นรังดำ (C. maxima, Hume)

โดยลักษณะการสร้างฟาร์มส่วนมากจะเป็นนกแอ่นกินรังชนิดรังขาว (Edible-nest Swiftlets : Aerodramus fuciphagus) ลักษณะพันธ์ที่มีขนาดปานกลาง (ลำตัวยาวประมาณ 12 เซนติเมตร) ลำตัวสีน้ำตาลเข้มเกือบดำหลังสีน้ำตาลเข้มเกือบดำ สะโพกสีน้ำตาลหรือเทาอ่อน หางแฉกเล็กน้อย ท้องสีน้ำตาล นัยน์ตาสีน้ำตาล จะงอยปากดำ เท้าสีดำ จึงสามารถสร้างเสียงนกแอ่นกินรังเพื่อดึงดูดนกแอ่นกินรังมาทำรังเป็นคู่เพื่อสร้างรังในฟาร์มและมีการเพิ่มปริมาณตามสภาพของฟาร์มต่อไป ดังนั้น สิ่งที่เกษตรกรอยากรู้เป็นอันดับต้นๆ ก็คือ ปริมาณจำนวนนกแอ่นกินรังที่สนใจเข้ามาทำรังภายในฟาร์มทั้งการบินเข้ามาสำรวจ การมาเกาะนอนชั่วคราว และการวางไข่ ซึ่งเป็นประเด็นน่าสนใจที่จะนำมาพัฒนาระบบตรวจจับนกแอ่นกินรังในระบบฟาร์ม แต่การนับจำนวนนกที่กำลังเคลื่อนที่ภายในห้องนั้นเป็นเรื่องที่ลำบาก ผู้วิจัยจึงคิดว่า การนำระบบการประมวลผลภาพที่สามารถนำภาพมาวิเคราะห์เป็นข้อมูล จะช่วยพัฒนาการตรวจสอบจำนวนนกและการปรับปรุงสภาพแวดล้อมของฟาร์มต่อไปได้เราสามารถนำวิธีการประมวลผลภาพ (Image Processing) นั้นได้มีการนำมาประยุกต์ใช้กับงานหลายประเภท ตัวอย่างเช่น ระบบตรวจวัดความเร็วของยานพาหนะ ระบบคัดกรองบุคคลเข้า-ออก เป็นต้น โดยผู้วิจัยได้ให้ความสนใจในเรื่องการนับจำนวนวัตถุเคลื่อนที่ผ่านการประมวลผลภาพ ซึ่งจะช่วยให้เราสามารถตรวจนับจำนวนได้อย่างรวดเร็วและแม่นยำมากยิ่งขึ้น จึงอยากจะนำระบบนี้มาทดสอบโดยการนำใช้กับฟาร์มนกแอ่นกินรัง ด้วยเหตุผลว่าลักษณะลำตัวของตัวนกแอ่นกินรังนั้นมีขนาดที่ค่อนข้างเล็ก โดยมีขนาดลำตัวยาวประมาณ 11.5 – 12.5 เซนติเมตร มีปีกยาว 113 -123.5 มิลลิเมตร ขนลำตัวมักมีสีน้ำตาลคล้ำด้านๆ จนถึงสีดำเป็นมัน และตัวนกแอ่นบินเร็วด้วย จึงยากต่อ

การมองและนับจำนวนด้วยตาเปล่า อีกทั้งอัตราการรอดของนกที่อาศัยตามบ้านนกหรือคอนโดนกมี 64.4 % ซึ่งมีโอกาสที่จำนวนตัวนกที่อาศัยอยู่มีปริมาณที่ไม่แน่นอน รวมถึงช่วงเวลาที่ตัวนกแอ่นจะหาอาหารช่วงเวลาประมาณ 5.30 น.- 6.30 น. และบินกลับรังประมาณ 18.15 น. - 19.15 น. บางครั้งนกแอ่นบางตัวอาจจะกลับรังประมาณ 12.00 น. เพื่อให้อาหารลูกอ่อน ขณะที่นกแอ่นบางส่วนกกไข่

รูปที่ 1.1 ตัวอย่างบ้านนกหรือคอนโดนก

รูปที่ 1.2 Video Streaming

การประมวลผลภาพดิจิทัล (Digital Image Processing) นั้นเป็นสาขาที่กล่าวถึงรูปแบบเทคนิคและอัลกอริทึมต่างๆ ที่จะนำมาใช้ในการประมวลผลภาพที่อยู่ในรูปแบบภาพดิจิทัล โดยภาพในที่นี้ จะรวมถึงความหมายเกี่ยวกับสัญญาณดิจิทัลใน 2 มิติอื่นๆ โดยทั่วไปคำนี้เมื่อใช้อย่างกว้างๆ จะครอบคลุมถึงสัญญาณวีดิโอ (video) หรือภาพเคลื่อนไหว ซึ่งจะเป็นชุดของภาพนิ่ง เรียกว่า เฟรม

(frame) หลายๆภาพต่อกันไปตามเวลาซึ่งก็คือสัญญาณ 3 มิติ เมื่อนับเวลาเป็นมิติที่ 3 หรืออาจจะครอบคลุมถึงสัญญาณ 3 มิติอื่นๆ เช่น ภาพ 3 มิติทางการแพทย์ หรือ อาจจะมากกว่านั้น เช่น ภาพ 3 มิติ และ หลายชนิด (multimodal image)

1.2 วัตถุประสงค์ของการวิจัย

1.2.1 เพื่อศึกษาระบบการตรวจจับวัตถุเคลื่อนที่

1.2.2 เพื่อนำระบบ Digital Image Processing มาประยุกต์ใช้ในการนับจำนวนนกแอ่นกินรัง

1.2.3 เพื่อออกแบบและใช้งานการตรวจนับจำนวนนกนางแอ่นกินรัง

1.3 ขอบเขตของการวิจัย

1.3.1 ศึกษาพฤติกรรมการเคลื่อนที่ของนกแอ่นกินรัง

1.3.2 ออกแบบระบบการตรวจจับวัตถุเคลื่อนที่ ไว้ใช้งานภายในพื้นที่ที่กำหนดไว้

1.3.3 แสดงจำนวนนกแอ่นกินรังที่กำลังเคลื่อนที่อยู่ภายในอาคาร ห้อง

1.4 ประโยชน์ที่คาดว่าจะได้รับ

1.4.1 สามารถช่วยให้การนับนกแอ่นกินรังมีจำนวนที่ชัดเจนและความแม่นยำยิ่งขึ้น

1.4.2 สามารถช่วยเพิ่มประสิทธิภาพในการตรวจสอบนกแอ่นกินรัง

1.5 นิยามศัพท์เฉพาะ

1.5.1 การประมวลผลภาพ (Image Processing) คือ การประมวลผลภาพ (Image Processing) หมายถึง การนำภาพมาประมวลผลหรือคิดคำนวณด้วยคอมพิวเตอร์ เพื่อให้ได้ข้อมูลที่เราต้องการทั้งในเชิงคุณภาพและปริมาณ โดยมีขั้นตอนหลักๆ คือ การแบ่งส่วนของวัตถุที่เราสนใจออกมาจากภาพ เพื่อนำภาพวัตถุที่ได้ไปวิเคราะห์หาข้อมูลเชิงปริมาณ เช่น ขนาด รูปร่าง และทิศทางการเคลื่อนของวัตถุในภาพ

1.5.2 การประมวลผลภาพดิจิทัล (Digital Image Processing) คือ การแปลงข้อมูลภาพให้อยู่ในรูปแบบข้อมูลดิจิตอล (Digital format) ซึ่งสามารถที่จะนำเอาข้อมูลนี้จัดผ่านกระบวนการต่างๆ ด้วยดิจิตอลคอมพิวเตอร์ได้ ในระบบของดิจิตอล อินพุต และเอาพุตของระบบจะอยู่ในรูปแบบดิจิตอลเท่านั้น

1.5.3 การตรวจจับวัตถุ (Object Detection) คือ เทคโนโลยีในทางคอมพิวเตอร์ หลักการที่เกี่ยวกับ Computer Vision และ Image Processing ที่ใช้ในงาน AI ตรวจจับวัตถุชนิดที่กำหนด เช่น มนุษย์ รถยนต์ อาคาร ที่อยู่ในรูปภาพ หรือวิดีโอ

บทที่ 2
แนวคิด ทฤษฎี และงานวิจัยที่เกี่ยวข้อง

จากการศึกษาเทคโนโลยีต่างๆ เพื่อประเมินประสิทธิภาพ ผู้วิจัยจึงได้สืบค้นข้อมูล ทฤษฎีและงานวิจัยที่เกี่ยวข้อง เพื่อเป็นแนวทางในการศึกษา โดยมีหัวข้อดังต่อไปนี้

2.1 ทฤษฎีที่เกี่ยวข้อง

2.1.1 นกแอ่นกินรัง

นกแอ่นกินรัง (Edible-nest Swiftlet) **[1,2]** เป็นนกกลุ่มหนึ่งสามารถให้ประโยชน์แก่มนุษย์ โดยมีขนาดตัวเล็กปีกเรียวโค้งเป็นคันธนู ท้ายของหางมน ขาและเท้ามีขนาดเล็ก มีเล็บโค้งงอ มีนิสัยหาแมลงกินในอากาศเป็นอาหาร โดยนกนางแอ่นนั้นมีหางหยักเป็นแฉกลึก ทำรังด้วยโคลนแล้วเสริมความแข็งแรงด้วยหญ้าติดอยู่ตามหน้าผา หรือสถานที่ก่อสร้างต่าง ๆ ในประเทศไทย

2.1.1.1 ลักษณะของนกแอ่นกินรัง

ในประเทศไทยจะพบนกแอ่นกินรังอยู่ 3 ชนิด

1. นกแอ่นกินรัง (Edible-Nest Swiftlet)

ลักษณะทั่วไป ตัวผู้และตัวเมียมีลักษณะคล้ายคลึงกันมาก โดยตัวผู้จะมีขนาดโตกว่าตัวเมียเล็กน้อย เป็นนกขนาดเล็กขนาดเท่านกกระจอกบ้าน จงอยปากสั้นกว้างมีสีดำ ม่านตาสีดำ วงปีกเป็นรูปโค้งคันธนู ปีกหมุนได้รอบ ขนด้านบนของตัวมีสีดำ อมน้ำตาล ส่วนล่างจะมีสีจางไล่ไปจนถึงส่วนท้าย ขนที่แข็งจะมีเล็กน้อยหรือไม่มี หางเป็นแฉกเล็กน้อย ตะโพกมีสีขาว ทางด้านท้องมีสีน้ำตาล เวลาบินจะกระพือปีกเร็วมาก รังมีสีขาวสร้างด้วยเมือกโปร่งแสงซึ่งประกอบด้วยสารพวกไกลโคโปรตีนที่สังเคราะห์มาจากต่อมน้ำลายนิสัย ชอบสร้างรังตามผนังถ้ำในเกาะแถบชายฝั่งหรือบางครั้งก็อยู่ตามอาคาร การสร้างรังใหม่มักจะสร้าง ณ จุดเดิม ชอบออกหาอาหารประเภทแมลงในตอนเช้าตรู่ เป็นพื้นที่กว้างบริเวณชายป่า ทุ่งนาและใกล้รังในช่วงเย็นสามารถบินกลับรังภายในถ้ำที่เป็นซอกหลืบ หรือโพรงที่มืดมิดได้อย่างแม่นยำ ซึ่งเกิดจากการส่งเสียงร้องให้เกิดเสียงสะท้อนน้ำทางซึ่งเรียกว่า Echolocation และจะสร้างรังใหม่ในเวลากลางคืน ถิ่นกำเนิด หมู่เกาะอันดามัน-นิโคบาส์, เกาะซุนดาส์ ประเทศอินโดนีเซีย, เกาะพาบาวานประเทศฟิลิปปินส์, เทือกเขาตะนาวศรี, พม่า, มาเลเซีย, ไทย, เวียดนาม, โคชินไชนา และอ่าวตังเกี๋ย

รูปที่ 2.1 ตัวอย่างภาพนกแอ่นกินรัง

2. นกแอ่นกินรังตะโพกขาว (White-Rumped Swift)

ลักษณะทั่วไป มีสะโพกสีขาว ขนทางด้านใต้ท้องมีลายขวางสีดำสลับขาว หางเป็นแฉกมองเห็นชัดเจน นกแอ่นกินรังตะโพกขาวและนกแอ่นกินรังเดิมจัดเป็นชนิดเดียวกัน แต่ปัจจุบันจัดไว้คนละชนิดนกแอ่นกินรังตะโพกขาวจะมีรังที่มีสีขาว (white nest) นิสัย เหมือนนกแอ่นกินรัง ถิ่นกำเนิด อินเดีย, จีน, ไหหลำ, ไต้หวัน, เกาะซุนดาส์ในประเทศออสเตรเลีย, พม่า, ไทย, ลาว, ฮ่องกง, ฟิลิปปินส์ และเวียดนาม

3. นกแอ่นหางสี่เหลี่ยมหรือนกแอ่นรังดำ (Black-nest Swiftlet)

ลักษณะทั่วไป สีด้านบนของตัวมีสีน้ำตาลแก่ ด้านล่างของตัวมีสีเทา หางเป็นรูปบากเล็กน้อย หน้าแข้งมีขนขึ้นเต็มไปหมด เสียงร้องเหมือนนกแอ่นกินรัง หรือเหมือนนกแอ่นพันธุ์หิมาลัย รังมีสีดำ (Black Nest) เพราะมีขนเป็นส่วนประกอบและมักจะมีถิ่นอาศัยที่เดียวกับนกแอ่นกินรัง นิสัย เหมือนนกแอ่นกินรัง ถิ่นกำเนิดเทือกเขาหิมาลัย อินเดีย, เกาะซุนดาส์ อินโดนิเซีย, เกาะพาลาวาน ฟิลิปปินส์, เทือกเขาตะนาวศรี ไทย และมาเลเซีย

2.1.1.2 พฤติกรรมของนกแอ่นกินรัง

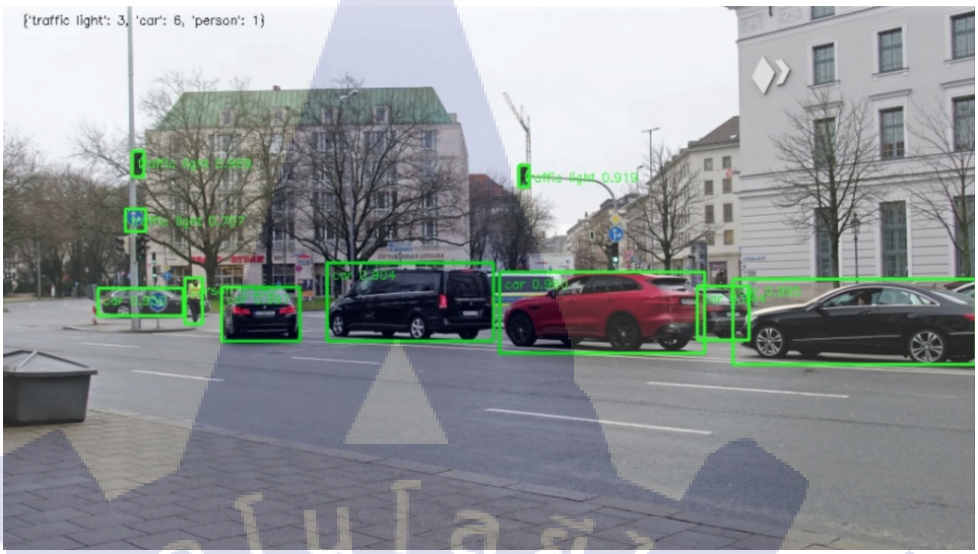
นกแอ่นกินรัง เป็นกลุ่มนกที่มักทำกิจกรรมต่าง ๆ ส่วนใหญ่อยู่ในอากาศในช่วงเวลากลางวัน ส่วนช่วงเวลากลางคืนเกาะนอนอยู่เป็นกลุ่มในที่อยู่อาศัย ตามผนังถ้ำ หรือสิ่งก่อสร้างของมนุษย์ กิจกรรมในรอบวันของนก ในช่วงเช้านกบินออกจากแหล่งที่อยู่อาศัยไปหากิน พื้นที่หากินของนกกว้างครอบคลุมทั้งบริเวณพื้นที่แนวชายฝั่งทะเล พื้นที่ป่าไม้ หุบเขาตามเกาะกลางทะเล ป่าชายเลน พื้นที่เกษตรกรรม หรือพื้นที่เปิดโล่งต่าง ๆ อีกทั้งยังเป็นกลุ่มนกที่มีความสามารถในการบินที่มี

ประสิทธิภาพสูง โดยมีการปรับตัวของปีกในส่วนของกระดูกต้นปีกให้มีลักษณะป้อม สั้น กระดูกชิ้นต่อไปยาว และมีมัดกล้ามเนื้อมาเกาะทำให้ปีกมีความแข็งแรง การบินของนกเป็นการบินในลักษณะตรงและเร็ว ฉวัดเฉวียนน้อยกว่านกนางแอ่น ลักษณะการบินของนกเป็นการกระพือปีกเป็นจังหวะเร็ว สลับการร่อนในอากาศ การโบกปีกทั้ง 2 ข้างของนกเป็นจังหวะที่ไม่สอดคล้องกัน โดยนกสามารถโบกปีกข้างใดข้างหนึ่งให้เร็วกว่าอีกข้างหนึ่งได้ เนื่องจากขนหางของนกมีความยาวไม่มากเมื่อเทียบกับขนาดลำตัวของนก ทำให้ใช้ประโยชน์ในการบังคับทิศทางได้ไม่ดีนัก จึงต้องใช้การโบกปีกช่วยในการบังคับทิศทางการบินความเร็วในการบินของนกแอ่นกินรัง ประมาณ 20 - 30 ไมล์ต่อชั่วโมง โดยนกแอ่นกินรังจะใช้เวลาบินในอากาศอย่างน้อย 11 ชั่วโมงต่อวัน ในระดับความสูงมากกว่า 1,000 ฟุต รวมระยะทางที่ นกบินในรอบวันประมาณ 50 - 70 ไมล์ ครอบคลุมพื้นที่ประมาณ 40 ตารางกิโลเมตร พบว่านกในกลุ่มนกแอ่นกินรังใช้เวลาการบินในอากาศถึง 16 - 18 ชั่วโมงต่อวัน ครอบคลุมพื้นที่ 90 - 100 ตารางกิโลเมตร ในช่วงที่นกมีลูกในรัง นกมีพฤติกรรมการเลี้ยงดูลูกอ่อน โดยการบินนา อาหารมาป้อนลูก ในรังเป็นครั้งคราวในช่วงเวลากลางวัน ส่วนในช่วงที่มีฝนตกในตอนเช้านกจะยังไม่บินออกจากแหล่งที่อยู่อาศัยจนกว่าฝนจะหยุดตกหรือซาเม็ดลงจึงบินออกไปหากินตามปกติ ในช่วงเย็นจะบินกลับที่อยู่อาศัย

2.1.2 การตรวจจับวัตถุ

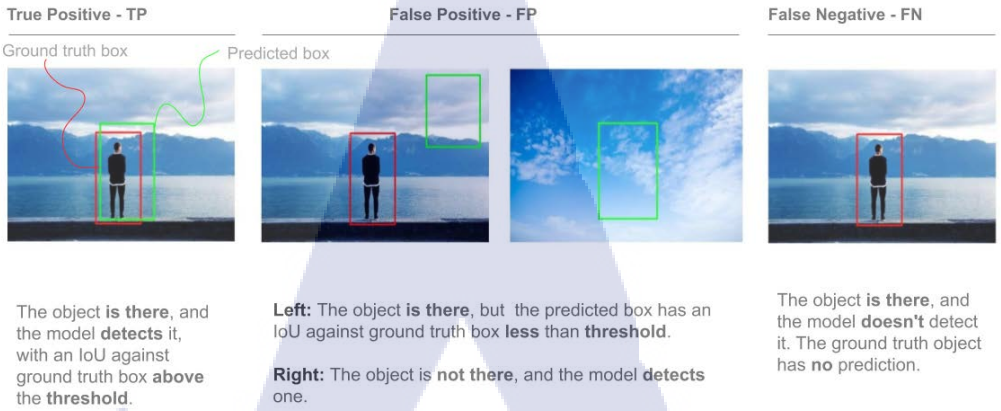
การตรวจจับวัตถุ (Object Detection) เป็นการค้นหาวัตถุในรูปภาพ ซึ่งสามารถจำแนกและเจาะลึกลงไปได้อีกหลายแขนง เช่น การตรวจจับหน้าคน (Face Detection) ตรวจจับคนเดินถนน (Pedestrian Detection) อีกทั้งยังสามารถนำมาประยุกต์ใช้กับงานได้อย่างหลากหลาย เช่น ใช้ในงานรักษาความปลอดภัย การนับจำนวนวัตถุ เป็นต้น

การตรวจจับวัตถุ **[3,4]** คือ AI ตรวจจับวัตถุ ในงานทางด้าน Computer Vision ที่จะจำแนก และตรวจจับวัตถุที่อยู่ในรูป การตรวจจับ มาร์กจุด มาร์กพื้นที่ โดยหลักการนี้ สามารถทำได้หลายวิธี การทำมาร์กพื้นที่ที่นิยมได้แก่ วาดกล่องรอบวัตถุ (Bounding Box) หรือ ถมสีให้ทุก Pixel ของวัตถุนั้น (เรียกว่า Segmentation) โดยตัวอย่างตามรูปที่ 2.2 และ 2.3



รูปที่ 2.2 ตัวอย่างการทำ Object Detection โดยการวาดกล่องรอบวัตถุ

รูปที่ 2.3 ตัวอย่างการทำ Object Detection โดยการถมสีให้ทุก Pixel ของวัตถุนั้น



รูปที่ 2.4 ตัวอย่างการหาค่า TP, FP, FN

จากรูปที่ 2.4 กรอบสีแดงคือกรอบวัตถุความจริง และกรอบสีเขียวคือกรอบทำนายวัตถุ ค่า IoU ที่ได้จากกรอบทำนายวัตถุและกรอบวัตถุความจริงซ้อนทับตามรูปที่ 2.5 เราสามารถตั้งค่า threshold ได้สำหรับค่า IoU เพื่อพิจารณาว่าการตรวจจับวัตถุนั้นถูกต้องหรือไม่ โดยสมมติว่าเราตั้งค่า IoU เป็น 0.5 โดย

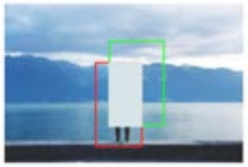
ถ้า IoU ≥ 0.5 ให้จัดประเภทการตรวจจับวัตถุเป็น True Positive (TP)

หาก IoU < 0.5 แสดงว่าเป็นการตรวจจับที่ผิดและจัดประเภทเป็น False Positive (FP)

เมื่อกรอบวัตถุความจริงปรากฏอยู่ในภาพและแบบจำลองไม่สามารถตรวจจับวัตถุได้ให้จัดประเภทเป็น False Negative (FN)

True Negative (TN) คือทุกส่วนของภาพที่เราไม่ได้ทำนายวัตถุ โดยเมตริกนี้จะไม่มีผลต่อการตรวจจับวัตถุดังนั้นเราจึงไม่สนใจ True Negative

ส่วนการตั้งค่า threshold ของ IoU เป็น 0.5 หรือสูงกว่า สามารถตั้งค่าเป็น 0.5, 0.75 0.9 หรือ 0.95 เป็นต้น การใช้ค่า Precision และค่า Recall ของเมตริกในการประเมินประสิทธิภาพ Precision และ Recall มาคำนวณโดยใช้ True Positive, False Positive และ False Negative โดยจะคำนวณตามรูปที่ 2.6

$$IOU = \frac{\text{Area of Intersection}}{\text{Area of Union}} = \frac{\text{  }}{\text{  }}$$

รูปที่ 2.5 วิธีการหาค่า IoU

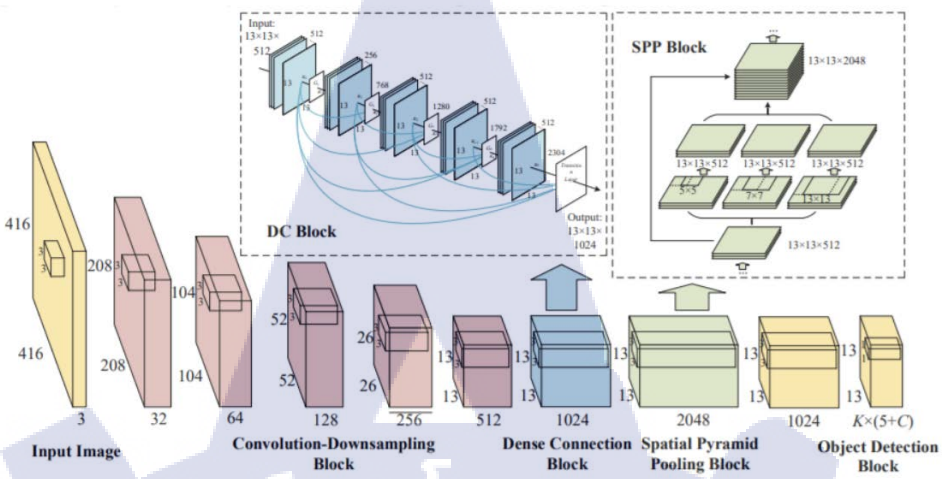
$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive + False Positive}}$$

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive + False Negative}}$$

รูปที่ 2.6 วิธีการหาค่า Precision และ Recall

2.1.3 YOLOv5

YOLO (You Only Look Once) เป็นอัลกอริธึมการรู้จำวัตถุแบบเรียลไทม์ที่มีประสิทธิภาพ ซึ่ง **[5]** จะกล่าวถึงแนวคิดของการตรวจจับวัตถุอัลกอริทึม โดย **[6,7]** อัลกอริธึม YOLO เป็น Object Detection Model ตัวหนึ่งที่มีความโดดเด่นอย่างมากในด้านของความเร็ว ตลอดจนการปรับปรุงบางอย่างในขั้นตอนการคาดคะเนกรอบขอบเขตที่แตกต่างกันเพื่อแยกคุณสมบัติ

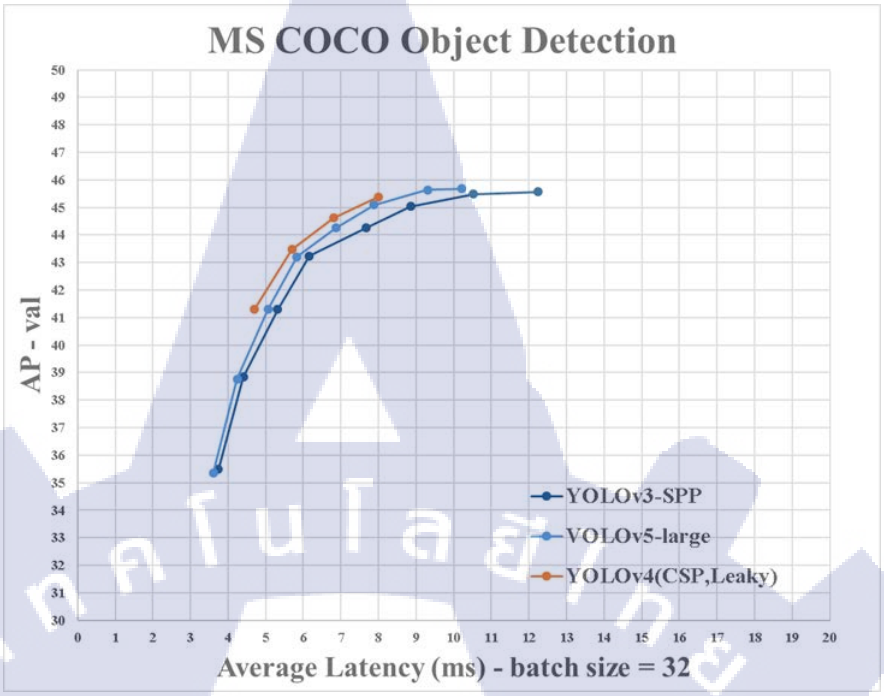


รูปที่ 2.7 โครงสร้างของ YOLO ที่เป็นชั้นๆ แบบ neural network

โดยจะมีอัลกอริทึมที่แตกต่างกันสำหรับการตรวจจับวัตถุ (Object Detection) และสามารถแบ่งออกเป็นสองกลุ่ม **[8,9]**

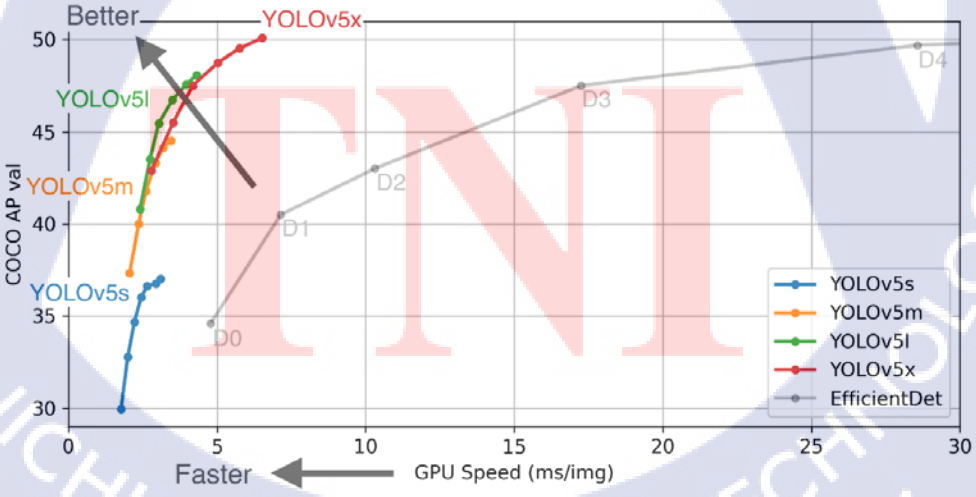
อัลกอริทึมที่ใช้กับ Classification มีการดำเนินการในสองขั้นตอน ขั้นแรกเลือกภูมิภาคที่สนใจในรูปภาพ ต่อมาก็จำแนกภูมิภาคเหล่านี้โดยใช้โครงข่ายประสาทเทียม การแก้ปัญหานี้อาจช้าเนื่องจากต้องเรียกใช้การคาดการณ์สำหรับทุกภูมิภาคที่เลือก ตัวอย่างของอัลกอริทึมประเภทนี้คือ Region-based convolutional neural network (RCNN)

อัลกอริทึมที่ใช้กับ Regression แทนที่จะเลือกส่วนที่น่าสนใจของรูปภาพ ก็จะทำนายคลาสและขอบเขตสำหรับรูปภาพทั้งหมดในการรันอัลกอริทึมเดียว ตัวอย่างที่ดีที่สุดจากอัลกอริทึมกลุ่มนี้คือ YOLO และ SSD (Single Shot Multibox Detector) มักใช้สำหรับการตรวจจับวัตถุแบบเรียลไทม์เนื่องจากโดยทั่วไปแล้วอาจต้องแลกกับการลดความแม่นยำลงเล็กน้อยเพื่อการปรับปรุงให้ความเร็วเพิ่มขึ้น



รูปที่ 2.8 ตัวอย่างภาพเปรียบเทียบการทำงานของโมเดล

สำหรับเหตุผลที่เลือก Yolov5s มาใช้แทน Yolov4 เพราะด้วยขนาดของโมเดลที่เล็กกว่ามาก มีความเร็วในการประมวณผลสูง แม้ว่าความแม่นยำอาจจะยังน้อยกว่ากันอยู่บ้าง แต่การนำมาใช้งานนั้นง่ายกว่ามาก เหมาะกับงานที่ต้องการความเร็วในการทำงานและประมวณผลของระบบ **[10]**



รูปที่ 2.9 ภาพการเปรียบเทียบการทำงานของโมเดล Yolov5 แต่ละขนาด

2.2 งานวิจัยที่เกี่ยวข้อง

งานวิจัยที่เกี่ยวข้องกับการประเมินประสิทธิภาพและวิเคราะห์การทำงานมีดังนี้

2.2.1 Development of a Vehicle Counting Algorithms by Image Processing from Video Camera

งานวิจัยนี้ได้ประยุกต์ใช้กล้องวิดีโอ และเทคโนโลยีเกี่ยวกับการประมวลผลภาพเข้ามาใช้เพื่อแก้ปัญหาเนื่องจากต้นทุนต่ำและมีประสิทธิภาพโดยใช้วิธีการตรวจจับภาพเคลื่อนไหว การวิเคราะห์บล็อบ การปรับมุมมองภาพในแนวลึก (Perspective) และประมาณค่าความหนาแน่นของการจราจรแบบ Real-Time ซึ่งเป็นวิธีที่ง่าย ไม่ซับซ้อน มีประสิทธิภาพ สามารถทดสอบได้ในสถานที่จริงและได้ผลลัพธ์ที่ดี เพื่อใช้ในการจัดการจราจร ให้เป็นไปอย่างถูกต้อง **[11]**

รูปที่ 2.10 ภาพที่วางกรอบตรวจจับและทำการตรวจจับและปรับมุมมองภาพในแนวลึก

2.2.2 Implementation of a Low-Cost Real-Time People Counting System on Raspberry Pi Based on Applied Tiny YOLO V3

สร้างระบบ Classification นับบุคคลที่เดินไปมาแบบ Real-Time บน Raspberry Pi ด้วยการประยุกต์ใช้ Tiny YOLO V3 ซึ่งเป็นอัลกอริทึมการตรวจจับวัตถุรุ่นที่มีความเร็วและความถูกต้องแม่นยำสูง โดย Tiny YOLO v3 มีความซับซ้อนน้อยกว่า YOLO แบบปกติมาก ซึ่งเหมาะกับแพลตฟอร์มของระบบสมองกลฝังตัว Raspberry Pi ที่มีทรัพยากรและความเร็วในการประมวลผลที่จำกัด **[12]**

Bright-ness (%)	Threshold (%)	Total number of people	System counting	Counting accuracy (%)
100	80	24	16	67
	60		24	100
90	80		16	67
	60		24	100
80	80		15	62
	60		22	91
70	80		6	25
	60		12	50
60	80		0	0
	60		5	21
50	80		0	0
	60		0	0

รูปที่ 2.11 รายละเอียดประสิทธิภาพการตรวจนับบุคคล โดยแสดงเปรียบเทียบความแม่นยำของการตรวจนับด้วยระบบ ระหว่างการกำหนดเกณฑ์ต่ำสุดของอัลกอริทึม Tiny YOLO V3 ในการตรวจจับวัตถุที่ 80% และ 60%

2.2.3 Automatic detection, tracking and counting of birds in marine video content

การตรวจจับวัตถุที่เคลื่อนไหวโดยอัตโนมัติที่มีประสิทธิภาพในทะเล เป็นปัญหาที่มีหลายแง่มุมเนื่องจากความซับซ้อนของฉากที่สังเกตได้ ธรรมชาติของทะเลที่เกิดจากคลื่นเรือและสภาพอากาศทำให้เกิดความลำบากอย่างมากสำหรับการพัฒนา Dynamic background subtraction (DBGS) ถือเป็นวิธีการแก้ปัญหาที่เหมาะสมในขอบเขตของการตรวจจับเรือสำหรับการวิเคราะห์ทางทะเล ในงานวิจัยนี้เทคนิค DBGS เหมาะสำหรับเรือได้รับการตรวจสอบและปรับให้เหมาะสมสำหรับการตรวจสอบและติดตามนกในวิดีโอทางทะเล นอกเหนือจากการลบพื้นหลัง รายละเอียดที่เหลือจะถูกแยกโดยลักษณะเฉพาะตามตัวบอกคุณสมบัติเพื่อลบวัตถุที่ไม่ใช่นกออก ส่วนตัวนกก็ถูกตรวจจับได้หลังที่ทำการคัดแยกเอาแต่วัตถุในกล่องสี่เหลี่ยม ซึ่งช่วยให้ประมวณผลได้เร็วและแม่นยำยิ่งขึ้น **[4]**

รูปที่ 2.12 เอาต์พุตของอัลกอริทึมที่ใช้: ไฟล์เอาต์พุต JSON และสตรีมวิดีโอที่มีคำอธิบาย

Dictionary	Kernel	Bird	Water	Time
5	Linear	92 %	86 %	91 s
5	**Intersect**	**92 %**	**90 %**	**89 s**
5	RBF	90 %	89 %	95 s
10	Linear	89 %	89 %	108 s
10	Intersect	91 %	91 %	107 s
10	RBF	91 %	92 %	107 s

รูปที่ 2.13 ผลลัพธ์ความแม่นยำและประสิทธิภาพจากอัลกอริทึม SIFT พื้นฐาน

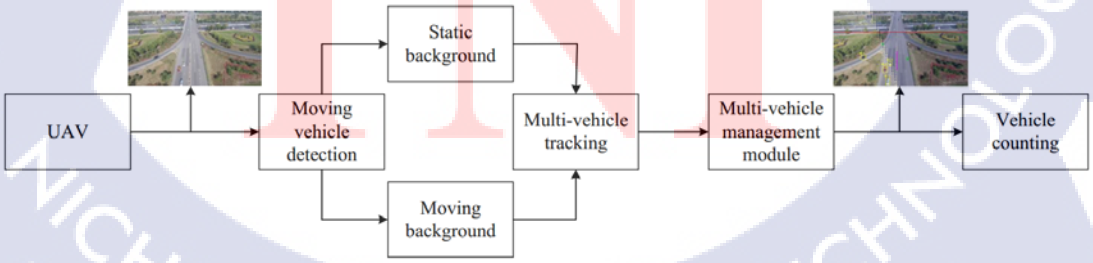
2.2.4 Swiftlets and edible bird's nest industry in Asia

นกนางแอ่นเป็นนกกินแมลงขนาดเล็กซึ่งแพร่พันธุ์ไปทั่วเอเชียตะวันออกเฉียงใต้และแปซิฟิกใต้ ในบรรดานกนางแอ่นหลายชนิดมีเพียงไม่กี่ชนิดเท่านั้นที่มีความโดดเด่นในการสร้างรังนกที่กินได้ (EBN) จากน้ำลายที่หลั่งออกมาในช่วงฤดูผสมพันธุ์ อนุกรมวิธานของนกนางแอ่นยังคงเป็นหนึ่งในสิ่งที่ถกเถียงกันมากที่สุดในสายพันธุ์นกเนื่องจากลักษณะทางสัณฐานวิทยาในสายพันธุ์มีความคล้ายคลึงกันสูง ซึ่งสามารถพบได้ทั่วเอเชียตะวันออกเฉียงใต้และแปซิฟิกใต้ โดยทั่วไปนกนางแอ่นจะสร้างอาณานิคมขนาดใหญ่ในถ้ำมืดหรือสภาพแวดล้อมที่เหมือนถ้ำเนื่องจากความสามารถในการนำทางในความมืดโดยใช้การระบุตำแหน่ง สิ่งมีชีวิตที่คัดเลือกมามีความโดดเด่นในเรื่องความสามารถในการผลิตรังที่กินได้ซึ่งส่วนใหญ่ประกอบด้วยมิวซินเช่นไกลโคโปรตีนที่มาจากการหลั่งน้ำลายซีเมนต์ นกนางแอ่นรังขาว (Aerodramus fuciphagus) และนกนางแอ่นรังดำ (Aerodramus maximus) ได้รับการยกย่อง

อย่างสูงในฐานะรังเกรดพรีเมี่ยมเนื่องจากทั้งสองชนิดเกิดขึ้นทั้งหมดหรือส่วนใหญ่มาจากการหลั่งน้ำลายโดยไม่รวมวัสดุอื่น ๆ เช่นหญ้าโคลนและขนนก ดังนั้นการเก็บเกี่ยวรังนกจึงถือเป็นอุตสาหกรรมที่มีกำไรในหลายประเทศในเอเชียตะวันออกเฉียงใต้ **[1]**

2.2.5 Vehicle Counting Based on Vehicle Detection and Tracking from Aerial Videos

การนับยานพาหนะจากอากาศยานไร้คนขับ (UAV) กำลังกลายเป็นหัวข้อวิจัยยอดนิยมในการตรวจสอบสภาพการจราจร กล้องที่ติดตั้งบน UAV ถือได้ว่าเป็นเซ็นเซอร์ภาพสำหรับรวบรวมวิดีโอทางอากาศ เมื่อเปรียบเทียบกับเซ็นเซอร์แบบเดิมแล้ว UAV สามารถปรับใช้กับพื้นที่ที่ต้องการตรวจสอบได้อย่างยืดหยุ่นและให้มุมมองที่กว้างขึ้น ในบทความนี้มีการเสนอกรอบการทำงานใหม่สำหรับการนับยานพาหนะตามวิดีโอทางอากาศ ในเฟรมเวิร์กของงานเครื่องตรวจจับวัตถุเคลื่อนที่สามารถจัดการกับสองสถานการณ์ต่อไปนี้: พื้นหลังคงที่และพื้นหลังที่เคลื่อนไหวสำหรับพื้นหลังแบบคงที่ตัวตรวจจับวิดีโอเบื้องหน้าระดับพิกเซลจะได้รับเพื่อตรวจจับยานพาหนะซึ่งสามารถอัปเดตโมเดลพื้นหลังได้อย่างต่อเนื่อง สำหรับพื้นหลังที่เคลื่อนไหวจะมีการใช้การลงทะเบียนภาพเพื่อประมาณการเคลื่อนที่ของกล้องซึ่งทำให้สามารถตรวจจับยานพาหนะได้ในระบบพิกัดอ้างอิง นอกจากนี้เพื่อเอาชนะการเปลี่ยนแปลงขนาดและรูปร่างของยานพาหนะในรูปภาพเราใช้เครื่องมือติดตามการเรียนรู้ออนไลน์ซึ่งสามารถอัปเดตตัวอย่างที่ใช้สำหรับการฝึกอบรม สุดท้ายเราออกแบบโมดูลการจัดการหลายวัตถุซึ่งสามารถวิเคราะห์และตรวจสอบสถานะของยานพาหนะที่ติดตามได้อย่างมีประสิทธิภาพด้วยเทคนิคมัลติเธรด วิธีการของเราได้รับการทดสอบกับวิดีโอทางอากาศของฉากทางหลวงจริงที่มีพื้นหลังคงที่และพื้นหลังที่เคลื่อนไหว ผลการทดลองแสดงให้เห็นว่าวิธีการที่เสนอสามารถบรรลุความแม่นยำมากกว่า 90% และ 85% ของการนับรถในวิดีโอพื้นหลังคงที่และวิดีโอพื้นหลังที่เคลื่อนไหวตามลำดับ **[13,14]**



รูปที่ 2.14 เฟรมเวิร์คของงานที่ประกอบด้วยการตรวจจับยานพาหนะการติดตามรถหลายคันการจัดการรถหลายคันและการตรวจนับรถ UAV ติดตั้งเซ็นเซอร์ภาพ

ในบทความนี้มีการเสนอกรอบการตรวจจับและติดตามยานพาหนะหลายคันตาม UAV ซึ่งสามารถใช้สำหรับการนับยานพาหนะและสามารถจัดการได้ทั้งพื้นหลังคงที่และพื้นหลังที่เคลื่อนที่ ขั้นแรก UAV จะรวบรวมลำดับภาพและส่งไปยังเครื่องตรวจจับซึ่งแบ่งออกเป็นสองส่วนคือพื้นหลังแบบคงที่และพื้นหลังที่เคลื่อนไหว เพื่อยืนยันเอกลักษณ์เฉพาะของยานพาหนะในวิดีโอลำดับยาว ยานพาหนะที่ตรวจพบทั้งหมดจะถูกติดตามโดยตัวติดตาม เพื่อจัดการยานพาหนะที่ถูกติดตามอย่างมีประสิทธิภาพและหลีกเลี่ยงความวุ่นวายในการติดตามเราได้ออกแบบโมดูลการจัดการหลายวัตถุซึ่งจัดการยานพาหนะที่ติดตามภายใต้โมดูลแบบรวมและให้ข้อมูลสถานะของยานพาหนะที่ติดตามแต่ละคันเพื่อการวิเคราะห์อัจฉริยะในภายหลัง **[14]**

2.3 ระยะเวลาการดำเนินงานวิจัย

จากการศึกษาแนวคิด ทฤษฎี และงานวิจัยที่เกี่ยวข้องเพื่อเป็นความรู้สำหรับการศึกษาที่เกี่ยวข้องกับเทคโนโลยีประมวณผลภาพในครั้งนี้ ผู้ศึกษาได้นำความรู้ที่ได้รับมาดำเนินงานตามขั้นตอนโดยแผนการดำเนินงานวิจัยมีทั้งหมด 3 ขั้นตอน ตามตารางที่ 2.1 ดังนี้

1. ขั้นตอนเบื้องต้น เป็นขั้นตอนในการหาหัวข้องานวิจัย ผู้วิจัยจึงได้สืบค้นข้อมูล ทฤษฎี และงานวิจัยที่เกี่ยวข้องโดยเลือกที่จะทำการเปรียบเทียบข้อมูลระหว่างการใช้ภาพนิ่ง (snapshot) กับ ภาพวีดีโอเฟรม (videoframe image) ที่ได้จาการถ่ายวีดีโอจากรังนกแอ่นกินรังอยู่ในคอนโดรังนกแอ่น เพื่อเก็บรวบรวมข้อมูลที่เหมาะสมต่อหัวข้อวิจัย

2. ขั้นตอนการดำเนินการ เป็นขั้นตอนในการดำเนินงานวิจัย ผู้วิจัยจึงได้ศึกษาพฤติกรรมของนกแอ่นกินรังจากเกษตรกร เตรียมอุปกรณ์และเขียนสคริปต์โดยใช้หลักการตรวจจับวัตถุ (Object Detection) สำหรับการตรวจนับจำนวนและวัดประสิทธิภาพ โดยอัลกอ-ลิธึม Yolov5 จากนั้นจึงเริ่มดำเนินการทดลอง

3. ขั้นตอนการสรุป เป็นขั้นตอนสุดท้ายในการทำงานวิจัยการสรุปผลงานวิจัยทั้งหมดที่ได้ทำมาในทั้งขั้นตอนเบื้องต้นและขั้นตอนการดำเนินการ ทั้งสองขั้นตอนข้างต้นเพื่อนำไปเขียนรูปเล่มวิทยานิพนธ์และบทความวิชาการ

ตารางที่ 2.1 แผนการดำเนินงานวิจัย

การดำเนินงาน	ระยะเวลา								
	ปี 2563				ปี 2564				
	พ.ค.	ต.ค.	พ.ย.	ธ.ค.	ม.ค.	ก.พ.	มี.ค.	เม.ย.	พ.ค.
ขั้นตอนเบื้องต้น									
การหาหัวข้อ									
การทบทวนวรรณกรรม									
การทดลองเบื้องต้น									
ร่างเค้าโครงวิทยานิพนธ์									
นำเสนอเค้าโครงวิทยานิพนธ์									
ขั้นตอนการดำเนินการ									
การเก็บข้อมูลและเตรียมข้อมูล									
การประมวลผล									
ขั้นตอนการสรุป									
สรุปและตีพิมพ์งานวิจัย									
เขียนวิทยานิพนธ์เล่มสมบูรณ์									
สอบป้องกันวิทยานิพนธ์									

บทที่ 3
วิธีการดำเนินงาน

จากการศึกษาแนวคิด ทฤษฎี และงานวิจัยที่เกี่ยวข้องเพื่อเป็นความรู้สำหรับการศึกษาที่เกี่ยวข้องกับเทคโนโลยีประมวณผลภาพในครั้งนี้ ผู้ศึกษาได้นำความรู้ที่ได้รับมาดำเนินงานตามขั้นตอนดังนี้

1. การวิเคราะห์ข้อมูลปัญหาของงานวิจัย
2. การศึกษาข้อมูลและเทคโนโลยีที่นำมาใช้
3. ออกแบบการทดลองและดำเนินการทดลอง

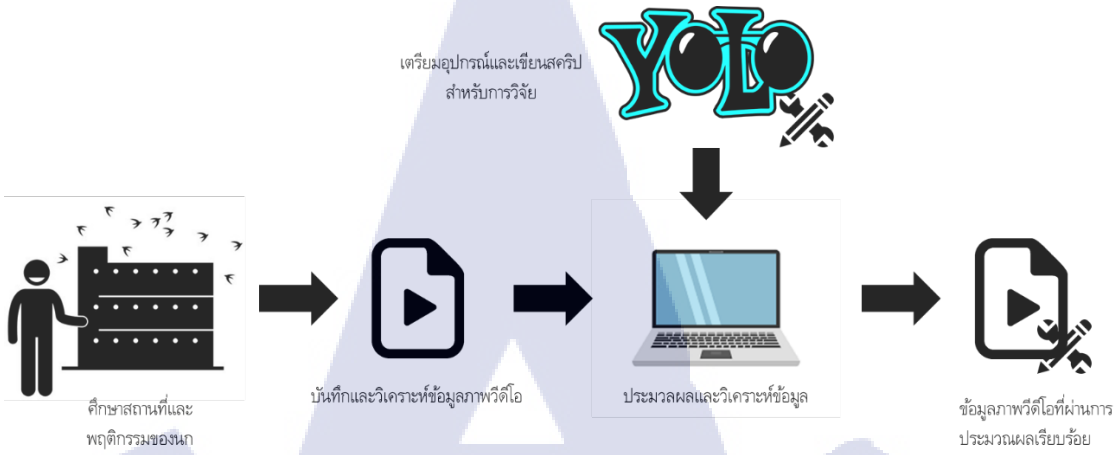
โดยแผนการดำเนินงานวิจัยมีทั้งหมด 3 ขั้นตอน

ขั้นตอนเบื้องต้น เป็นขั้นตอนในการหาหัวข้องานวิจัย ผู้วิจัยได้ทำการศึกษาและสืบค้นข้อมูล ทฤษฎีและงานวิจัยที่เกี่ยวข้องโดยเลือกที่จะทำการเปรียบเทียบข้อมูลระหว่างการใช้ภาพนิ่ง (snapshot) กับ ภาพวีดีโอเฟรม (videoframe image) ที่ได้จาการถ่ายวีดีโอจากรังนกแอ่นกินรังอยู่ในคอนโดรังนกแอ่น เพื่อเก็บรวบรวมข้อมูลที่เหมาะสมต่อหัวข้อวิจัย

ขั้นตอนการดำเนินการ เป็นขั้นตอนในการดำเนินงานวิจัย ผู้วิจัยจึงได้ศึกษาพฤติกรรมของนกแอ่นกินรังจากเกษตรกร เตรียมอุปกรณ์และเขียนสคริปต์โดยใช้หลักการตรวจจับวัตถุ (Object Detection) สำหรับการตรวจนับจำนวนและวัดประสิทธิภาพ โดยอัลกอ-ลิธึม Yolov5 จากนั้นจึงเริ่มดำเนินการทดลอง

ขั้นตอนการสรุป เป็นขั้นตอนสุดท้ายในการทำงานวิจัยการสรุปผลงานวิจัยทั้งหมดที่ได้ทำมาในทั้งขั้นตอนเบื้องต้นและขั้นตอนการดำเนินการ ทั้งสองขั้นตอนข้างต้นเพื่อนำไปเขียนรูปเล่มวิทยานิพนธ์และบทความวิชาการ

ซึ่งในการดำเนินงานวิจัยจะเป็นไปตามขั้นตอนการดำเนินงานวิจัยตามรูปที่ 3.1



รูปที่ 3.1 ขั้นตอนการดำเนินงานวิจัย

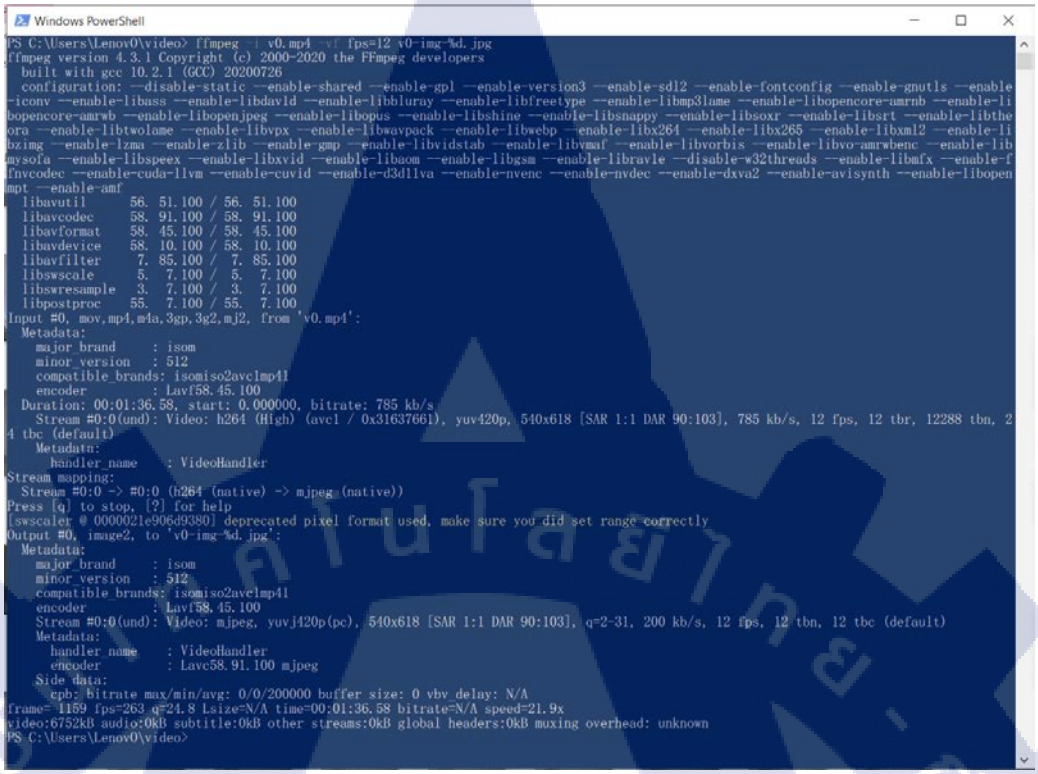
3.1 การวิเคราะห์ข้อมูลปัญหาของงานวิจัย

ผู้วิจัยจะทำการศึกษาปัญหาของงานวิจัยเพื่อศึกษาเกี่ยวกับเทคโนโลยีประมวลผลภาพและฟาร์มนกแอ่นกินรัง โดยผู้วิจัยได้ทำการวิเคราะห์ปัญหาของงานร่วมกับอาจารย์ที่ปรึกษาซึ่งมีความสนใจในเทคโนโลยีประมวลผลภาพอยู่ก่อนหน้านี้แล้ว ซึ่งเป็นเทคโนโลยีที่ใหม่ที่สามารถใช้งานได้จริงและมีความน่าสนใจที่นำมาศึกษาเพื่อการวิจัย โดยในช่วงปี 2563 ที่ฟาร์มนกแอ่นกินรังยังไม่ได้มีการให้เข้าชม และการยอมรับได้อย่างเป็นทางการทำให้ยากต่อการเก็บข้อมูล ภาพและวิดีโอ อีกทั้งยังไม่สามารถนำอุปกรณ์เข้าไปติดตั้งในสถานที่ที่ต้องการได้อีก จึงส่งผลให้การวัดประสิทธิภาพการทำงานในแต่ละครั้งมีผลลัพธ์ออกมาต่างกัน ซึ่งข้อมูลวีดีโอที่ได้รับมาจากความร่วมมือของอาจารย์ที่ปรึกษาทำให้ผู้วิจัยสามารถนำข้อมูลมาวิเคราะห์หาปัญหาของงานวิจัยต่อไปได้ จากนั้นผู้วิจัยจึงได้ข้อสรุปออกมาว่า จะใช้หลักการการตรวจจับวัตถุซึ่งเป็น 1 ในเทคโนโลยีการทำประมวลผลภาพเพื่อใช้ในการตรวจจับวัตถุเคลื่อนที่ในอากาศ ซึ่งมีผลต่อการนับจำนวนนกแอ่นกินรังที่บินอยู่ในฟาร์มในแต่ละวัน

จากการศึกษาทำให้ผู้วิจัยได้ปรึกษาและเสนอหัวข้อกับอาจารย์ที่ปรึกษาที่จะเลือกทำงานวิจัยใน “ประสิทธิภาพของระบบตรวจจับวัตถุเคลื่อนที่ กรณีศึกษา ฟาร์มนกแอ่นกินรัง” โดยการนำข้อมูลภาพวิดีโอที่มีนกแอ่นกินรังอยู่เคลื่อนที่ภายในพื้นที่ ซึ่งที่มาข้อมูลนั้นได้รับความกรุณามาจากอาจารย์ที่ปรึกษา แล้วมาทำการศึกษาข้อมูลเพื่อค้นหาเทคนิคหรือเทคโนโลยีที่จะสามารถวิเคราะห์ข้อมูลที่มีได้อย่างเหมาะสม และผู้วิจัยได้ทำการนำหลักการตรวจจับวัตถุของ YOLOv5 ที่มีความโดดเด่นในการประมวณผลที่รวดเร็วมาใช้ในการตรวจจับวัตถุเคลื่อนที่ เพื่อทำการนับจำนวนนกภายในพื้นที่ที่ต้องการ

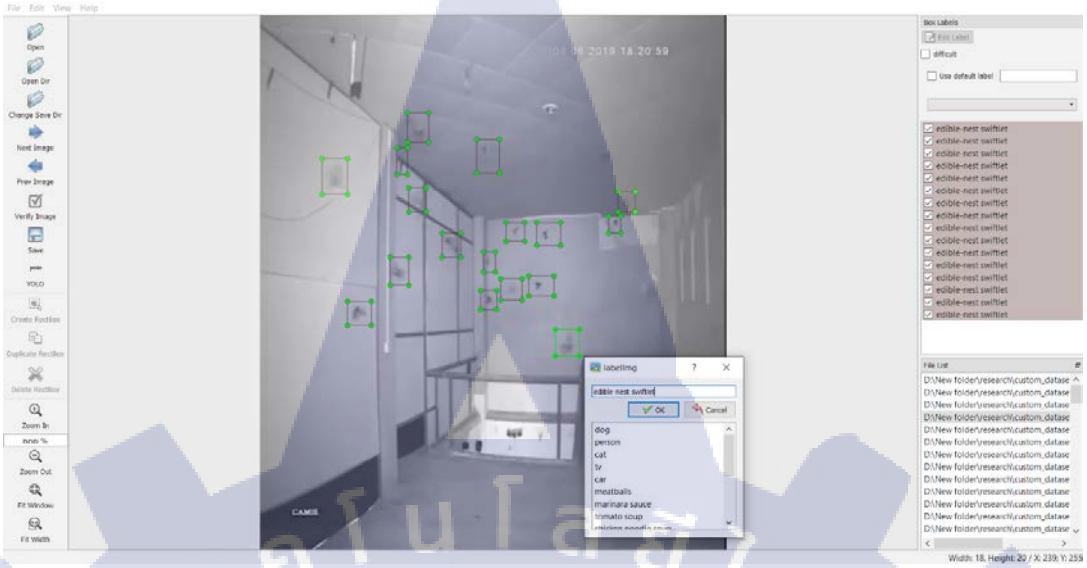
3.2 การศึกษาข้อมูลและเทคโนโลยีที่นำมาใช้

จากบันทึกวีดีโอจะพบว่า จากพื้นที่ภายในห้องทิศทางการบินของนกแอ่นกินรังนั้นไม่มีความคงที่ ไม่เป็นเส้นตรงชัดเจน บางครั้งนกแอ่นกินรังก็บินไปเกาะตามผนังกำแพง เสา มุมมืดของห้อง ทำให้มีความลำบากในการนับจำนวนนกที่เข้าออกรัง ผู้วิจัยจึงคิดว่าควรที่จะนับจำนวนนกแอ่นกินรังที่บินอยู่ หรือเกาะตามผนังที่สามารถมองเห็นได้ชัดเจน โดยจะทำการนับจากจำนวนที่ตรวจจับได้ภายในบันทึกวีดีโอ จากนั้นจึงทำการศึกษาเกี่ยวกับกระบวนการทำงานของเทคโนโลยีการตรวจจับวัตถุ และศึกษาเกี่ยวกับตัวแปรที่ส่งผลต่อประสิทธิภาพการทำงาน เช่น ปัจจัยที่จะทำให้การนับจำนวนมีความผิดพลาดได้ ค่าตัวเลขที่ไม่ตรงตามความเป็นจริง เป็นต้น จากการเปรียบเทียบการใช้งานข้อมูล 2 ชนิดระหว่างภาพนิ่ง กับ บันทึกวีดีโอ แล้วจึงเลือกใช้ภาพวีดีโอเฟรมด้วยเหตุผลที่ว่า การนำภาพนิ่งมาใช้ในการสร้างแบบจำลองนั้นจะเหมาะกับการตรวจจับวัตถุในภาพนิ่งมากกว่าในขณะที่การนำบันทึกวีดีโอมาใช้ในการสร้างแบบจำลองนั้นเหมาะกับการตรวจจับวัตถุในบันทึกวีดีโอ อีกทั้งพื้นที่ห้องที่จะนำมาทดลองการตรวจจับวัตถุควรจะเป็นห้องที่มีลักษณะที่ใกล้เคียงกันด้วยซึ่งอาจจะเป็นช่วงเวลาที่แตกต่างกันได้ ซึ่งการตรวจจับวัตถุนี้จะใช้หลักการ Object detection ในการทำงานโดยนำ Yolov5 ที่มีความโดดเด่นในด้านความเร็วในการการตรวจจับและประมวณผลมาใช้งานซึ่งเขียนด้วยโปรแกรมภาษา Python โดยไฟล์บันทึกวีดีโอ(*.mp4)จะไม่สามารถทำการ Image Labeling ได้ในไฟล์บันทึกวีดีโอ จึงจะต้องนำไฟล์บันทึกวีดีโอมาแปลงเป็นไฟล์รูปภาพ(*.jpg)เสียก่อนซึ่งจะแบ่งจำนวนเฟรมต่อวินาที ตามรูปที่ 3.2 โดยการใช้ ffmpeg



รูปที่ 3.2 การแปลงไฟล์ .mp4 เป็นไฟล์ .jpg

จากนั้นให้นำไฟล์รูปภาพที่ได้นำมาทำการ Image Labeling ตามรูปที่ 3.3 เพื่อใช้ในการตรวจจับสิ่งของหรือวัตถุ (Object) ที่อยู่ในรูปภาพซึ่งจำเป็นต่อการสร้างแบบจำลอง และให้เครื่องมือมีการเรียนรู้ รูปร่าง ลักษณะของวัตถุที่ผู้วิจัยต้องการตรวจนับจำนวน ซึ่งจะจำแนกคลาสของวัตถุเพื่อให้สามารถคัดแยกวัตถุที่ต้องการให้ชัดเจนมากขึ้น

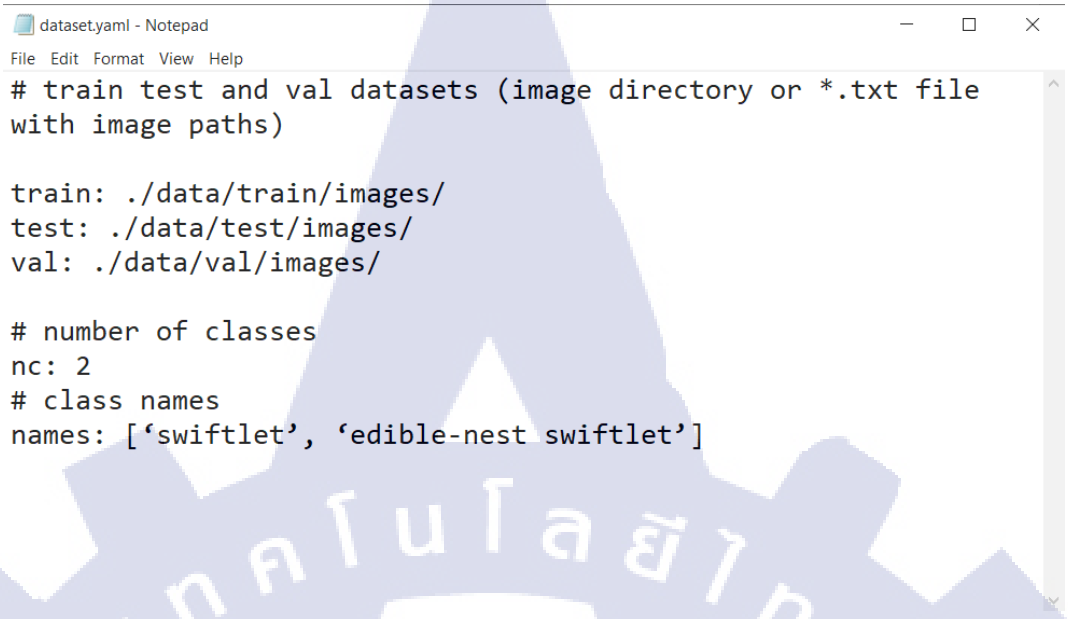


รูปที่ 3.3 การทำ Image Labeling

3.3 ออกแบบการทดลองและดำเนินการทดลอง

ผู้วิจัยได้ทำการออกแบบลำดับกระบวนการทำงานของระบบ เริ่มต้นผู้วิจัยได้ทำการแบ่งข้อมูลภาพที่ผ่านการทำ Image Labeling เป็นข้อมูล Training ,Validation และ Test โดยการใช้ข้อมูลรูปจำนวนทั้งหมด 1559 รูป ด้วยอัตราส่วนดังนี้

- Training Dataset = 75% ของ Dataset (1159 รูป)
- Validation Dataset = 35% ของ Training Dataset (400 รูป)
- Test Dataset = 25% ของ Dataset (400 รูป)



รูปที่ 3.4 ตัวอย่างไฟล์ Dataset config ในไฟล์ .yaml

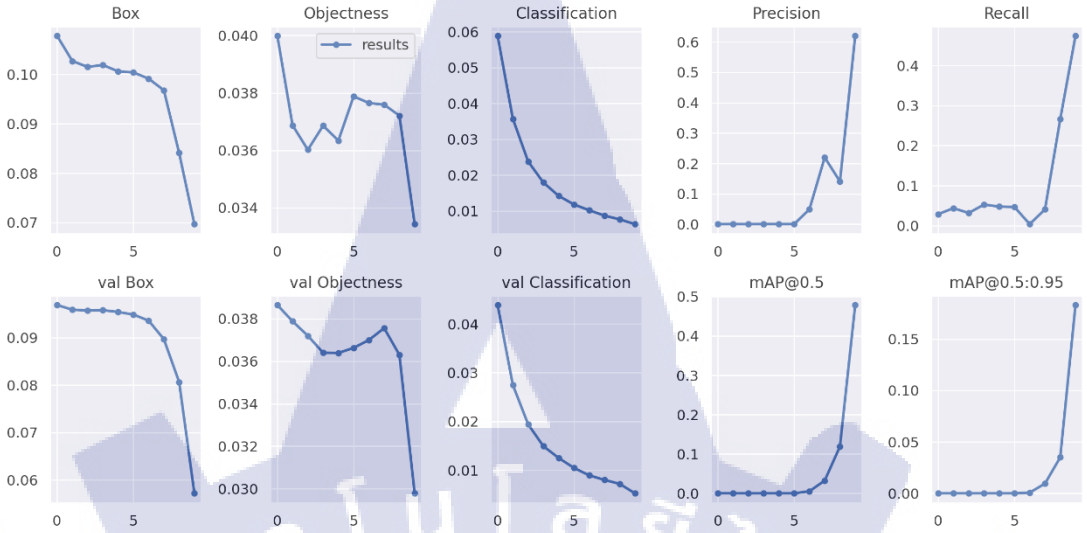
ในการสร้างแบบจำลองการตรวจจับวัตถุโดย Yolov5 เพื่อใช้ในการตรวจนับจำนวนของนกแอ่นกินรังในฟาร์มนั้น จะต้องทำการป้อนคำสั่งเพื่อสร้างค่า weights เริ่มต้นสำหรับการสร้างแบบจำลอง Yolov5 โดยจะมีการตั้งค่าข้อมูลที่จะนำมาสร้างแบบจำลอง ขนาดของโมเดล จำนวนรอบของการเรียนรู้ในแต่ละครั้ง พื้นที่ใช้ประมวณผล ที่มีการป้อนคำสั่งตามรูปที่ 3.5 และ 3.6 และควรนำ GPU มาช่วยประมวณผลในแต่ละครั้งด้วย

```
$ python train.py --data coco.yaml --cfg yolov5s.yaml --weights '' --batch-size 64
                                         yolov5m                                 40
                                         yolov5l                                 24
                                         yolov5x                                 16
```

รูปที่ 3.5 ตัวอย่างคำสั่งการตั้งค่า train.py

```
!python train.py --data custom_dataset/custom_data.yaml --cfg
models/yolov5s.yaml --weights '' --epochs 100 --batch-size 16
```

รูปที่ 3.6 ตัวอย่างคำสั่งการสร้างค่า weights เริ่มต้นของแบบจำลอง



รูปที่ 3.7 ตัวอย่างผลลัพธ์จากคำสั่งการสร้างค่า weights เริ่มต้น

จากนั้นเมื่อเราได้ค่า weights เริ่มต้นแล้วควรดูผลลัพธ์ที่ได้จากการสร้างค่า weights เริ่มต้นก่อน เพื่อกำหนดจำนวนพารามิเตอร์ที่จะใช้ครั้งต่อไปโดยดูจากรูปที่ 3.7 แล้วจึงทำการป้อนคำสั่งสร้างค่า weights ของแบบจำลองที่ใช้สำหรับการตรวจจับนกแอ่นกินรังในพื้นที่ที่ต้องการ และกำหนดจำนวนรอบของการเรียนรู้เพิ่มขึ้นให้เพียงพอ เพราะถ้าหากจำนวนรอบการเรียนรู้ครั้งนี้น้อยเกินไป ความแม่นยำและความถูกต้องอาจมีความคลาดเคลื่อนได้ ซึ่งจะทำการป้อนคำสั่งสร้างค่า weights สำหรับการสร้างแบบจำลอง Yolov5 นี้ ด้วยการเพิ่มพารามิเตอร์จากรูปที่ 3.8 โดยครั้งนี้จะใช้ค่า weights เริ่มต้นที่ได้มาก่อนหน้านี้โดยป้อนคำสั่งตามรูปที่ 3.9 ซึ่งมีเป็นแบบจำลองต้นแบบที่ใช้ตรวจจับวัตถุที่ได้ออกมา 2 ชุด ดังนี้

- ./weights/last.pt : แบบจำลองที่ได้ครั้งล่าสุด
- ./weights/best.pt : แบบจำลองที่ค่าการทำงานดีที่สุด

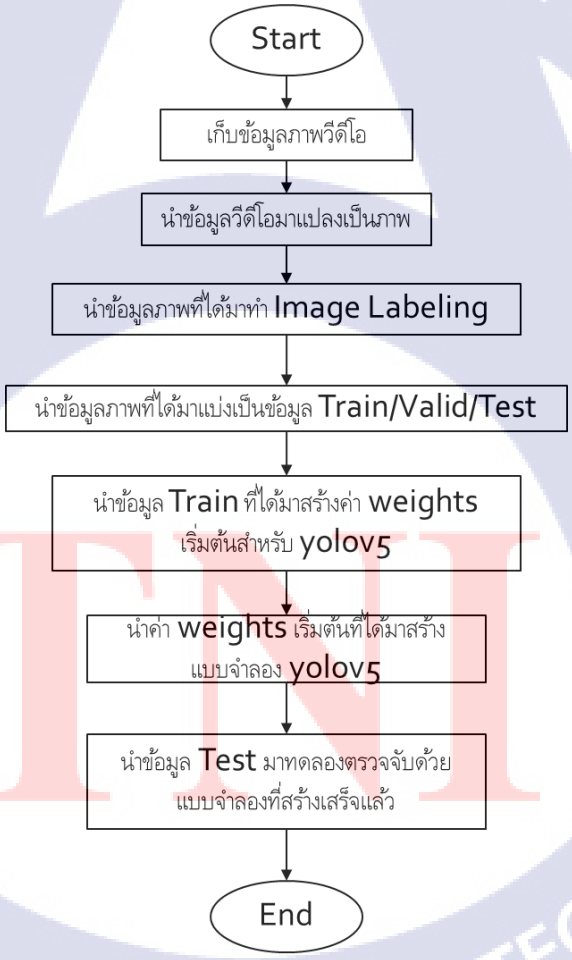
โดยไฟล์ best.pt นี้คือแบบจำลองที่เราจะนำไปใช้สร้างแบบจำลองที่สมบูรณ์และมีค่าการทำงานที่ดีที่สุด เหตุผลที่ว่าทำไมจึงไม่นำค่า weights เริ่มต้นมาใช้งานทันที เพราะว่า ค่าแม่นยำที่ได้เริ่มต้นยังสูงมาก อันเนื่องจากจำนวนรอบการเรียนรู้ที่น้อย อีกทั้งการใช้จำนวนรอบการเรียนรู้ที่มากในการสร้างค่า weights เริ่มต้น จะทำให้การสร้างแบบจำลองที่สมบูรณ์ใช้เวลามากขึ้น

```
--img <> : ขนาดของ Image ขาเข้า(มีการปรับ Resolution ของ Image)
--batch <> : จำนวน Batch size
--epochs <> : จำนวน Epochs
--data <> : Path ที่ชี้ไปยังไฟล์ Data config
--cfg <> : Path ที่ชี้ไปยังไฟล์โครงสร้างโมเดลที่เลือกใช้
--weights <> : Path ที่ชี้ไปยังไฟล์ Pretrained model สำหรับทำ Transfer learning
```

รูปที่ 3.8 ตัวอย่างรายละเอียดของแต่ละ Arguments ภายในคำสั่ง

```
!python train.py --data custom_dataset/custom_data.yaml --cfg
yolov5s.yaml --weights runs/train/exp4/weights/best.pt --epochs 500
```

รูปที่ 3.9 ตัวอย่างคำสั่งการสร้างค่า weights โดยใช้ค่า weights เริ่มต้น

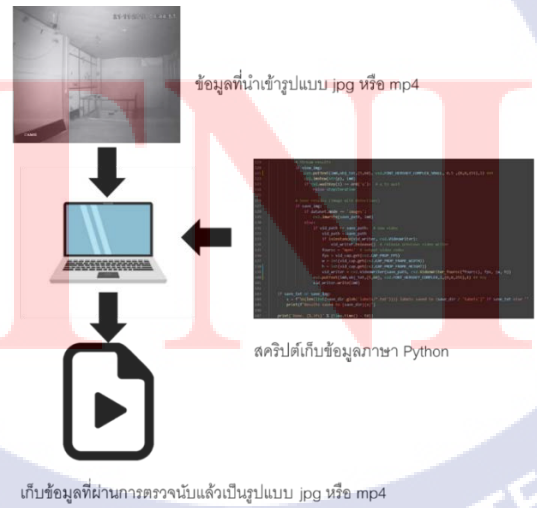


รูปที่ 3.10 Flow Chart ขั้นตอนแสดงการทำงานของระบบ

โดยในรูปที่ 3.10 แสดง Flowchart ขั้นตอนการทำงานของระบบ โดยจะอธิบายตั้งแต่ขั้นตอนการเก็บรวบรวมข้อมูลภาพวีดีโอต่างๆจนถึงการนำข้อมูลมาทดสอบประสิทธิภาพของแบบจำลองที่ได้ซึ่งผู้วิจัยใช้สคริปต์เดียวกับรูปที่ 3.11 ในการแสดงผลของการตรวจจับให้ออกมาเป็นจำนวนวัตถุที่พบ และบันทึกวีดีโอที่ผ่านการทดลองแล้วในรูปแบบไฟล์ที่นำมาทดลองตามรูปที่ 3.12 โดยสคริปต์แบบเต็มจะมีอยู่ในภาคผนวก

```
# Stream results
if view_img:
    cv2.putText(im0,obj_txt,(5,60), cv2.FONT_HERSHEY_COMPLEX_SMALL, 0.5 ,(0,0,255),1) ### Show text
    cv2.imshow(str(p), im0)
    if cv2.waitKey(1) == ord('q'):  # q to quit
        raise StopIteration

# Save results (image with detections)
if save_img:
    x = x + 1
    if dataset.mode == 'images':
        cv2.imwrite(save_path, im0)
    else:
        if vid_path != save_path:  # new video
            vid_path = save_path
            if isinstance(vid_writer, cv2.VideoWriter):
                vid_writer.release()  # release previous video writer
            fourcc = 'mp4v'  # output video codec
            fps = vid_cap.get(cv2.CAP_PROP_FPS)
            w = int(vid_cap.get(cv2.CAP_PROP_FRAME_WIDTH))
            h = int(vid_cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
            vid_writer = cv2.VideoWriter(save_path, cv2.VideoWriter_fourcc(*fourcc), fps, (w, h))
    cv2.putText(im0,obj_txt,(5,30), cv2.FONT_HERSHEY_COMPLEX_SMALL,1,(0,0,255),1) ## Show text
    cv2.putText(im0,obj_avg,(5,50), cv2.FONT_HERSHEY_COMPLEX_SMALL,1,(0,0,255),1) ## Show text
```

รูปที่ 3.11 ตัวอย่างสคริปต์การแสดงจำนวนวัตถุที่ตรวจพบของระบบการตรวจจับ



รูปที่ 3.12 ภาพรวมการวิเคราะห์ข้อมูลของระบบ

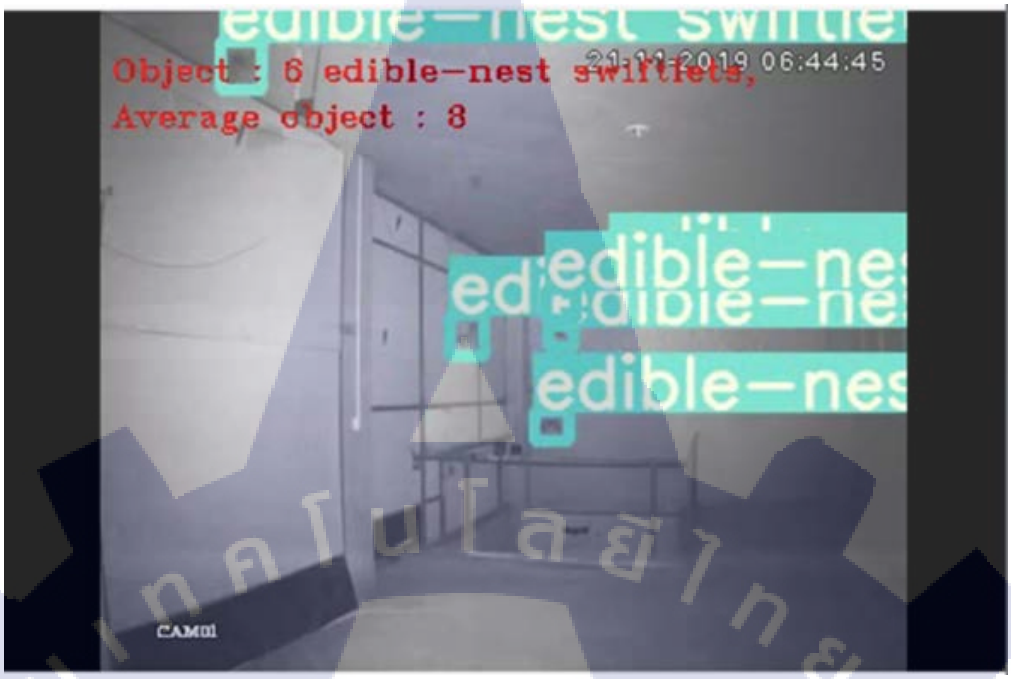
ในทำการป้อนคำสั่งการตรวจจับและนับจำนวนนกแอ่นกินรัง โดยข้อมูลที่จะนำมาการตรวจจับและวิเคราะห์ด้วยการกำหนดพารามิเตอร์ตามรูปที่ 3.13 แล้วสั่งเริ่มการทำงาน หลังจาก ป้อนคำสั่งตามที่รูปที่ 3.14 ระบบจะทำการวิเคราะห์และบันทึกข้อมูลการตรวจจับแล้วจึงทำการนับจำนวนนกแอ่นกินรังภายในบันทึกวิดีโอซึ่งจะแสดงข้อมูลตัวเลขในภาพวิดีโอตามรูปที่ 3.15

```
$ python detect.py --source 0  # webcam
                           file.jpg  # image
                           file.mp4  # video
                           path/  # directory
                           path/*.jpg  # glob
                           rtsp://170.93.143.139/rtplive/470011e600ef003a004ee33696235daa  # rtsp stream
                           rtmp://192.168.1.105/live/test  # rtmp stream
                           http://112.50.243.8/PLTV/88888888/224/3221225900/1.m3u8  # http stream
```

รูปที่ 3.13 ตัวอย่างคำสั่งการใช้งาน detect.py

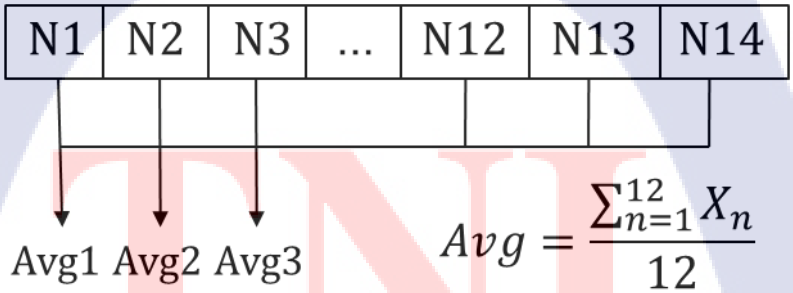
```
1 !python detect.py --source custom_dataset/valid/images --weights runs/train/exp3/weights/best.pt --conf 0.6 --save-txt

Namespace(agnostic_nms=False, augment=False, classes=None, conf_thres=0.4, device='', exist_ok=False, img_size=640, iou_thres=0.45,
YOLOv5 v4.0-102-g71dd276 torch 1.7.0+cu101 CUDA:0 (Tesla T4, 15109.75MB)

Fusing layers...
Model Summary: 224 layers, 7094365 parameters, 0 gradients, 16.5 GFLOPS
image 1/189 /content/yolov5/custom_dataset/valid/t/v1-img-1.jpg: 640x576 18 edible-nest swiftlets, Done. (0.013s)
image 2/189 /content/yolov5/custom_dataset/valid/t/v1-img-10.jpg: 640x576 22 edible-nest swiftlets, Done. (0.013s)
image 3/189 /content/yolov5/custom_dataset/valid/t/v1-img-100.jpg: 640x576 15 edible-nest swiftlets, Done. (0.011s)
image 4/189 /content/yolov5/custom_dataset/valid/t/v1-img-11.jpg: 640x576 20 edible-nest swiftlets, Done. (0.011s)
image 5/189 /content/yolov5/custom_dataset/valid/t/v1-img-12.jpg: 640x576 19 edible-nest swiftlets, Done. (0.011s)
image 6/189 /content/yolov5/custom_dataset/valid/t/v1-img-13.jpg: 640x576 21 edible-nest swiftlets, Done. (0.011s)
image 7/189 /content/yolov5/custom_dataset/valid/t/v1-img-14.jpg: 640x576 21 edible-nest swiftlets, Done. (0.011s)
image 8/189 /content/yolov5/custom_dataset/valid/t/v1-img-15.jpg: 640x576 19 edible-nest swiftlets, Done. (0.012s)
image 9/189 /content/yolov5/custom_dataset/valid/t/v1-img-16.jpg: 640x576 19 edible-nest swiftlets, Done. (0.011s)
image 10/189 /content/yolov5/custom_dataset/valid/t/v1-img-17.jpg: 640x576 22 edible-nest swiftlets, Done. (0.011s)
```

รูปที่ 3.14 ตัวอย่างคำสั่งการตรวจจับและนับจำนวนนกแอ่นกินรัง



รูปที่ 3.15 ตัวอย่างการแสดงผลการตรวจจับวัตถุ

จากรูปที่ 3.15 วิธีการคำนวณหาจำนวนนกเฉลี่ยนั้น จะนำเอาจำนวนนกในแต่ละมารวมกันแล้วไปกับจำนวนเฟรมทั้งหมดต่อวินาที และแสดงค่าเฉลี่ยออกมาตามรูปที่ 3.16



รูปที่ 3.16 วิธีการคำนวณหาจำนวนนกเฉลี่ย

จากผลลัพธ์การวิจัย ทำให้ผู้วิจัยนำข้อมูลที่ได้จากการทดลองมาทำการสรุปเป็นรายงานในแต่ละวันและทำการเปรียบเทียบกันว่าในแต่ละส่วนมีความแม่นยำมากน้อยเพียงใด มีผลลัพธ์เป็นอย่างไร และค่า weights จากพื้นที่นี้ สามารถนำไปใช้งานได้อย่างไรในแต่ละจุด โดยจะป้อนคำสั่งการประเมินประสิทธิภาพของแบบจำลองที่มีค่า weights ดีที่สุด โดยใช้ข้อมูล Test เพื่อทำการวัดผล

และประเมินประสิทธิภาพของแบบจำลองว่ามีค่าความถูกต้องมากเพียงพอต่อการนำมาใช้งานในสถานที่จริงแค่ไหน โดยสามารถตรวจสอบค่าความถูกต้องและค่าอื่นๆได้จากรูปที่ 3.17

รูปที่ 3.17 ตัวอย่างคำสั่งการประเมินประสิทธิภาพของแบบจำลอง

และในส่วนของการสรุปผลการวิจัย จะกล่าวถึง สรุปผลที่ได้จากการทดลองว่าการนับจำนวนมีประสิทธิภาพเท่าไรและข้อมูลที่นำมาใช้ในการวิจัยมีผลต่อการทำงานอย่างไร เพื่อนำมาใช้ในการพัฒนาระบบในฟาร์มต่อไป และเปรียบเทียบกันว่าในแต่ละส่วนมีความแม่นยำมากน้อยเพียงใด มีผลลัพธ์เป็นอย่างไรจากแบบจำลองที่ใช้ค่า weights นี้สามารถนำไปใช้งานในแต่ละจุดได้อย่างไร แล้วเก็บเป็นข้อมูลวิจัยต่อไป

บทที่ 4
ผลการวิจัย

จากการศึกษาแนวคิด ทฤษฎี และงานวิจัยที่เกี่ยวข้องเพื่อเป็นความรู้สำหรับการศึกษาในครั้งนี้ ผู้ศึกษาได้ทำการทดลองเพื่อศึกษาประสิทธิภาพของอัลกอลิทึม Yolov5 โดยได้เก็บผลการทดลอง นำข้อมูลที่ได้รับมาสรุป และนำข้อมูลมาวิเคราะห์เพิ่มเติม โดยแบ่งเป็น 2 ส่วนได้ดังนี้

4.1 ตารางแสดงผลลัพธ์การวิจัย

4.2 สรุปผลผลลัพธ์จากการวิจัย

4.3 วิเคราะห์ผลลัพธ์การวิจัย

4.1 ตารางแสดงผลลัพธ์การวิจัย

จากการนำข้อมูลภาพวิดีโอของฟาร์มนกแอ่นกินรังที่บันทึกได้มาทำการ labeling เพื่อใช้ในการสร้างแบบจำลองสำหรับการตรวจจับวัตถุในภาพวิดีโอ โดยจะนำแบบจำลองที่ได้มาทำการวัดความแม่นยำในการตรวจจับวัตถุจากภาพวิดิโอขนาด 640 x 640 ด้วยเวลาไม่เกิน 13 วินาที ซึ่งภาพวีดีโอเฟรมจำนวน 400 รูป กับเป้าหมายในการตรวจจับ 3563 จุด จึงได้ค่า mean Average Precision (mAP) อยู่ที่ประมาณ 92 % และค่าอื่นๆ ดังตารางที่ 4.1

ตารางที่ 4.1 ค่า Precision, Recall และ ค่าความถูกต้องของแบบจำลอง

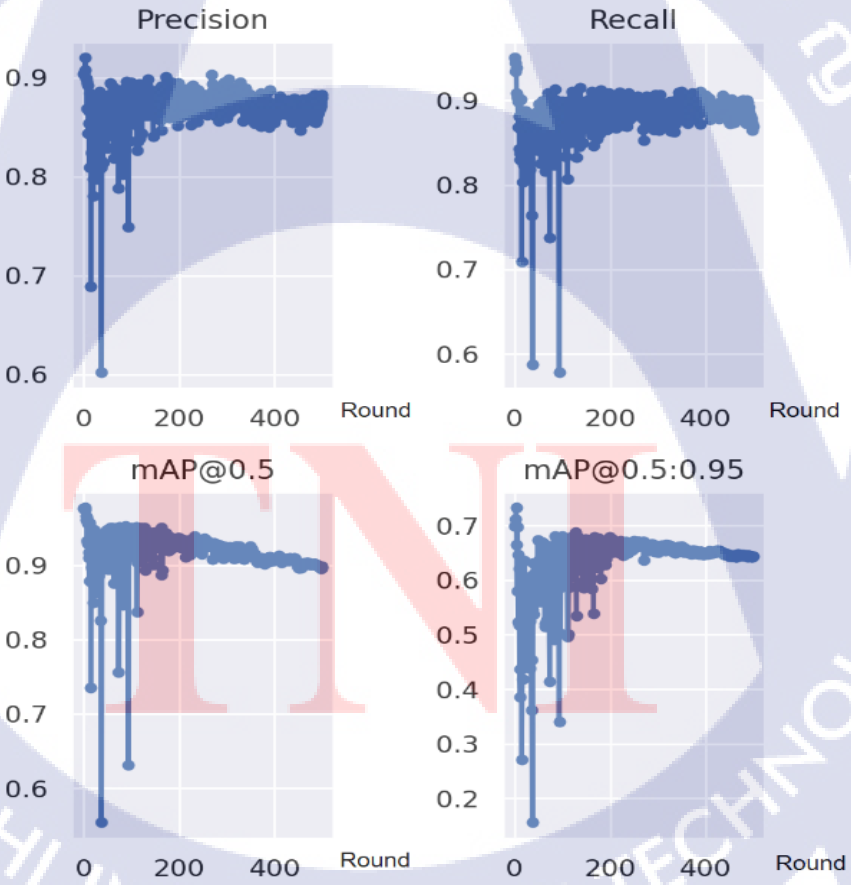
Object Class	Images	Labels	Precision	Recall	mAP@.5	mAP@.5:.95:
edible-nest swiftlet	400	3563	0.974	0.956	0.919	0.829
all	400	3563	0.974	0.956	0.919	0.829

4.2 สรุปผลผลลัพธ์จากการวิจัย

จากข้อมูลภาพวิดีโอของฟาร์มนกแอ่นกินรังที่บันทึกได้นั้น มีองค์ประกอบที่แตกต่างกันมากเช่น สถานที่ มุมมองของกล้อง เวลา เป็นต้น ทำให้โมเดลที่ได้มีการใช้งานแตกต่างออกไปตามสถานที่และมุมมองของกล้อง ซึ่งถ้าหากนำแบบจำลองที่ผ่านการ Train มาใช้ในสถานที่ซึ่งองค์ประกอบไม่สอดคล้องกัน อาจทำให้ผลลัพธ์มีความคลาดเคลื่อนเป็นอย่างมากและไม่สามารถนำมาใช้งานได้จริง อีกทั้งการนำข้อมูลภาพวิดีโอมาใช้กับแบบจำลองดั้งเดิมที่ไม่ได้มีการปรับปรุงแก้ไขและพัฒนาให้เหมาะกับการตรวจจับนกแอ่นกินรังนั้น จะไม่สามารถระบุประเภทของวัตถุและตรวจจับ

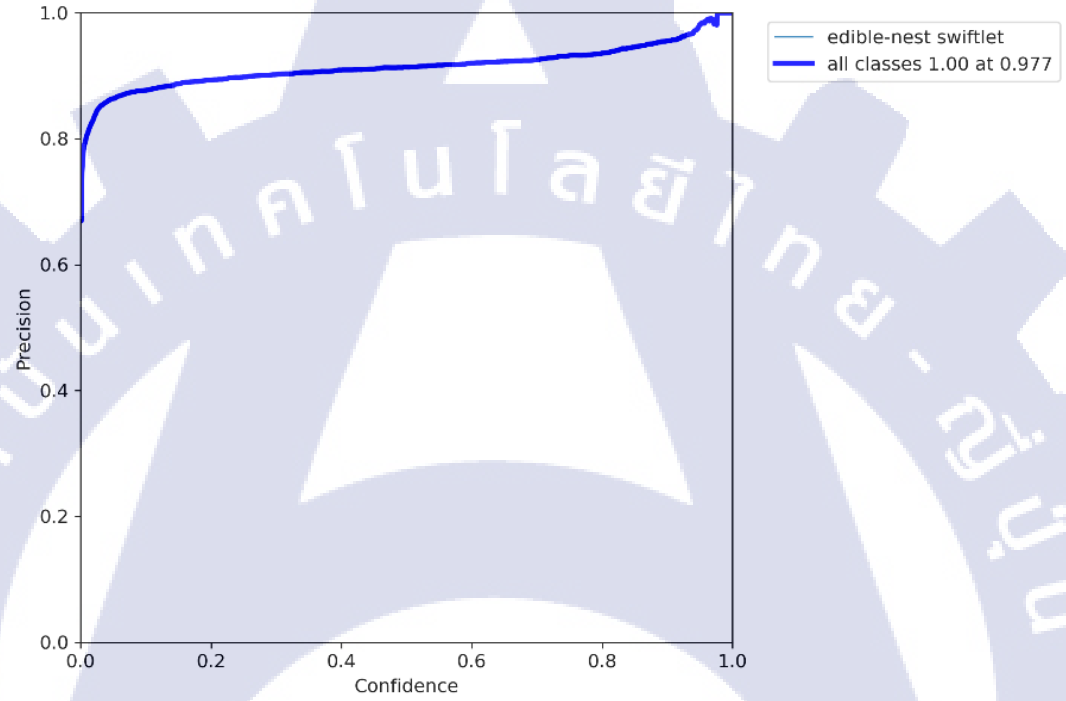
วัตถุได้ โดยดูจากการเปรียบเทียบระหว่างแบบจำลองดั้งเดิมกับแบบจำลองที่ผ่านการ Train ด้วยข้อมูลที่เหมาะสมและเพิ่มวิธีการนับจำนวนตามรูปที่ 4.1

รูปที่ 4.1 การเปรียบเทียบระหว่างแบบจำลองดั้งเดิมกับแบบจำลองที่ผ่านการ Train ด้วยข้อมูลที่เหมาะสมและเพิ่มวิธีการนับจำนวน



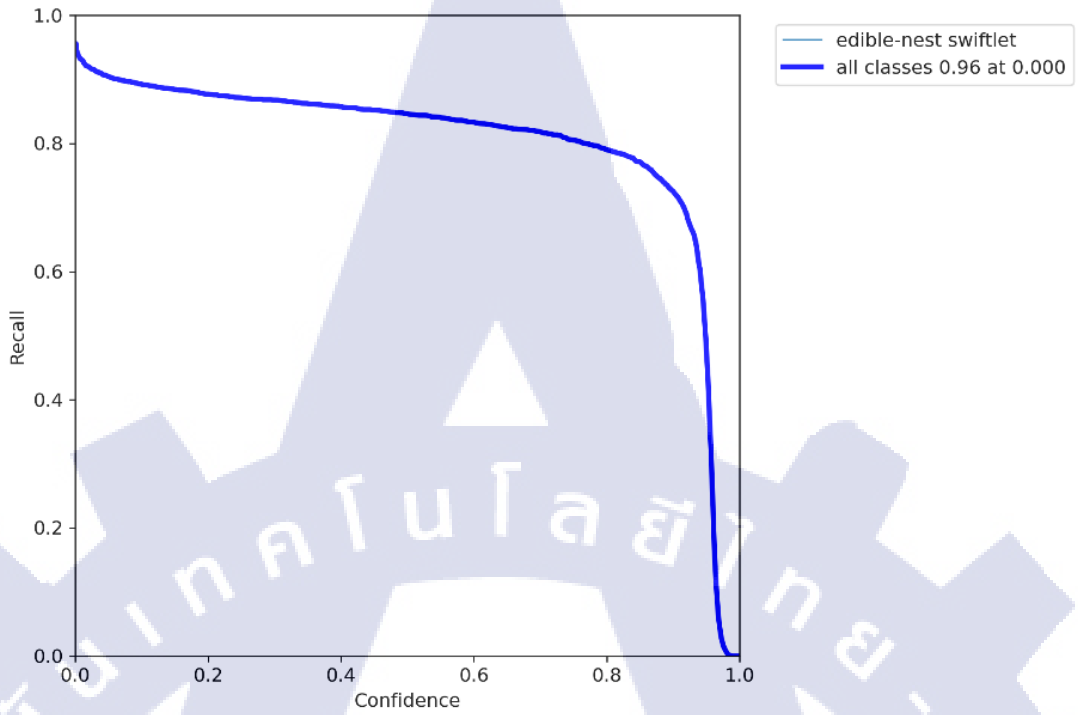
รูปที่ 4.2 กราฟแสดงผลลัพธ์จากการสร้างแบบจำลอง

จากรูปที่ 4.2 ผลลัพธ์การวิจัยจากการทำการสร้างแบบจำลอง เพื่อนำมาทำการสร้างแบบจำลองสำหรับการตรวจจับวัตถุและนับจำนวนนกแอ่นกินรัง โดยใช้การตั้งค่าตามรูปที่ 3.9 ผลการวิจัยพบว่า ค่าความถูกต้องเฉลี่ยสูงสุดที่ได้จะอยู่ที่ประมาณ 95 % จากค่าความถูกต้องเฉลี่ยต่ำสุดประมาณ 20 % และเพิ่มขึ้นตามจำนวนรอบการเรียนรู้สำหรับการสร้างแบบจำลอง โดยค่าความถูกต้องเฉลี่ยจะเริ่มคงที่อยู่ในช่วง 85 – 95 % หลังจากผ่านการเรียนรู้ประมาณ 100 รอบ



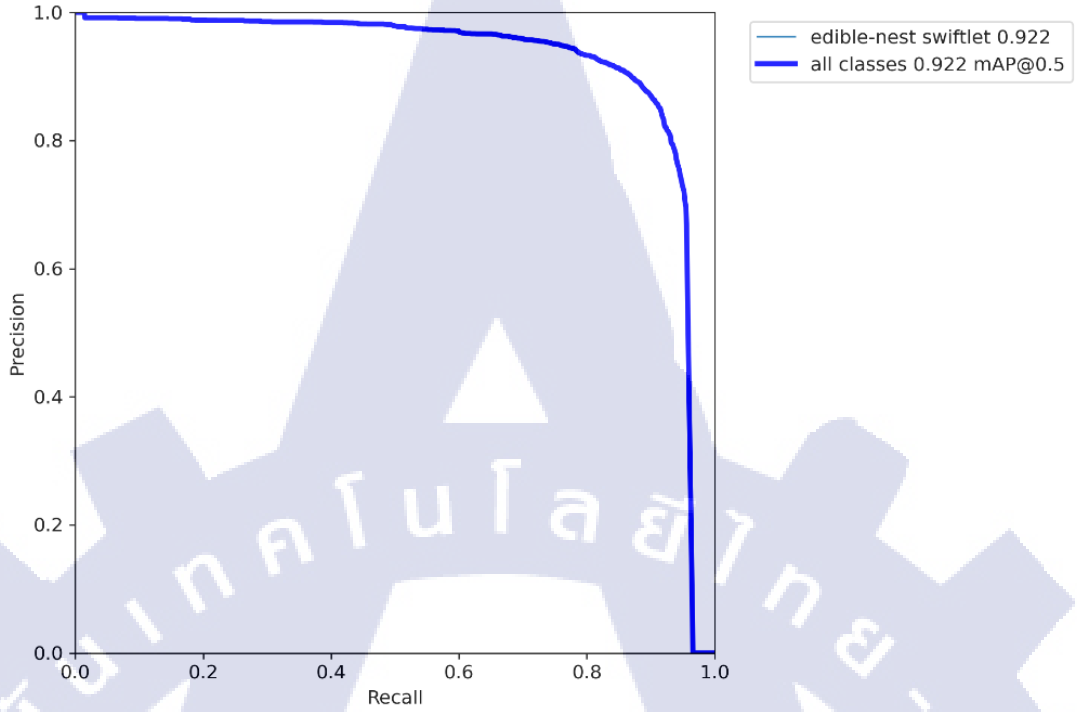
รูปที่ 4.3 กราฟแสดงค่า Precision จากการทำการสร้างแบบจำลอง

จากรูปที่ 4.3 ผลลัพธ์การวิจัยจากการทำการสร้างแบบจำลอง เพื่อนำมาทำการสร้างแบบจำลองสำหรับการตรวจจับวัตถุและนับจำนวนนกแอ่นกินรัง โดยผลการวิจัยพบว่า ค่า Precision สูงสุดที่ได้จะอยู่ที่ประมาณ 97 % จากค่า Precision ต่ำสุดประมาณ 70 % และเพิ่มขึ้นตามจำนวนรอบการเรียนรู้สำหรับการสร้างแบบจำลอง



รูปที่ 4.4 กราฟแสดงค่า Recall จากการทำการสร้างแบบจำลอง

จากรูปที่ 4.4 ผลลัพธ์การวิจัยจากการทำการสร้างแบบจำลอง เพื่อนำมาทำการสร้างแบบจำลองสำหรับการตรวจจับวัตถุและนับจำนวนนกแอ่นกินรัง โดยผลการวิจัยพบว่า ค่า Recall สูงสุดที่ได้จะอยู่ที่ประมาณ 96 % จากค่า Recall ต่ำสุดประมาณ 20 % และเพิ่มขึ้นตามจำนวนรอบการเรียนรู้สำหรับการสร้างแบบจำลอง

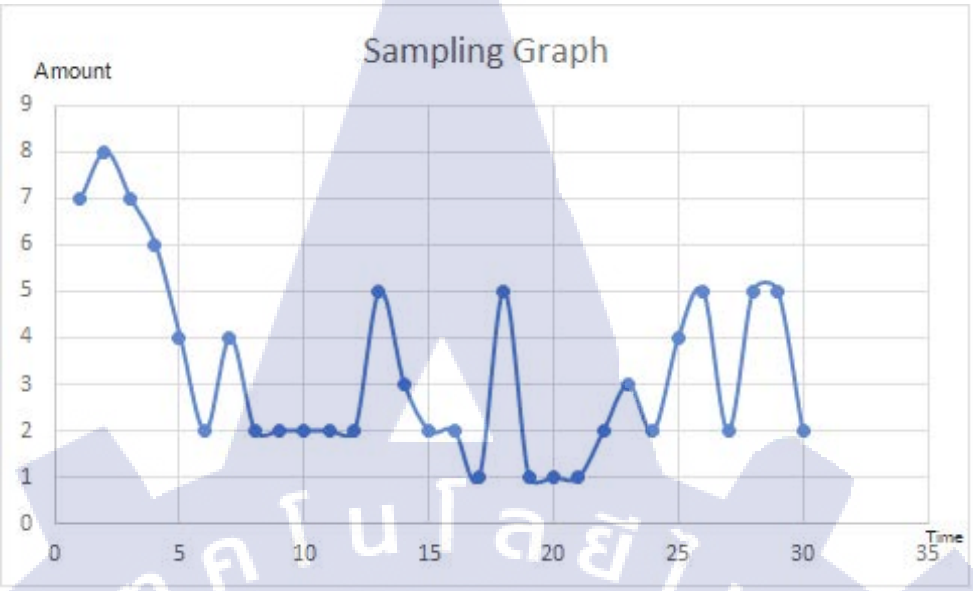


รูปที่ 4.5 กราฟแสดงค่าความถูกต้องจากการคำนวณระหว่าง Precision และ Recall

จากรูปที่ 4.5 ผลลัพธ์การวิจัยจากการทำการสร้างแบบจำลอง เพื่อนำมาทำการสร้างแบบจำลองสำหรับการตรวจจับวัตถุและนับจำนวนนกแอ่นกินรัง โดยผลการวิจัยพบว่าจากการนำค่า Precision สูงสุดประมาณ 97 % และค่า Recall สูงสุดประมาณ 96 % มาคำนวณหาค่าความถูกต้องที่มาจากการตรวจจับวัตถุ แล้วได้ความถูกต้องเฉลี่ยประมาณ 92 %

4.3 วิเคราะห์ผลลัพธ์การวิจัย

หลังจากผลลัพธ์การนำข้อมูลภาพวิดีโอของฟาร์มนกแอ่นกินรังที่บันทึกได้นั้นมาทำการตรวจจับวัตถุเป้าหมายและวัดประสิทธิภาพของระบบตรวจจับวัตถุแล้วและในการวัดประสิทธิภาพของระบบนั้นยังมีการทดลองนำจำนวนวัตถุที่ตรวจจับได้มาอัตราการตรวจจับวัตถุในชุดข้อมูลทดสอบที่นำมาทดสอบ โดยการสุ่มเอาจำนวนนก 1 เฟรมจาก 1 วินาที ด้วยระยะที่มีความแตกต่างกัน 2 วินาทีตามรูปที่ 4.6



รูปที่ 4.6 กราฟแสดงการสุ่มจำนวนนกที่มีระยะห่าง 2 วินาทีใน 60 วินาที

จากรูปที่ 4.6 จะเห็นว่าผลลัพธ์การนำข้อมูลภาพวิดีโอของฟาร์มนกแอ่นกินรังที่บันทึกได้นั้นมาทำการตรวจจับวัตถุเป้าหมายและวัดประสิทธิภาพของระบบตรวจจับวัตถุแล้วได้ผลออกมาเป็นที่พอใจเป็นอย่างมาก โดยจะสังเกตุได้ว่า ภายในกราฟนี้จำนวนนกที่ได้มีอัตราค่าเฉลี่ยอยู่ที่ประมาณ 3-4 ตัวในการทดลองหลายๆครั้ง ทำให้เราเชื่อว่าแบบจำลองที่สร้างขึ้นตามการตั้งค่าจากรูปที่ 3.9 ด้วยชุดข้อมูลนี้ได้ผลลัพธ์ออกมาตามที่เราคาดไว้

บทที่ 5
บทสรุป อภิปราย และข้อเสนอแนะ

การศึกษาวิจัยเรื่องประสิทธิภาพของระบบตรวจจับวัตถุเคลื่อนที่ ซึ่งการวิจัยนี้ได้มีการนำเสนอข้อสรุปตามลำดับดังนี้

5.1 สรุปผลการวิจัย

5.2 ปัญหาและอุปสรรคในการทำวิจัย

5.3 ข้อเสนอแนะงานวิจัย

5.1 สรุปผลการวิจัย

จากโครงการทำฟาร์มเลี้ยงนกแอ่นกินรังมีวัตถุประสงค์ของการทำฟาร์มนกแอ่นกินรัง คือ การพยายามปรับสภาพฟาร์มให้เหมาะสมกับการทำรังวางไข่แบบธรรมชาติของนกแอ่นกินรังตามถิ่นธรรมชาติรวมทั้งการปรับอุณหภูมิ ความชื้น การไหลเวียนของอากาศและกลิ่น ป้องกันศัตรูผู้ล่าเช่น เหยี่ยวนกเขา เสียงเรียกนกที่ใช้ดึงดูดนกให้เข้ามาทำรัง และอื่นๆ เกษตรกรจำเป็นต้องศึกษาและเรียนรู้พฤติกรรมนกแอ่นกินรัง สถานที่ตั้งของการทำฟาร์มเลี้ยงนกแอ่นกินรัง รวมไปถึงการใช้งานอุปกรณ์ต่างๆที่ติดตั้งอยู่ภายในฟาร์มเลี้ยงนกแอ่นกินรัง ช่วยให้ข้อมูลภาพวิดีโอของฟาร์มนกแอ่นกินรังที่บันทึกได้มีคุณภาพมากขึ้น หลังจากนำข้อมูลภาพวิดีโอของฟาร์มนกแอ่นกินรังที่บันทึกได้มาทำการสร้างแบบจำลองแล้วนำมาประมวลผลว่าสามารถตรวจจับและนับจำนวนนกแอ่นกินรังได้เท่าไร เพราะถ้ารู้จำนวนนกก็รู้จำนวนผลผลิตที่ได้ โดยนำโมเดลที่ได้มาทำประเมินผลและวัดประสิทธิภาพแล้วได้ผลออกมาว่า ค่าความถูกต้องอยู่ที่ 92 % ซึ่งองค์ประกอบที่แตกต่างกันเช่น สถานที่ มุมมองของกล้อง เวลา เป็นต้น มีผลต่อการนำข้อมูลวิเคราะห์เพื่อใช้ในการสร้างแบบจำลอง จึงควรมีการแบ่งองค์ประกอบการใช้งานที่แตกต่างกันให้ชัดเจน ตามสถานที่และมุมมองของกล้อง ซึ่งถ้าหากนำโมเดลที่ผ่าน Train มาใช้ในสถานที่ซึ่งองค์ประกอบไม่สอดคล้องกัน อาจทำให้ผลลัพธ์มีความคลาดเคลื่อนเป็นอย่างมากและไม่สามารถนำมาใช้งานได้จริง

5.2 ปัญหาและอุปสรรคในการทำวิจัย

ในระหว่างการทำวิจัยเกี่ยวกับการสร้างแบบจำลองนั้น ภาพวิดีโอที่มีมุมมองการถ่ายที่แตกต่างกันมาก ทำให้ไม่สามารถตรวจจับวัตถุที่ต้องการได้อย่างแม่นยำ อีกทั้งไม่สามารถแยกได้ว่าวัตถุไหนคือเป้าหมายที่ต้องการเพราะหากทำการนับจำนวนวัตถุที่ไม่ใช้เป้าหมาย อาจทำให้ผลลัพธ์ใน

การตรวจจับเกิดความคลาดเคลื่อน และอาจต้องทำให้มีการออกแบบแผนการดำเนินการวิจัยขึ้นใหม่อีกครั้งซึ่งเป็นการสิ้นเปลืองทรัพยากรเป็นอย่างมาก

5.3 ข้อเสนอแนะงานวิจัย

สำหรับหัวการวิจัยนี้ ในการนับจำนวนในบันทึกวิดีโอแล้วนั้น ควรมีการปรับ Resolution ของกล้องวิดีโอให้สูงขึ้น วิธีการคำนวณจำนวนนกที่อาจแตกต่างจากงานวิจัยนี้ การตรวจจับทิศทางการบินของนก ช่วยให้สามารถนับจำนวนนกได้แม่นยำยิ่งขึ้น

บรรณานุกรม

บรรณานุกรม

[1] Q. H., Looi and A., Omar, "Swiftlets and edible bird's nest industry in Asia," *Pertanika Journal of Scholarly Research Reviews*, vol. 2, no. 1, pp. 32-48, April 2016.

[2] C. C. Thorburn, "The edible nest swiftlet industry in Southeast Asia: Capitalism meets commensalism," *Human Ecology an Interdisciplinary Journal*, vol. 43, no. 1, pp. 179 - 184, January 2015.

[3] Z. Zhong-Qiu et al., "Object Detection with Deep Learning: A Review," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 99, pp. 1 - 21, January 2019.

[4] T. Roeland and H. Francisco, "Automatic detection, tracking and counting of birds in marine video content," *2016 Sixth International Conference on Image Processing Theory, Tools and Applications (IPTA)*, Oulu, Finland, December 12 – 15, 2016, pp. 1 – 6.

[5] J. Redmon and A. Farhadi, "*Yolov3: An incremental improvement,"* CoRR [Online]. Available: http://arxiv.org/abs/1804.02767. [Accessed: April 8, 2018].

[6] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, June 27 - 30, 2016, pp. 779-788.

[7] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI, July 21 - 26 2017, pp. 6517–6525.

[8] T. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, et al., "*Microsoft COCO: Common Objects in Context,"* 2014. [Online]. Available: https://arxiv.org/abs/1405.0312. [Accessed: May 1, 2014].

[9] D. Arya, H. Maeda, S. K. Ghosh, D. Toshniwal, A. Mraz, T. Kashiyamaand Y. Sekimoto, "*Transfer Learning-based Road Damage Detection for Multiple Countries,"* [Online]. Available: https://arxiv.org/abs/2008.13101. [Accessed: August 30, 2020].

[10] J. Glenn, S. Alex, B. Jirka et al., "*YOLOv5-P6 1280 Models, AWS, Supervise.ly and YouTube Integrations,*" [Online]. Available: http://doi.org/10.5281/zenodo.4679653. [Accessed: April 11, 2021].

[11] ชลธิศา เวทโอสถ และ นิคม สุวรรณวร, "การพัฒนาอัลกอริทึมเพื่อตรวจนับปริมาณรถบนถนนด้วยการประมวลผลภาพจากกล้องวิดีโอ," *Princess of Naradhiwas University Journal*, vol. 5, no. 1, pp. 35 – 48, January 2013.

[12] P. Punyato et al, "Implementation of a low-cost real-time people counting system on raspberry Pi based on applied tiny YOLO V3", *Engineering Transactions*, vol. 22, no. 2, pp. 47, July - December 2019.

[13] A. Tourani and A. Shahbahrami, "Vehicle counting method based on digital image processing algorithms," *2015 2nd International Conference on Pattern Recognition and Image Analysis (IPRIA),* Rasht, Iran, March 11-12, 2015, pp. 1-6.

[14] X. Xiang, M. Zhai, N. Lv, and A. El Saddik, "Vehicle counting based on vehicle detection and tracking from aerial videos," *Sensors*, vol. 18, no. 8, p. 2560, August 2018.

ภาคผนวก

ภาคผนวก ก.
โค้ดโปรแกรม

โค้ดโปรแกรมภาษา Python

Train.py

```
import argparse
import logging
import os
import random
import time
from pathlib import Path
from threading import Thread
from warnings import warn

import math
import numpy as np
import torch.distributed as dist
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torch.optim.lr_scheduler as lr_scheduler
import torch.utils.data
import yaml
from torch.cuda import amp
from torch.nn.parallel import DistributedDataParallel as DDP
from torch.utils.tensorboard import SummaryWriter
from tqdm import tqdm
import test  # import test.py to get mAP after each epoch
from models.yolo import Model
from utils.autoanchor import check_anchors
from utils.datasets import create_dataloader
from utils.general import labels_to_class_weights, increment_path,
```

```
init_seeds, \
    fitness, strip_optimizer, get_latest_run, check_dataset, check_file,
check_git_status, check_img_size, \
    print_mutation, set_logging
from utils.google_utils import attempt_download
from utils.loss import compute_loss
from utils.plots import plot_images, plot_labels, plot_results, plot_evolution
from utils.torch_utils import ModelEMA, select_device, intersect_dicts,
torch_distributed_zero_first
logger = logging.getLogger(__name__)
try:
    import wandb
except ImportError:
    wandb = None
    logger.info("Install Weights & Biases for experiment logging via 'pip install wandb'
(recommended)")

def train(hyp, opt, device, tb_writer=None, wandb=None):
    logger.info(f'Hyperparameters {hyp}')
    save_dir, epochs, batch_size, total_batch_size, weights, rank = \
        Path(opt.save_dir), opt.epochs, opt.batch_size, opt.total_batch_size,
opt.weights, opt.global_rank
    # Directories
    wdir = save_dir / 'weights'
    wdir.mkdir(parents=True, exist_ok=True)  # make dir
    last = wdir / 'last.pt'
    best = wdir / 'best.pt'
    results_file = save_dir / 'results.txt'
```

```
    # Save run settings
    with open(save_dir / 'hyp.yaml', 'w') as f:
        yaml.dump(hyp, f, sort_keys=False)
    with open(save_dir / 'opt.yaml', 'w') as f:
        yaml.dump(vars(opt), f, sort_keys=False)
    plots = not opt.evolve  # create plots
    cuda = device.type != 'cpu'
    init_seeds(2 + rank)
    with open(opt.data) as f:
        data_dict = yaml.load(f, Loader=yaml.FullLoader)  # data dict
    with torch_distributed_zero_first(rank):
        check_dataset(data_dict)  # check
    train_path = data_dict['train']
    test_path = data_dict['val']
    nc, names = (1, ['item']) if opt.single_cls else (int(data_dict['nc']),
data_dict['names'])  # number classes, names
    assert len(names) == nc, '%g names found for nc=%g dataset in %s' %
(len(names), nc, opt.data)  # check
    # Model
    pretrained = weights.endswith('.pt')
    if pretrained:
        with torch_distributed_zero_first(rank):
            attempt_download(weights)  # download if not found locally
        ckpt = torch.load(weights, map_location=device)  # load checkpoint
        if hyp.get('anchors'):
            ckpt['model'].yaml['anchors'] = round(hyp['anchors'])  # force autoanchor
        model = Model(opt.cfg or ckpt['model'].yaml, ch=3, nc=nc).to(device)  # create
        exclude = ['anchor'] if opt.cfg or hyp.get('anchors') else []  # exclude keys
        state_dict = ckpt['model'].float().state_dict()  # to FP32
```

```
        state_dict = intersect_dicts(state_dict, model.state_dict(), exclude=exclude)  # intersect
        model.load_state_dict(state_dict, strict=False)  # load
        logger.info('Transferred %g/%g items from %s' % (len(state_dict), len(model.state_dict()), weights))  # report
    else:
        model = Model(opt.cfg, ch=3, nc=nc).to(device)  # create

    # Freeze
    freeze = []  # parameter names to freeze (full or partial)
    for k, v in model.named_parameters():
        v.requires_grad = True  # train all layers
        if any(x in k for x in freeze):
            print('freezing %s' % k)
            v.requires_grad = False

    # Optimizer
    nbs = 64  # nominal batch size
    accumulate = max(round(nbs / total_batch_size), 1)  # accumulate loss before optimizing
    hyp['weight_decay'] *= total_batch_size * accumulate / nbs  # scale weight_decay

    pg0, pg1, pg2 = [], [], []  # optimizer parameter groups
    for k, v in model.named_modules():
        if hasattr(v, 'bias') and isinstance(v.bias, nn.Parameter):
            pg2.append(v.bias)  # biases
        if isinstance(v, nn.BatchNorm2d):
            pg0.append(v.weight)  # no decay
        elif hasattr(v, 'weight') and isinstance(v.weight, nn.Parameter):
```

```
    else:
        optimizer = optim.SGD(pg0, lr=hyp['lr0'], momentum=hyp['momentum'],
nesterov=True)
    optimizer.add_param_group({'params': pg1, 'weight_decay': hyp['weight_decay']})
# add pg1 with weight_decay
    optimizer.add_param_group({'params': pg2})  # add pg2 (biases)
    logger.info('Optimizer groups: %g .bias, %g conv.weight, %g other' % (len(pg2),
len(pg1), len(pg0)))
    del pg0, pg1, pg2
    # Scheduler https://arxiv.org/pdf/1812.01187.pdf
    #
https://pytorch.org/docs/stable/_modules/torch/optim/lr_scheduler.html#OneCycleLR
    lf = lambda x: ((1 + math.cos(x * math.pi / epochs)) / 2) * (1 - hyp['lrf']) + hyp['lrf']  #
cosine
    scheduler = lr_scheduler.LambdaLR(optimizer, lr_lambda=lf)
    # plot_lr_scheduler(optimizer, scheduler, epochs)
    # Logging
    if wandb and wandb.run is None:
        opt.hyp = hyp  # add hyperparameters
        wandb_run = wandb.init(config=opt, resume="allow",project='YOLOv5'
if opt.project == 'runs/train' else Path(opt.project).stem,name=save_dir.stem,
id=ckpt.get('wandb_id') if 'ckpt' in locals() else None)
    loggers = {'wandb': wandb}  # loggers dict
    # Resume
    start_epoch, best_fitness = 0, 0.0
    if pretrained:
        # Optimizer
        if ckpt['optimizer'] is not None:
            optimizer.load_state_dict(ckpt['optimizer'])
```

```
        # Results
        if ckpt.get('training_results') is not None:
            with open(results_file, 'w') as file:
                file.write(ckpt['training_results'])  # write results.txt
        # Epochs
        start_epoch = ckpt['epoch'] + 1
        if opt.resume:
            assert start_epoch > 0, '%s training to %g epochs is finished, nothing to
resume.' % (weights, epochs)
        if epochs < start_epoch:
            logger.info('%s has been trained for %g epochs. Fine-tuning for %g additional
epochs.' %
                        (weights, ckpt['epoch'], epochs))
            epochs += ckpt['epoch']  # finetune additional epochs
        del ckpt, state_dict
    # Image sizes
    gs = int(max(model.stride))  # grid size (max stride)
    imgsz, imgsz_test = [check_img_size(x, gs) for x in opt.img_size]  # verify imgsz
are gs-multiples
    # DP mode
    if cuda and rank == -1 and torch.cuda.device_count() > 1:
        model = torch.nn.DataParallel(model)
    # SyncBatchNorm
    if opt.sync_bn and cuda and rank != -1:
        model = torch.nn.SyncBatchNorm.convert_sync_batchnorm(model).to(device)
        logger.info('Using SyncBatchNorm()')
    # EMA
    ema = ModelEMA(model) if rank in [-1, 0] else None
```

```
    # DDP mode
    if cuda and rank != -1:
        model = DDP(model, device_ids=[opt.local_rank],
output_device=opt.local_rank)

    # Trainloader
    dataloader, dataset = create_dataloader(train_path, imgsz, batch_size, gs, opt,
                                            hyp=hyp, augment=True, cache=opt.cache_images,
rect=opt.rect, rank=rank,
                                            world_size=opt.world_size, workers=opt.workers,
                                            image_weights=opt.image_weights)
    mlc = np.concatenate(dataset.labels, 0)[:, 0].max()  # max label class
    nb = len(dataloader)  # number of batches
    assert mlc < nc, 'Label class %g exceeds nc=%g in %s. Possible class labels are
0-%g' % (mlc, nc, opt.data, nc - 1)
    # Process 0
    if rank in [-1, 0]:
        ema.updates = start_epoch * nb // accumulate  # set EMA updates
        testloader = create_dataloader(test_path, imgsz_test, total_batch_size, gs, opt,
         hyp=hyp, cache=opt.cache_images and not opt.notest, rect=True,
         rank=-1, world_size=opt.world_size, workers=opt.workers)[0]  # testloader

        if not opt.resume:
            labels = np.concatenate(dataset.labels, 0)
            c = torch.tensor(labels[:, 0])  # classes
            # cf = torch.bincount(c.long(), minlength=nc) + 1.  # frequency
            # model._initialize_biases(cf.to(device))
            if plots:
                Thread(target=plot_labels, args=(labels, save_dir, loggers),
```

```
        # Anchors
        if not opt.noautoanchor:
            check_anchors(dataset, model=model, thr=hyp['anchor_t'], imgsz=imgsz)

    # Model parameters
    hyp['cls'] *= nc / 80.  # scale coco-tuned hyp['cls'] to current dataset
    model.nc = nc  # attach number of classes to model
    model.hyp = hyp  # attach hyperparameters to model
    model.gr = 1.0  # iou loss ratio (obj_loss = 1.0 or iou)
    model.class_weights = labels_to_class_weights(dataset.labels, nc).to(device)  # attach class weights
    model.names = names

    # Start training
    t0 = time.time()
    nw = max(round(hyp['warmup_epochs'] * nb), 1000)  # number of warmup iterations, max(3 epochs, 1k iterations)
    # nw = min(nw, (epochs - start_epoch) / 2 * nb)  # limit warmup to < 1/2 of training
    maps = np.zeros(nc)  # mAP per class
    results = (0, 0, 0, 0, 0, 0, 0)  # P, R, mAP@.5, mAP@.5-.95, val_loss(box, obj, cls)
    scheduler.last_epoch = start_epoch - 1  # do not move
    scaler = amp.GradScaler(enabled=cuda)
    logger.info('Image sizes %g train, %g test\n'
                'Using %g dataloader workers\nLogging results to %s\n'
                'Starting training for %g epochs...' % (imgsz, imgsz_test, dataloader.num_workers, save_dir, epochs))
    for epoch in range(start_epoch, epochs):  # epoch ------------------------------------------------------------------
```

```
        # Generate indices
        if rank in [-1, 0]:
            cw = model.class_weights.cpu().numpy() * (1 - maps) ** 2  # class weights
            iw = labels_to_image_weights(dataset.labels, nc=nc, class_weights=cw)  # image weights
            dataset.indices = random.choices(range(dataset.n), weights=iw, k=dataset.n)  # rand weighted idx
        # Broadcast if DDP
        if rank != -1:
            indices = (torch.tensor(dataset.indices) if rank == 0 else torch.zeros(dataset.n)).int()
            dist.broadcast(indices, 0)
            if rank != 0:
                dataset.indices = indices.cpu().numpy()

    # Update mosaic border
    # b = int(random.uniform(0.25 * imgsz, 0.75 * imgsz + gs) // gs * gs)
    # dataset.mosaic_border = [b - imgsz, -b]  # height, width borders

    mloss = torch.zeros(4, device=device)  # mean losses
    if rank != -1:
        dataloader.sampler.set_epoch(epoch)
    pbar = enumerate(dataloader)
    logger.info(('\n' + '%10s' * 8) % ('Epoch', 'gpu_mem', 'box', 'obj', 'cls', 'total', 'targets', 'img_size'))
    if rank in [-1, 0]:
        pbar = tqdm(pbar, total=nb)  # progress bar
    optimizer.zero_grad()
    for i, (imgs, targets, paths, _) in pbar:  # batch -------------------------------------------
```

```
            if ni <= nw:
                xi = [0, nw]  # x interp
                # model.gr = np.interp(ni, xi, [0.0, 1.0])  # iou loss ratio (obj_loss = 1.0 or
iou)
                accumulate = max(1, np.interp(ni, xi, [1, nbs / total_batch_size]).round())
                for j, x in enumerate(optimizer.param_groups):
                    # bias lr falls from 0.1 to lr0, all other lrs rise from 0.0 to lr0
                    x['lr'] = np.interp(ni, xi, [hyp['warmup_bias_lr'] if j == 2 else 0.0,
x['initial_lr'] * lf(epoch)])
                    if 'momentum' in x:
                        x['momentum'] = np.interp(ni, xi, [hyp['warmup_momentum'],
hyp['momentum']])

            # Multi-scale
            if opt.multi_scale:
                sz = random.randrange(imgsz * 0.5, imgsz * 1.5 + gs) // gs * gs  # size
                sf = sz / max(imgs.shape[2:])  # scale factor
                if sf != 1:
                    ns = [math.ceil(x * sf / gs) * gs for x in imgs.shape[2:]]  # new shape
(stretched to gs-multiple)
                    imgs = F.interpolate(imgs, size=ns, mode='bilinear',
align_corners=False)

            # Forward
            with amp.autocast(enabled=cuda):
                pred = model(imgs)  # forward
                loss, loss_items = compute_loss(pred, targets.to(device), model)  # loss
scaled by batch_size
                if rank != -1:
```

```
            scaler.update()
            optimizer.zero_grad()
            if ema:
                ema.update(model)

        # Print
        if rank in [-1, 0]:
            mloss = (mloss * i + loss_items) / (i + 1)  # update mean losses
            mem = '%.3gG' % (torch.cuda.memory_reserved() / 1E9 if
torch.cuda.is_available() else 0)  # (GB)
            s = ('%10s' * 2 + '%10.4g' * 6) % (
                '%g/%g' % (epoch, epochs - 1), mem, *mloss, targets.shape[0],
imgs.shape[-1])
            pbar.set_description(s)

            # Plot
            if plots and ni < 3:
                f = save_dir / f'train_batch{ni}.jpg'  # filename
                Thread(target=plot_images, args=(imgs, targets, paths, f),
daemon=True).start()
                # if tb_writer:
                #                 tb_writer.add_image(f, result, dataformats='HWC',
global_step=epoch)
                #     tb_writer.add_graph(model, imgs)  # add model to tensorboard
            elif plots and ni == 3 and wandb:
                wandb.log({"Mosaics": [wandb.Image(str(x), caption=x.name) for x in
save_dir.glob('train*.jpg')]})

        # end batch ------------------------------------------------------------------------
```

```
        if rank in [-1, 0]:
            # mAP
            if ema:
                ema.update_attr(model, include=['yaml', 'nc', 'hyp', 'gr', 'names', 'stride'])
            final_epoch = epoch + 1 == epochs
            if not opt.notest or final_epoch:  # Calculate mAP
                results, maps, times = test.test(opt.data,
                                                 batch_size=total_batch_size,
                                                 imgsz=imgsz_test,
                                                 model=ema.ema,
                                                 single_cls=opt.single_cls,
                                                 dataloader=testloader,
                                                 save_dir=save_dir,
                                                 plots=plots and final_epoch,
                                                 log_imgs=opt.log_imgs if wandb else 0)

            # Write
            with open(results_file, 'a') as f:
                f.write(s + '%10.4g' * 7 % results + '\n')  # P, R, mAP@.5, mAP@.5-.95, val_loss(box, obj, cls)
            if len(opt.name) and opt.bucket:
                os.system('gsutil cp %s gs://%s/results/results%s.txt' % (results_file, opt.bucket, opt.name))

            # Log
            tags = ['train/box_loss', 'train/obj_loss', 'train/cls_loss',  # train loss
                    'metrics/precision', 'metrics/recall', 'metrics/mAP_0.5', 'metrics/mAP_0.5:0.95',
                    'val/box_loss', 'val/obj_loss', 'val/cls_loss',  # val loss
```

```
            d combination of [P, R, mAP@.5, mAP@.5-.95]
        if fi > best_fitness:
            best_fitness = fi

        # Save model
        save = (not opt.nosave) or (final_epoch and not opt.evolve)
        if save:
            with open(results_file, 'r') as f:  # create checkpoint
                ckpt = {'epoch': epoch,
                        'best_fitness': best_fitness,
                        'training_results': f.read(),
                        'model': ema.ema,
                        'optimizer': None if final_epoch else optimizer.state_dict(),
                        'wandb_id': wandb_run.id if wandb else None}

            # Save last, best and delete
            torch.save(ckpt, last)
            if best_fitness == fi:
                torch.save(ckpt, best)
            del ckpt
    # end epoch ----------------------------------------------------------------------------------
----------
    # end training

    if rank in [-1, 0]:
        # Strip optimizers
        n = opt.name if opt.name.isnumeric() else ''
        fresults, flast, fbest = save_dir / f'results{n}.txt', wdir / f'last{n}.pt', wdir /
f'best{n}.pt'
```

```
#upload
        # Finish
        if plots:
            plot_results(save_dir=save_dir)  # save as results.png
            if wandb:
                files = ['results.png', 'precision_recall_curve.png', 'confusion_matrix.png']
                wandb.log({"Results": [wandb.Image(str(save_dir / f), caption=f) for f in files
                                       if (save_dir / f).exists()]})
        logger.info('%g epochs completed in %.3f hours.\n' % (epoch - start_epoch + 1,
(time.time() - t0) / 3600))
    else:
        dist.destroy_process_group()

    wandb.run.finish() if wandb and wandb.run else None
    torch.cuda.empty_cache()
    return results


if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--weights', type=str, default='yolov5s.pt', help='initial
weights path')
    parser.add_argument('--cfg', type=str, default='', help='model.yaml path')
    parser.add_argument('--data', type=str, default='data/coco128.yaml',
help='data.yaml path')
    parser.add_argument('--hyp', type=str, default='data/hyp.scratch.yaml',
help='hyperparameters path')
    parser.add_argument('--epochs', type=int, default=300)
```

```
    parser.add_argument('--nosave', action='store_true', help='only save final checkpoint')
    parser.add_argument('--notest', action='store_true', help='only test final epoch')
    parser.add_argument('--noautoanchor', action='store_true', help='disable autoanchor check')
    parser.add_argument('--evolve', action='store_true', help='evolve hyperparameters')
    parser.add_argument('--bucket', type=str, default='', help='gsutil bucket')
    parser.add_argument('--cache-images', action='store_true', help='cache images for faster training')
    parser.add_argument('--image-weights', action='store_true', help='use weighted image selection for training')
    parser.add_argument('--device', default='', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
    parser.add_argument('--multi-scale', action='store_true', help='vary img-size +/- 50%%')
    parser.add_argument('--single-cls', action='store_true', help='train as single-class dataset')
    parser.add_argument('--adam', action='store_true', help='use torch.optim.Adam() optimizer')
    parser.add_argument('--sync-bn', action='store_true', help='use SyncBatchNorm, only available in DDP mode')
    parser.add_argument('--local_rank', type=int, default=-1, help='DDP parameter, do not modify')
    parser.add_argument('--log-imgs', type=int, default=16, help='number of images for W&B logging, max 100')
    parser.add_argument('--workers', type=int, default=8, help='maximum number of dataloader workers')
```

```
        check_git_status()

    # Resume
    if opt.resume:  # resume an interrupted run
        ckpt = opt.resume if isinstance(opt.resume, str) else get_latest_run()  # specified or most recent path
        assert os.path.isfile(ckpt), 'ERROR: --resume checkpoint does not exist'
        with open(Path(ckpt).parent.parent / 'opt.yaml') as f:
            opt = argparse.Namespace(**yaml.load(f, Loader=yaml.FullLoader))  # replace
        opt.cfg, opt.weights, opt.resume = '', ckpt, True
        logger.info('Resuming training from %s' % ckpt)
    else:
        # opt.hyp = opt.hyp or ('hyp.finetune.yaml' if opt.weights else 'hyp.scratch.yaml')
        opt.data, opt.cfg, opt.hyp = check_file(opt.data), check_file(opt.cfg), check_file(opt.hyp)  # check files
        assert len(opt.cfg) or len(opt.weights), 'either --cfg or --weights must be specified'
        opt.img_size.extend([opt.img_size[-1]] * (2 - len(opt.img_size)))  # extend to 2 sizes (train, test)
        opt.name = 'evolve' if opt.evolve else opt.name
        opt.save_dir = increment_path(Path(opt.project) / opt.name, exist_ok=opt.exist_ok | opt.evolve)  # increment run

    # DDP mode
    device = select_device(opt.device, batch_size=opt.batch_size)
    if opt.local_rank != -1:
        assert torch.cuda.device_count() > opt.local_rank
```

```
    # Hyperparameters
    with open(opt.hyp) as f:
        hyp = yaml.load(f, Loader=yaml.FullLoader)  # load hyps
        if 'box' not in hyp:
            warn('Compatibility: %s missing "box" which was renamed from "giou" in %s' %
                 (opt.hyp, 'https://github.com/ultralytics/yolov5/pull/1120'))
            hyp['box'] = hyp.pop('giou')

    # Train
    logger.info(opt)
    if not opt.evolve:
        tb_writer = None  # init loggers
        if opt.global_rank in [-1, 0]:
            logger.info(f'Start Tensorboard with "tensorboard --logdir {opt.project}", view at http://localhost:6006/')
            tb_writer = SummaryWriter(opt.save_dir)  # Tensorboard
        train(hyp, opt, device, tb_writer, wandb)

    # Evolve hyperparameters (optional)
    else:
        # Hyperparameter evolution metadata (mutation scale 0-1, lower_limit, upper_limit)
        meta = {'lr0': (1, 1e-5, 1e-1),  # initial learning rate (SGD=1E-2, Adam=1E-3)
                'lrf': (1, 0.01, 1.0),  # final OneCycleLR learning rate (lr0 * lrf)
                'momentum': (0.3, 0.6, 0.98),  # SGD momentum/Adam beta1
                'weight_decay': (1, 0.0, 0.001),  # optimizer weight decay
                'warmup_epochs': (1, 0.0, 5.0),  # warmup epochs (fractions ok)
```

```
            'obj_pw': (1, 0.5, 2.0),  # obj BCELoss positive_weight
            'iou_t': (0, 0.1, 0.7),  # IoU training threshold
            'anchor_t': (1, 2.0, 8.0),  # anchor-multiple threshold
            'anchors': (2, 2.0, 10.0),  # anchors per output grid (0 to ignore)
            'fl_gamma': (0, 0.0, 2.0),  # focal loss gamma (efficientDet default gamma=1.5)
            'hsv_h': (1, 0.0, 0.1),  # image HSV-Hue augmentation (fraction)
            'hsv_s': (1, 0.0, 0.9),  # image HSV-Saturation augmentation (fraction)
            'hsv_v': (1, 0.0, 0.9),  # image HSV-Value augmentation (fraction)
            'degrees': (1, 0.0, 45.0),  # image rotation (+/- deg)
            'translate': (1, 0.0, 0.9),  # image translation (+/- fraction)
            'scale': (1, 0.0, 0.9),  # image scale (+/- gain)
            'shear': (1, 0.0, 10.0),  # image shear (+/- deg)
            'perspective': (0, 0.0, 0.001),  # image perspective (+/- fraction), range 0-0.001
            'flipud': (1, 0.0, 1.0),  # image flip up-down (probability)
            'fliplr': (0, 0.0, 1.0),  # image flip left-right (probability)
            'mosaic': (1, 0.0, 1.0),  # image mixup (probability)
            'mixup': (1, 0.0, 1.0)}  # image mixup (probability)

        assert opt.local_rank == -1, 'DDP mode not implemented for --evolve'
        opt.notest, opt.nosave = True, True  # only test/save final epoch
        # ei = [isinstance(x, (int, float)) for x in hyp.values()]  # evolvable indices
        yaml_file = Path(opt.save_dir) / 'hyp_evolved.yaml'  # save best result here
        if opt.bucket:
            os.system('gsutil cp gs://%s/evolve.txt .' % opt.bucket)  # download evolve.txt if exists

        for _ in range(300):  # generations to evolve
```

```
            if parent == 'single' or len(x) == 1:
                # x = x[random.randint(0, n - 1)]  # random selection
                x = x[random.choices(range(n), weights=w)[0]]  # weighted selection
            elif parent == 'weighted':
                x = (x * w.reshape(n, 1)).sum(0) / w.sum()  # weighted combination

            # Mutate
            mp, s = 0.8, 0.2  # mutation probability, sigma
            npr = np.random
            npr.seed(int(time.time()))
            g = np.array([x[0] for x in meta.values()])  # gains 0-1
            ng = len(meta)
            v = np.ones(ng)
            while all(v == 1):  # mutate until a change occurs (prevent duplicates)
                v = (g * (npr.random(ng) < mp) * npr.randn(ng) * npr.random() * s +
1).clip(0.3, 3.0)
            for i, k in enumerate(hyp.keys()):  # plt.hist(v.ravel(), 300)
                hyp[k] = float(x[i + 7] * v[i])  # mutate

        # Constrain to limits
        for k, v in meta.items():
            hyp[k] = max(hyp[k], v[1])  # lower limit
            hyp[k] = min(hyp[k], v[2])  # upper limit
            hyp[k] = round(hyp[k], 5)  # significant digits

        # Train mutation
        results = train(hyp.copy(), opt, device, wandb=wandb)
        # Write mutation results
        print_mutation(hyp.copy(), results, yaml_file, opt.bucket)
```

Custom_data.yaml

```
train: custom_dataset/train/images
test: custom_dataset/test/images
val: custom_dataset/valid/images

nc: 16
names: ['edible-nest swiftlet','Black-nest Swiftlet','Seychelles swiftlet','Cave swiftlet','Philippine swiftlet','Moluccan swiftlet','Mountain swiftlet','Australian swiftlet','Uniform swiftlet','Bare-legged swiftlet','Palau swiftlet','Guam swiftlet','Marquesan swiftlet','Indochinese swiftlet','White-Rumped Swift','edible-nest
```

Detect1.py

```
import argparse
import time
from pathlib import Path

import cv2
import torch
import torch.backends.cudnn as cudnn
from numpy import random
import csv

from models.experimental import attempt_load
from utils.datasets import LoadStreams, LoadImages
from utils.general import check_img_size, non_max_suppression, apply_classifier,
scale_coords, xyxy2xywh, \
    strip_optimizer, set_logging, increment_path
from utils.plots import plot_one_box
from utils.torch_utils import select_device, load_classifier, time_synchronized
```

```
def detect(save_img=False):
    source, weights, view_img, save_txt, imgsz = opt.source, opt.weights, opt.view_img, opt.save_txt, opt.img_size
    webcam = source.isnumeric() or source.endswith('.txt') or source.lower().startswith(
        ('rtsp://', 'rtmp://', 'http://'))
    # Directories
    save_dir = Path(increment_path(Path(opt.project) / opt.name, exist_ok=opt.exist_ok))  # increment run
    (save_dir / 'labels' if save_txt else save_dir).mkdir(parents=True, exist_ok=True)  # make dir
    # Initialize
    set_logging()
    device = select_device(opt.device)
    half = device.type != 'cpu'  # half precision only supported on CUDA
    # Load model
    model = attempt_load(weights, map_location=device)  # load FP32 model
    imgsz = check_img_size(imgsz, s=model.stride.max())  # check img_size
    if half:
        model.half()  # to FP16
    # Second-stage classifier
    classify = False
    if classify:
        modelc = load_classifier(name='resnet101', n=2)  # initialize
        modelc.load_state_dict(torch.load('weights/resnet101.pt', map_location=device)['model']).to(device).eval()
    # Set Dataloader
    vid_path, vid_writer = None, None
    if webcam:
        view_img = True
```

```
    cudnn.benchmark = True  # set True to speed up constant image size inference
    dataset = LoadStreams(source, img_size=imgsz)
else:
    save_img = True
    dataset = LoadImages(source, img_size=imgsz)

# Get names and colors
names = model.module.names if hasattr(model, 'module') else model.names
colors = [[random.randint(0, 255) for _ in range(3)] for _ in names]

# Run inference
t0 = time.time()
img = torch.zeros((1, 3, imgsz, imgsz), device=device)  # init img
_ = model(img.half() if half else img) if device.type != 'cpu' else None  # run once

# add-on variable
x = 0
obj = []
obj1 = []
avg = []
avg_txt = 0
top = 0
last = None

for path, img, im0s, vid_cap in dataset:
    img = torch.from_numpy(img).to(device)
    img = img.half() if half else img.float()  # uint8 to fp16/32
    img /= 255.0  # 0 - 255 to 0.0 - 1.0
    if img.ndimension() == 3:
```

```
        pred = model(img, augment=opt.augment)[0]


        # Apply NMS
        pred = non_max_suppression(pred, opt.conf_thres, opt.iou_thres,
classes=opt.classes, agnostic=opt.agnostic_nms)
        t2 = time_synchronized()
        last = 1159
        # Apply Classifier
        if classify:
            pred = apply_classifier(pred, modelc, img, im0s)
        # Process detections
        for i, det in enumerate(pred):  # detections per image
            obj_txt = "Object : " ### object counting
            obj_avg = "Overall object : " ### overall counting
            if webcam:  # batch_size >= 1
                p, s, im0 = Path(path[i]), '%g: ' % i, im0s[i].copy()
            else:
                p, s, im0 = Path(path), '', im0s
            save_path = str(save_dir / p.name)
            txt_path = str(save_dir / 'labels' / p.stem) + ('_%g' % dataset.frame if
dataset.mode == 'video' else '')
            s += '%gx%g ' % img.shape[2:]  # print string
            gn = torch.tensor(im0.shape)[[1, 0, 1, 0]]  # normalization gain whwh
            if len(det):
                # Rescale boxes from img_size to im0 size
                det[:, :4] = scale_coords(img.shape[2:], det[:, :4], im0.shape).round()
                # Print results
                for c in det[:, -1].unique():
                    n = (det[:, -1] == c).sum()  # detections per class
```

```
                s += '%g %ss,  ' % (n, names[int(c)])  # add to string
                obj_txt += '%g %ss,  ' % (n, names[int(c)]) # add string to txt ###
                obj.append(int(n))
                obj1.append(int(n))
            # Write results
            for *xyxy, conf, cls in reversed(det):
                if save_txt:  # Write to file
                    xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) / gn).view(-1).tolist()
# normalized xywh
                    line = (cls, *xywh, conf) if opt.save_conf else (cls, *xywh)  # label format
                    with open(txt_path + '.txt', 'a') as f:
                        f.write(('%g ' * len(line)).rstrip() % line + '\n')
                if save_img or view_img:  # Add bbox to image
                    label = '%s %.2f' % (names[int(cls)], conf)
                    plot_one_box(xyxy,      im0,      label=label,      color=colors[int(cls)],
line_thickness=3)
            else:
                obj_txt += "0"
                obj.append(int(0))
                obj1.append(int(0))
            # Print time (inference + NMS)
            print('%sDone. (%.3fs)' % (s, t2 - t1))
            # Stream results
            if view_img:
                cv2.putText(im0,obj_txt,(5,60),    cv2.FONT_HERSHEY_COMPLEX_SMALL,
0.5 ,(0,0,255),1) ### Show text
                cv2.imshow(str(p), im0)
                if cv2.waitKey(1) == ord('q'):  # q to quit
                    raise StopIteration
```

```
            # Save results (image with detections)
            if save_img:
                x = x + 1
                if dataset.mode == 'images':
                    cv2.imwrite(save_path, im0)
                else:
                    if vid_path != save_path:  # new video
                        vid_path = save_path
                        if isinstance(vid_writer, cv2.VideoWriter):
                            vid_writer.release()  # release previous video writer
                        fourcc = 'mp4v'  # output video codec
                        fps = vid_cap.get(cv2.CAP_PROP_FPS)
                        w = int(vid_cap.get(cv2.CAP_PROP_FRAME_WIDTH))
                        h = int(vid_cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
                        vid_writer = cv2.VideoWriter(save_path,
cv2.VideoWriter_fourcc(*fourcc), fps, (w, h))
                    cv2.putText(im0,obj_txt,(5,30),
cv2.FONT_HERSHEY_COMPLEX_SMALL,1,(0,0,255),1) ## Show text

                    # Show Average Number
                    avg_num = 0
                    if x < 13:
                        for a in range(len(obj)):
                            avg_num += obj[a]
                        # avg_num = avg_num - top
                        # avg_num = avg_num / 12
                        avg_txt = int(round(avg_num))
                        avg.append(avg_txt)
                        obj_avg += str(avg_num)
```

```
                # obj = []
                print(i)
            else:
                top += obj[x-12]
                for a in range(len(obj)):
                    avg_num += obj[a]
                avg_num = avg_num - top
                # avg_num = avg_num / 12
                avg_txt = int(round(avg_num))
                avg.append(avg_txt)
                obj_avg += str(avg_num)
                if x == last:
                 sum_avg = "Average Object : " + str(avg_num)
                 cv2.putText(im0,sum_avg,(5,100),
cv2.FONT_HERSHEY_COMPLEX_SMALL,2,(0,0,255),1) ## Show text
                # obj = []
               # else:
               #   obj_avg += str(int(avg_txt))
            cv2.putText(im0,obj_avg,(5,50),
cv2.FONT_HERSHEY_COMPLEX_SMALL,1,(0,0,255),1) ## Show text
            vid_writer.write(im0)


    # Create *.csv file
    write_dir = save_path +".csv" ## Object per Frame
    with open(write_dir, mode='w') as obj_file1:
        writer = csv.writer(obj_file1, delimiter=',', quotechar='"',
quoting=csv.QUOTE_MINIMAL)
        writer.writerow(['Frame', 'Object Amount'])
```

```
    for a in range(x):
            writer.writerow([a, obj1[a]])
    if save_txt or save_img:
        s = f"\n{len(list(save_dir.glob('labels/*.txt')))} labels saved to {save_dir / 'labels'}"
if save_txt else ''
        print(f"Results saved to {save_dir}{s}"
    print('Done. (%.3fs)' % (time.time() - t0))

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--weights', nargs='+', type=str, default='yolov5s.pt',
help='model.pt path(s)')
    parser.add_argument('--source', type=str, default='data/images', help='source') #
file/folder, 0 for webcam
    parser.add_argument('--img-size', type=int, default=640, help='inference size
(pixels)')
    parser.add_argument('--conf-thres', type=float, default=0.25, help='object
confidence threshold')
    parser.add_argument('--iou-thres', type=float, default=0.45, help='IOU threshold
for NMS')
    parser.add_argument('--device', default='', help='cuda device, i.e. 0 or 0,1,2,3 or
cpu')
    parser.add_argument('--view-img', action='store_true', help='display results')
    parser.add_argument('--save-txt', action='store_true', help='save results to *.txt')
    parser.add_argument('--save-conf', action='store_true', help='save confidences in
--save-txt labels')
    parser.add_argument('--classes', nargs='+', type=int, help='filter by class: --class
0, or --class 0 2 3')
```

```
    parser.add_argument('--agnostic-nms', action='store_true', help='class-agnostic
NMS')
    parser.add_argument('--augment', action='store_true', help='augmented
inference')
    parser.add_argument('--update', action='store_true', help='update all models')
    parser.add_argument('--project', default='runs/detect', help='save results to
project/name')
    parser.add_argument('--name', default='exp', help='save results to project/name')
    parser.add_argument('--exist-ok', action='store_true', help='existing project/name
ok, do not increment')
    opt = parser.parse_args()
    print(opt)

    with torch.no_grad():
        if opt.update:  # update all models (to fix SourceChangeWarning)
            for opt.weights in ['yolov5s.pt', 'yolov5m.pt', 'yolov5l.pt', 'yolov5x.pt']:
                detect()
                strip_optimizer(opt.weights)
        else:
            detect()
```