

การจำแนกองค์ประกอบของโรคหูชั้นกลางอักเสบ
จากภาพอโตสโคป

CLASSIFICATION OF OTITIS MEDIA FACTORS FROM OTOSCOPE

IMAGE

เมธานนท์ แก้วกระจาง
เอกสิทธิ์ บุตรดา

ปัญหาพิเศษนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตร
ปริญญาวิทยาศาสตรบัณฑิต (วิทยาการคอมพิวเตอร์)
ภาควิชาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2565

**CLASSIFICATION OF OTITIS MEDIA FACTORS FROM OTOSCOPE
IMAGE**

Methanon Kaeokrachang

Aekkasit Budda

**A SPECIAL PROBLEM SUBMITTED IN PARTIALFULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF BACHELOR OF SCIENCE
DEPARTMENT OF COMPUTER SCIENCE
SCHOOL OF SCIENCE
KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG**

2565

หัวข้อโครงการพิเศษ การจำแนกองค์ประกอบของโรคหูชั้นกลางอักเสบจากภาพอสโคป

Classification of Otitis Media Factors from Otoscope Image

ชื่อนักศึกษา	นายเมธานนท์ แก้วกระจ่าง	รหัสนักศึกษา 62050214
	นายเอกสิทธิ์ บุตรดา	รหัสนักศึกษา 62050251
ปริญญา	วิทยาศาสตรบัณฑิต (วิทยาการคอมพิวเตอร์)	
ภาควิชา	วิทยาการคอมพิวเตอร์	
ปีการศึกษา	2565	
อาจารย์ที่ปรึกษา	ดร.อัคเดช อุดมชัยพร	
อาจารย์ที่ปรึกษาร่วม	ศ.พญ.กิติรัตน์ อังกานนท์	

คณะวิทยาศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง (สจล.) อนุมัติให้ปัญหาพิเศษนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรบัณฑิต (วิทยาการคอมพิวเตอร์) ประจำปีการศึกษา 2565

คณะกรรมการสอบ	ลายมือชื่อ
ผศ.ดร.ธีระ ศิริธีราภุณ	
ผศ.ดร.อันเนตพร ธรรมดุณนาณัย	
ดร.อัคเดช อุดมชัยพร	

ลิขสิทธิ์ของคณะวิทยาศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

หัวข้อโครงการพิเศษ การจำแนกองค์ประกอบของโรคหูชั้นกลางอักเสบจากภาพออตอสโคป

Classification of Otitis Media Factors from Otoscope Image

ชื่อนักศึกษา	นายเมธานันท์ แก้วกระจ่าง	รหัสนักศึกษา 62050214
	นายเอกสิทธิ์ บุตรดา	รหัสนักศึกษา 62050251
ปริญญา	วิทยาศาสตรบัณฑิต (วิทยาการคอมพิวเตอร์)	
ภาควิชา	วิทยาการคอมพิวเตอร์	
คณะ	วิทยาศาสตร์	
มหาวิทยาลัย	สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง	
ปีการศึกษา	2565	
อาจารย์ที่ปรึกษา	ดร.อัคเดช อุดมชัยพร	
อาจารย์ที่ปรึกษาร่วม	ศ.พญ.กิติรัตน์ อังกานนท์	

บทคัดย่อ

ปัญหาพิเศษนี้นำเสนอการนำเทคโนโลยี Machine Learning มาช่วยในการจำแนกองค์ประกอบที่ใช้วินิจฉัยโรคหูชั้นกลางอักเสบ เพื่อช่วยลดภาระของแพทย์ผู้เชี่ยวชาญ โดยใช้ Support Vector Machine, K-Nearest Neighbors, Decision Tree, XGBoost และ Neural Network โดย Neural Network จะใช้โครงข่ายประสาทเทียมแบบคอนโวลูชัน (Convolutional Neural Network) ซึ่งประกอบด้วยสถาปัตยกรรม Xception MobileNetV2 และ ConvNext สำหรับตัวอย่างภาพจากกล้อง Otoscope ถูกประมาณผลด้วยการเพิ่มตัวอย่างของภาพด้วยวิธีการหมุนภาพและการบีบบังความสว่างของภาพ ซึ่งจะแบ่งข้อมูลสำหรับฝึกฝน 70% และข้อมูลสำหรับทดสอบ 30% โดยการทดลองฝึกฝนโมเดลที่จำแนกองค์ประกอบที่ใช้ในการวินิจฉัยโรคทั้งหมด 5 องค์ประกอบ แล้วนำไปวินิจฉัยตามเงื่อนไขที่กำหนดโดยแพทย์ผู้เชี่ยวชาญ โดยโมเดลจะถูกประเมินประสิทธิภาพด้วยมาตรฐานดังต่อไปนี้คือ Accuracy, Sensitivity, Specificity, Precision, F1 และ AUC ผลการทดสอบพบว่าโมเดล Neural Networks ที่จำแนกองค์ประกอบที่ใช้ในการวินิจฉัยโรคทั้งหมด 5 องค์ประกอบ มีความแม่นยำที่ดีที่สุด ในแต่ละองค์ประกอบดังนี้ Perforation คือ MobileNetV2 มี Accuracy อยู่ที่ 68.9%, Fluid คือ Xception มี Accuracy อยู่ที่ 60%, Retraction คือ ConvNext มี Accuracy อยู่ที่ 51.1%, Transparency คือ Xception มี Accuracy อยู่ที่ 45.3%, และ Color คือ Xception มี Accuracy อยู่ที่ 25.9% ส่วนโมเดล Traditional Machine Learning มีความแม่นยำที่ดีที่สุดในแต่ละองค์ประกอบดังนี้ Perforation คือ SVM มี Accuracy อยู่ที่ 95%, Fluid คือ SVM มี Accuracy อยู่ที่ 68%, Retraction คือ Decision Tree มี Accuracy อยู่ที่ 57%, Transparency คือ XGBoost มี Accuracy อยู่ที่ 47% และ Color คือ XGBoost มี Accuracy อยู่ที่ 30%

คำสำคัญ : แมชชีนเลิร์นนิ่ง, โครงข่ายประสาทเทียม, ภาพจากกล้องออตอสโคป, โรคหูชั้นกลางอักเสบ

Title	Classification of Otitis Media Factors from Otoscope Image		
Students	Mr. Methanon Kaeokrachang	Student ID	62050214
	Mr. Aekkasit Budda	Student ID	62050251
Degree	Bachelor of Science (Computer Science)		
Department	Computer Science		
Faculty	School of Science		
University	King Mongkut's Institute of Technology Ladkrabang (KMITL)		
Academic year	2565		
Advisor	Dr.Akadej Udomchaiporn		
Co-advisor	Prof.Kitirat Ungkanont		

Abstract

This special issue proposes the use of Machine Learning technology to aid in the classification of middle ear inflammation components for the diagnosis of otitis media, with the aim of reducing the burden on expert physicians. The proposed models include Support Vector Machine, K-Nearest Neighbors, Decision Tree, XGBoost, and Neural Network, with the Neural Network utilizing a Convolutional Neural Network architecture comprising Xception, MobileNetV2, and ConvNext. For the example images captured by an Otoscope camera, data augmentation techniques such as image rotation and brightness adjustment are applied to increase the image dataset. The dataset is then divided into 70% for training and 30% for testing. The models are trained to classify the five components used for diagnosing the disease. The performance of the models is evaluated using metrics such as Accuracy, Sensitivity, Specificity, Precision, F1 score, and Area Under the Curve (AUC). The testing results reveal that the Neural Network model achieved the highest accuracy for each component: MobileNetV2 achieved 68.9% accuracy for Perforation, Xception achieved 60% accuracy for Fluid, ConvNext achieved 51.1% accuracy for Retraction, Xception achieved 45.3% accuracy for Transparency, and Xception achieved 25.9% accuracy for Color. On the other hand, the Traditional Machine Learning models achieved the highest accuracy for each component as follows: SVM achieved 95% accuracy for Perforation, SVM achieved 68% accuracy for Fluid, Decision Tree achieved 57% accuracy for Retraction, XGBoost achieved 47% accuracy for Transparency, and XGBoost achieved 30% accuracy for Color.

Keywords: Machine Learning, Neural Network, Otitis Media Disease, Otoscope camera images

กิตติกรรมประกาศ

บัญชาพิเศษเรื่องการจำแนกงบประจำรอบของโรคหูชั้นกลางอักเสบจากภาพօอโสโคปในระดับปริญญาตรีนี้สามารถสำเร็จลุล่วงไปด้วยดี ผู้จัดทำข้อขอขอบพระคุณอาจารย์ที่ปรึกษา ดร.อัคเดช อุดมชัย พร และอาจารย์ที่ปรึกษาร่วม ศ.พญ.กิตติรัตน์ อังกานนท์ ที่ได้ให้คำแนะนำชี้แนะแนวทางการแก้ไขบัญชา เกี่ยวกับการแพทย์และการประยุกต์ใช้ให้เกิดประโยชน์สูงสุดในการพัฒนาปรับปรุงบัญชาพิเศษนี้

ขอขอบพระคุณคณาจารย์ผู้ควบคุมการสอบบัญชาพิเศษ ผศ.ดร. อนันตพร ธรรมคุณณาย และ ผศ.ดร. ธีระ ศิริธีรากุล ที่ช่วยในการตรวจสอบ และให้คำแนะนำ ทำให้บัญชาพิเศษนี้มีความสมบูรณ์ยิ่งขึ้น

สุดท้ายนี้ ขอบพระคุณสถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ซึ่งเป็นหนึ่งในปัจจัยที่ทำให้ได้เรียนรู้กับคณาจารย์ที่มีศักยภาพ ในการช่วยพัฒนาทักษะและมอบความรู้ให้แก่นักศึกษา จนสามารถลุล่วงการทำบัญชาพิเศษนี้ไปได้ด้วยดี

เมธานนท์ แก้วกระจ่าง
เอกสิทธิ์ บุตรดา

สารบัญ

หน้า

บทบทคัดย่อภาษาไทย	ก
บทคัดย่อภาษาอังกฤษ	ค
กิตติกรรมประกาศ	จ
สารบัญ	ฉ
สารบัญตาราง	ช
สารบัญรูป	ภ
บทที่ 1 บทนำ	1
1.1 ความเป็นมาและความสำคัญ	1
1.2 วัตถุประสงค์ของงานวิจัย	2
1.3 ขอบเขตของงานวิจัย	2
1.4 ประโยชน์ที่คาดว่าจะได้รับ	2
1.5 ขั้นตอนการดำเนินงาน	3
บทที่ 2 ทฤษฎีและงานวิจัยที่เกี่ยวข้อง	3
2.1 โรคหูชั้นกลางอักเสบ (Otitis Media Disease)	3
2.2 การวินิจฉัยโรคหูชั้นกลางอักเสบ (Diagnosis of Otitis Media)	4
2.3 โครงข่ายประสาทเทียมแบบconvoluted neural network (Convolutional Neural Network)	4
2.3.1 Input layer	5
2.3.2 Kernel หรือ Filter	5
2.3.3 Padding	6
2.3.4 Stride	6
2.3.5 Pooling layer	7

2.3.6 Flatten Layer.....	7
2.3.7 Fully Connected Layer.....	7
2.3.8 Activation Function.....	8
2.3.10 Data Augmentation.....	9
2.3.12 Gradient Descent.....	9
2.3.13 Learning Rate	10
2.3.14 Optimizer	10
2.3.15 Loss.....	12
2.5 Decision Tree.....	13
2.6 KNN.....	14
2.7 XGBoost.....	15
2.8 มาตรวัด	16
2.9 งานวิจัยที่เกี่ยวข้อง.....	16
2.9.1 Artificial Intelligence to classify ear disease from otoscopy: A systematic review and meta-analysis	16
2.9.2 Otitis media detection using tympanic membrane images with a novel multi-class machine learning algorithm	17
2.9.3 OtoMatch: Content-based eardrum image retrieval using deep learning.....	17
2.9.4 MobileNetV2: Inverted Residuals and Linear Bottlenecks.....	18
2.9.5 Xception: Deep Learning with Depthwise Separable Convolutions.....	18
2.9.6 A ConvNet for the 2020s	19
บทที่ 3 วิธีการดำเนินงาน.....	20
3.1 ระเบียบวิธีวิจัย.....	20
3.1.2 คัดเลือกภาพที่ไม่ใช่บริเวณหูชั้นกลางออกจากชุดข้อมูล	21

3.1.3 เพิ่มจำนวนข้อมูลเพื่อลดปัญหาความไม่สมดุลของคุณลักษณะ	22
3.1.4 สร้างโมเดลด้วย Neural Networks และ Traditional Machine Learning	22
3.2 การประมวลผลข้อมูลก่อนการฝึกฝน.....	23
3.2.1 การเพิ่มจำนวนข้อมูลด้วย Augmentation	23
3.2.2 ตั้งค่า Neural Network Model	24
บทที่ 4 ผลการดำเนินงานและการอภิปราย	88
4.1 ผลการทดลอง	88
4.1.1 ประสิทธิภาพโมเดลแบบ Neural Networks และ Traditional Machine Learning สำหรับการวิเคราะห์องค์ประกอบ Perforation.....	88
4.1.2 ประสิทธิภาพโมเดลแบบ Neural Networks และ Traditional Machine Laerning สำหรับการวิเคราะห์องค์ประกอบ Fluid.....	91
4.1.4 ประสิทธิภาพโมเดลแบบ Neural Networks และ Traditional Machine Laerning สำหรับการวิเคราะห์องค์ประกอบ Transparency.....	96
4.1.5 ประสิทธิภาพโมเดลแบบ Neural Networks และ Traditional Machine Laerning สำหรับการวิเคราะห์องค์ประกอบ Color	99
4.2 การอภิปรายผล.....	102
4.2.1 องค์ความรู้จากการทดลองโมเดลแบบ Neural Networks	102
4.2.2 องค์ความรู้จากการทดลองโมเดลแบบ Traditional Machine Learning.....	102
4.2.3 องค์ความรู้จากการเพิ่มจำนวนข้อมูล	102
บทที่ 5 สรุปผลการดำเนินงานและข้อเสนอแนะ	103
5.1 สรุปผลดำเนินงาน	103
5.2 ข้อเสนอแนะ	103
5.2.1 เพิ่มจำนวนข้อมูล	103
5.2.2 เพิ่มการวิเคราะห์แบบระบุตำแหน่งขององค์ประกอบ.....	104

5.2.2 เพิ่มการวิเคราะห์แบบภาพเคลื่อนไหว.....	104
บรรณานุกรม	104

สารบัญตาราง

หน้า

ตารางที่ 3.1 ตารางแสดงคุณลักษณะขององค์ประกอบของโรคหูชั้นกลางอักเสบ	21
ตารางที่ 3.2 ตารางแสดงคุณลักษณะของการวินิจฉัยโรคหูชั้นกลางอักเสบ.....	21
ตารางที่ 3.3 ตารางแสดงคุณลักษณะขององค์ประกอบของโรคหูชั้นกลางอักเสบหลังแก้ไขคุณลักษณะ	22
ตารางที่ 3.4 ตารางแสดงคุณลักษณะของการวินิจฉัยโรคหูชั้นกลางอักเสบหลังแก้ไขคุณลักษณะ	22
ตารางที่ 3.5 ตาราง Parameter ของสถาปัตยกรรม Xception	25
ตารางที่ 3.6 ตาราง Parameter ของสถาปัตยกรรม MobileNet V2	26
ตารางที่ 3.7 ตาราง Parameter ของสถาปัตยกรรม ConvNext Tiny	28
ตารางที่ 3.8 ตาราง Parameter ของสถาปัตยกรรม SVM	29
ตารางที่ 3.8 ตาราง Parameter ของสถาปัตยกรรม KNN.....	29
ตารางที่ 3.9 ตาราง Parameter ของสถาปัตยกรรม Decision Tree.....	30
ตารางที่ 3.10 ตาราง Parameter ของสถาปัตยกรรม XGBoost.....	31
ตารางที่ 4.1 ตารางแสดงประสิทธิภาพโมเดลแบบ Neural Networks สำหรับ Perforation	91
ตารางที่ 4.2 ตารางแสดงประสิทธิภาพโมเดลแบบ Traditional Machine Learning สำหรับ Perforation	91
ตารางที่ 4.3 ตารางแสดงประสิทธิภาพโมเดลแบบ Neural Networks สำหรับ Fluid	93
ตารางที่ 4.4 ตารางแสดงประสิทธิภาพโมเดลแบบ Traditional Machine Learning สำหรับ Fluid.....	93
ตารางที่ 4.5 ตารางแสดงประสิทธิภาพโมเดลแบบ Neural Networks สำหรับ Retraction	95
ตารางที่ 4.6 ตารางแสดงประสิทธิภาพโมเดลแบบ Traditional Machine Learning สำหรับ Retraction	96
ตารางที่ 4.7 ตารางแสดงประสิทธิภาพโมเดลแบบ Neural Networks สำหรับ Transparency.....	97
ตารางที่ 4.8 ตารางแสดงประสิทธิภาพโมเดลแบบ Traditional Machine Learning สำหรับ Transparency	98
ตารางที่ 4.9 ตารางแสดงประสิทธิภาพโมเดลแบบ Neural Networks สำหรับ Color	100
ตารางที่ 4.10 ตารางแสดงประสิทธิภาพโมเดลแบบ Traditional Machine Learning สำหรับ Color..	100

สารบัญรูป

หน้า

รูปที่ 1.1 ความถี่โรคหูน้ำหนวก ráy แรงระห่วงปี 2557-2561 ข้อมูลจากการสาธารณสุขและการแพทย์ ปีที่ 46	46
ฉบับที่ 2 เมษาคม - มิถุนายน 2564	1
รูปที่ 2.1 Input Layer.....	5
รูปที่ 2.2 Input Layer และ Kernel.....	6
รูปที่ 2.3 Output หลังจากสกัด Feature ด้วย Kernel	6
รูปที่ 2.4 Padding.....	6
รูปที่ 2.5 Max Pooling	7
รูปที่ 2.6 Flatten Layer.....	8
รูปที่ 2.7 Artificial Neural Network.....	8
รูปที่ 2.8 จุดต่ำสุดของสมการตัวแปรเดียว	10
รูปที่ 2.9 จุดต่ำสุดของสมการสองตัวแปร	10
รูปที่ 2.10 SVM	13
รูปที่ 2.11 ต้นไม้ตัดสินใจ	14
รูปที่ 2.12 KNN	15
รูป 2.13 XGBoost	16
รูปที่ 2.14 MobileNet V2	18
รูปที่ 2.15 Xception	19
รูปที่ 2.16 ConvNext Tiny	19
รูปที่ 3.2 ภาพที่ไม่ใช่บริเวณหูชั้นกลาง	22
รูปที่ 3.3 ตัวอย่างจำนวนข้อมูลในแต่ละ Class ของ Perforation Attributes	23
รูปที่ 3.4 ตัวอย่างคำสั่งที่ใช้ในการหมุนภาพ.....	24
รูปที่ 3.5 ตัวอย่างคำสั่งที่ใช้ในการเพิ่ม-ลดแสง	24
รูปที่ 3.6 ตัวอย่างการ Augmentation	24

รูปที่ 3.7 พังก์ชันสำหรับตั้งค่าการฝึกฝนโมเดล	25
รูปที่ 3.8 ตั้งค่า Layer	25
รูปที่ 3.9 ตั้งค่าการ Compile โมเดล	26
รูปที่ 3.10 พังก์ชันตั้งค่าการฝึกฝนโมเดล MobileNet V2	27
รูปที่ 3.11 ตั้งค่า Layer MobileNet V2	27
รูปที่ 3.12 ตั้งค่าการ compile โมเดล	27
รูปที่ 3.13 พังก์ชันตั้งค่าการฝึกฝนโมเดล ConvNext Tiny	28
รูปที่ 3.14 ตั้งค่า Layer ConvNext Tiny	28
รูปที่ 3.15 ตั้งค่าการ compile โมเดล	28
รูปที่ 3.16 การตั้งค่าโมเดลแบบ SVM	29
รูปที่ 3.17 การตั้งค่าโมเดลแบบ KNN	30
รูปที่ 3.18 การตั้งค่าโมเดลแบบ Decision Tree	30
รูปที่ 3.19 การตั้งค่าโมเดลแบบ XGBoost	31
รูปที่ 3.20 ตัวอย่างคุณลักษณะ Perforation แบบ Yes และ No	32
รูปที่ 3.21 จำนวนข้อมูลในแต่ละ Class ของ Perforation	32
รูปที่ 3.22 จำนวนข้อมูลในแต่ละ Class ของ Perforation หลังทำการ Augmentation	33
รูปที่ 3.23 ตัวอย่างคำสั่งการฝึกฝนโมเดล Perforation ด้วย Xception	33
รูปที่ 3.24 ตัวอย่างผลลัพธ์ระหว่างการฝึกฝนโมเดล Perforation ด้วย Xception	33
รูปที่ 3.26 ตัวอย่าง Loss ของโมเดล Perforation ที่ฝึกฝนด้วย Xception	34
รูปที่ 3.27 ตัวอย่าง AUC ของโมเดล Perforation ที่ฝึกฝนด้วย Xception	34
รูปที่ 3.28 ตัวอย่าง Confusion ของโมเดล Perforation ที่ฝึกฝนด้วย Xception	34
รูปที่ 3.29 ตัวอย่างคำสั่งในการฝึกฝนโมเดล Perforation ด้วย MobileNet V2	35
รูปที่ 3.30 ตัวอย่างผลลัพธ์ระหว่างการฝึกฝนโมเดล Perforation ด้วย MobileNet V2	35
รูปที่ 3.31 ตัวอย่าง Accuracy ของโมเดล Perforation ที่ฝึกฝนด้วย MobileNet V2	35
รูปที่ 3.33 ตัวอย่าง AUC ของโมเดล Perforation ที่ฝึกฝนด้วย MobileNet V2	36

รูปที่ 3.34 ตัวอย่าง Confusion Matrix ของโมเดล Perforation ที่ฝึกฝนด้วย MobileNet V2.....	36
รูปที่ 3.35 ตัวอย่างคำสั่งในการฝึกฝนโมเดล Perforation ด้วย ConvNext Tiny	36
รูปที่ 3.36 ตัวอย่างผลลัพธ์ระหว่างการฝึกฝนโมเดล Perforation ด้วย ConvNext Tiny	36
รูปที่ 3.37 ตัวอย่าง AUC ของโมเดล Perforation ที่ฝึกฝนด้วย ConvNext Tiny	37
รูปที่ 3.38 ตัวอย่าง Accuracy ของโมเดล Perforation ที่ฝึกฝนด้วย ConvNext Tiny	37
รูปที่ 3.39 ตัวอย่าง Loss ของโมเดล Perforation ที่ฝึกฝนด้วย ConvNext Tiny	37
รูปที่ 3.40 ตัวอย่าง Confusion Matrix ของโมเดล Perforation ที่ฝึกฝนด้วย ConvNext Tiny.....	37
รูปที่ 3.41 ตัวอย่างคุณลักษณะ None, Bubble, Level และ Full.....	38
รูปที่ 3.42 จำนวนข้อมูลในแต่ละ Class ของ Fluid	38
รูปที่ 3.43 จำนวนข้อมูลในแต่ละ Class ของ Fluid หลังทำการ Augmentation	39
รูปที่ 3.44 ตัวอย่างคำสั่งการฝึกฝนโมเดล Fluid ด้วย Xception	39
รูปที่ 3.45 ตัวอย่างผลลัพธ์ระหว่างการฝึกฝนโมเดล Fluid ด้วย Xception	39
รูปที่ 3.46 ตัวอย่าง Accuracy ของโมเดล Fluid ที่ฝึกฝนโดยใช้ Xception.....	40
รูปที่ 3.47 ตัวอย่าง Loss ของโมเดล Fluid ที่ฝึกฝนโดยใช้ Xception	40
รูปที่ 3.48 ตัวอย่าง AUC ของการฝึกฝน Fluid ด้วย Xception	40
รูปที่ 3.49 ตัวอย่าง Confusion Matrix ของการฝึกฝน Fluid ด้วย Xception.....	40
รูปที่ 3.50 ตัวอย่างคำสั่งการฝึกฝนโมเดล Fluid ด้วย MobileNet V2	41
รูปที่ 3.51 ตัวอย่างผลลัพธ์ระหว่างการฝึกฝนโมเดล Fluid ด้วย MobileNet V2	41
รูปที่ 3.53 ตัวอย่าง Loss ของโมเดล Fluid ที่ฝึกฝนโดยใช้ MobileNet V2	42
รูปที่ 3.54 ตัวอย่าง AUC ของโมเดล Fluid ที่ฝึกฝนโดยใช้ MobileNet V2	42
รูปที่ 3.55 ตัวอย่าง Confusion Matrix ของโมเดล Fluid ที่ฝึกฝนโดยใช้ MobileNet V2	42
รูปที่ 3.56 ตัวอย่างคำสั่งการฝึกฝนโมเดล Fluid ด้วย ConvNext Tiny	42
รูปที่ 3.57 ตัวอย่างผลลัพธ์ระหว่างการฝึกฝนโมเดล Fluid ด้วย ConvNext Tiny	43
รูปที่ 3.58 ตัวอย่าง Accuracy ของโมเดล Fluid ที่ฝึกฝนโดยใช้ ConvNext Tiny	43
รูปที่ 3.59 ตัวอย่าง Loss ของโมเดล Fluid ที่ฝึกฝนโดยใช้ ConvNext Tiny	43

รูปที่ 3.60 ตัวอย่าง AUC ของโมเดล Fluid ที่ฝึกฝนโดยใช้ ConvNext Tiny	44
รูปที่ 3.61 ตัวอย่าง Confusion Matrix ของโมเดล Fluid ที่ฝึกฝนโดยใช้ ConvNext Tiny	44
รูปที่ 3.62 ตัวอย่างคำสั่งการฝึกฝนโมเดล Fluid ด้วย SVM	44
รูปที่ 3.63 ตัวอย่างผลลัพธ์โดยรวมของการฝึกฝนโมเดล Fluid ด้วย SVM	45
รูปที่ 3.64 ตัวอย่างผลลัพธ์ AUC ของการฝึกฝนโมเดล Fluid ด้วย SVM	45
รูปที่ 3.65 ตัวอย่าง Confusion Matrix ของการฝึกฝนโมเดล Fluid ด้วย SVM	45
รูปที่ 3.66 ตัวอย่างคำสั่งการฝึกฝนโมเดล Fluid ด้วย KNN	46
รูปที่ 3.67 ตัวอย่างผลลัพธ์โดยรวมของการฝึกฝนโมเดล Colot ด้วย KNN	46
รูปที่ 3.68 ตัวอย่าง AUC ของการฝึกฝนโมเดล Fluid ด้วย KNN	47
รูปที่ 3.69 ตัวอย่าง Confusion Matrix ของการฝึกฝนโมเดล Fluid ด้วย KNN	47
รูปที่ 3.70 ตัวอย่างคำสั่งการฝึกฝนโมเดล Fluid ด้วย Decision Tree	47
รูปที่ 3.71 ตัวอย่างผลลัพธ์โดยรวมของการฝึกฝน Fluid ด้วย Decision Tree	48
รูปที่ 3.72 ตัวอย่าง AUC ของการฝึกฝนโมเดล Fluid ด้วย Decision Tree	48
รูปที่ 3.73 ตัวอย่าง Confusion Matrix ของการฝึกฝนโมเดล Fluid ด้วย Decision Tree	48
รูปที่ 3.74 ตัวอย่างคำสั่งการฝึกฝน Fluid ด้วย XGBoost	49
รูปที่ 3.75 ตัวอย่างผลลัพธ์โดยรวมของการฝึกฝน Fluid ด้วย XGBoost	49
รูปที่ 3.76 ตัวอย่าง AUC ของการฝึกฝนโมเดล Fluid ด้วย XGBoost	50
รูปที่ 3.77 ตัวอย่าง Confusion Matrix ของการฝึกฝนโมเดล Fluid ด้วย XGBoost	50
รูปที่ 3.78 ตัวอย่างคุณลักษณะ None, Mild, Moderate, Severe และ Bulging	50
รูปที่ 3.79 จำนวนข้อมูลในแต่ละ Class ของ Retraction	51
รูปที่ 3.80 จำนวนข้อมูลในแต่ละ Class ของ Retraction หลังทำการ Augmentation	51
รูปที่ 3.81 ตัวอย่างคำสั่งการฝึกฝนโมเดล Retraction ด้วย Xception	52
รูปที่ 3.82 ตัวอย่างผลลัพธ์ระหว่างการฝึกฝนโมเดล Retraction ด้วย Xception	52
รูปที่ 3.83 ตัวอย่าง Accuracy ของโมเดล Retraction ที่ฝึกฝนโดยใช้ Xception	52
รูปที่ 3.84 ตัวอย่าง Loss ของโมเดล Retraction ที่ฝึกฝนโดยใช้ Xception	53

รูปที่ 3.85 ตัวอย่าง AUC ของการฝึกฝน Retraction ด้วย Xception.....	53
รูปที่ 3.86 ตัวอย่าง Confusion Matrix ของการฝึกฝน Retraction ด้วย Xception.....	53
รูปที่ 3.87 ตัวอย่างคำสั่งการฝึกฝนโมเดล Retraction ด้วย MobileNet V2	54
รูปที่ 3.88 ตัวอย่างผลลัพธ์ระหว่างการฝึกฝนโมเดล Retraction ด้วย MobileNet V2	54
รูปที่ 3.89 ตัวอย่าง Accuracy ของโมเดล Retraction ที่ฝึกฝนโดยใช้ MobileNet V2	54
รูปที่ 3.90 ตัวอย่าง Loss ของโมเดล Retraction ที่ฝึกฝนโดยใช้ MobileNet V2	54
รูปที่ 3.91 ตัวอย่าง AUC ของโมเดล Retraction ที่ฝึกฝนโดยใช้ MobileNet V2	55
รูปที่ 3.92 ตัวอย่าง Confusion Matrix ของโมเดล Retraction ที่ฝึกฝนโดยใช้ MobileNet V2	55
รูปที่ 3.93 ตัวอย่างคำสั่งการฝึกฝนโมเดล Retraction ด้วย ConvNext Tiny	55
รูปที่ 3.94 ตัวอย่างผลลัพธ์ระหว่างการฝึกฝนโมเดล Retraction ด้วย ConvNext Tiny	56
รูปที่ 3.95 ตัวอย่าง Accuracy ของโมเดล Retraction ที่ฝึกฝนโดยใช้ ConvNext Tiny	56
รูปที่ 3.96 ตัวอย่าง Loss ของโมเดล Retraction ที่ฝึกฝนโดยใช้ ConvNext Tiny	56
รูปที่ 3.97 ตัวอย่าง AUC ของโมเดล Retraction ที่ฝึกฝนโดยใช้ ConvNext Tiny	57
รูปที่ 3.98 ตัวอย่าง Confusion Matrix ของโมเดล Retraction ที่ฝึกฝนโดยใช้ ConvNext Tiny	57
รูปที่ 3.99 ตัวอย่างคำสั่งการฝึกฝนโมเดล Retraction ด้วย SVM	57
รูปที่ 3.100 ตัวอย่างผลลัพธ์โดยรวมของการฝึกฝนโมเดล Retraction ด้วย SVM	58
รูปที่ 3.101 ตัวอย่างผลลัพธ์ AUC ของการฝึกฝนโมเดล Retraction ด้วย SVM	58
รูปที่ 3.102 ตัวอย่าง Confusion Matrix ของการฝึกฝนโมเดล Retraction ด้วย SVM	58
รูปที่ 3.103 ตัวอย่างคำสั่งการฝึกฝนโมเดล Retraction ด้วย KNN.....	59
รูปที่ 3.104 ตัวอย่างผลลัพธ์โดยรวมของการฝึกฝนโมเดล Colot ด้วย KNN.....	59
รูปที่ 3.105 ตัวอย่าง AUC ของการฝึกฝนโมเดล Retraction ด้วย KNN	59
รูปที่ 3.106 ตัวอย่าง Confusion Matrix ของการฝึกฝนโมเดล Retraction ด้วย KNN.....	60
รูปที่ 3.107 ตัวอย่างคำสั่งการฝึกฝนโมเดล Retraction ด้วย Decision Tree.....	60
รูปที่ 3.108 ตัวอย่างผลลัพธ์โดยรวมของการฝึกฝน Retraction ด้วย Decision Tree	60
รูปที่ 3.109 ตัวอย่าง AUC ของการฝึกฝนโมเดล Retraction ด้วย Decision Tree	61

รูปที่ 3.110 ตัวอย่าง Confusion Matrix ของการฝึกฝนโมเดล Retraction ด้วย Decision Tree	61
รูปที่ 3.111 ตัวอย่างคำสั่งการฝึกฝน Retraction ด้วย XGBoost	61
รูปที่ 3.112 ตัวอย่างผลลัพธ์โดยรวมของการฝึกฝน Retraction ด้วย XGBoost	62
รูปที่ 3.113 ตัวอย่าง AUC ของการฝึกฝนโมเดล Retraction ด้วย XGBoost.....	62
รูปที่ 3.114 ตัวอย่าง Confusion Matrix ของการฝึกฝนโมเดล Retraction ด้วย XGBoost.....	62
รูปที่ 3.115 ตัวอย่างคุณลักษณะ Clear, Dull, Opaque และ Tympanosclerosis	63
รูปที่ 3.116 จำนวนข้อมูลในแต่ละ Class ของ Transparency.....	63
รูปที่ 3.117 จำนวนข้อมูลในแต่ละ Class ของ Transparency หลังทำการ Augmentation.....	64
รูปที่ 3.118 ตัวอย่างคำสั่งการฝึกฝนโมเดล Transparency ด้วย Xception.....	64
รูปที่ 3.119 ตัวอย่างผลลัพธ์ระหว่างการฝึกฝนโมเดล Transparency ด้วย Xception	64
รูปที่ 3.120 ตัวอย่าง Accuracy ของโมเดล Transparency ที่ฝึกฝนโดยใช้ Xception	65
รูปที่ 3.121 ตัวอย่าง Loss ของโมเดล Transparency ที่ฝึกฝนโดยใช้ Xception	65
รูปที่ 3.122 ตัวอย่าง AUC ของการฝึกฝน Transparency ด้วย Xception	65
รูปที่ 3.123 ตัวอย่าง Confusion Matrix ของการฝึกฝน Transparency ด้วย Xception	66
รูปที่ 3.124 ตัวอย่างคำสั่งการฝึกฝนโมเดล Transparency ด้วย MobileNet V2.....	66
รูปที่ 3.125 ตัวอย่างผลลัพธ์ระหว่างการฝึกฝนโมเดล Transparency ด้วย MobileNet V2	66
รูปที่ 3.126 ตัวอย่าง Accuracy ของโมเดล Transparency ที่ฝึกฝนโดยใช้ MobileNet V2	67
รูปที่ 3.127 ตัวอย่าง Loss ของโมเดล Transparency ที่ฝึกฝนโดยใช้ MobileNet V2	67
รูปที่ 3.128 ตัวอย่าง AUC ของโมเดล Transparency ที่ฝึกฝนโดยใช้ MobileNet V2	67
รูปที่ 3.129 ตัวอย่าง Confusion Matrix ของโมเดล Transparency ที่ฝึกฝนโดยใช้ MobileNet V2.....	67
รูปที่ 3.130 ตัวอย่างคำสั่งการฝึกฝนโมเดล Transparency ด้วย ConvNext Tiny.....	68
รูปที่ 3.131 ตัวอย่างผลลัพธ์ระหว่างการฝึกฝนโมเดล Transparency ด้วย ConvNext Tiny	68
รูปที่ 3.132 ตัวอย่าง Accuracy ของโมเดล Transparency ที่ฝึกฝนโดยใช้ ConvNext Tiny	68
รูปที่ 3.133 ตัวอย่าง Loss ของโมเดล Transparency ที่ฝึกฝนโดยใช้ ConvNext Tiny.....	69
รูปที่ 3.134 ตัวอย่าง AUC ของโมเดล Transparency ที่ฝึกฝนโดยใช้ ConvNext Tiny.....	69

รูปที่ 3.135 ตัวอย่าง Confusion Matrix ของโมเดล Transparency ที่ฝึกฝนโดยใช้ ConvNext Tiny ...	69
รูปที่ 3.136 ตัวอย่างคำสั่งการฝึกฝนโมเดล Transparency ด้วย SVM	70
รูปที่ 3.137 ตัวอย่างผลลัพธ์โดยรวมของการฝึกฝนโมเดล Transparency ด้วย SVM	70
รูปที่ 3.138 ตัวอย่างผลลัพธ์ AUC ของการฝึกฝนโมเดล Transparency ด้วย SVM	70
รูปที่ 3.139 ตัวอย่าง Confusion Matrix ของการฝึกฝนโมเดล Transparency ด้วย SVM	71
รูปที่ 3.140 ตัวอย่างคำสั่งการฝึกฝนโมเดล Transparency ด้วย KNN	71
รูปที่ 3.141 ตัวอย่างผลลัพธ์โดยรวมของการฝึกฝนโมเดล Colot ด้วย KNN	71
รูปที่ 3.142 ตัวอย่าง AUC ของการฝึกฝนโมเดล Transparency ด้วย KNN	72
รูปที่ 3.143 ตัวอย่าง Confusion Matrix ของการฝึกฝนโมเดล Transparency ด้วย KNN	72
รูปที่ 3.144 ตัวอย่างคำสั่งการฝึกฝนโมเดล Transparency ด้วย Decision Tree	72
รูปที่ 3.145 ตัวอย่างผลลัพธ์โดยรวมของการฝึกฝน Transparency ด้วย Decision Tree	73
รูปที่ 3.146 ตัวอย่าง AUC ของการฝึกฝนโมเดล Transparency ด้วย Decision Tree	73
รูปที่ 3.147 ตัวอย่าง Confusion Matrix ของการฝึกฝนโมเดล Transparency ด้วย Decision Tree	73
รูปที่ 3.148 ตัวอย่างคำสั่งการฝึกฝน Transparency ด้วย XGBoost	74
รูปที่ 3.149 ตัวอย่างผลลัพธ์โดยรวมของการฝึกฝน Transparency ด้วย XGBoost	74
รูปที่ 3.150 ตัวอย่าง AUC ของการฝึกฝนโมเดล Transparency ด้วย XGBoost	75
รูปที่ 3.151 ตัวอย่าง Confusion Matrix ของการฝึกฝนโมเดล Transparency ด้วย XGBoost	75
รูปที่ 3.152 ตัวอย่างคุณลักษณะ Normal, White, Amber, Blue, Pearly White, Milky, Light Yellow และ Red	75
รูปที่ 3.153 จำนวนข้อมูลในแต่ละ Class ของ Color	76
รูปที่ 3.154 จำนวนข้อมูลในแต่ละ Class ของ Color หลังทำการ Augmentation	76
รูปที่ 3.155 ตัวอย่างคำสั่งการฝึกฝนโมเดล Color ด้วย Xception	77
รูปที่ 3.156 ตัวอย่างผลลัพธ์ระหว่างการฝึกฝนโมเดล Color ด้วย Xception	77
รูปที่ 3.157 ตัวอย่าง Accuracy ของโมเดล Color ที่ฝึกฝนโดยใช้ Xception	77
รูปที่ 3.158 ตัวอย่าง Loss ของโมเดล Color ที่ฝึกฝนโดยใช้ Xception	78

รูปที่ 3.159 ตัวอย่าง AUC ของการฝึกฝน Color ด้วย Xception.....	78
รูปที่ 3.160 ตัวอย่าง Confusion Matrix ของการฝึกฝน Color ด้วย Xception.....	78
รูปที่ 3.161 ตัวอย่างคำสั่งการฝึกฝนโมเดล Color ด้วย MobileNet V2	79
รูปที่ 3.162 ตัวอย่างผลลัพธ์ระหว่างการฝึกฝนโมเดล Color ด้วย MobileNet V2	79
รูปที่ 3.163 ตัวอย่าง Accuracy ของโมเดล Color ที่ฝึกฝนโดยใช้ MobileNet V2.....	79
รูปที่ 3.164 ตัวอย่าง Loss ของโมเดล Color ที่ฝึกฝนโดยใช้ MobileNet V2	79
รูปที่ 3.165 ตัวอย่าง AUC ของโมเดล Color ที่ฝึกฝนโดยใช้ MobileNet V2	80
รูปที่ 3.166 ตัวอย่าง Confusion Matrix ของโมเดล Color ที่ฝึกฝนโดยใช้ MobileNet V2	80
รูปที่ 3.167 ตัวอย่างคำสั่งการฝึกฝนโมเดล Color ด้วย ConvNext Tiny	80
รูปที่ 3.168 ตัวอย่างผลลัพธ์ระหว่างการฝึกฝนโมเดล Color ด้วย ConvNext Tiny	81
รูปที่ 3.169 ตัวอย่าง Accuracy ของโมเดล Color ที่ฝึกฝนโดยใช้ ConvNext Tiny	81
รูปที่ 3.170 ตัวอย่าง Loss ของโมเดล Color ที่ฝึกฝนโดยใช้ ConvNext Tiny	81
รูปที่ 3.171 ตัวอย่าง AUC ของโมเดล Color ที่ฝึกฝนโดยใช้ ConvNext Tiny	82
รูปที่ 3.172 ตัวอย่าง Confusion Matrix ของโมเดล Color ที่ฝึกฝนโดยใช้ ConvNext Tiny	82
รูปที่ 3.173 ตัวอย่างคำสั่งการฝึกฝนโมเดล Color ด้วย SVM	82
รูปที่ 3.174 ตัวอย่างผลลัพธ์โดยรวมของการฝึกฝนโมเดล Color ด้วย SVM	83
รูปที่ 3.175 ตัวอย่างผลลัพธ์ AUC ของการฝึกฝนโมเดล Color ด้วย SVM	83
รูปที่ 3.176 ตัวอย่าง Confusion Matrix ของการฝึกฝนโมเดล Color ด้วย SVM	83
รูปที่ 3.177 ตัวอย่างคำสั่งการฝึกฝนโมเดล Color ด้วย KNN	84
รูปที่ 3.178 ตัวอย่างผลลัพธ์โดยรวมของการฝึกฝนโมเดล Colot ด้วย KNN.....	84
รูปที่ 3.179 ตัวอย่าง AUC ของการฝึกฝนโมเดล Color ด้วย KNN	85
รูปที่ 3.180 ตัวอย่าง Confusion Matrix ของการฝึกฝนโมเดล Color ด้วย KNN.....	85
รูปที่ 3.180 ตัวอย่างคำสั่งการฝึกฝนโมเดล Color ด้วย Decision Tree.....	85
รูปที่ 3.181 ตัวอย่างผลลัพธ์โดยรวมของการฝึกฝน Color ด้วย Decision Tree	86
รูปที่ 3.182 ตัวอย่าง AUC ของการฝึกฝนโมเดล Color ด้วย Decision Tree	86

รูปที่ 3.183 ตัวอย่าง Confusion Matrix ของการฝึกฝนโมเดล Color ด้วย Decision Tree.....	86
รูปที่ 3.184 ตัวอย่างคำสั่งการฝึกฝน Color ด้วย XGBoost	87
รูปที่ 3.185 ตัวอย่างผลลัพธ์โดยรวมของการฝึกฝน Color ด้วย XGBoost	87
รูปที่ 3.186 ตัวอย่าง AUC ของการฝึกฝนโมเดล Color ด้วย XGBoost	88
รูปที่ 3.187 ตัวอย่าง Confusion Matrix ของการฝึกฝนโมเดล Color ด้วย XGBoost.....	88
รูปที่ 3.188 ตัวอย่างประวัติการวินิจฉัยในรูปแบบของ Data Frame	88
รูปที่ 3.189 ตัวอย่างการสร้าง Rule Base ของการวินิจฉัยด้วย Decision Tree	89
รูปที่ 3.190 ตัวอย่าง Rule Base ของการวินิจฉัยที่ได้จาก Decision Tree	89
รูปที่ 4.2 ตัวอย่างผลลัพธ์ AUC ของโมเดล Perforation ที่ฝึกฝนโดย SVM	92
รูปที่ 4.3 ตัวอย่างผลลัพธ์ Confusion Matrix ของโมเดล Perforation ที่ฝึกฝนโดย SVM.....	92
รูปที่ 4.4 ตัวอย่างผลลัพธ์ AUC ของโมเดล Fluid ที่ฝึกฝนโดย SVM	94
รูปที่ 4.5 ตัวอย่างผลลัพธ์ Confusion Matrix ของโมเดล Fluid ที่ฝึกฝนโดย SVM.....	94
รูปที่ 4.6 ตัวอย่างผลลัพธ์ AUC ของโมเดล Retraction ที่ฝึกฝนโดย Decision Tree	96
รูปที่ 4.7 ตัวอย่างผลลัพธ์ Confusion Matrix ของโมเดล Retraction ที่ฝึกฝนโดย XGBoost	96
รูปที่ 4.8 ตัวอย่างผลลัพธ์ AUC ของโมเดล Transparency ที่ฝึกฝนโดย SVM	99
รูปที่ 4.9 ตัวอย่างผลลัพธ์ Confusion Matrix ของโมเดล Transparency ที่ฝึกฝนโดย SVM	99
รูปที่ 4.10 ตัวอย่างผลลัพธ์ AUC ของโมเดล Color ที่ฝึกฝนด้วย SVM	101
รูปที่ 4.11 ตัวอย่างผลลัพธ์ Confusion Matrix ของโมเดล Color ที่ฝึกฝนด้วย SVM	101
รูปที่ 4.12 ตัวอย่างผลลัพธ์ Confusion Matrix ของการวินิจฉัยด้วยโมเดลที่ีที่สุดในแต่ละองค์ประกอบ	102

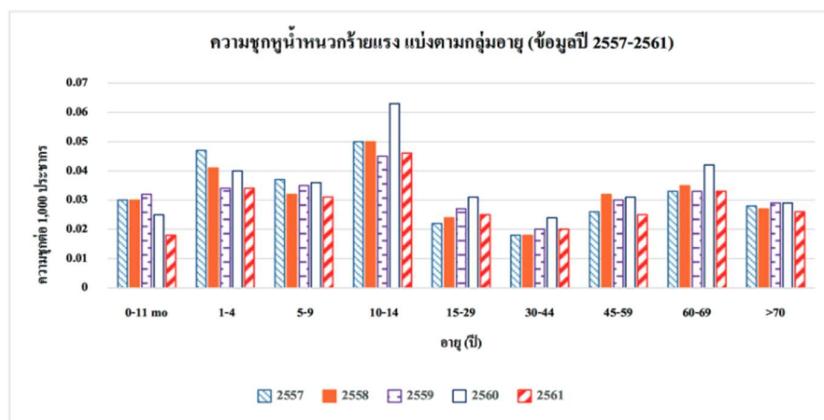
บทที่ 1

บทนำ

1.1 ความเป็นมาและความสำคัญ

โรคหูชั้นกลางอักเสบเป็นโรคที่พบเห็นได้ทั่วไปในประเทศไทยการวินิจฉัยในปัจจุบันมีวิธีการวินิจฉัยเบื้องต้นด้วยการสังเกตอาการ เช่น มีหนองในหลู มีไข้สูง มีอาการปวดหู หรือมีปัญหาการได้ยิน นอกจากการสังเกตอาการแล้วยังมีการใช้เครื่องมือบางชนิดมาช่วยในการวินิจฉัย เช่น การใช้เทคนิคการหาปฏิกิริยาตอบสนองต่อพลังงานคลื่น หรือ Wideband Energy Reflectance เพื่อตรวจสอบของเหลวในหูว่าผิดปกติหรือไม่ โดยใช้เครื่องมือในการปล่อยคลื่นเสียงตั้งแต่ความถี่ 0.25 ถึง 8 กิโลเฮิรตซ์และรับคลื่นสะท้อนกลับมาเพื่อนำไปประมวลผลต่อไป

สำหรับประเทศไทยแนวโน้มของผู้ป่วยมีการเพิ่มขึ้นแต่ยังไม่อยู่ในระดับวิกฤตซึ่งเกณฑ์ของกรมอนามัยโลกได้กำหนดไว้ว่าหากมีผู้ป่วยเกิน 4% ของประชากรในประเทศจะถือว่าอยู่ในระดับวิกฤต ซึ่งในประเทศไทยมีจำนวนผู้ป่วยที่เป็นหูชั้นกลางอักเสบระดับร้ายแรงคืออายุระหว่าง 10-14 ปี



รูปที่ 1.1 ความถี่โรคหูน้ำหนวก咽炎 แบ่งตามอายุ (ข้อมูลปี 2557-2561)

ฉบับที่ 2 เมษายน - มิถุนายน 2564

การวินิจฉัยด้วยการสังเกตอาการเบื้องต้นในบางครั้งอาจไม่สามารถตรวจสอบได้อย่างละเอียด เพราะอาจมีความผิดพลาดที่เกิดจากมนุษย์หรือ Human Error เกิดขึ้นได้ การใช้เทคนิคการวิเคราะห์ภาพด้วยคอมพิวเตอร์มาช่วยในขั้นตอนนี้จะช่วยทั้งในด้านการเพิ่มขึ้นของความแม่นยำการลดกำลังคน

และลดระยะเวลาในการวินิจฉัยเบื้องต้น โดยปัจจุบันการวินิจฉัยโรคต่าง ๆ มีการใช้วิธีการจำแนกภาพด้วย Neural Network Model หรือแบบ Traditional Machine Learning Model

1.2 วัตถุประสงค์ของงานวิจัย

1. เพื่อหาวิธีการจำแนกภาพที่เหมาะสมกับการวิเคราะห์ในแต่ละองค์ประกอบที่สุด
2. เพื่อสร้างโมเดลเพื่อจำแนกแต่ละองค์ประกอบที่เป็นสาเหตุของโรคหูน้ำหนวก
3. เพื่อลดกำลังคน เวลา และแบ่งเบาภาระงานของแพทย์ในการวินิจฉัยโรคหูน้ำหนวก

1.3 ขอบเขตของงานวิจัย

1. การศึกษาครั้งนี้จึงมุ่งเน้นไปที่การทำโมเดลแบบ Classification เพื่อหาลักษณะของแต่ละองค์ประกอบของหูชั้นกลางดังต่อไปนี้
 - ลักษณะของเหลวในหูชั้นกลาง (Fluid)
 - ลักษณะการหดตัวของเยื่อหูชั้นกลาง (Retraction)
 - สีของเหลวในหูชั้นกลาง (Color)
 - การทะลุของเยื่อหูชั้นกลาง (Perforation)
 - ความโปร่งใสของเยื่อหูชั้นกลาง (Transparency)
2. ศึกษาโดยใช้ข้อมูลจากโรงพยาบาลศิริราชโดยมีผู้ป่วยจำนวน 181 ราย โดยภาพที่ได้มาจากการใช้กล้องอโตสโคป (Otoscope) ส่องเข้าไปในหูชั้นกลาง

1.4 ประโยชน์ที่คาดว่าจะได้รับ

1. ได้ทราบวิธีการจำแนกภาพที่เหมาะสมกับแต่ละองค์ประกอบ
2. ได้โน้มเดลที่เหมาะสมกับการจำแนกของค์ประกอบ
3. ได้ลดกำลังคน เวลา และแบ่งเบาภาระงานของแพทย์ในการวินิจฉัยโรค

1.5 ขั้นตอนการดำเนินงาน

1. ศึกษาข้อมูลของโรคหูชั้นกลางอักเสบ (Otitis Media)
2. ศึกษาและทำความเข้าใจความต้องการของแพทย์
3. ศึกษาการใช้ปัญญาประดิษฐ์ในการวิเคราะห์ข้อมูลจากภาพ
4. วิเคราะห์และศึกษาขุดข้อมูลที่ต้องใช้
5. ทำความสะอาดข้อมูลเพื่อพร้อมสำหรับการสร้างโมเดล
6. แบ่งชุดข้อมูลฝึกฝนและชุดข้อมูลทดสอบ
7. ทดลองสร้างโมเดลเพื่อจำแนกภาพ
8. ประเมินประสิทธิภาพของโมเดลและปรับปรุงโมเดลให้มีประสิทธิภาพดีขึ้น
9. วิเคราะห์และสรุปผลการทดลอง

บทที่ 2

ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

2.1 โรคหูชั้นกลางอักเสบ (Otitis Media Disease)

โรคหูชั้นกลางอักเสบ (Otitis Media Disease) คือโรคที่มีการอักเสบของหูชั้นกลางซึ่งอยู่ระหว่างหูชั้นนอกและหูชั้นในซึ่งผู้ป่วยจะมีอาการปวดหูและอาจเป็นไข้ร่วมด้วย ลักษณะของโรคถูกจำแนกออกเป็น 3 ชนิด ประ กอนด้วย โรคหูชั้นกลางอักเสบแบบเฉียบพลัน (Acute Otitis Media: AOM) โรคหูชั้นกลางอักเสบแบบเรื้อรัง (Chronic Suppurative Otitis Media: CSOM) และโรคหูชั้นกลางอักเสบแบบมีน้ำไหล (Otitis Media with Effusion: OME)

สาเหตุแห่งการเกิดโรคมีหลายสาเหตุ เช่น ภูมิคุ้มกันลดลงเนื่องจากภาวะ HIV โรคเบาหวาน ภาวะภูมิคุ้มกันบกพร่องอื่นๆ รวมทั้งความบกพร่องทางพันธุกรรม (Genetic Predisposition) หรือความผิดปกติทางกายวิภาคของเดานปากและกล้ามเนื้อเทนเซอร์เวลิพาลาตินี (Tensor Veli Palatini) ทำให้ Eustachian Tube ซึ่งมีหน้าที่ป้องกันสิ่งแปรปรวนหลุดเข้าไปในบริเวณหูชั้นกลางทำงานผิดปกติจนทำให้เกิดการติดเชื้อ

2.2 การวินิจฉัยโรคหูชั้นกลางอักเสบ (Diagnosis of Otitis Media)

การวินิจฉัยโรคหูชั้นกลางอักเสบจะพิจารณาจากองค์ประกอบ 5 อย่างเข้าด้วยกันจึงจะได้ผลพิชิตว่าเป็นโรคหูชั้นกลางอักเสบประเภทใด องค์ประกอบทั้ง 5 อย่างประกอบไปด้วย

1. ลักษณะของเหลวในหูชั้นกลาง (Fluid) จะเป็นการวิเคราะห์ลักษณะของเหลวในหูชั้นกลางว่า มีลักษณะอย่างไร ซึ่งจะมีลักษณะที่เป็นไปได้คือ ของเหลวเต็มบริเวณหูชั้นกลาง (Full) ของเหลวมีฟองอากาศ(Bubble) ของเหลวมีการแยกระดับชั้น (Level) และ ไม่มีของเหลวในหูชั้นกลาง (None)

2. ลักษณะการหดหรือปูดของหูชั้นกลาง (Retraction) จะเป็นการวิเคราะห์ว่าเยื่อหูชั้นกลางมีการปูดหรือหดตัวในลักษณะใดซึ่งลักษณะการหดตัวของเยื่อหูชั้นกลางที่เป็นไปได้คือหดตัวเล็กน้อย (Mild) หดตัวปานกลาง (Moderate) หดตัวรุนแรง (Severe) ยึดแน่น (Adhesive) การปูดออกด้านนอกของหูชั้นกลาง(Bulging) และไม่มีการหดตัวของเยื่อหูชั้นกลาง (None)

3. สีของเหลวในหูชั้นกลาง (Color) จะเป็นการวิเคราะห์ว่าเยื่อหูชั้นกลางมีสีเป็นแบบใด ซึ่งสีที่เป็นไปได้คือ สีขาว (White) สี琥珀 (Amber) สีน้ำเงิน (Blue) สีขาวมุก (Pearly White) สีนม (Milky) สีเหลืองสว่าง (Light Yellow) สีแดง (Red) และสีปกติ (Normal)

4. การทะลุของเยื่อหูชั้นกลาง (Perforation) จะเป็นการวิเคราะห์ว่าเยื่อหูชั้นกลางมีการทะลุหรือไม่มีการทะลุ จึงสามารถแบ่งคุณลักษณะได้เป็น ไม่ทะลุ (No) และทะลุ (Yes)

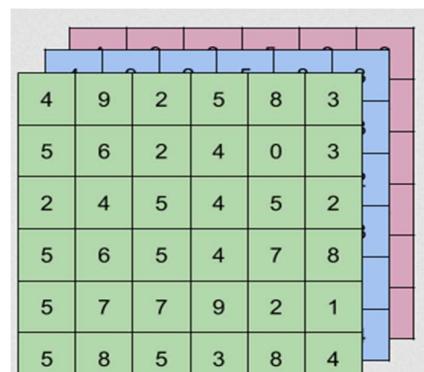
5. ความโปร่งใสของเยื่อหุ้นกลาง (Transparency) จะเป็นการวิเคราะห์ว่าหุ้นกลางมีความโปร่งใสในระดับใด ซึ่งระดับความโปร่งใสที่เป็นไปได้คือ โปร่งใส (Clear) ขุ่นมัว (Dull) ทึบแสง (Opaque) และเป็นพังผืด (Tympanosclerosis)

2.3 โครงข่ายประสาทเทียมแบบ convolutional neural network (Convolutional Neural Network)

เป็น Neural Networks ชนิดหนึ่งที่ใช้ในการจำแนกภาพและมีวิธีการมองรูปภาพที่อ้างอิงวิธีการมาจากการมองเห็นของมนุษย์โดยเป็นการมองเป็นส่วนประกอบอยู่ เช่น เส้น มุน ขอบ และนำมารวมกันเป็นภาพใหญ่ ซึ่งในแต่ละส่วนของโครงข่ายประสาทเทียมแบบ convolutional neural network จะมีหน้าที่แตกต่างกัน โดยเริ่มจากการรับข้อมูลเข้า (Input) การสกัดคุณลักษณะ (Feature Extraction) และนำคุณลักษณะต่างๆ ที่ได้มาประกอบกันเพื่อนำไปสู่ผลลัพธ์

2.3.1 Input Layer

Input Layer คือ Layer แรกที่รับข้อมูลภาพเข้ามาในโครงข่ายประสาทเทียมแบบ convolutional neural network ทำให้ทำการรับข้อมูลต้องรับข้อมูลของสีทั้ง 3 Channel ได้แก่ RED GREEN BLUE ทำให้มิติของ Input Layer มีขนาด $X \times Y \times 3$ โดยที่ X คือจำนวน Pixel ในแนวอน ส่วน Y คือจำนวน Pixel แนวตั้ง



รูปที่ 2.1 Input Layer

2.3.2 Kernel หรือ Filter

Kernel หรือ Filter มีหน้าที่ใช้สำหรับสกัด Feature บน Input เช่น เส้นตรง เส้นโค้ง ขอบของวัตถุหรือ Feature อื่นที่เราสนใจ ซึ่ง Kernel หนึ่งแบบจะสกัด Feature เพียงแบบเดียวเท่านั้น และขนาดของ Kernel ที่นิยมใช้กันได้แก่ขนาด 1×1 3×3 และ 5×5

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Input

1	0	1
0	1	0
1	0	1

Filter / Kernel

รูปที่ 2.2 Input Layer และ Kernel

1	1x1	1x0	0x1	0
0	1x0	1x1	1x0	0
0	0x1	1x0	1x1	1
0	0	1	1	0
0	1	1	0	0

Input x Filter

4	3	0
0	0	0
0	0	0

Feature Map

1x1	1x0	1x1	0	0
0x0	1x1	1x0	1	0
0x1	0x0	1x1	1	1
0	0	1	1	0
0	1	1	0	0

Input x Filter

4	0	0
0	0	0
0	0	0

Feature Map

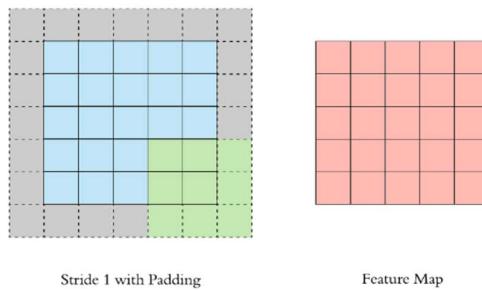
1	1	1	0	0
0	1	1	1	0
0	0	1x1	1x0	1x1
0	0	1x0	1x1	0x0
0	1	1x1	0x0	0x1

Output

รูปที่ 2.3 Output หลังจากสกัด Feature ด้วย Kernel

2.3.3 Padding

ในการนี้ที่การเลื่อนของ Kernel เพื่อสกัด Feature เกิดเหตุการณ์ที่ Kernel ลับบริเวณของ Input ออกไปบางส่วนจนทำให้ไม่สามารถคำนวณ Output จากการสกัดได้ วิธีแก้ไขสามารถทำได้โดยการเพิ่มขอบเขตของ Input เพื่อให้ครอบคลุมการเลื่อนของ Kernel ทั้งหมด ซึ่งบริเวณที่เพิ่มขึ้นมาจะนักจะนิยมกำหนดไว้ให้มีค่าเป็น 0 เพื่อป้องกัน Kernel ว่าบริเวณที่เพิ่มขึ้นมาใหม่ไม่ได้มีผลต่อการสกัด Feature



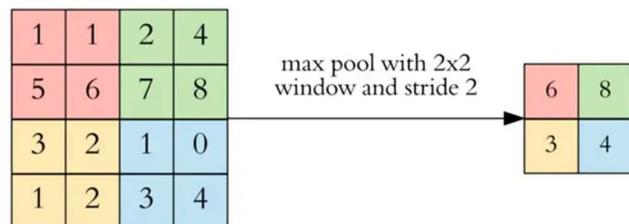
รูปที่ 2.4 Padding

2.3.4 Stride

ในกระบวนการของการใช้ Kernel เพื่อสกัด Feature ที่เราสนใจจะมีการนำ Kernel มาวางทับไว้ที่ตำแหน่งต่าง ๆ ทั่วทั้ง Input ซึ่งการสกัด Feature จะขยับ Kernel ไปเรื่อย ๆ จนกว่าจะทั่วบริเวณ ในการขยับ Kernel นั้น เราสามารถเลือกที่จะขยับทีละเท่าใดก็ได้ตามที่ Kernel ไม่ลับออกนอกกรอบของ Input ในการขยับเราจะสามารถเลือกได้ว่าจะเลื่อนทีละยังไงได้โดยปกติจะกำหนดให้ขยับทีละ 1 ช่อง

2.3.5 Pooling layer

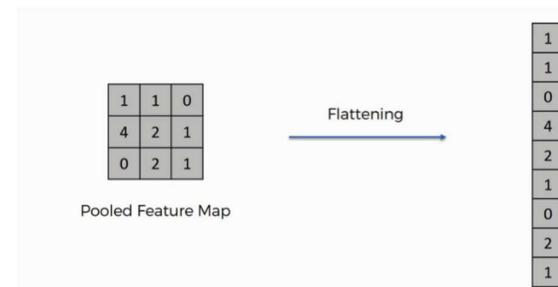
Pooling Layer เป็นวิธีการที่ใช้ในการลดขนาดของ Input เพื่อให้สามารถนำไปคำนวณใน Layer ถัดไปได้รวดเร็วขึ้น การลดขนาดของ Input ควรที่จะคงไว้ซึ่งความหมายของ Input เดิมอยู่เพียงแต่อ้าจะมีขนาดความละเอียดที่ลดลง วิธีการทำ Pooling Layer จะนิยมใช้สองแบบ ได้แก่ Max Pooling และ Average Pooling



รูปที่ 2.5 Max Pooling

2.3.6 Flatten Layer

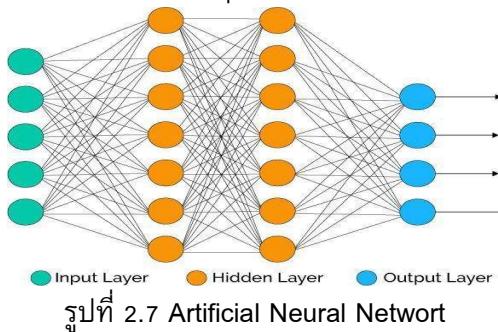
Flatten Layer คือ Layer ที่ใช้สำหรับการเตรียมความพร้อมให้ Convolutional Layer เปลี่ยนรูปแบบมิติของข้อมูลเพื่อให้สามารถนำข้อมูลไปเข้าสู่กระบวนการของ Neural Network ได้ซึ่งวิธีการนี้จะเรียกว่าการ扁平化 (Flattening)



รูปที่ 2.6 Flatten Layer

2.3.7 Fully Connected Layer

Fully Connected Layer คือ Layer ที่รับข้อมูลต่อจาก Flatten Layer โดยหลังจากรับข้อมูลมาแล้วจะนำไปเข้าสู่กระบวนการของการทำ Neural Network ได้แก่ Forward Propagation และ Backward Propagation เพื่อปรับปรุงให้มี Weight ที่เหมาะสมต่อไป



2.3.8 Activation Function

Activation Function เป็นฟังก์ชันทางคณิตศาสตร์ที่ใช้ในการจัดการกับ Input ที่รับมาจาก Layer ก่อนหน้า ว่าควรจะปล่อย Output ออกไปให้อยู่ในเกณฑ์ใด ซึ่งในแต่ละฟังก์ชัน จะมีความแตกต่างกันตามแต่ลักษณะของ Neural Network ที่เลือกใช้

ReLU Function

ใช้สำหรับเป็น Activation Function ของ Hidden Layer โดยจะไม่ยอมให้ค่าที่ต่ำกว่า 0 หลุดออกไป ส่วนค่าที่มากกว่าหรือเท่ากับ 0 จะสาม

$$f(x) = (0, x) = \begin{cases} 0 & \text{สำหรับ } x \leq 0 \\ x & \text{สำหรับ } x > 0 \end{cases}$$

$$f(x) = \frac{1}{1 + e^{-x}}$$

SoftMax Function

ใช้สำหรับเป็น Activation Function ของ Output Layer เนื่องจากสามารถให้คำตอบของฟังก์ชันเป็นค่าความน่าจะเป็น ทำให้สามารถใช้กับ Model ที่มีหลายคลาส

$$f(x_n) = \frac{e^{x_n}}{\sum_{i=1}^K e^{x_i}}$$

2.3.9 Overfitting และ Underfitting

Overfitting คือโมเดลที่มีประสิทธิภาพดีกับข้อมูลสอน (Training Dataset) แต่เมื่อทดสอบกับข้อมูลทดสอบ (Testing Dataset) หรือโมเดลตอบสนองกับการรบกวน (Noise) ของข้อมูลฝึกสอนเป็นจำนวนมาก ส่วน Underfitting คือโมเดลที่มีประสิทธิภาพต่ำไม่สามารถหารูปแบบของข้อมูลได้ เนื่องจากข้อมูลมีจำนวนน้อยไปหรือมีจำนวนของคุณลักษณะที่มากเกินไป โดยวิธีแก้ปัญหาเหล่านี้สามารถทำได้หลากหลายวิธี เช่น Cross-Validation, Data Augmentation เป็นต้น

2.3.10 Data Augmentation

Data Augmentation คือการปรับขนาด รูปร่าง การหมุน หรืออื่นๆโดยยังคงให้ภาพมีความแบบเดิม สำหรับข้อดีของการทำ Data Augmentation คือทำให้ได้ชุดข้อมูลที่มีขนาดใหญ่

2.3.12 Early Stopping

Early Stopping คือการหยุดการฝึกฝนของโมเดลเมื่อถึงจุดหนึ่งเพื่อไม่ให้โมเดลเกิดการเรียนรู้มากเกินไปจนกลายเป็นปัญหา Over fitting

2.3.13 Data Augmentation

คือการปรับขนาด รูปร่าง การหมุน หรืออื่นๆโดยยังคงให้ภาพมีความหมายแบบเดิม ข้อดีของการทำ Data Augmentation คือทำให้ได้ชุดข้อมูลที่มีขนาดใหญ่ขึ้นและมีความหลากหลาย

2.3.14 Dropout

คือการสุ่มเพื่อไม่ให้โมเดลทำการฝึกฝนในบางโหนดทำให้สามารถลดปัญหา Overfitting ได้

2.3.12 Gradient Descent

Gradient Descent คือการหาจุดที่ค่า Loss ต่ำที่สุดของฟังก์ชันนั้นๆที่ใช้ โดยมีขั้นตอนการค้นหาค่าดังสมการ สำหรับฟังก์ชัน loss แบบ 2 และ 3 ตัวแปร แสดงดังรูป และรูปสมการ Gradient Decent

$$w_{t+1} = w_t - \alpha \nabla f(w_t)$$

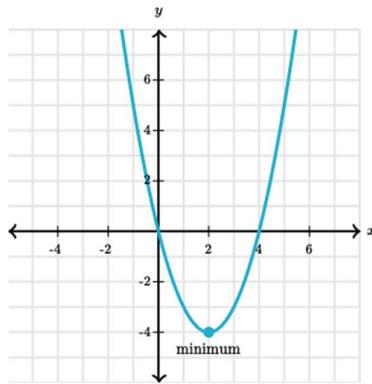
โดยที่

w_{t+1} คือ Weight ใหม่ ณ เวลา $t+1$

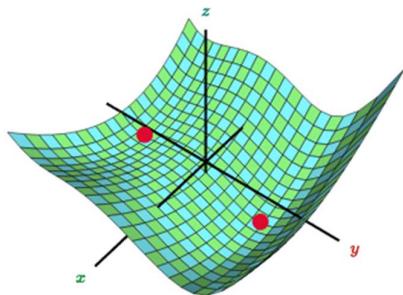
w_t คือ Weight เก่าที่มีความผิดพลาด ณ เวลา t

α คือ Learning Rate

$f(w_t)$ คือ ฟังก์ชัน Loss



รูปที่ 2.8 จุดต่ำสุดของสมการตัวแปรเดียว



รูปที่ 2.9 จุดต่ำสุดของสมการสองตัวแปร

2.3.13 Learning Rate

Learning Rate คือการกำหนดความเร็วของการเรียนรู้ในแต่ละรอบว่าจะเรียนรู้ในระดับที่เร็วขนาดไหน หากปรับค่า Learning Rate ไม่พอดีอาจติดอยู่ในปัญหา Gradient Descent

2.3.14 Optimizer

Optimizer คือกระบวนการที่นำค่าความผิดพลาดในการทำนายของโมเดลไปคำนวนหาจุดที่โมเดลจะพัฒนาค่าตอบให้ดีขึ้น

- **Momentum**

สมการ Gradient Descent

$$w_{t+1} = w_t - \alpha m_t$$

สมการ Momentum

$$m_t = \beta m_{t-1} + (1 - \beta) \left[\frac{\delta L}{\delta w_t} \right]$$

โดยที่

w_{t+1}	คือ weight ใหม่ ณ เวลา $t + 1$
w_t	คือ weight เก่าที่มีความผิดพลาด ณ เวลา t
α	คือ learning rate
m_t	คือ momentum ณ เวลา t
m_{t-1}	คือ momentum ณ เวลา $t - 1$
β	คือ moving average
L	คือ พังก์ชัน loss

▪ RMSP

สมการของ Gradient descent

$$w_{t+1} = w_t - \frac{\alpha_t}{(v_t \epsilon)^{\frac{1}{2}}} * \left[\frac{\delta L}{\delta w_t} \right]$$

สมการผลรวม Gradient

$$v_t = \beta v_{t-1} + (1 - \beta) * \left[\frac{\delta L}{\delta w_t} \right]^2$$

โดยที่

w_{t+1}	คือ weight ใหม่ ณ เวลา $t + 1$
w_t	คือ weight เก่าที่มีความผิดพลาด ณ เวลา t
α	คือ learning rate
α_t	คือ learning rate ณ เวลา t
v_t	คือ ผลรวมกำลังสอง gradient รอบที่แล้ว
ϵ	คือ ค่าคงที่เป็นบวก
β	คือ ค่า moving average
L	คือพังก์ชัน LOSS

▪ Adam

สมการของ Gradient descent

$$w_{t+1} = w_t - \hat{m}_t \left(\frac{\alpha}{\sqrt{\hat{v}_t} + \epsilon} \right)$$

โดยที่

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

และ

$$\hat{\nu}_t = \frac{\nu_t}{1 - \beta_2^t}$$

ซึ่ง m_t คือ Momentum และ β_1^t คือ decay rate ของ Momentum
 ν_t คือ RMSP และ β_2^t คือ decay rate ของ RMSP

2.3.15 Loss

Loss คือการคำนวณค่าความผิดพลาดไม่เดลจากการทำนาย

Categorical Cross Entropy

Categorical Cross Entropy สำหรับคำตอบแบบ binary class

$$\text{Loss} = -\frac{1}{\text{output size}} \sum_{i=1}^{\text{output size}} y_i * \log \hat{y}_i + (1 - y_i) * \log(1 - \hat{y}_i)$$

Categorical Cross Entropy สำหรับคำตอบแบบ multiple class

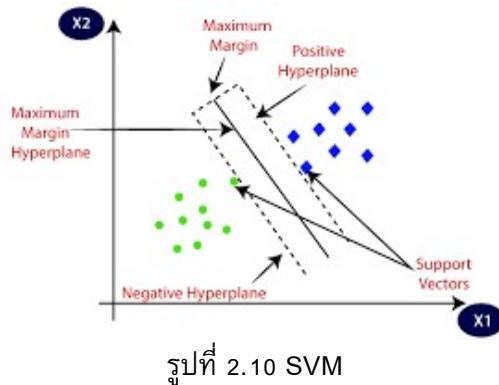
$$\text{Loss} = -\sum_{i=1}^{\text{output size}} y_i * \log \hat{y}_i$$

2.4 SVM

เครื่องมือ Support Vector Machine (SVM) คือหนึ่งในอัลกอริทึมสำหรับการเรียนรู้ของเครื่องจักร (Machine Learning) ที่ใช้ในการจัดกลุ่มข้อมูลและสร้างโมเดลที่สามารถนำไปใช้กับการจำแนกและคาดการณ์ข้อมูลได้ อัลกอริทึม SVM มีความแตกต่างจากอัลกอริทึมอื่น ๆ โดยพิเศษที่สุดคือการใช้เวกเตอร์เพื่อแบ่งกลุ่มข้อมูลอย่างชัดเจน SVM จะพยายามสร้างเส้น (หรือพื้นที่หลายมิติในกรณีข้อมูลหลายมิติ) เพื่อแบ่งกลุ่มข้อมูลอย่างชัดเจนที่สุดหรือมีระยะที่เยอะที่สุดระหว่างข้อมูลแต่ละกลุ่มจุดที่สนใจสำคัญใน SVM คือเวกเตอร์สนับสนุน (Support Vectors) ซึ่งเป็นข้อมูลที่อยู่ใกล้กับเส้นแบ่งกลุ่มและมีบทบาทสำคัญในการสร้างแบบจำลองพารามิเตอร์หรือแบบจำลองที่ใช้ในการฝึกฝนเพื่อให้ได้ผลลัพธ์ที่ดีที่สุด โดยมีพารามิเตอร์ที่สำคัญสำหรับ SVM ดังต่อไปนี้ได้แก่

- Kernel คือพารามิเตอร์ใช้ชนิดของ Kernel Function ในการแปลงและปรับปรุงข้อมูลของเวกเตอร์เพื่อให้ง่ายต่อการจำแนก บางตัวอย่างเช่น Linear Kernel, Polynomial Kernel, Radial Basis Function (RBF) Kernel และอื่นๆ เลือก Kernel Function ที่เหมาะสมสำหรับปัญหา

- Gamma คือพารามิเตอร์ที่ใช้ใน Kernel Function เช่น RBF Kernel โดย Gamma ควบคุมระยะห่างของจุดข้อมูลที่มีผลกับการแยกแยะ ค่า Gamma ที่สูงจะสร้างการแยกแยะที่เป็นเส้นโด้งและเข้มข้นของตัวแบบ ในขณะที่ค่า Gamma ที่ต่ำจะสร้างการแยกแยะที่เป็นเส้นตรงและคลาดเคลื่อนได้มาก
- Degree คือพารามิเตอร์ที่ใช้ใน Polynomial เพื่อกำหนดรูปทรงของเส้นโค้งแบบยกกำลัง



รูปที่ 2.10 SVM

2.5 Decision Tree

Decision Tree และ Random Forest เป็นวิธีการในการจัดกลุ่มและตัดสินใจที่ใช้ในการเรียนรู้ของเครื่องจักร (Machine Learning) ที่ใช้สร้างโมเดลที่สามารถทำงานอย่างแพลตฟอร์มหรือตัดสินใจจากข้อมูลได้ตันไม่การตัดสินใจทำงานโดยการสร้างโครงสร้างที่ประกอบด้วยบล็อกหรือโหนดตัดสินใจซึ่งแต่ละโหนดจะมีการตรวจสอบเงื่อนไขของข้อมูลและการแบ่งกลุ่มข้อมูลออกเป็นกลุ่มย่อยโดยที่การตัดสินใจเกิดขึ้นตามลำดับจากโหนดหลักไปยังโหนดย่อยจนกว่าจะมีกลุ่มข้อมูลเล็กสุดที่สามารถตัดสินใจได้ โดย Decision Tree และ Random Forest มีพารามิเตอร์ที่สำคัญดังต่อไปนี้ได้แก่

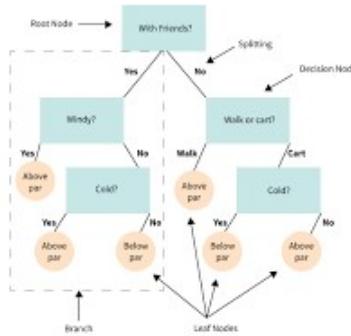
Criterion เป็นพารามิเตอร์ที่กำหนดวิธีการคำนวณค่า Information gain หรือค่า Gini impurity เพื่อใช้ในการตัดสินใจในการแยกสาขาของตันไม้ ซึ่งสามารถเลือก criterion เพื่อให้สอดคล้องกับประเภทของข้อมูลและปัญหาที่คุณกำลังจะแก้ไข

Max Depth พารามิเตอร์ที่ใช้กำหนดความลึกสูงสุดของตันไม้ การตั้งค่าค่า Max Depth สามารถช่วยลดความซับซ้อนของโมเดลและป้องกันการเกิด Overfitting ในข้อมูลฝึกฝน ค่า Max Depth ที่สูงอาจทำให้ตันไม้เต็มไปด้วยกิ่งย่อยและสามารถเกิด Overfitting ได้ ในขณะที่ค่า Max Depth ที่ต่ำอาจทำให้ไม่สามารถจัดสรรข้อมูลได้อย่างมีประสิทธิภาพ

Min Samples Split เป็นพารามิเตอร์ที่ใช้กำหนดจำนวนขั้นต่ำของตัวอย่างที่จำเป็นในการแยกสาขาของตันไม้ ค่า Min Samples Split ที่สูงจะทำให้ตันไม้มีความซับซ้อนน้อยลง ในขณะที่ค่า Min Samples Split ที่ต่ำอาจทำให้เกิด Overfitting ได้

Max Samples Leaf เป็นพารามิเตอร์ที่ใช้กำหนดจำนวนขั้นต่ำของตัวอย่างที่จำเป็นในการแยกสาขาของต้นไม้

Estimator เป็นพารามิเตอร์ที่มีเฉพาะใน Random Forest ใช้สำหรับกำหนดจำนวนของต้นไม้ตัดสินใจทั้งหมด



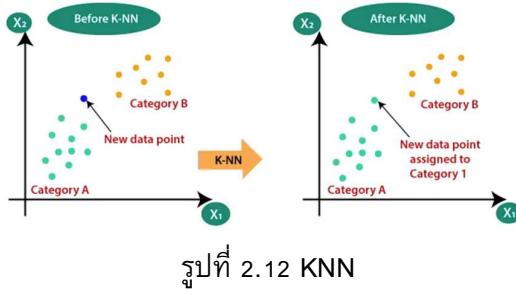
รูปที่ 2.11 ต้นไม้ตัดสินใจ

2.6 KNN

KNN คือเครื่องมือที่ใช้กับปัญหา Classification โดยการวัดระยะโดยใช้จุดศูนย์กลางเป็นข้อมูลที่เราต้องการจัดกลุ่มและมองหาข้อมูลอื่นรอบข้อมูลเป้าหมายจำนวน K ตัว จากนั้นทำการตรวจสอบว่าจากข้อมูล K ตัวมีคำตอบส่วนใหญ่เป็นคำตอบกลุ่มใด และคำตอบที่เราทำนายจะเป็นคำตอบตามคำตอบส่วนใหญ่ ซึ่งในการฝึกฝนโมเดลด้วยวิธี KNN จะมีพารามิเตอร์ที่สำคัญในการฝึกฝนโมเดลดังต่อไปนี้ได้แก่

K เป็นพารามิเตอร์ที่ใช้กำหนดจำนวนเพื่อบันทึกไว้ในการตัดสินใจ ใน KNN โดยค่า K หมายถึงจำนวนเพื่อบันทึกที่ใกล้ที่สุดที่จะถูกใช้ในการจำแนกข้อมูลใหม่ ค่า K ที่สูงสัมพันธ์กับการปรับปรุงความซับซ้อนของโมเดล ค่า K ที่เล็กมากจะทำให้เกิดความเสถียรและความแม่นยำในข้อมูลฝึกฝนสูงขึ้น ในขณะที่ค่า K ที่ใหญ่อาจทำให้เกิดความยืดหยุ่นและความแม่นยำในข้อมูลทดสอบต่ำลง

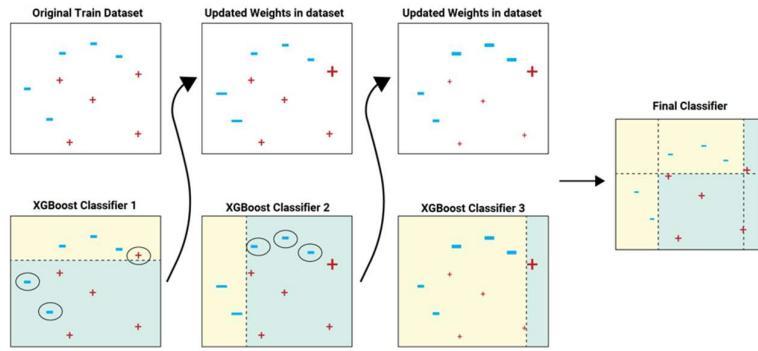
Metric เป็นพารามิเตอร์ที่ใช้กำหนดวิธีการคำนวณระยะห่างระหว่างตัวอย่างข้อมูล ใน KNN, ระยะห่างสามารถคำนวณได้โดยใช้วิธีต่างๆ เช่น Euclidean Distance, Manhattan Distance หรือ Minkowski Distance โดยเลือก Metric ที่เหมาะสมกับประเภทของข้อมูลและปัญหาที่คุณกำลังจะแก้ไข Weighting เป็นพารามิเตอร์ที่ใช้กำหนดน้ำหนักให้กับเพื่อบันทึกไว้ในการตัดสินใจใน



2.7 XGBoost

XGBoost (Extreme Gradient Boosting) มีพารามิเตอร์หลักหลายที่สามารถปรับแต่งเพื่อเพิ่มประสิทธิภาพของโมเดลได้ นี่คือบางพารามิเตอร์สำคัญของ XGBoost

1. N_Estimators คือ พารามิเตอร์นี้กำหนดจำนวนรอบการบูสต์ติ้งหรือจำนวนต้นไม้ตัดสินใจที่จะสร้าง การเพิ่มค่านี้อาจเพิ่มประสิทธิภาพของโมเดล แต่ก็เพิ่มเวลาการคำนวณด้วย มักใช้เทคนิค Cross-Validation ในการปรับแต่ง
2. Learning_Rate คือ พารามิเตอร์นี้ควบคุมอัตราการลดขนาดของขั้นตอนในกระบวนการบูสต์ติ้ง
3. Max_Depth คือ พารามิเตอร์นี้กำหนดระดับความลึกสูงสุดของแต่ละต้นไม้ในองค์ประกอบ ค่าสูงสามารถทำให้เกิดการโอเวอร์ฟิตติ้งได้ ในขณะที่ค่าต่ำสามารถทำให้เกิดการอัตราส่วนไม่เพียงพอ การปรับพารามิเตอร์นี้เป็นสิ่งสำคัญในการค้นหาสมดุลระหว่างความซับซ้อนของโมเดลและความทั่วไป
4. Subsample คือ พารามิเตอร์นี้กำหนดสัดส่วนของตัวอย่างที่ใช้สำหรับการฝึกต้นไม้ตัดสินใจ แต่ละต้น การตั้งค่าต่ำกว่า 1.0 จะนำเข้าความสุ่มในโมเดล เพื่อเพิ่มความทวaal และลดการโอเวอร์ฟิตติ้ง
5. Colsample_Bytree คือพารามิเตอร์นี้กำหนดสัดส่วนของคุณลักษณะ (คอลัมน์) ที่ใช้สำหรับการฝึกต้นไม้ตัดสินใจแต่ละต้น คล้ายกับการสุ่มตัวอย่างและช่วยลดการโอเวอร์ฟิตติ้ง
6. Gamma คือพารามิเตอร์นี้ควบคุมการลดค่าของความสูญเสียขั้นต่ำที่จำเป็นในการแยกโหนด ในการกระบวนการสร้างต้นไม้ ค่าสูงส่งผลให้โมเดลอยู่ในระดับที่รักษาสถานะที่ผ่าเชื้อถือ และลดการ Overfitting มีประโยชน์เฉพาะอย่างยิ่งในการควบคุมความซับซ้อนของต้นไม้



รูป 2.13 XGBoost

2.8 มาตรวัด

2.8.1 มาตรวัดของ Image Classification

สมการ Accuracy

$$\text{Accuracy} = \frac{TP + TN}{\text{Total Population}}$$

โดยที่

Accuracy คือ ประสิทธิภาพ Accuracy ของโมเดล

TP คือ จำนวนคำตอบเป็นบวกที่โมเดลทำนายถูกทั้งหมด

TN คือ จำนวนที่คำตอบเป็นลบที่โมเดลทำนายถูกทั้งหมด

Total Population คือ จำนวนตัวอย่างทั้งหมด

สมการ Precision

$$\text{Precision} = \frac{TP}{TP + FP}$$

โดยที่

Precision คือ ประสิทธิภาพ Precision ของโมเดล

TP คือ จำนวนคำตอบเป็นบวกที่โมเดลทำนายถูกทั้งหมด

FP คือ จำนวนที่คำตอบเป็นบวกที่โมเดลทำนายผิดทั้งหมด

สมการ Sensitivity (Recall)

$$\text{Recall} = \frac{TP}{TP + FN}$$

โดยที่

- Sensitivity** คือ ประสิทธิภาพ *Sensitivity* ของโมเดล
TP คือ จำนวนคำตอบเป็นบวกที่โมเดลทำนายถูกทั้งหมด
FN คือ จำนวนที่คำตอบเป็นลบที่โมเดลทำนายผิดทั้งหมด

สมการ Specificity

$$\text{Specificity} = \frac{TP}{TP + FP}$$

โดยที่

- Specificity** คือ ประสิทธิภาพ *Specificity* ของโมเดล
TP คือ จำนวนคำตอบเป็นบวกที่โมเดลทำนายถูกทั้งหมด
FP คือ จำนวนที่คำตอบเป็นบวกที่โมเดลทำนายผิดทั้งหมด

สมการ F1 Score

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

โดยที่

- F1** คือ ประสิทธิภาพ *F1 Score* ของโมเดล
Precision คือ ผลลัพธ์ของตัววัดประสิทธิภาพ *Precision*
Sensitivity คือ ผลลัพธ์ของตัววัดประสิทธิภาพ *Sensitivity*

2.9 งานวิจัยที่เกี่ยวข้อง

2.9.1 Artificial Intelligence to classify ear disease from otoscopy: A systematic review and meta-analysis

มีจุดประสงค์เพื่อต้องการเปรียบเทียบว่า AI มีความสามารถในการจำแนกภาพจากกล้องส่องหูได้ดีกว่ามนุษย์หรือไม่ โดยนำผลการ วิจัยจำนวน 39 ฉบับ ที่ทำการเปรียบเทียบระหว่าง AI และมนุษย์มาหาข้อสรุป ผลลัพธ์การพัฒนาโมเดลพบว่าในการจำแนกระหว่าง หูปกติ (Normal) และ หูที่ผิดปกติ (Abnormal) ซึ่งเป็นบัญหาแบบ Binary Classification พบร่วม CNN แบบ pre-trained โดยเฉพาะอย่างยิ่ง VGG16 และ VGG19 ให้ผลที่ดีกว่าโมเดลแบบอื่น บัญหาต่อมาคือบัญหาแบบ Multiclassification

โดยเป็นการจำแนกระหว่าง หูปกติ (Normal) หูที่เป็นโรคหูชั้นกลางเรื้อรัง (AOM) และหูที่เป็นโรคหูชั้นกลางอักเสบแบบมีน้ำ (OME) ซึ่ง Wu และคนอื่นๆใช้ Xception และ MobileNet-V2 โดยความความถูกต้องของหูที่เป็นโรคหูชั้นกลางอักเสบแบบมีน้ำ (OME) อยู่ที่ 96.7% ส่วนของ หูปกติ (normal) คือ 98.3% และ ที่เป็นโรคหูชั้นกลางเรื้อรัง (AOM) คือ 98.5% และในกรณีของ Byun จะใช้ RestNet-18 ใน การพัฒนาโมเดล ในการทดลองของนักวิจัยแต่ละคนจำโมเดลไปเปรียบเทียบกับเซิร์ฟเวอร์ด้านโรคทางหู โดยพบว่าในปัญหาการจำแนกระหว่าง หูปกติ (Normal) และ หูที่ผิดปกติ (Abnormal) มีความถูกต้อง 90.7% และการจำแนกระหว่าง หูปกติ (Normal) หูที่เป็นโรคหูชั้นกลางเรื้อรัง (AOM) และหูที่เป็นโรคหูชั้นกลางอักเสบแบบมีน้ำ (OME) มีความถูกต้องอยู่ที่ 97.6% ซึ่งมุ่งยกระดับให้เพียง 73.2%

2.9.2 Otitis media detection using tympanic membrane images with a novel multi-class machine learning algorithm

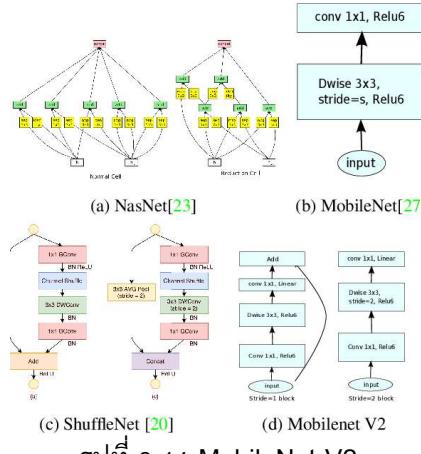
มีจุดประสงค์เพื่อพัฒนาโมเดลที่ใช้สำหรับการตรวจจับโรคหูชั้นกลางอักเสบโดยเฉพาะ และ ต้องการโมเดลที่มีความเหนือกว่า Transfer Learning เช่น AlexNet, VGG-Nets, GoogLeNet, และ ResNets โดยวิธีการที่ใช้จะนำวิธีการของ ResNets มาช่วยในการทำ Residual blocks และมีการฝึกฝน โมเดลแบบคู่ขนานคล้ายกับวิธีการของ InceptionNet สุดท้ายจะนำมาร่วมกันด้วย Global Average Pooling โมเดลแบบผสมประเภทนี้จะเรียกว่า Inception-ResNet ข้อมูลที่ใช้ในการฝึกฝนโมเดลได้แก่ หู ที่มีไขมัน (Wax) เคลือบอยู่ หูที่มีความปกติ (normal) หู ที่เป็นโรคหูชั้นกลางอักเสบแบบเฉียบพลัน (AOM) และหูที่เป็นโรคหูชั้นกลางอักเสบแบบมีน้ำ (OME) จากนั้นทำการทดลองร่วมกับ AlexNet VGG-16 VGG-19 GoogLeNet ResNet-50 ซึ่งจากผลลัพธ์พบว่าโมเดลที่สร้างเองแบบ Inception-ResNet มี ความถูกต้องมากกว่าโมเดลแบบ Transfer Learning ทุกโมเดล

2.9.3 OtoMatch: Content-based eardrum image retrieval using deep learning

มีจุดประสงค์เพื่อสร้าง Deep Learning เพื่อมาช่วยในระบบ Content-Based Image Retrieval (CBIR) ซึ่งระบบที่สร้างมีชื่อเรียกสั้นๆว่า OtoMatch ซึ่งแม้ว่าแพทย์ พยาบาล หรือผู้เชี่ยวชาญจะ สามารถวินิจฉัยได้อยู่แล้ว แต่การที่มีระบบแนะนำจาก CBIR ก็จะยิ่งช่วยให้เพิ่มประสิทธิภาพการทำงาน ได้มากยิ่งขึ้น โดยเฉพาะหากนำระบบนี้ไปใช้ในโรงพยาบาลระดับปฐมภูมิ การวิจัยนี้ใช้ Inception-ResNet V2 ในการสร้างโมเดลโดยข้อมูลที่ใช้ในการฝึกฝนโมเดล ประกอบไปด้วย Middle Ear Effusion 179 ภาพ Normal 179 ภาพ และ Tympanostomy Tube 96 ภาพ ผลลัพธ์จากการวิจัยพบว่า KNN ให้ ความถูกต้องที่ 65.59% ส่วนให้ความถูกต้อง SVM 61.00% และ OtoMatch ให้ความถูกต้องสูงที่สุดคือ 82%

2.9.4 MobileNet V2: Inverted Residuals and Linear Bottlenecks

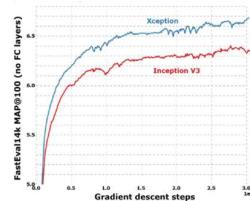
MobileNet V2 เป็นโมเดลที่ถูกพัฒนาขึ้นเพื่อการประมวลผลภาพในอุปกรณ์เคลื่อนที่ (Mobile Devices) โดยเน้นความเร็วและประสิทธิภาพสูงในการระบุและระบุคุณลักษณะของวัตถุ (Feature Extraction) ซึ่งช่วยให้สามารถใช้งานได้ง่ายและมีประสิทธิภาพในอุปกรณ์ที่มีทรัพยากรจำกัด MobileNet V2 ใช้โครงสร้างที่เรียนรู้แบบลึกและเสถียร (Deep Residual Learning) เพื่อสกัดลักษณะที่สำคัญของภาพ โดยใช้เทคนิคของการใช้แทนสัญญาณ (Abstraction) เพื่อเข้าใจและระบุลักษณะที่สำคัญของวัตถุในระดับสูง การทำงานของ MobileNet V2



รูปที่ 2.14 MobileNet V2

2.9.5 Xception: Deep Learning with Depthwise Separable Convolutions

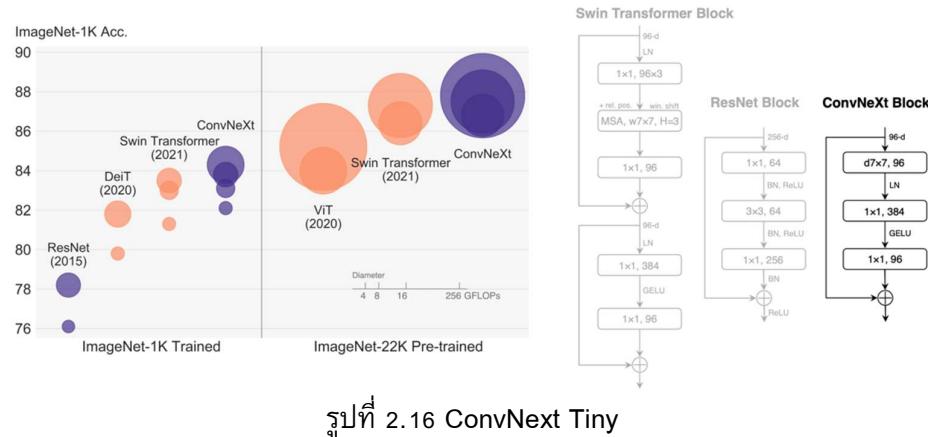
XceptionNet เป็นโมเดลที่ถูกนำเสนอด้วยงานวิจัย Xception: Deep Learning with Depthwise Separable Convolutions ซึ่งเป็นเทคนิคที่ใช้ convolution โอลูชันเชิงลึกแยกแยะ (Depthwise Separable Convolution) เพื่อประมวลผลภาพอย่างมีประสิทธิภาพและประหยัดทรัพยากรในการคำนวณโดย XceptionNet ได้ถูกออกแบบขึ้นโดยเรียนรู้จากแนวคิดของ InceptionNet และปรับปรุงโดยใช้เทคนิคของ convolution โอลูชันเชิงลึกแยกแยะ ทำให้สามารถระบุและสกัดลักษณะที่สำคัญของภาพได้อย่างมีประสิทธิภาพ



รูปที่ 2.15 Xception

2.9.6 A ConvNet for the 2020s

ConvNetX เป็นโมเดลที่พัฒนาขึ้นโดยใช้แนวคิดของ Convolutional Neural Network (ConvNet) และเทคนิค Network-in-Network (NiN) เพื่อปรับปรุงประสิทธิภาพในการประมวลผลของโมเดล โดย ConvNetX มีการพัฒนาแบบ Extra-Wide Network-in-Network ซึ่งเป็นการเพิ่มความกว้างของโครงสร้าง Network-in-Network เพื่อเพิ่มความสามารถในการสกัดลักษณะที่ซับซ้อนของภาพ และปรับปรุงประสิทธิภาพในการจำแนกประเภทของวัตถุในภาพ ConvNetX ใช้ตัวกรองที่เรียนรู้แบบเชิงลึก (Deep Filter) เพื่อสกัดลักษณะที่ซับซ้อนและเฉพาะเจาะจง ซึ่งช่วยให้โมเดลสามารถรู้จำและจำแนกวัตถุในภาพได้อย่างแม่นยำ นอกจากนี้ ConvNetX ยังมีการใช้เทคนิคของ Batch Normalization และ Dropout เพื่อลดการเกิด Overfitting ใน การฝึกฝนโมเดล และเพิ่มประสิทธิภาพในการทำนาย ผลลัพธ์ของการทดลองแสดงให้เห็นว่า ConvNetX มีประสิทธิภาพที่ดีกว่าโมเดล ConvNet และโมเดล Network-in-Network ในการจำแนกประเภทของวัตถุในภาพ

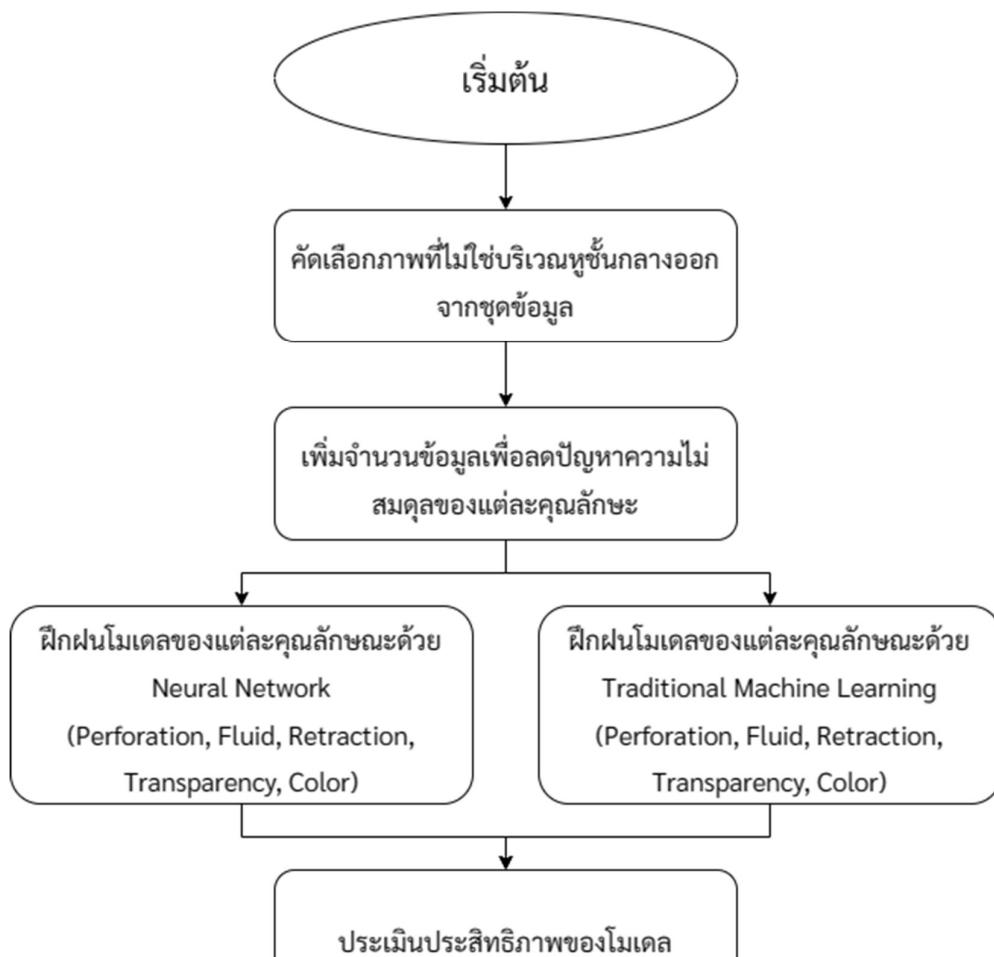


บทที่ 3

วิธีการดำเนินงาน

3.1 ระเบียบวิธีวิจัย

ขั้นตอนการศึกษาทฤษฎี การทดลอง การประเมินผลและสรุปผลมีดังนี้



รูปที่ 3.1 ระเบียบวิธีวิจัย

3.1.1 ศึกษาลักษณะข้อมูลของผู้ป่วยโรคหูชั้นกลางอักเสบและทฤษฎีที่เกี่ยวข้อง

ศึกษาองค์ประกอบที่ใช้ในการวินิจฉัยโรคหูชั้นกลางอักเสบจากแพทย์ผู้เชี่ยวชาญและศึกษาทฤษฎีที่เกี่ยวข้องในการวิเคราะห์ภาพของโรคหูชั้นกลางอักเสบด้วยคอมพิวเตอร์ได้แก่ Neural Networks และ Traditional Machine Learning รวมทั้งทฤษฎีการวัดประสิทธิภาพเพื่อใช้ในการเปรียบเทียบ

องค์ประกอบ		คุณลักษณะขององค์ประกอบ							
Perforation		No		Yes					
Fluid	Full	Bubble		Level	None				
Transparency	Clear		Dull	Opaque	Tympanosclerosis				
Retraction	None	Mild	Moderate	Severe	Adhesive	Bulging			
Color	White		Amber	Blue	Pearly White	Milky	Light Yellow	Red	Normal

ตารางที่ 3.1 ตารางแสดงคุณลักษณะขององค์ประกอบของโรคหูชั้นกลางอักเสบ

Diagnosis				
Normal	OME	Previous OME	AOM	Perforation

ตารางที่ 3.2 ตารางแสดงคุณลักษณะของการวินิจฉัยโรคหูชั้นกลางอักเสบ

ข้อมูลของผู้ป่วยมีทั้งหมดจำนวน 181 ราย แบ่งเป็นผู้ป่วยที่มีหูชั้นกลาง Normal 23 ราย Previous 30 ราย OME 105 ราย AOM 9 ราย และ Perforation 14 ราย ในส่วนของ Perforation มีคุณลักษณะของแก้วหูทะลุ 14 ภาพและไม่มีการทะลุ 167 ภาพ, Fluid มีคุณลักษณะที่มีของเหลวเต็มบริเวณ 62 ภาพ มีของเหลวและฟองอากาศ 14 ภาพ มีของเหลวระดับปานกลาง 31 ภาพ ไม่มีของเหลว 74 ภาพ, Retraction มีคุณลักษณะที่แก้วหูหดตัวเล็กน้อย 63 ภาพ แก้วหูหดตัวปานกลาง 42 ภาพ แก้วหูหดตัวรุนแรง 7 ภาพ มีการโป่งพอง 7 ภาพและไม่มีการหดตัว 62 ภาพ, Transparency มีคุณลักษณะแก้วหูที่มีความโปร่งใส 57 ภาพ ขุนมาว 104 ภาพ ขุนมาวเต็มบริเวณหู 4 ภาพ และเป็นพังผืด 16 ภาพ, Color มีคุณลักษณะที่สีของของเหลวในหูเป็นสีขาว 56 ภาพ น้ำตาลทอง 25 ภาพ น้ำเงิน 5 ภาพ ขาวมูก 17 ภาพ น้ำนม 4 ภาพ เหลืองสว่าง 21 ภาพ แดง 7 ภาพ และสีปกติ 46 ภาพ

จากการปรึกษาแพทย์ผู้เชี่ยวชาญพบว่าการในการวินิจฉัยโรคหูชั้นกลางอักเสบ เราสามารถนำคุณลักษณะของ Fluid มากวิเคราะห์รวมกันเป็นคุณลักษณะที่ไม่มีน้ำในหูชั้นกลางกับมีน้ำในหูชั้นกลางจึงทำให้คุณลักษณะของ Fluid เหลือเพียง None และ Full สำหรับการวิเคราะห์ Transparency คุณลักษณะ

Dull กับ Opaque มีความใกล้เคียงกันในการวินิจฉัยความสามารถรวมกันได้เป็น Dull ทำให้คุณลักษณะของ Transparency เหลือเพียง Clear Dull และ Tympanosclerosis สำหรับการวิเคราะห์ Retraction สามารถรวมคุณลักษณะ None และ Mild ให้เหลือเพียง None เนื่องจาก None และ Mild มีความใกล้เคียงกันในการวินิจฉัย และสามารถรวม Severe และ Moderate ให้เหลือเพียง Severe ทำให้การวิเคราะห์ Retraction จะมีคุณลักษณะ None Severe และ Bulging สำหรับการวิเคราะห์ Color สี Amber และ Light Yellow มีความใกล้เคียงกันการวินิจฉัยความสามารถรวมคุณลักษณะต่างกันให้เหลือเพียง Amber ทำให้การวิเคราะห์ Color จะมีคุณลักษณะ White, Amber, Blue, Milky ,Pearly White, Red และ Normal

องค์ประกอบ		คุณลักษณะขององค์ประกอบ					
Perforation		No			Yes		
Fluid		None			Full		
Transparency		Clear		Dull		Tympanosclerosis	
Retraction		None		Severe		Bulging	
Color	White	Amber	Blue	Pearly White	Red	Normal	Milky

ตารางที่ 3.3 ตารางแสดงคุณลักษณะขององค์ประกอบของโรคหูชั้นกลางอักเสบหลังแก้ไข

คุณลักษณะ

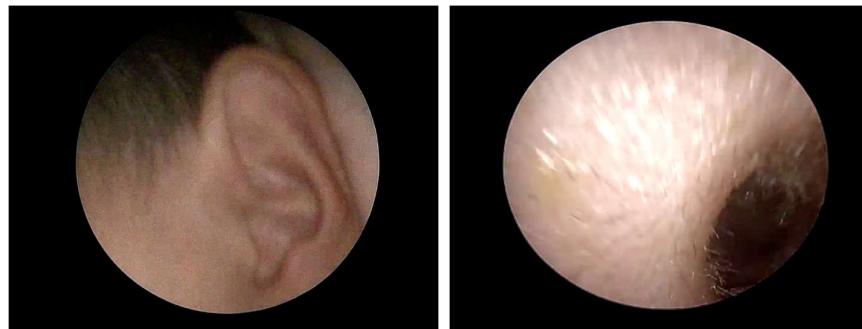
สำหรับการวิเคราะห์คุณลักษณะของโรคหูชั้นกลางในส่วนของคุณลักษณะ Normal และ Previous OME มีความหมายเดียวกันคือการไม่เป็นโรคหูชั้นกลางอักเสบ จึงสามารถรวมคุณลักษณะ Normal และ Previous OME ให้เหลือเพียง Normal ทำให้การวิเคราะห์คุณลักษณะของโรคหูชั้นกลางจะวิเคราะห์คุณลักษณะ Normal, OME, AOM และ Perforation

Diagnosis			
Normal	OME	AOM	Perforation

ตารางที่ 3.4 ตารางแสดงคุณลักษณะของการวินิจฉัยโรคหูชั้นกลางอักเสบหลังแก้ไขคุณลักษณะ

3.1.2 คัดเลือกภาพที่ไม่ใช่บริเวณหูชั้นกลางออกจากชุดข้อมูล

เนื่องจากภาพที่ได้เกิดจากการบันการตั้งแต่สอดกล้องเข้าหูจนถึงการเอากล้องออกจากรูจีงทำให้มีบางภาพที่ไม่ใช่บริเวณหูชั้นกลาง จึงต้องทำการนำภาพเหล่านั้นออกจากชุดข้อมูล โดยใน 1 คลิปวิดีโอนำภาพที่ดีที่สุดออกมาเพื่อเป็น Dataset จำนวน 1 รูป ซึ่งใน Dataset ของผู้ป่วยจำนวน 181 คน จะมีข้อมูลแบบคลิปวิดีโอทั้งหมดจำนวน 99 คลิป และมีข้อมูลแบบภาพนิ่งจำนวน 82 ภาพ



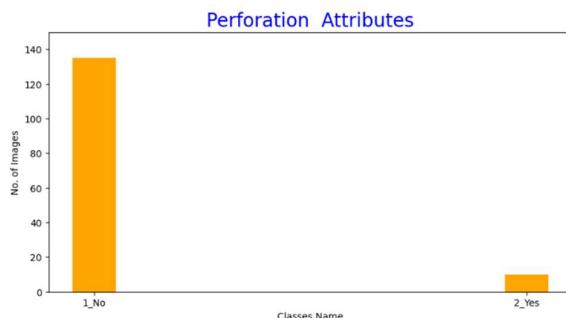
รูปที่ 3.2 ภาพที่ไม่ใช่บริเวณหูชั้นกลาง

3.1.3 เพิ่มจำนวนข้อมูลเพื่อลดปัญหาความไม่สมดุลของคุณลักษณะ

ทดลองเพิ่มจำนวนข้อมูลให้มากขึ้นเนื่องจากข้อมูลภาพนิ่งต้นฉบับมีจำนวนที่น้อยเกินไป ซึ่งจะทำการเพิ่มจำนวนด้วย 2 วิธีได้แก่การหมุนภาพ การปรับเพิ่มและลดแสงของภาพ เพื่อให้ท้ายที่สุดแล้วจำนวนของแต่ละคุณลักษณะมีปริมาณเท่ากันและมีจำนวนมากพอในการทำโมเดล

สำหรับการหมุนภาพจะหมุนภาพทีละ 10 องศา เพื่อเพิ่มปริมาณข้อมูลฝึกฝน โดย Perforation จะหมุนจนได้ภาพเพิ่มขึ้นจำนวน 300 ภาพ Fluid จะหมุนจนได้ภาพเพิ่มขึ้นจำนวน 420 ภาพ Retraction จะหมุนจนได้ภาพเพิ่มขึ้นจำนวน 410 ภาพ Transparency จะหมุนจนได้ภาพเพิ่มขึ้นจำนวน 480 ภาพ Color จะหมุนจนได้ภาพเพิ่มขึ้นจำนวน 536 ภาพ

สำหรับการปรับแสง จะปรับแสงให้ความสว่างลดลงโดยจะลดลงเป็น 60%, 70%, 80% และ 90% จากภาพต้นฉบับ และปรับเพิ่มแสงสว่างเป็น 110% 120% 130% 140% โดย Perforation จะเพิ่มขึ้นเป็นจำนวน 450 ภาพ



รูปที่ 3.3 ตัวอย่างจำนวนข้อมูลในแต่ละ Class ของ Perforation Attributes

3.1.4 สร้างโมเดลด้วย Neural Networks และ Traditional Machine Learning

ทดลองสร้างโมเดลสำหรับการวินิจฉัยโรคหูชั้นกลางอักเสบโดยสำหรับ Neural Networks จะใช้สถาปัตยกรรม Xception, MobileNet V2 และ ConvNext Tiny จากนั้นทำการประเมินประสิทธิภาพของโมเดล สำหรับ Traditional Machine Learning จะใช้สถาปัตยกรรม SVM, KNN, Decision Tree และ XGBoost จากนั้นทำการประเมินประสิทธิภาพของโมเดล

3.2 การประมวลผลข้อมูลก่อนการฝึกฝน

3.2.1 การเพิ่มจำนวนข้อมูลด้วย Augmentation

เนื่องจากในหลายแต่ละองค์ประกอบไม่มีความสมดุลกันของจำนวนคุณลักษณะซึ่งจะทำให้โมเดลมีความแม่นยำสูงเฉพาะกับคุณลักษณะที่มีตัวอย่างจำนวนมากกว่า ซึ่งจะแก้ไขปัญหานี้ด้วยการทำ Augmentation เพื่อให้คุณลักษณะมีความสมดุลกันซึ่งจะใช้ทั้งวิธีหมุนภาพและปรับความสว่างเพื่อเพิ่มจำนวนตัวอย่างให้สมดุลกัน นอกจากนี้การทำ Augmentation ดังกล่าว ยังทำให้โมเดลมีความทนทานต่อภาพที่หมุนในมุมที่หลากหลายและทนต่อภาพที่มีความสว่างน้อยหรือมากเกินไป

```
def rotate(degree, directory, filename):
    os.chdir(directory) # change to directory where image is located
    picture= Image.open(filename)
    picture.rotate(degree, expand=True).save('rotated_'+str(int(degree+0.875))+filename)
    print(directory, " ",filename, " ",degree , " ")
```

รูปที่ 3.4 ตัวอย่างคำสั่งที่ใช้ในการหมุนภาพ

```
def brightness(directory, target, classes):
    classes = classes
    # target is number of all dataset of each attributes after it was augmented
    # num is number of existing dataset of each attributes

    for i in classes: # Directory Loop
        num = lenfiles(directory, i)
        to_augmentate = target-num

        if to_augmentate<0:
            to_augmentate=0
            pass

        dir_list = os.listdir(directory+i)

        print("num=",num,"to_brightness=", to_augmentate,"dir_list", dir_list)

        run=True
        count=0
        num0image = len(dir_list)
        index=0
        based_degree = 0.45
        degree = based_degree

while(count<to_augmentate):
    print(i, "remain =", to_augmentate-count)
    #rotate(degree, directory+i, dir_list[index])
    os.chdir(directory)
    #read the image
    im = Image.open(directory+i+"/"+dir_list[index])
    #image brightness enhancer
    enhancer = ImageEnhance.Brightness(im)

    factor = degree #darkens the image
    im_output = enhancer.enhance(factor)
    im_output.save(i+"/"+'brightness_'+str(factor)+dir_list[index])

    print("factor=", factor, "brightness=", directory+i+"/"+dir_list[index])
    index+=1
    count+=1
    if index==num0image:
        index=0
        degree +=0.2
        if degree >=1.2:
            based_degree+=0.005
            degree = based_degree
```

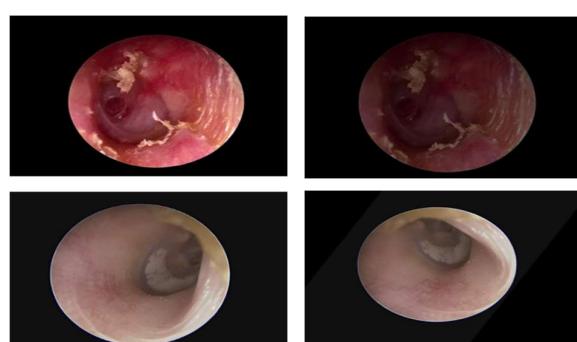
(a)

(b)

รูปที่ 3.5 ตัวอย่างคำสั่งที่ใช้ในการเพิ่ม-ลดแสง

(a) วน Loop ส่งภาพเข้าไปปรับแสงโดยเริ่มต้นที่ความสว่าง 45% จากภาพทั้งฉบับ

(b) วน Loop ปรับแสงขึ้นทีละ 20% โดยไม่ให้ภาพสว่างเกิน 180%



รูปที่ 3.6 ตัวอย่างการ Augmentation

3.2.2 ตั้งค่า Neural Network Model

- **Xception**

ทำการสร้างฟังก์ชันกำหนดค่า Parameter ของสถาปัตยกรรม Xception จากนั้นกำหนดรายละเอียดของ Layer ที่ใช้ในการฝึกฝนดังรูปที่ 3.7 ซึ่ง Parameter จะแสดงดังตารางที่ 3.5

Parameter	Value
Base Model	Xception
Learning Rate	10^{-4} , 10^{-5} และ 10^{-6}
Optimizer	Adam และ Stochastic Gradient Descent
Batch Size	16
Epoch	ไม่เกิน 100 ครั้ง
ฟังก์ชัน Loss	Categorical Cross Entropy (Multiple) Binary Categorical Cross Entropy (Binary)
Activation Function (Hidden Layer)	Relu
Activation Function (Last Layer)	Softmax (Multiple) Sigmoid (Binary)
Early Stopping	10
Dropout	0.2

ตารางที่ 3.5 ตาราง Parameter ของสถาปัตยกรรม Xception

```
def xception(IMG_SIZE, num_classes, dataset, lr, epoch):
    train = dataset[0]
    val = dataset[1]
    print("num of classes = ", num_classes)
    normalization_layer = layers.Rescaling(1./255)
    preprocess_input = tf.keras.applications.mobilenet_v2.preprocess_input
    IMG_SHAPE = IMG_SIZE + (3,)
    base_model = tf.keras.applications.xception.Xception(
        include_top=False,
        weights='imagenet',
        input_shape=IMG_SHAPE)
    base_model.trainable = True
    global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
    prediction_layer = tf.keras.layers.Dense(num_classes, activation='softmax')

    data_augmentation = tf.keras.Sequential([
        tf.keras.layers.RandomFlip('horizontal'),
        tf.keras.layers.RandomRotation(0.2),
    ])
    w = IMG_SIZE[0]
    h = IMG_SIZE[1]
```

รูปที่ 3.7 ฟังก์ชันสำหรับตั้งค่าการฝึกฝนโมเดล

```

inputs = tf.keras.Input(shape=(w, h, 3))
x = data_augmentation(inputs)
x = preprocess_input(x)
x = base_model(x, training=True)
x = global_average_layer(x)
x = tf.keras.layers.Dropout(0.2)(x)
outputs = prediction_layer(x)
model = tf.keras.Model(inputs, outputs)

```

รูปที่ 3.8 ตั้งค่า Layer

เมื่อทำการกำหนด Layer เสร็จแล้วก็ทำการกำหนด Optimizer, Loss และ Metric ดังรูปที่ 3.9

```

base_learning_rate = lr
model.compile(optimizer = tf.keras.optimizers.Adam(learning_rate=base_learning_rate),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
              metrics=['accuracy'],
              )
len(model.trainable_variables)

```

รูปที่ 3.9 ตั้งค่าการ Compile โมเดล

▪ MobileNet V2

ทำการสร้างฟังก์ชันกำหนดค่า Parameter ของสถาปัตยกรรม MobileNet V2 จากนั้นกำหนดรายละเอียดของ Layer ที่ใช้ในการฝึกฝนดังรูปที่ 3.10 ซึ่ง Parameter จะแสดงดังตาราง 3.6

Parameter	Value
Base Model	MobileNet V2
Learning Rate	10^{-4} , 10^{-5} และ 10^{-6}
Optimizer	Adam และ Stochastic Gradient Descent
Batch Size	16
Epoch	ไม่เกิน 100 ครั้ง
ฟังก์ชัน Loss	Categorical Cross Entropy (Multiple) Binary Categorical Cross Entropy (Binary)
Activation Function (Hidden Layer)	Relu
Activation Function (Last Layer)	Softmax (Multiple) Sigmoid (Binary)
Early Stopping	10
Dropout	0.2

ตารางที่ 3.6 ตาราง Parameter ของสถาปัตยกรรม MobileNet V2

```

def MobileNetV2(IMG_SIZE, num_classes, dataset, lr, epoch):
    train = dataset[0]
    val = dataset[1]
    print("num of classes = ", num_classes)
    normalization_layer = layers.Rescaling(1./255)
    preprocess_input = tf.keras.applications.mobilenet_v2.preprocess_input
    IMG_SHAPE = IMG_SIZE + (3,)
    base_model = tf.keras.applications.MobileNetV2(
        include_top=False,
        weights='imagenet',
        input_shape=IMG_SHAPE)

    base_model.trainable = True
    global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
    prediction_layer = tf.keras.layers.Dense(1, activation='sigmoid')

    data_augmentation = tf.keras.Sequential([
        tf.keras.layers.RandomFlip('horizontal'),
        tf.keras.layers.RandomRotation(0.2),
    ])

    w = IMG_SIZE[0]
    h = IMG_SIZE[1]

```

รูปที่ 3.10 พังก์ชันตั้งค่าการฝึกฝนโมเดล MobileNet V2

```

inputs = tf.keras.Input(shape=(w, h, 3))
x = data_augmentation(inputs)
x = preprocess_input(x)
x = base_model(x, training=True)
x = global_average_layer(x)
x = tf.keras.layers.Dropout(0.2)(x)
outputs = prediction_layer(x)
model = tf.keras.Model(inputs, outputs)

```

รูปที่ 3.11 ตั้งค่า Layer MobileNet V2

เมื่อทำการกำหนด Layer เสร็จแล้วก็ทำการกำหนด Optimizer, Loss และ Metric ดังรูปที่ 3.12

```

base_learning_rate = lr
model.compile(optimizer = tf.keras.optimizers.Adam(learning_rate=base_learning_rate),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
              metrics=[accuracy],
              )
len(model.trainable_variables)

```

รูปที่ 3.12 ตั้งค่าการ compile โมเดล

■ ConvNext Tiny

ทำการสร้างพังก์ชันกำหนดค่า Parameter ของสถาปัตยกรรม ConvNext Tiny จากนั้นกำหนดรายละเอียดของ Layer ที่ใช้ในการฝึกฝนดังรูปที่ 3.13 ซึ่ง Parameter จะแสดงดังตารางที่ 3.7

Parameter	Value
Base Model	ConvNext Tiny
Learning Rate	10^{-4} , 10^{-5} และ 10^{-6}
Optimizer	Adam และ Stochastic Gradient Descent
Batch Size	16
Epoch	ไม่เกิน 100 ครั้ง

ฟังก์ชัน Loss	Categorical Cross Entropy (Multiple) Binary Categorical Cross Entropy (Binary)
Activation Function (Hidden Layer)	Relu
Activation Function (Last Layer)	Softmax (Multiple) Sigmoid (Binary)
Early Stopping	10
Dropout	0.2

ตารางที่ 3.7 ตาราง Parameter ของสถาปัตยกรรม ConvNext Tiny

```
def convnext(IMG_SIZE, num_classes, dataset, lr, epoch):
    train = dataset[0]
    val = dataset[1]
    print("num of classes = ", num_classes)
    normalization_layer = layers.Rescaling(1./255)
    preprocess_input = tf.keras.applications.convnext.preprocess_input
    IMG_SHAPE = IMG_SIZE + (3,)
    base_model = tf.keras.applications.convnext.ConvNeXtSmall(
        include_top=False,
        weights='imagenet',
        input_shape=IMG_SHAPE)
    base_model.trainable = True
    global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
    prediction_layer = tf.keras.layers.Dense(num_classes, activation='softmax')

    data_augmentation = tf.keras.Sequential([
        tf.keras.layers.RandomFlip('horizontal'),
        tf.keras.layers.RandomRotation(0.2),
    ])

    w = IMG_SIZE[0]
    h = IMG_SIZE[1]
```

รูปที่ 3.13 ฟังก์ชันตั้งค่าการฝึกฝนโมเดล ConvNext Tiny

```
inputs = tf.keras.Input(shape=(w, h, 3))
x = data_augmentation(inputs)
x = preprocess_input(x)
x = base_model(x, training=True)
x = global_average_layer(x)
x = tf.keras.layers.Dropout(0.2)(x)
outputs = prediction_layer(x)
model = tf.keras.Model(inputs, outputs)
```

รูปที่ 3.14 ตั้งค่า Layer ConvNext Tiny

เมื่อทำการกำหนด Layer เสร็จแล้วก็ทำการกำหนด Optimizer, Loss และ Metric ดังรูปที่ 3.15

```
base_learning_rate = lr
model.compile(optimizer = tf.keras.optimizers.Adam(learning_rate=base_learning_rate),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
              metrics=['accuracy'],
              )
len(model.trainable_variables)
```

รูปที่ 3.15 ตั้งค่าการ compile โมเดล

3.2.3 ตั้งค่า Traditional Machine Learning

- **SVM**

ทำการตั้งค่าโมเดลแบบ SVM โดยจะมีการทดลองกับ Parameter ต่าง ๆ ดังตารางที่ 3.8 สำหรับฝึกฝนและแสดงผลลัพธ์จะแสดงดังรูปที่ 3.16

Parameter	Value
Kernal	Linear, Poly, RBF และ Sigmoid
Gamma	0.25 ถึง 0.8
C	1
Degree (Poly)	1, 2, 3

ตารางที่ 3.8 ตาราง Parameter ของสถาปัตยกรรม SVM

```
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.decomposition import PCA
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
%matplotlib inline

X_train = X_train_aug1_128x128
y_train = y_train_aug1_128x128

X_test = X_test_aug1_128x128
y_test = y_test_aug1_128x128

# Apply PCA for dimensionality reduction
pca = PCA(n_components=85)
X_pca = pca.fit_transform(X_train)
|
X_pca_test = pca.fit_transform(X_test)

classifier = SVC(kernel='linear', probability=True)
#classifier = SVC(kernel='rbf', probability=True)
#classifier = SVC(kernel='poly', degree=2, probability=True)
#classifier = SVC(kernel='sigmoid', probability=True)

classifier.fit(X_pca, y_train)

y_pca_pred = classifier.predict(X_pca_test)
```

รูปที่ 3.16 การตั้งค่าโมเดลแบบ SVM

- **KNN**

ทำการตั้งค่าโมเดลแบบ KNN โดยจะมีการทดลองกับ Parameter ต่าง ๆ ดังตารางที่ 3.8 สำหรับฝึกฝนและแสดงผลลัพธ์จะแสดงดังรูปที่ 3.17

Parameter	Value
Neighbor	2 ถึง 20
Distance Equation	Minkowski

ตารางที่ 3.8 ตาราง Parameter ของสถาปัตยกรรม KNN

```

from sklearn.neighbors import KNeighborsClassifier
from sklearn.decomposition import PCA
from sklearn import metrics
from sklearn.metrics import roc_curve, auc, confusion_matrix
%matplotlib inline

X_train = X_train_aug1_128x128
y_train = y_train_aug1_128x128

X_test = X_test_aug1_128x128
y_test = y_test_aug1_128x128

# Apply PCA for dimensionality reduction
pca = PCA(n_components=85)
X_pca = pca.fit_transform(X_train)

X_pca_test = pca.fit_transform(X_test)

classifier = KNeighborsClassifier(n_neighbors=1)

classifier.fit(X_pca, y_train)

y_pca_pred = classifier.predict(X_pca_test)

```

รูปที่ 3.17 การตั้งค่าโมเดลแบบ KNN

▪ Decision Tree

ทำการตั้งค่าโมเดลแบบ SVM โดยจะมีการทดลองกับ Parameter ต่าง ๆ ดังตารางที่ 3.9 สำหรับผู้อ่านและแสดงผลลัพธ์จะแสดงดังรูปที่ 3.18

Parameter	Value
Criterion	Gini
Max Depth	2 ถึง 20
Min Samples Leaf	1 ถึง 20

ตารางที่ 3.9 ตาราง Parameter ของสถาปัตยกรรม Decision Tree

```

from sklearn import tree
from sklearn.decomposition import PCA
from sklearn import metrics
from sklearn.metrics import roc_curve, auc, confusion_matrix
%matplotlib inline

X_train = X_train_aug1_128x128
y_train = y_train_aug1_128x128

X_test = X_test_aug1_128x128
y_test = y_test_aug1_128x128

# Apply PCA for dimensionality reduction
pca = PCA(n_components=85)
X_pca = pca.fit_transform(X_train)

X_pca_test = pca.fit_transform(X_test)

classifier = tree.DecisionTreeClassifier()

classifier.fit(X_pca, y_train)

y_pca_pred = classifier.predict(X_pca_test)

```

รูปที่ 3.18 การตั้งค่าโมเดลแบบ Decision Tree

■ XGBoost

ทำการตั้งค่าโมเดลแบบ SVM โดยจะมีการทดลองกับ Parameter ต่าง ๆ ดังตารางที่ 3.10 สำหรับฝึกฝนและแสดงผลลัพธ์จะแสดงดังรูปที่ 3.19

Parameter	Value
Booster	Gradient Booster Tree
Learning Rate	10^{-1} ถึง 10^{-4}
N Estimator	10 ถึง 100
Max Depth	3 ถึง 20
Max Leaves	2 ถึง 20
Gamma	0.25 ถึง 0.8

ตารางที่ 3.10 ตาราง Parameter ของสถาปัตยกรรม XGBoost

```
from xgboost import XGBClassifier
from sklearn.decomposition import PCA
from sklearn import metrics
from sklearn.metrics import roc_curve, auc, confusion_matrix
%matplotlib inline

X_train = X_train_aug1_128x128
y_train = y_train_aug1_128x128

X_test = X_test_aug1_128x128
y_test = y_test_aug1_128x128

# Apply PCA for dimensionality reduction
pca = PCA(n_components=85)
X_pca = pca.fit_transform(X_train)

X_pca_test = pca.fit_transform(X_test)

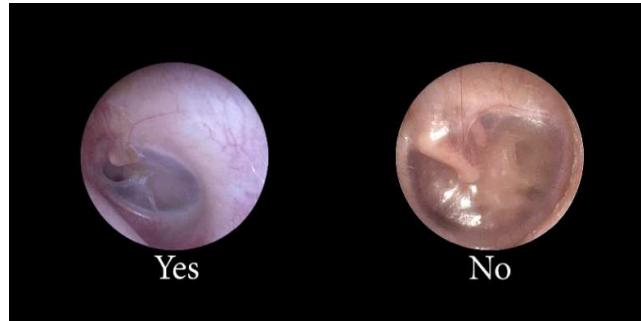
classifier = XGBClassifier()
classifier.fit(X_pca, y_train)

y_pca_pred = classifier.predict(X_pca_test)
```

รูปที่ 3.19 การตั้งค่าโมเดลแบบ XGBoost

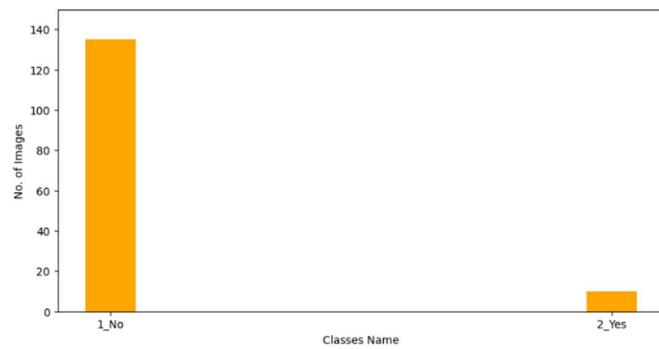
3.2.2.1 Perforation

สำหรับ Perforation จะประกอบไปด้วย 2 คุณลักษณะ ได้แก่ Yes และ No โดยที่ Yes หมายถึงหูที่มีลักษณะที่เป็นการหลุดและ No หมายถึงหูที่ไม่มีการหลุด โดยจะแสดงดังภาพที่ 3.20



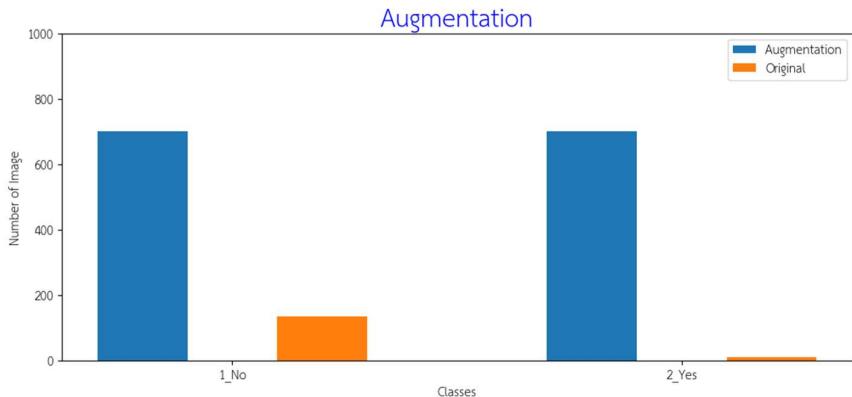
รูปที่ 3.20 ตัวอย่างคุณลักษณะ Perforation แบบ Yes และ No

จากภาพภาพที่ จะเห็นว่า Perforation เป็น Imbalanced Class ซึ่งจำเป็นต้องมีการเพิ่มข้อมูลของคลาสให้สมดุลกันทุกคลาส



รูปที่ 3.21 จำนวนข้อมูลในแต่ละ Class ของ Perforation

เพิ่มตัวอย่างของ Class ให้สมดุลกันด้วยวิธี Augmentation ด้วยการปรับความสว่างและหมุนภาพ



รูปที่ 3.22 จำนวนข้อมูลในแต่ละ Class ของ Perforation หลังทำการ Augmentation

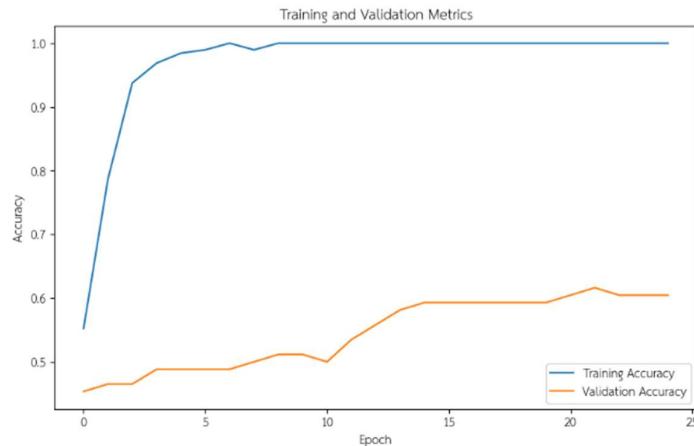
■ Xception

```
# Train the model
history = model.fit(train_ds, validation_data=val_ds, epochs=10)
```

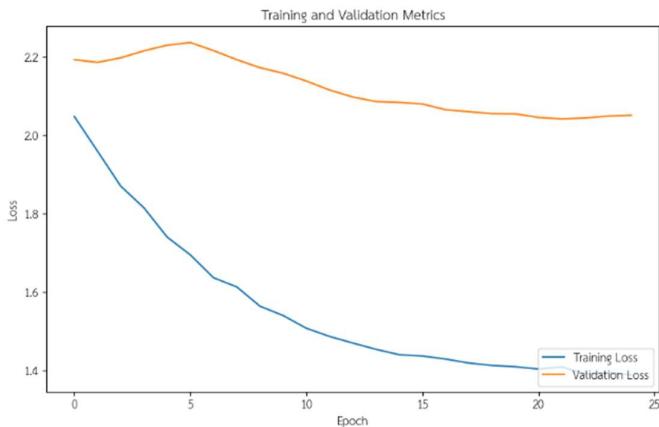
รูปที่ 3.23 ตัวอย่างคำสั่งการฝึกฝนโมเดล Perforation ด้วย Xception

```
Epoch 1/10
38/38 [=====] - 102s 1s/step - loss: 140.1383 - accuracy: 0.1908 - F1Score: 0.0000e+00 - SpecificityAtSensitivity: 0.4465 - SensitivityAtSpecificity: 0.4975 - AUC: 0.5083 - val_loss: 139.8581 - val_accuracy: 0.2650 - val_F1Score: 0.0000e+00 - val_SpecificityAtSensitivity: 0.5987 - val_SensitivityAtSpecificity: 0.5542 - val_AUC: 0.5803 - val_val_loss: 139.5990 - val_val_accuracy: 0.3042 - val_val_F1Score: 0.0189 - val_val_SpecificityAtSensitivity: 0.6162 - val_val_SensitivityAtSpecificity: 0.6117 - val_val_AUC: 0.6330 - val_val_loss: 139.4342 - val_val_accuracy: 0.3742 - val_val_F1Score: 0.0472 - val_val_SpecificityAtSensitivity: 0.7027 - val_val_SensitivityAtSpecificity: 0.6450 - val_val_AUC: 0.6747 - val_val_loss: 139.4083 - val_val_accuracy: 0.4235 - val_val_F1Score: 0.0894 - val_val_SpecificityAtSensitivity: 0.7208 - val_val_SensitivityAtSpecificity: 0.7758 - val_val_AUC: 0.7190 - val_val_loss: 139.2843 - val_val_accuracy: 0.4533 - val_val_F1Score: 0.1069 - val_val_SpecificityAtSensitivity: 0.8342 - val_val_SensitivityAtSpecificity: 0.7758 - val_val_AUC: 0.7510 - val_val_loss: 139.05734 - val_val_accuracy: 0.4858 - val_val_F1Score: 0.1510 - val_val_SpecificityAtSensitivity: 0.8785 - val_val_SensitivityAtSpecificity: 0.8108 - val_val_AUC: 0.7888 - val_val_loss: 138.83316 - val_val_accuracy: 0.5325 - val_val_F1Score: 0.1723 - val_val_SpecificityAtSensitivity: 0.9103 - val_val_SensitivityAtSpecificity: 0.8650 - val_val_AUC: 0.8096 - val_val_loss: 138.40791 - val_val_accuracy: 0.5668 - val_val_F1Score: 0.2042 - val_val_SpecificityAtSensitivity: 0.9267 - val_val_SensitivityAtSpecificity: 0.8817 - val_val_AUC: 0.8344 - val_val_loss: 138.28279 - val_val_accuracy: 0.6000 - val_val_F1Score: 0.2323 - val_val_SpecificityAtSensitivity: 0.9557 - val_val_SensitivityAtSpecificity: 0.9175 - val_val_AUC: 0.8527 - val_val_loss: 138.2
```

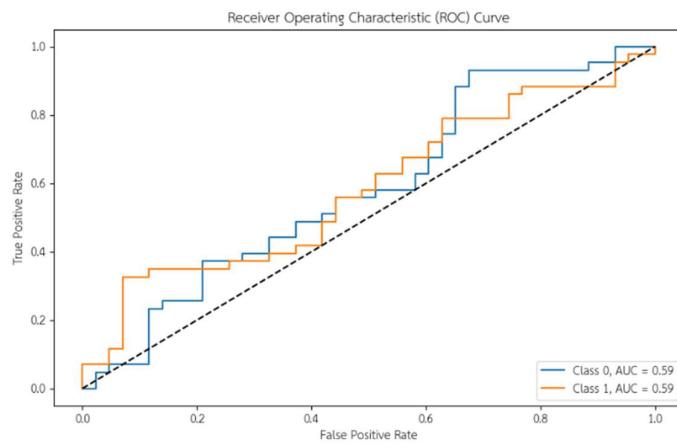
รูปที่ 3.24 ตัวอย่างผลลัพธ์ระหว่างการฝึกฝนโมเดล Perforation ด้วย Xception



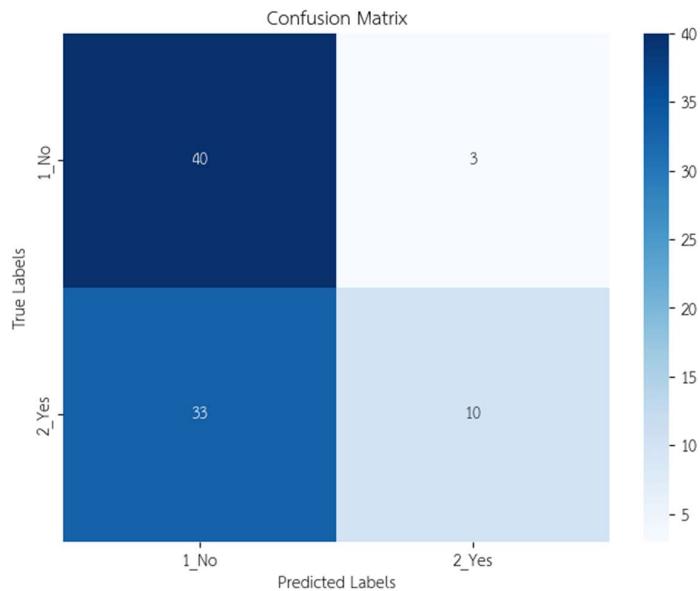
รูปที่ 3.25 ตัวอย่าง Accuracy ของโมเดล Perforation ที่ฝึกฝนด้วย Xception



รูปที่ 3.26 ตัวอย่าง Loss ของโมเดล Perforation ที่ฝึกฝนด้วย Xception



รูปที่ 3.27 ตัวอย่าง AUC ของโมเดล Perforation ที่ฝึกฝนด้วย Xception



รูปที่ 3.28 ตัวอย่าง Confusion ของโมเดล Perforation ที่ฝึกฝนด้วย Xception

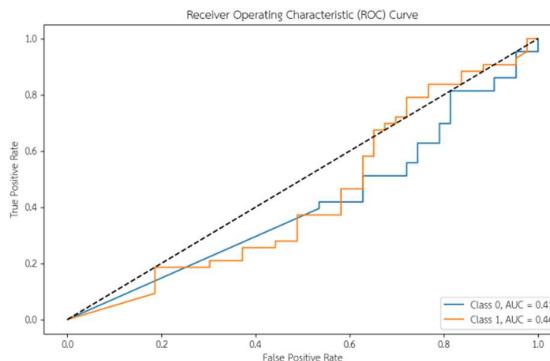
■ MobileNet V2

```
# Train the model
history = model.fit(train_ds, validation_data=val_ds, epochs=10)
```

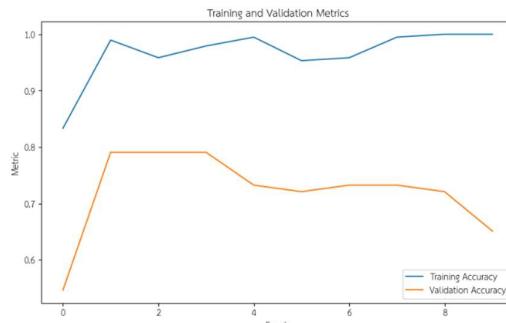
รูปที่ 3.29 ตัวอย่างคำสั่งในการฝึกฝนโมเดล Perforation ด้วย MobileNet V2

```
Epoch 1/10
25/25 [=====] - 60s: 35ms/step - loss: 0.0640 - accuracy: 0.6913 - F1Score: 0.6975 - SpecificityAtSensitivity: 0.9728 - SensitivityAtSpecificity: 0.9656 - AUC: 0.8997 - val_loss: 3.40
Epoch 2/10
25/25 [=====] - 10s: 33ms/step - loss: 0.3116 - accuracy: 0.8916 - F1Score: 0.8916 - SpecificityAtSensitivity: 0.9979 - SensitivityAtSpecificity: 0.9911 - AUC: 0.9817 - val_loss: 6.22
Epoch 3/10
25/25 [=====] - 10s: 33ms/step - loss: 0.2225 - accuracy: 0.9464 - F1Score: 0.9457 - SpecificityAtSensitivity: 0.9979 - SensitivityAtSpecificity: 0.9936 - AUC: 0.9911 - val_loss: 9.75
Epoch 4/10
25/25 [=====] - 8s: 258ms/step - loss: 0.2046 - accuracy: 0.9349 - F1Score: 0.9348 - SpecificityAtSensitivity: 0.9983 - SensitivityAtSpecificity: 0.9923 - AUC: 0.9987 - val_loss: 6.100
Epoch 5/10
25/25 [=====] - 10s: 341ms/step - loss: 0.1414 - accuracy: 0.9452 - F1Score: 0.9480 - SpecificityAtSensitivity: 0.9996 - SensitivityAtSpecificity: 0.9987 - AUC: 0.9955 - val_loss: 8.10
Epoch 6/10
25/25 [=====] - 16s: 597ms/step - loss: 0.1295 - accuracy: 0.9630 - F1Score: 0.9622 - SpecificityAtSensitivity: 0.9987 - SensitivityAtSpecificity: 0.9949 - AUC: 0.9951 - val_loss: 10.4
Epoch 7/10
25/25 [=====] - 10s: 257ms/step - loss: 0.0447 - accuracy: 0.9911 - F1Score: 0.9904 - SpecificityAtSensitivity: 0.9996 - SensitivityAtSpecificity: 0.9987 - AUC: 0.9988 - val_loss: 12.3
Epoch 8/10
25/25 [=====] - 10s: 341ms/step - loss: 0.0190 - accuracy: 0.9898 - F1Score: 0.9911 - SpecificityAtSensitivity: 1.0000 - SensitivityAtSpecificity: 1.0000 - AUC: 0.9999 - val_loss: 15.0
Epoch 9/10
25/25 [=====] - 11s: 370ms/step - loss: 0.0488 - accuracy: 0.9809 - F1Score: 0.9808 - SpecificityAtSensitivity: 0.9996 - SensitivityAtSpecificity: 0.9987 - AUC: 0.9988 - val_loss: 10.0
Epoch 10/10
25/25 [=====] - 10s: 262ms/step - loss: 0.1049 - accuracy: 0.9732 - F1Score: 0.9712 - SpecificityAtSensitivity: 0.9987 - SensitivityAtSpecificity: 0.9962 - AUC: 0.9961 - val_loss: 10.1
```

รูปที่ 3.30 ตัวอย่างผลลัพธ์ระหว่างการฝึกฝนโมเดล Perforation ด้วย MobileNet V2



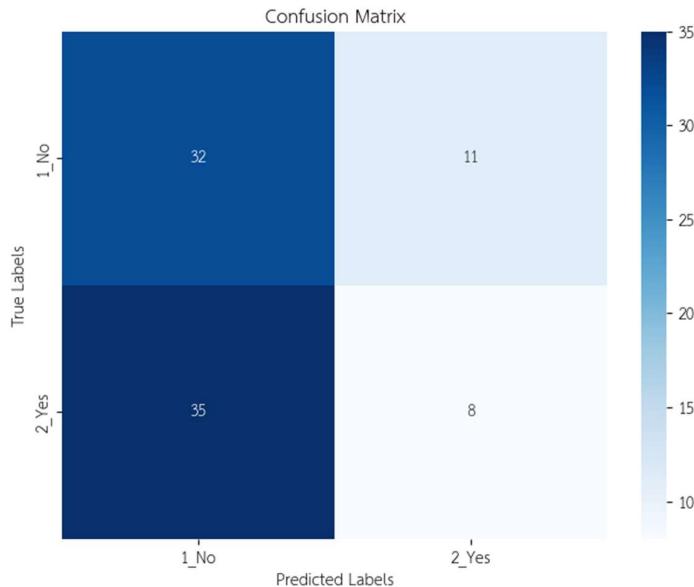
รูปที่ 3.31 ตัวอย่าง Accuracy ของโมเดล Perforation ที่ฝึกฝนด้วย MobileNet V2



รูปที่ 3.32 ตัวอย่าง Loss ของโมเดล Perforation ที่ฝึกฝนด้วย MobileNet V2



รูปที่ 3.33 ตัวอย่าง AUC ของโมเดล Perforation ที่ฝึกฝนด้วย MobileNet V2



รูปที่ 3.34 ตัวอย่าง Confusion Matrix ของโมเดล Perforation ที่ฝึกฝนด้วย MobileNet V2

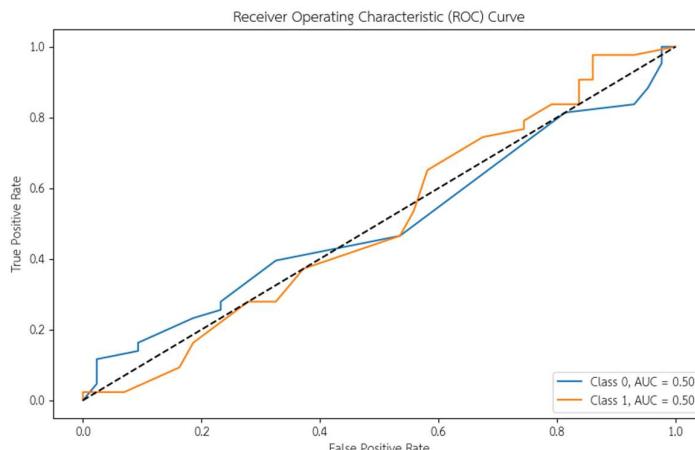
▪ ConvNext Tiny

```
# Train the model
history = model.fit(train_ds, validation_data=val_ds, epochs=10)
```

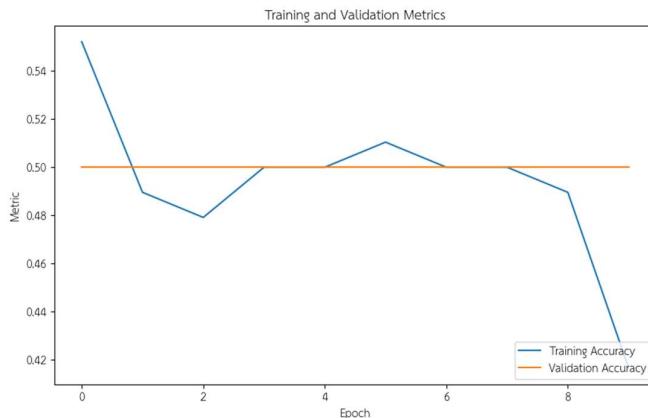
รูปที่ 3.35 ตัวอย่างคำสั่งในการฝึกฝนโมเดล Perforation ด้วย ConvNext Tiny

```
Epoch 1/10
[00000/4000] - 96s: 942ms/step - loss: 1.5470 - accuracy: 0.2638 - F1Score: 0.0039 - Specificity@Specificity: 0.0102 - AUC: 0.5126 - val_loss: 1.0
[00001/4000] - 15s: 55ms/step - loss: 1.4463 - accuracy: 0.2270 - F1Score: 0.0000e+00 - Specificity@Specificity: 0.4396 - AUC: 0.4756 - val_loss:
[00002/4000] - 15s: 55ms/step - loss: 1.4463 - accuracy: 0.2270 - F1Score: 0.0000e+00 - Specificity@Specificity: 0.4375 - AUC: 0.4756 - val_loss:
[00003/4000] - 15s: 55ms/step - loss: 1.4463 - accuracy: 0.2270 - F1Score: 0.0000e+00 - Specificity@Specificity: 0.4375 - AUC: 0.4756 - val_loss:
[00004/4000] - 15s: 55ms/step - loss: 1.4463 - accuracy: 0.2270 - F1Score: 0.0000e+00 - Specificity@Specificity: 0.4375 - AUC: 0.4756 - val_loss:
[00005/4000] - 15s: 55ms/step - loss: 1.4463 - accuracy: 0.2270 - F1Score: 0.0000e+00 - Specificity@Specificity: 0.4375 - AUC: 0.4756 - val_loss:
[00006/4000] - 15s: 55ms/step - loss: 1.4463 - accuracy: 0.2270 - F1Score: 0.0000e+00 - Specificity@Specificity: 0.4375 - AUC: 0.4756 - val_loss:
[00007/4000] - 15s: 55ms/step - loss: 1.4463 - accuracy: 0.2270 - F1Score: 0.0000e+00 - Specificity@Specificity: 0.4375 - AUC: 0.4756 - val_loss:
[00008/4000] - 15s: 55ms/step - loss: 1.4463 - accuracy: 0.2270 - F1Score: 0.0000e+00 - Specificity@Specificity: 0.4375 - AUC: 0.4756 - val_loss:
[00009/4000] - 15s: 55ms/step - loss: 1.4463 - accuracy: 0.2270 - F1Score: 0.0000e+00 - Specificity@Specificity: 0.4375 - AUC: 0.4756 - val_loss:
[00010/4000] - 15s: 55ms/step - loss: 1.4463 - accuracy: 0.2270 - F1Score: 0.0000e+00 - Specificity@Specificity: 0.4375 - AUC: 0.4756 - val_loss:
```

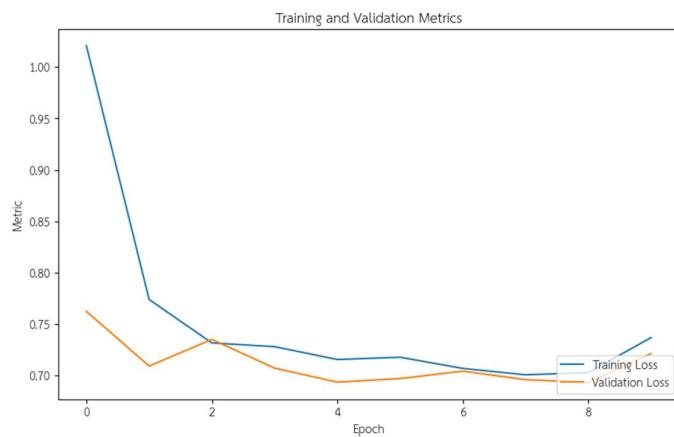
รูปที่ 3.36 ตัวอย่างผลลัพธ์ระหว่างการฝึกฝนโมเดล Perforation ด้วย ConvNext Tiny



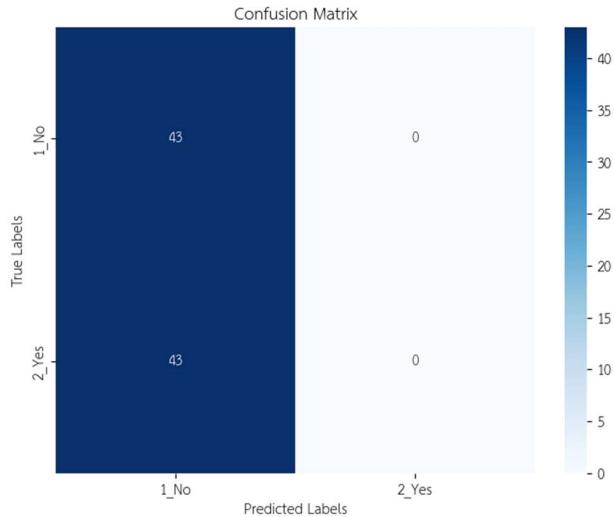
รูปที่ 3.37 ตัวอย่าง AUC ของโมเดล Perforation ที่ฝึกฝนด้วย ConvNext Tiny



รูปที่ 3.38 ตัวอย่าง Accuracy ของโมเดล Perforation ที่ฝึกฝนด้วย ConvNext Tiny



รูปที่ 3.39 ตัวอย่าง Loss ของโมเดล Perforation ที่ฝึกฝนด้วย ConvNext Tiny



รูปที่ 3.40 ตัวอย่าง Confusion Matrix ของโมเดล Perforation ที่ฝึกฝนด้วย ConvNext Tiny

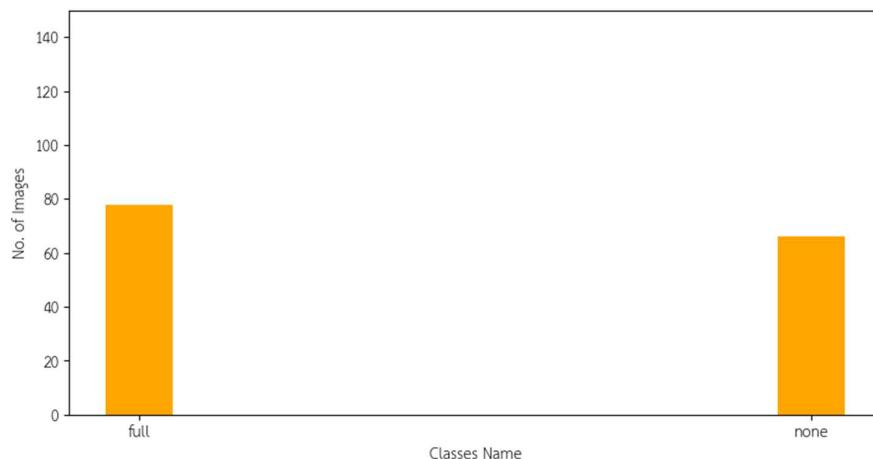
3.2.2.2 Fluid

สำหรับ Fluid จะประกอบไปด้วย 4 คุณลักษณะ ได้แก่ None, Bubble, Level และ Full โดยที่ None หมายถึงหูชั้นกลางที่ไม่มีลักษณะที่ไม่มีของเหลว Bubble หมายถึงหูชั้นกลางที่มีลักษณะของเหลวเป็นฟองอากาศ Level หมายถึงหูชั้นกลางที่มีลักษณะแบ่งเป็นชั้นภายใน และ Full หมายถึงหูชั้นกลางที่มีน้ำเต็มหูโดยจะแสดงดังรูปที่ 3.41



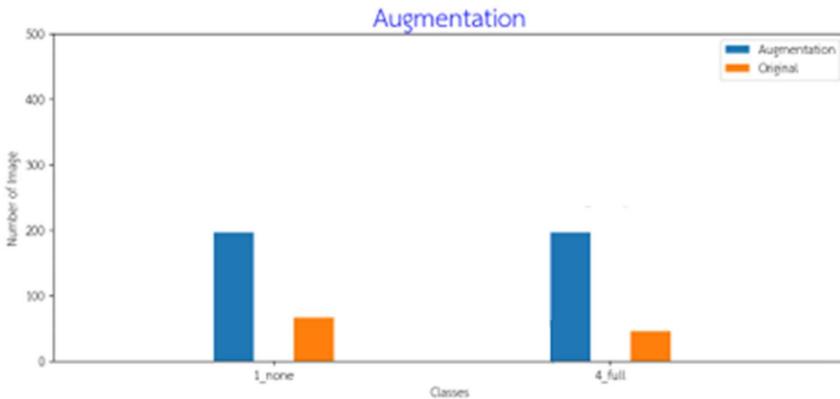
รูปที่ 3.41 ตัวอย่างคุณลักษณะ None, Bubble, Level และ Full

จากรูปที่ 3.42 จะเห็นว่า Fluid เป็น Imbalanced Class ซึ่งจำเป็นต้องมีการเพิ่มข้อมูลของคลาสให้สมดุลกันทุกคลาส



รูปที่ 3.42 จำนวนข้อมูลในแต่ละ Class ของ Fluid

เพิ่มตัวอย่างของ Class ให้สมดุลกันด้วยวิธี Augmentation ด้วยการปรับความ
สว่างและหมุนภาพ



รูปที่ 3.43 จำนวนข้อมูลในแต่ละ Class ของ Fluid หลังทำการ Augmentation

3.2.2.2.1 การฝึกฝนโมเดลด้วย Neural Networks

■ Xception

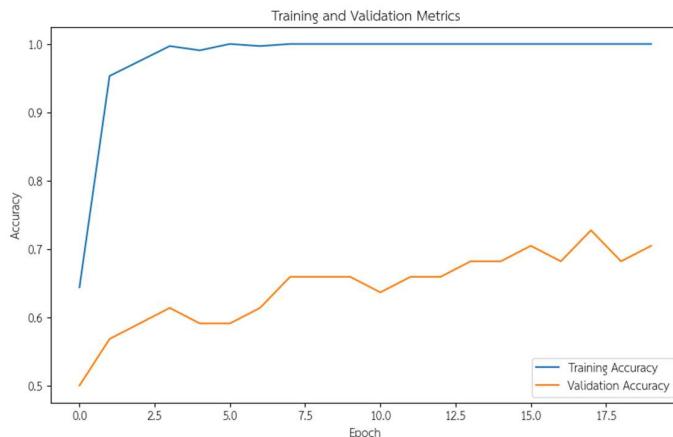
ตัวอย่างการฝึกฝนโมเดลของ Fluid ด้วยสถาปัตยกรรม Xception โดยจะเรียกคำสั่ง `model.fit` เพื่อทำการฝึกฝนโมเดลดังรูปที่ 3.44 ส่วนในระหว่างการฝึกฝนโมเดลจะแสดงผลลัพธ์ดังรูปที่ 3.45 และเมื่อสิ้นสุดการฝึกฝนจะแสดงผลลัพธ์ดังรูปที่ 3.46

```
# Train the model
history = model.fit(train_ds, validation_data=val_ds, epochs=10)
```

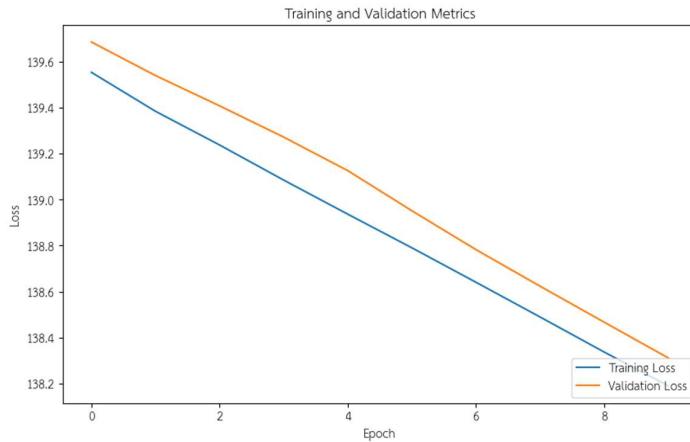
รูปที่ 3.44 ตัวอย่างคำสั่งการฝึกฝนโมเดล Fluid ด้วย Xception

```
Epoch 1/10
[0/1000] [====] - 102s 1s/step - loss: 140.1381 - accuracy: 0.1980 - F1Score: 0.0000e+00 - SpecificityAtSensitivity: 0.4465 - SensitivityAtSpecificity: 0.4975 - AUC: 0.5083 - val_loss: 139.4
Epoch 2/10
[0/1000] [====] - 102s 1s/step - loss: 139.8581 - accuracy: 0.2658 - F1Score: 0.0000e+00 - SpecificityAtSensitivity: 0.5987 - SensitivityAtSpecificity: 0.5542 - AUC: 0.5803 - val_loss: 139.4
Epoch 3/10
[0/1000] [====] - 102s 1s/step - loss: 139.5990 - accuracy: 0.3042 - F1Score: 0.0119 - SpecificityAtSensitivity: 0.6362 - SensitivityAtSpecificity: 0.6117 - AUC: 0.6358 - val_loss: 139.4
Epoch 4/10
[0/1000] [====] - 102s 1s/step - loss: 139.3423 - accuracy: 0.3742 - F1Score: 0.0472 - SpecificityAtSensitivity: 0.7627 - SensitivityAtSpecificity: 0.6454 - AUC: 0.6747 - val_loss: 139.4
Epoch 5/10
[0/1000] [====] - 102s 1s/step - loss: 139.1853 - accuracy: 0.4233 - F1Score: 0.0694 - SpecificityAtSensitivity: 0.7268 - SensitivityAtSpecificity: 0.7759 - AUC: 0.7190 - val_loss: 139.2
Epoch 6/10
[0/1000] [====] - 102s 1s/step - loss: 139.0283 - accuracy: 0.4533 - F1Score: 0.1069 - SpecificityAtSensitivity: 0.8342 - SensitivityAtSpecificity: 0.7758 - AUC: 0.7510 - val_loss: 139.0
Epoch 7/10
[0/1000] [====] - 102s 1s/step - loss: 138.8713 - accuracy: 0.4858 - F1Score: 0.1510 - SpecificityAtSensitivity: 0.8795 - SensitivityAtSpecificity: 0.8108 - AUC: 0.7688 - val_loss: 138.8
Epoch 8/10
[0/1000] [====] - 102s 1s/step - loss: 138.7143 - accuracy: 0.5125 - F1Score: 0.1723 - SpecificityAtSensitivity: 0.9183 - SensitivityAtSpecificity: 0.8656 - AUC: 0.8096 - val_loss: 138.8
Epoch 9/10
[0/1000] [====] - 102s 1s/step - loss: 138.5574 - accuracy: 0.5325 - F1Score: 0.2042 - SpecificityAtSensitivity: 0.9267 - SensitivityAtSpecificity: 0.8837 - AUC: 0.8344 - val_loss: 138.4
Epoch 10/10
[0/1000] [====] - 102s 1s/step - loss: 138.0791 - accuracy: 0.5648 - F1Score: 0.2323 - SpecificityAtSensitivity: 0.9557 - SensitivityAtSpecificity: 0.9179 - AUC: 0.8527 - val_loss: 138.2
```

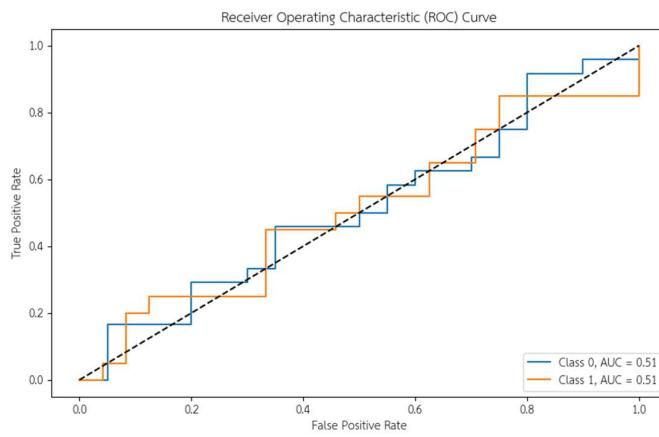
รูปที่ 3.45 ตัวอย่างผลลัพธ์ระหว่างการฝึกฝนโมเดล Fluid ด้วย Xception



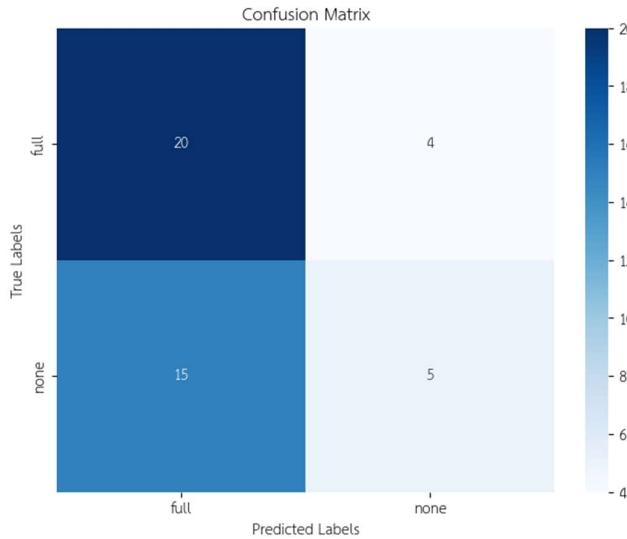
รูปที่ 3.46 ตัวอย่าง Accuracy ของโมเดล Fluid ที่ฝึกฝนโดยใช้ Xception



รูปที่ 3.47 ตัวอย่าง Loss ของโมเดล Fluid ที่ฝึกฝนโดยใช้ Xception



รูปที่ 3.48 ตัวอย่าง AUC ของการฝึกฝน Fluid ด้วย Xception



รูปที่ 3.49 ตัวอย่าง Confusion Matrix ของการฝึกฝน Fluid ด้วย Xception

■ MobileNet V2

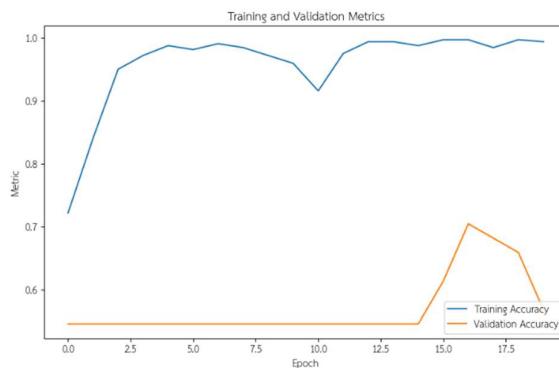
ตัวอย่างการฝึกฝนโมเดลของ Fluid ด้วยสถาปัตยกรรม MobileNet V2 โดยจะเรียกคำสั่ง model.fit เพื่อทำการฝึกฝนโมเดลดังแสดงดังรูปที่ 3.50 ส่วนในระหว่างการฝึกฝนโมเดลจะแสดงดังรูปที่ 3.51 และเมื่อสิ้นสุดการฝึกฝนจะแสดงผลลัพธ์ดังรูปที่ 3.52

```
# Train the model
history = model.fit(train_ds, validation_data=val_ds, epochs=10)
```

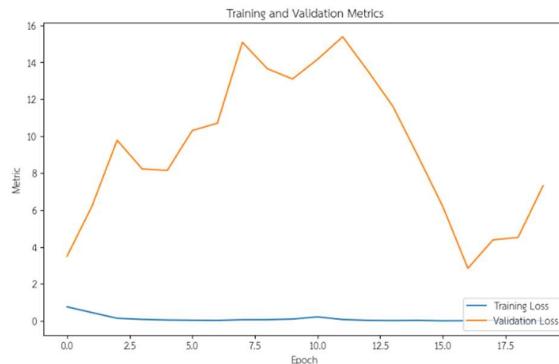
รูปที่ 3.50 ตัวอย่างคำสั่งการฝึกฝนโมเดล Fluid ด้วย MobileNet V2

```
Epoch 1/10
  0ms/1s/step - loss: 0.5584 - accuracy: 0.8083 - F1Score: 0.8162 - SpecificityAtSensitivity: 0.9963 - SensitivityAtSpecificity: 0.9992 - AUC: 0.9993 - val_loss:
Epoch 2/10
  0ms/1s/step - loss: 0.3218 - accuracy: 0.9558 - F1Score: 0.9581 - SpecificityAtSensitivity: 0.9993 - SensitivityAtSpecificity: 0.9997 - AUC: 0.9997 - val_loss:
Epoch 3/10
  0ms/1s/step - loss: 0.1863 - accuracy: 0.9598 - F1Score: 0.9608 - SpecificityAtSensitivity: 0.9992 - SensitivityAtSpecificity: 0.9992 - AUC: 0.9998 - val_loss:
Epoch 4/10
  0ms/1s/step - loss: 0.1373 - accuracy: 0.9481 - F1Score: 0.9478 - SpecificityAtSensitivity: 0.9997 - SensitivityAtSpecificity: 0.9995 - AUC: 0.9999 - val_loss:
Epoch 5/10
  0ms/1s/step - loss: 0.1273 - accuracy: 0.9481 - F1Score: 0.9488 - SpecificityAtSensitivity: 0.9992 - SensitivityAtSpecificity: 0.9994 - AUC: 0.9999 - val_loss:
Epoch 6/10
  0ms/1s/step - loss: 0.1158 - accuracy: 0.9592 - F1Score: 0.9588 - SpecificityAtSensitivity: 0.9998 - SensitivityAtSpecificity: 0.9992 - AUC: 0.9998 - val_loss:
Epoch 7/10
  0ms/1s/step - loss: 0.1217 - accuracy: 0.9515 - F1Score: 0.9538 - SpecificityAtSensitivity: 0.9995 - SensitivityAtSpecificity: 0.9997 - AUC: 0.9995 - val_loss:
Epoch 8/10
  0ms/1s/step - loss: 0.1217 - accuracy: 0.9517 - F1Score: 0.9597 - SpecificityAtSensitivity: 0.9995 - SensitivityAtSpecificity: 0.9997 - AUC: 0.9995 - val_loss:
Epoch 9/10
  0ms/1s/step - loss: 0.1146 - accuracy: 0.9611 - F1Score: 0.9611 - SpecificityAtSensitivity: 0.9993 - SensitivityAtSpecificity: 0.9998 - AUC: 0.9996 - val_loss:
Epoch 10/10
  0ms/1s/step - loss: 0.0681 - accuracy: 0.9880 - F1Score: 0.9881 - SpecificityAtSensitivity: 0.9997 - SensitivityAtSpecificity: 0.9993 - AUC: 0.9997 - val_loss:
```

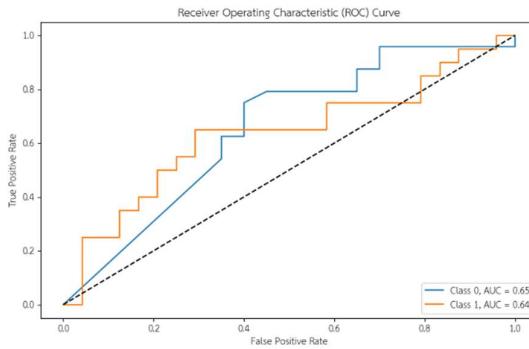
รูปที่ 3.51 ตัวอย่างผลลัพธ์ระหว่างการฝึกฝนโมเดล Fluid ด้วย MobileNet V2



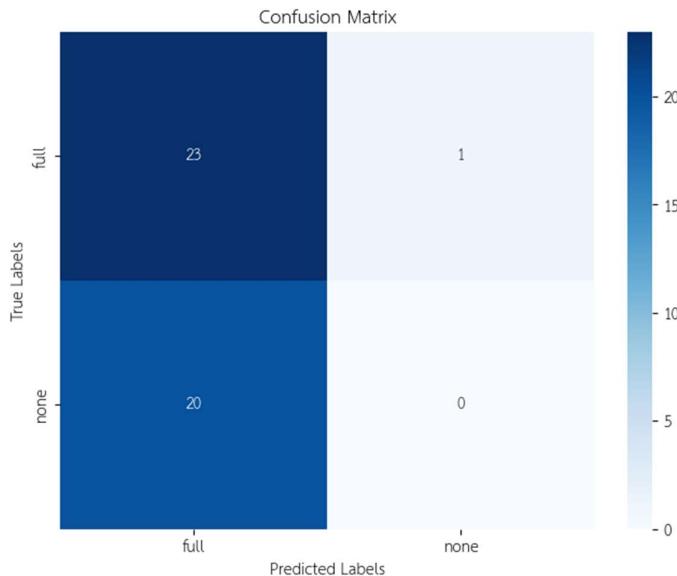
รูปที่ 3.52 ตัวอย่าง Accuracy ของโมเดล Fluid ที่ฝึกฝนโดยใช้ MobileNet V2



รูปที่ 3.53 ตัวอย่าง Loss ของโมเดล Fluid ที่ฝึกฝนโดยใช้ MobileNet V2



รูปที่ 3.54 ตัวอย่าง AUC ของโมเดล Fluid ที่ฝึกฝนโดยใช้ MobileNet V2



รูปที่ 3.55 ตัวอย่าง Confusion Matrix ของโมเดล Fluid ที่ฝึกฝนโดยใช้ MobileNet V2

■ ConvNext Tiny

ตัวอย่างการฝึกฝนโมเดลของ Fluid ด้วยสถาปัตยกรรม ConvNext Tiny โดยจะเรียกคำสั่ง `model.fit` เพื่อทำการฝึกฝนโมเดลดังแสดงดังรูปที่ 3.56 ส่วนในระหว่างการฝึกฝนโมเดลจะแสดงดังรูปที่ 3.57 และเมื่อสิ้นสุดการฝึกฝนจะแสดงผลลัพธ์ดังรูปที่ 3.58

```
# Train the model
history = model.fit(train_ds, validation_data=val_ds, epochs=10)
```

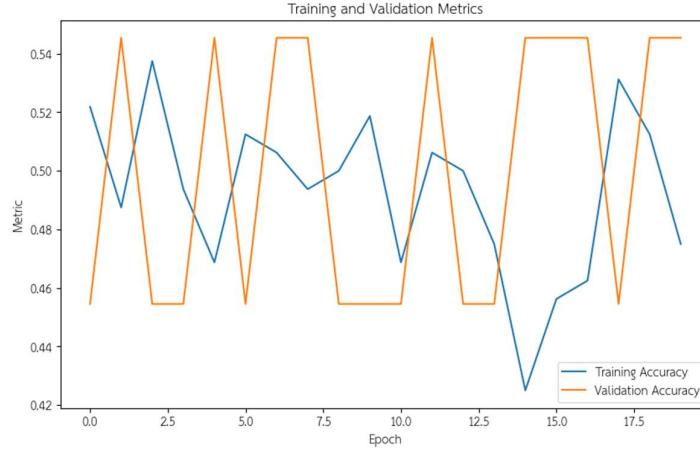
รูปที่ 3.56 ตัวอย่างคำสั่งการฝึกฝนโมเดล Fluid ด้วย ConvNext Tiny

```

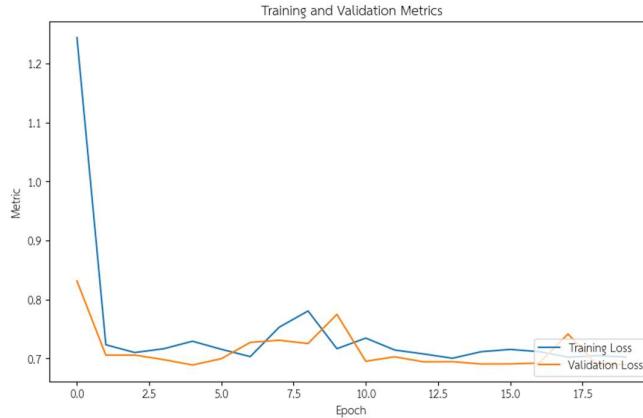
Epoch 1/10
Epoch 2/10 [=====] - 128s 2s/step - loss: 1.0887 - accuracy: 0.1625 - SpecificityAtSensitivity: 0.4958 - SensitivityAtSpecificity: 0.4583 - AUC: 0.4991 - val_loss: 1.06
Epoch 3/10 [=====] - 50s 1s/step - loss: 1.0354 - accuracy: 0.1817 - F1Score: 0.0000e+00 - SpecificityAtSensitivity: 0.4608 - SensitivityAtSpecificity: 0.4983 - AUC: 0.5064 - val_loss: 1.06
Epoch 4/10 [=====] - 57s 1s/step - loss: 1.0512 - accuracy: 0.1700 - F1Score: 0.0000e+00 - SpecificityAtSensitivity: 0.5037 - SensitivityAtSpecificity: 0.5153 - AUC: 0.5123 - val_loss: 1.06
Epoch 5/10 [=====] - 56s 1s/step - loss: 1.0370 - accuracy: 0.1700 - F1Score: 0.0000e+00 - SpecificityAtSensitivity: 0.4925 - SensitivityAtSpecificity: 0.4992 - AUC: 0.4979 - val_loss: 1.06
Epoch 6/10 [=====] - 56s 1s/step - loss: 1.0297 - accuracy: 0.1835 - F1Score: 0.0000e+00 - SpecificityAtSensitivity: 0.4917 - SensitivityAtSpecificity: 0.4942 - AUC: 0.4934 - val_loss: 1.06
Epoch 7/10 [=====] - 56s 1s/step - loss: 1.0316 - accuracy: 0.1853 - F1Score: 0.0000e+00 - SpecificityAtSensitivity: 0.4905 - SensitivityAtSpecificity: 0.4967 - AUC: 0.4959 - val_loss: 1.06
Epoch 8/10 [=====] - 56s 1s/step - loss: 1.0315 - accuracy: 0.1863 - F1Score: 0.0000e+00 - SpecificityAtSensitivity: 0.4903 - SensitivityAtSpecificity: 0.4975 - AUC: 0.4962 - val_loss: 1.06
Epoch 9/10 [=====] - 56s 1s/step - loss: 1.0215 - accuracy: 0.1725 - F1Score: 0.0000e+00 - SpecificityAtSensitivity: 0.4918 - SensitivityAtSpecificity: 0.4750 - AUC: 0.5023 - val_loss: 1.06
Epoch 10/10 [=====] - 55s 1s/step - loss: 1.0257 - accuracy: 0.1700 - F1Score: 0.0000e+00 - SpecificityAtSensitivity: 0.4932 - SensitivityAtSpecificity: 0.4800 - AUC: 0.5060 - val_loss: 1.06
Epoch 11/10 [=====] - 55s 1s/step - loss: 1.0188 - accuracy: 0.1775 - F1Score: 0.0000e+00 - SpecificityAtSensitivity: 0.4545 - SensitivityAtSpecificity: 0.4942 - AUC: 0.5109 - val_loss: 1.06

```

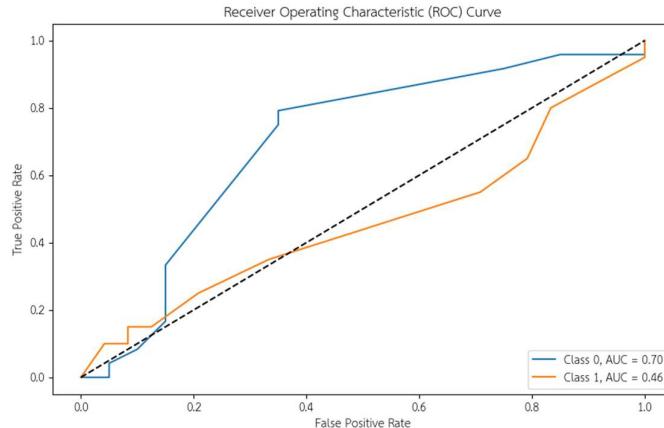
รูปที่ 3.57 ตัวอย่างผลลัพธ์ระหว่างการฝึกฝนโมเดล Fluid ด้วย ConvNext Tiny



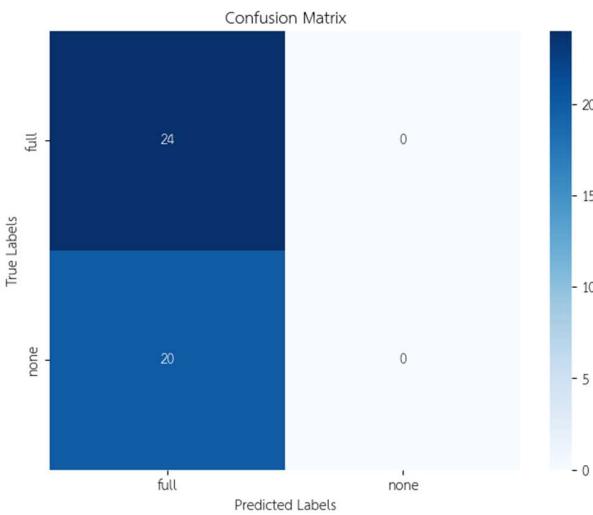
รูปที่ 3.58 ตัวอย่าง Accuracy ของโมเดล Fluid ที่ฝึกฝนโดยใช้ ConvNext Tiny



รูปที่ 3.59 ตัวอย่าง Loss ของโมเดล Fluid ที่ฝึกฝนโดยใช้ ConvNext Tiny



รูปที่ 3.60 ตัวอย่าง AUC ของโมเดล Fluid ที่ฝึกฝนโดยใช้ ConvNext Tiny



รูปที่ 3.61 ตัวอย่าง Confusion Matrix ของโมเดล Fluid ที่ฝึกฝนโดยใช้ ConvNext Tiny

3.2.2.2.2 การฝึกฝนโมเดลด้วย Traditional Machine Learning

■ SVM

ตัวอย่างการฝึกฝนโมเดลของ Fluid ด้วยสถาปัตยกรรม SVM โดยจะเรียกคำสั่ง classifier.fit เพื่อทำการฝึกฝนโมเดลดังรูปที่ 3.62 ส่วนในระหว่างการฝึกฝนโมเดลจะแสดงดังรูปที่ 3.63 และเมื่อสิ้นสุดการฝึกฝนจะแสดงผลลัพธ์ดังรูปที่ 3.64

```
# Train the classifier
classifier = SVC(kernel='linear', probability=True)
classifier.fit(X_pca, y_train)
```

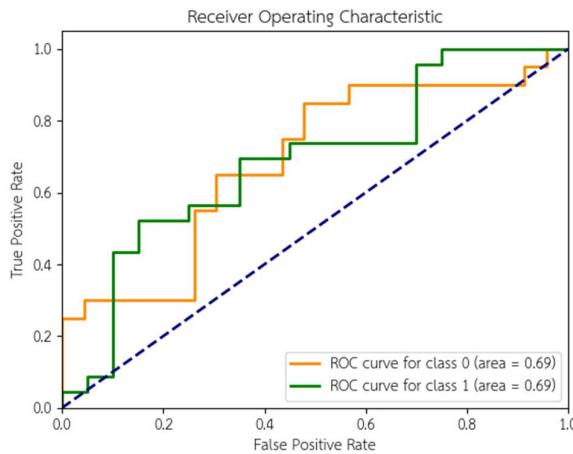
รูปที่ 3.62 ตัวอย่างคำสั่งการฝึกฝนโมเดล Fluid ด้วย SVM

```
Classification report for -
SVC(kernel='sigmoid', probability=True):
precision    recall    f1-score   support
          0       0.62      0.65      0.63      20
          1       0.68      0.65      0.67      23

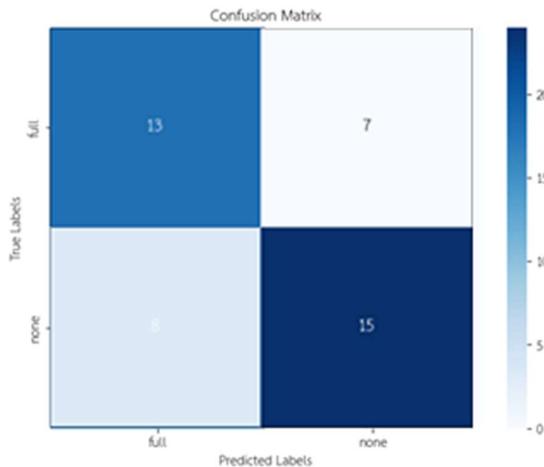
accuracy                           0.65      43
macro avg       0.65      0.65      0.65      43
weighted avg    0.65      0.65      0.65      43

specificity macro avg {.2f} 0.6510869565217392
specificity micro avg {.2f} 0.6511627906976745
specificity weighted avg {.2f} 0.6510111223458038
AUC 0.6510869565217391
Confusion Matrix:
```

รูปที่ 3.63 ตัวอย่างผลลัพธ์โดยรวมของการฝึกฝนโมเดล Fluid ด้วย SVM



รูปที่ 3.64 ตัวอย่างผลลัพธ์ AUC ของการฝึกฝนโมเดล Fluid ด้วย SVM



รูปที่ 3.65 ตัวอย่าง Confusion Matrix ของการฝึกฝนโมเดล Fluid ด้วย SVM

■ KNN

ตัวอย่างการฝึกฝนโมเดลของ Fluid ด้วยสถาปัตยกรรม KNN โดยจะเรียกคำสั่ง `classifier.fit` เพื่อทำการฝึกฝนโมเดลดังรูปที่ 3.66 ส่วนในระหว่างการฝึกฝนโมเดลจะแสดงดังรูปที่ 3.67 และเมื่อสิ้นสุดการฝึกฝนจะแสดงผลลัพธ์ดังรูปที่ 3.68

```
# Train the classifier
classifier = KNeighborsClassifier(n_neighbors=2)
classifier.fit(X_pca, y_train)
```

รูปที่ 3.66 ตัวอย่างคำสั่งการฝึกฝนโมเดล Fluid ด้วย KNN

```

Classification report for -
KNeighborsClassifier(n_neighbors=11):
      precision    recall  f1-score   support

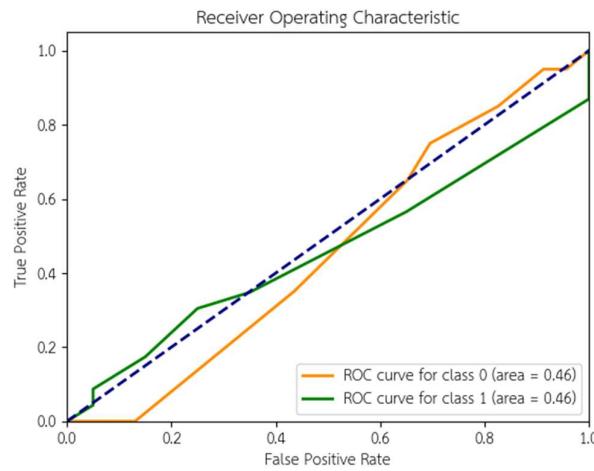
          0       0.48      0.75      0.59      20
          1       0.58      0.30      0.40      23

   accuracy                           0.51      43
macro avg       0.53      0.53      0.49      43
weighted avg    0.54      0.51      0.49      43

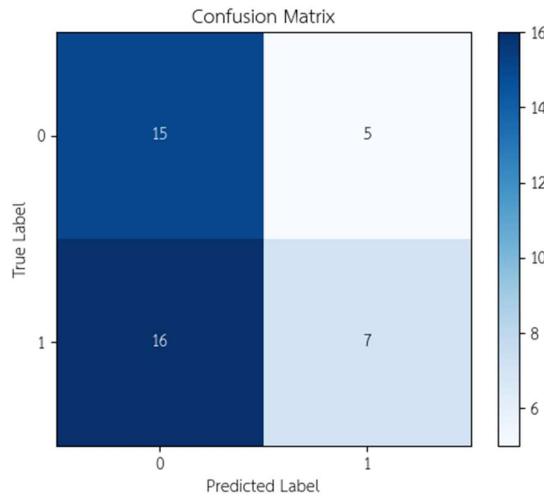
specificity macro avg {.2f} 0.5271739130434783
specificity micro avg {.2f} 0.5116279069767442
specificity weighted avg {.2f} 0.5427199191102123
AUC 0.5271739130434783

```

รูปที่ 3.67 ตัวอย่างผลลัพธ์โดยรวมของการฝึกฝนโมเดล Colot ด้วย KNN



รูปที่ 3.68 ตัวอย่าง AUC ของการฝึกฝนโมเดล Fluid ด้วย KNN



รูปที่ 3.69 ตัวอย่าง Confusion Matrix ของการฝึกฝนโมเดล Fluid ด้วย KNN

■ Decision Tree

ตัวอย่างการฝึกฝนโมเดลของ Fluid ด้วยสถาปัตยกรรม Decision Tree โดยจะเรียกคำสั่ง classifier.fit เพื่อทำการฝึกฝนโมเดลดังรูปที่ 3.70 ส่วนในระหว่างการฝึกฝนโมเดลจะแสดงดังรูปที่ 3.71 และเมื่อสิ้นสุดการฝึกฝนจะแสดงผลลัพธ์ดังรูปที่ 3.72

```
# Train the classifier
classifier = tree.DecisionTreeClassifier()
classifier.fit(X_pca, y_train)
```

รูปที่ 3.70 ตัวอย่างคำสั่งการฝึกฝนโมเดล Fluid ด้วย Decision Tree

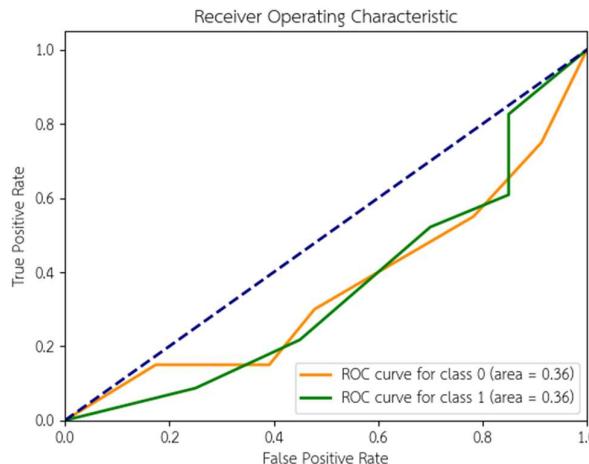
```
Classification report for -
DecisionTreeClassifier(max_depth=3, min_samples_leaf=20):
              precision    recall  f1-score   support

             0       0.35     0.30     0.32      20
             1       0.46     0.52     0.49      23

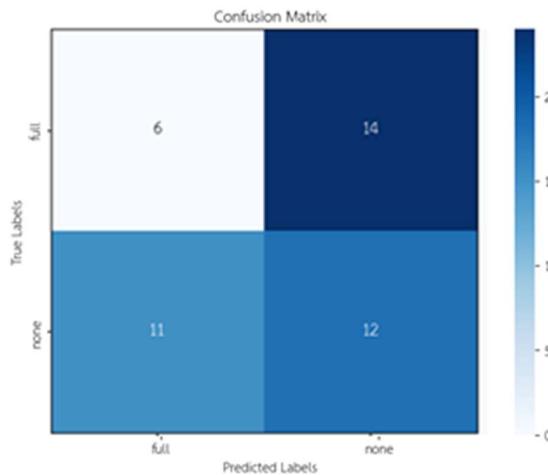
      accuracy                           0.42      43
   macro avg       0.41     0.41     0.41      43
weighted avg       0.41     0.42     0.41      43

specification macro avg {.2f} 0.41086956521739126
specification micro avg {.2f} 0.4186046511627907
specification weighted avg {.2f} 0.40313447927199186
AUC 0.4108695652173913
```

รูปที่ 3.71 ตัวอย่างผลลัพธ์โดยรวมของการฝึกฝน Fluid ด้วย Decision Tree



รูปที่ 3.72 ตัวอย่าง AUC ของการฝึกฝนโมเดล Fluid ด้วย Decision Tree



รูปที่ 3.73 ตัวอย่าง Confusion Matrix ของการฝึกฝนโมเดล Fluid ด้วย Decision Tree

■ XGBoost

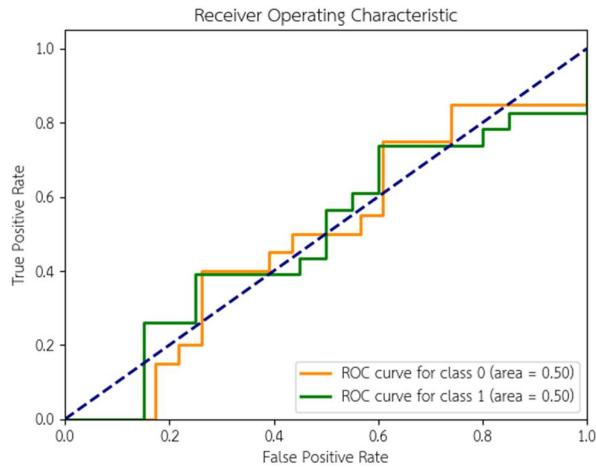
ตัวอย่างการฝึกฝนโมเดลของ Fluid ด้วยสถาบัตยกรรม Decision Tree โดยจะเรียกคำสั่ง `classifier.fit` เพื่อทำการฝึกฝนโมเดลโดยจะแสดงดังรูปที่ 3.74 ส่วนในระหว่างการฝึกฝนโมเดลจะแสดงดังรูปที่ 3.75 และเมื่อสิ้นสุดการฝึกฝนจะแสดงผลลัพธ์ดังรูปที่ 3.76

```
# Train the classifier
classifier = XGBClassifier(
    n_estimators=50, # Number of boosting rounds or decision trees
    learning_rate=0.01, # Step size shrinkage
    max_depth=4, # Maximum depth of each decision tree
    subsample=1.0, # Fraction of samples used for training each tree
    colsample_bytree=1.0, # Fraction of features used for training each tree
    reg_alpha=0, # L1 regularization term
    reg_lambda=1, # L2 regularization term
    gamma=0.5, # Minimum loss reduction required to split a node
)
classifier.fit(X_pca, y_train)
```

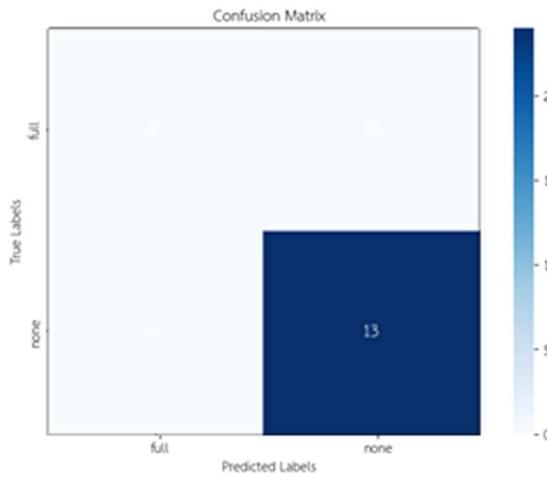
รูปที่ 3.74 ตัวอย่างคำสั่งการฝึกฝน Fluid ด้วย XGBoost

```
Classification report for -
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=None, monotone_constraints=None,
              n_estimators=100, n_jobs=-1, num_parallel_trees=1,
              objectives='multi:softmax', predictor=None, ...):
Precision      recall   f1-score   support
0            0.46     0.33     0.39      18
1            0.44     0.79     0.57      19
2            0.31     0.50     0.38      18
3            0.78     0.23     0.36      30
accuracy          0.44      85
macro avg       0.50     0.46     0.42      85
weighted avg    0.54     0.44     0.42      85
```

รูปที่ 3.75 ตัวอย่างผลลัพธ์โดยรวมของการฝึกฝน Fluid ด้วย XGBoost



รูปที่ 3.76 ตัวอย่าง AUC ของการฝึกฝนโมเดล Fluid ด้วย XGBoost



รูปที่ 3.77 ตัวอย่าง Confusion Matrix ของการฝึกฝนโมเดล Fluid ด้วย XGBoost

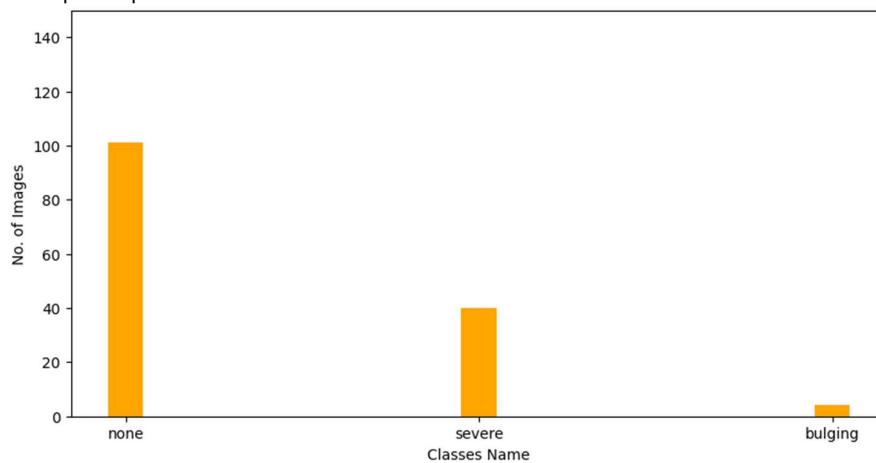
3.2.2.3 Retraction

สำหรับ Retraction จะประกอบไปด้วย 4 คุณลักษณะ ได้แก่ None, Mild, Moderate, Severe และ Bulging โดยที่ None หมายถึงหูชี้นกลางที่ไม่มีลักษณะของการหดตัว Mild หมายถึงหูชี้นกลางที่มีลักษณะการหดตัวเล็กน้อย Moderate หมายถึงหูชี้นกลางที่มีลักษณะการหดตัวปานกลาง Severe หมายถึงหูชี้นกลางที่มีลักษณะการหดตัวอย่างรุนแรง และ Bulging หมายถึงหูชี้นกลางที่มีการปูด โดยจะแสดงดังรูปที่ 3.78



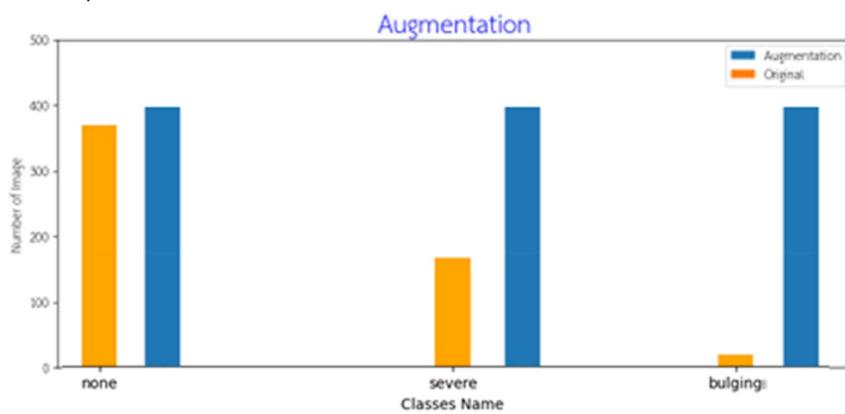
รูปที่ 3.78 ตัวอย่างคุณลักษณะ None, Mild, Moderate, Severe และ Bulging

จากรูปที่ 3.79 จะเห็นว่า Retraction เป็น Imbalanced Class ซึ่งจำเป็นต้องมีการเพิ่มข้อมูลของคลาสให้สมดุลกันทุกคลาส



รูปที่ 3.79 จำนวนข้อมูลในแต่ละ Class ของ Retraction

เพิ่มตัวอย่างของ class ให้สมดุลกันด้วยวิธี Augmentation ด้วยการปรับความสว่างและหมุนภาพ



รูปที่ 3.80 จำนวนข้อมูลในแต่ละ Class ของ Retraction หลังทำการ Augmentation

3.2.2.3.1 การฝึกฝนโมเดลด้วย Neural Networks

▪ Xception

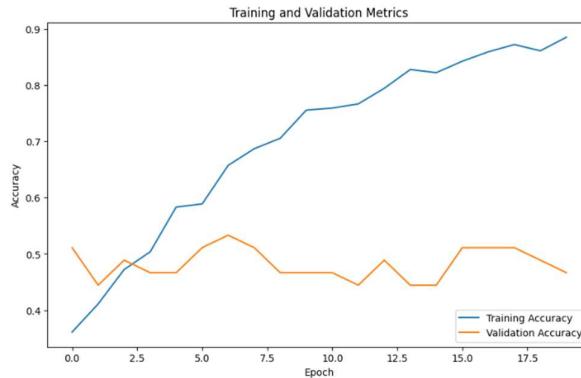
ตัวอย่างการฝึกฝนโมเดลของ Retraction ด้วยสถาปัตยกรรม Xception โดยจะเรียกคำสั่ง `model.fit` เพื่อทำการฝึกฝนโมเดลโดยจะแสดงดังรูปที่ 3.81 ส่วนในระหว่างการฝึกฝนโมเดลจะแสดงดังรูปที่ 3.82 และเมื่อสิ้นสุดการฝึกฝนจะแสดงผลลัพธ์ดังรูปที่ 3.83

```
# Train the model
history = model.fit(train_ds, validation_data=val_ds, epochs=10)
```

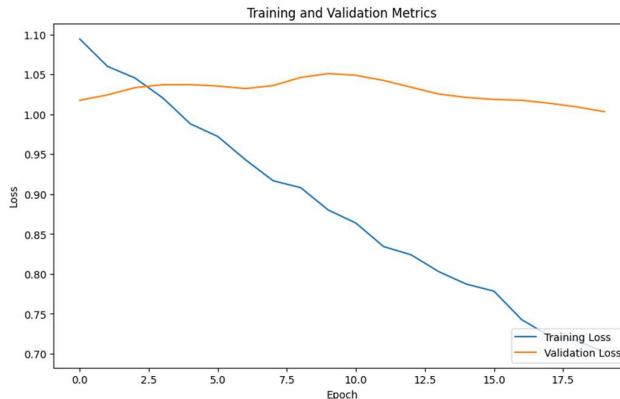
รูปที่ 3.81 ตัวอย่างคำสั่งการฝึกฝนโมเดล Retraction ด้วย Xception

```
Epoch 3/10
Epoch 3/10 [=====] - 102s/1step - loss: 140.1381 - accuracy: 0.1980 - PIScore: 0.0000e+00 - SpecificityAtSensitivity: 0.4465 - SensitivityAtSpecificity: 0.4975 - AUC: 0.5983 - val_loss: 139.4
Epoch 3/10 [=====] - 56s/1step - loss: 139.8581 - accuracy: 0.2658 - PIScore: 0.0000e+00 - SpecificityAtSensitivity: 0.5987 - SensitivityAtSpecificity: 0.5342 - AUC: 0.5880 - val_loss: 139.4
Epoch 4/10 [=====] - 102s/1step - loss: 139.5990 - accuracy: 0.3042 - PIScore: 0.0119 - SpecificityAtSensitivity: 0.6117 - SensitivityAtSpecificity: 0.6162 - AUC: 0.6350 - val_loss: 139.4
Epoch 4/10 [=====] - 56s/1step - loss: 139.3423 - accuracy: 0.3742 - PIScore: 0.0472 - SpecificityAtSensitivity: 0.7027 - SensitivityAtSpecificity: 0.6558 - AUC: 0.6747 - val_loss: 139.4
Epoch 5/10 [=====] - 102s/1step - loss: 139.0863 - accuracy: 0.4231 - PIScore: 0.0804 - SpecificityAtSensitivity: 0.7268 - SensitivityAtSpecificity: 0.7058 - AUC: 0.7198 - val_loss: 139.4
Epoch 5/10 [=====] - 56s/1step - loss: 139.0343 - accuracy: 0.4533 - PIScore: 0.1049 - SpecificityAtSensitivity: 0.7342 - SensitivityAtSpecificity: 0.7558 - AUC: 0.7540 - val_loss: 139.4
Epoch 6/10 [=====] - 102s/1step - loss: 138.3314 - accuracy: 0.5325 - PIScore: 0.1723 - SpecificityAtSensitivity: 0.8108 - SensitivityAtSpecificity: 0.8795 - AUC: 0.8288 - val_loss: 138.4
Epoch 6/10 [=====] - 56s/1step - loss: 138.0791 - accuracy: 0.5680 - PIScore: 0.2042 - SpecificityAtSensitivity: 0.8267 - SensitivityAtSpecificity: 0.8817 - AUC: 0.8344 - val_loss: 138.4
Epoch 7/10 [=====] - 102s/1step - loss: 137.8279 - accuracy: 0.6000 - PIScore: 0.2323 - SpecificityAtSensitivity: 0.8957 - SensitivityAtSpecificity: 0.9175 - AUC: 0.8527 - val_loss: 138.4
Epoch 7/10 [=====] - 56s/1step - loss: 137.8279 - accuracy: 0.6000 - PIScore: 0.2323 - SpecificityAtSensitivity: 0.8957 - SensitivityAtSpecificity: 0.9175 - AUC: 0.8527 - val_loss: 138.4
```

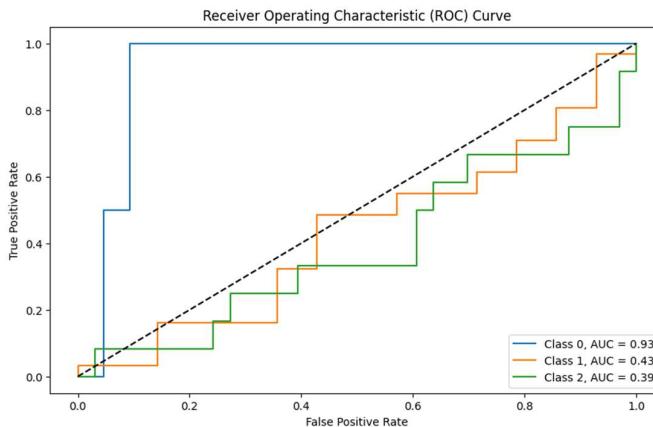
รูปที่ 3.82 ตัวอย่างผลลัพธ์ระหว่างการฝึกฝนโมเดล Retraction ด้วย Xception



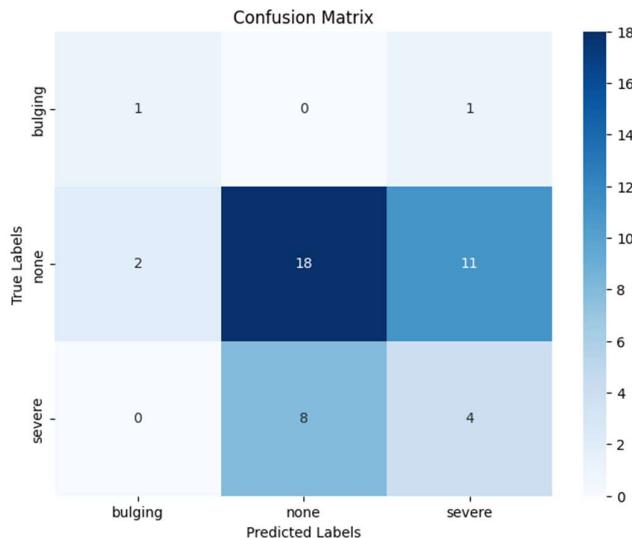
รูปที่ 3.83 ตัวอย่าง Accuracy ของโมเดล Retraction ที่ฝึกฝนโดยใช้ Xception



รูปที่ 3.84 ตัวอย่าง Loss ของโมเดล Retraction ที่ฝึกฝนโดยใช้ Xception



รูปที่ 3.85 ตัวอย่าง AUC ของการฝึกฝน Retraction ด้วย Xception



รูปที่ 3.86 ตัวอย่าง Confusion Matrix ของการฝึกฝน Retraction ด้วย Xception

■ MobileNet V2

ตัวอย่างการฝึกฝนโมเดลของ Retraction ด้วยสถาปัตยกรรม MobileNet V2 โดยจะเรียกคำสั่ง `model.fit` เพื่อทำการฝึกฝนโมเดลโดยจะแสดงดังรูปที่ 3.87 ส่วนในระหว่างการฝึกฝนโมเดลจะแสดงดังรูปที่ 3.88 และเมื่อสิ้นสุดการฝึกฝนจะแสดงผลลัพธ์ดังรูปที่ 3.89

```
# Train the model
history = model.fit(train_ds, validation_data=val_ds, epochs=10)
```

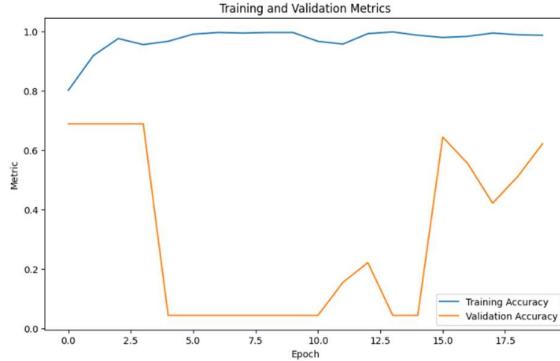
รูปที่ 3.87 ตัวอย่างคำสั่งการฝึกฝนโมเดล Retraction ด้วย MobileNet V2

```

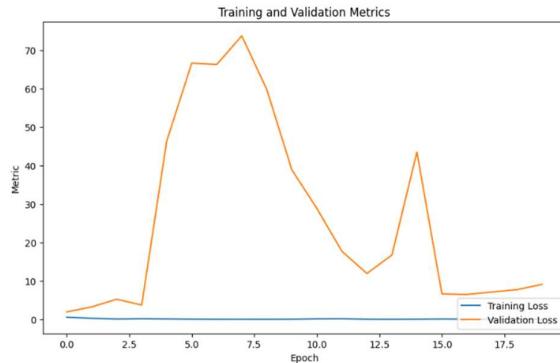
Epoch 1/10
1/10 [=====] - 104s 1s/step - loss: 0.5584 - accuracy: 0.8083 - F1Score: 0.8162 - SpecificityAtSensitivity: 0.9963 - SensitivityAtSpecificity: 0.9982 - AUC: 0.9693 - val_loss:
2/10 [=====] - 61s 1s/step - loss: 0.3218 - accuracy: 0.9558 - F1Score: 0.9561 - SpecificityAtSensitivity: 0.9993 - SensitivityAtSpecificity: 0.9967 - AUC: 0.9967 - val_loss:
3/10 [=====] - 57s 1s/step - loss: 0.1863 - accuracy: 0.9701 - F1Score: 0.9704 - SpecificityAtSensitivity: 0.9994 - SensitivityAtSpecificity: 0.9989 - AUC: 0.9971 - val_loss:
4/10 [=====] - 57s 1s/step - loss: 0.1853 - accuracy: 0.9488 - F1Score: 0.9498 - SpecificityAtSensitivity: 0.9992 - SensitivityAtSpecificity: 0.9942 - AUC: 0.9938 - val_loss:
5/10 [=====] - 54s 1s/step - loss: 0.1875 - accuracy: 0.9463 - F1Score: 0.9479 - SpecificityAtSensitivity: 0.9991 - SensitivityAtSpecificity: 0.9925 - AUC: 0.9955 - val_loss:
6/10 [=====] - 47s 1s/step - loss: 0.1138 - accuracy: 0.9952 - F1Score: 0.9538 - SpecificityAtSensitivity: 0.9998 - SensitivityAtSpecificity: 0.9992 - AUC: 0.9980 - val_loss:
7/10 [=====] - 48s 1s/step - loss: 0.1373 - accuracy: 0.9525 - F1Score: 0.9538 - SpecificityAtSensitivity: 0.9999 - SensitivityAtSpecificity: 0.9967 - AUC: 0.9966 - val_loss:
8/10 [=====] - 53s 1s/step - loss: 0.2217 - accuracy: 0.9517 - F1Score: 0.9507 - SpecificityAtSensitivity: 0.9995 - SensitivityAtSpecificity: 0.9937 - AUC: 0.9933 - val_loss:
9/10 [=====] - 53s 1s/step - loss: 0.1390 - accuracy: 0.9653 - F1Score: 0.9611 - SpecificityAtSensitivity: 0.9958 - SensitivityAtSpecificity: 0.9958 - AUC: 0.9961 - val_loss:
10/10 [=====] - 48s 1s/step - loss: 0.1468 - accuracy: 0.9625 - F1Score: 0.9639 - SpecificityAtSensitivity: 0.9993 - SensitivityAtSpecificity: 0.9942 - AUC: 0.9955 - val_loss:
11/10 [=====] - 53s 1s/step - loss: 0.0681 - accuracy: 0.9880 - F1Score: 0.9890 - SpecificityAtSensitivity: 0.9983 - SensitivityAtSpecificity: 0.9983 - AUC: 0.9987 - val_loss:

```

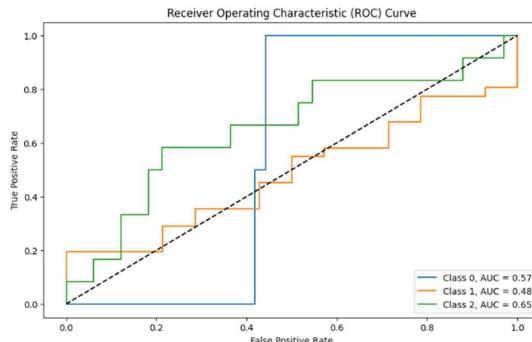
รูปที่ 3.88 ตัวอย่างผลลัพธ์ระหว่างการฝึกฝนโมเดล Retraction ด้วย MobileNet V2



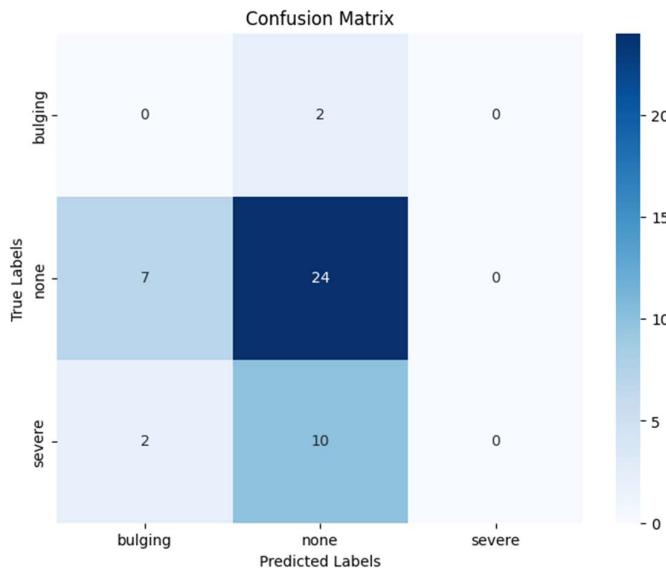
รูปที่ 3.89 ตัวอย่าง Accuracy ของโมเดล Retraction ที่ฝึกฝนโดยใช้ MobileNet V2



รูปที่ 3.90 ตัวอย่าง Loss ของโมเดล Retraction ที่ฝึกฝนโดยใช้ MobileNet V2



รูปที่ 3.91 ตัวอย่าง AUC ของโมเดล Retraction ที่ฝึกฝนโดยใช้ MobileNet V2



รูปที่ 3.92 ตัวอย่าง Confusion Matrix ของโมเดล Retraction ที่ฝึกฝนโดยใช้ MobileNet V2

▪ ConvNext Tiny

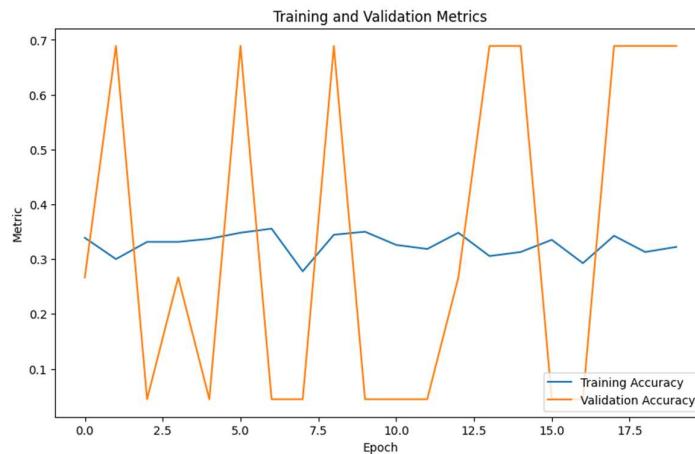
ตัวอย่างการฝึกฝนโมเดลของ Retraction ด้วยสถาปัตยกรรม ConvNext Tiny โดยจะเรียกคำสั่ง `model.fit` เพื่อทำการฝึกฝนโมเดลโดยจะแสดงดังรูปที่ 3.93 ส่วนในระหว่างการฝึกฝนโมเดลจะแสดงดังรูปที่ 3.94 และเมื่อสิ้นสุดการฝึกฝนจะแสดงผลลัพธ์ดังรูปที่ 3.95

```
# Train the model
history = model.fit(train_ds, validation_data=val_ds, epochs=10)
```

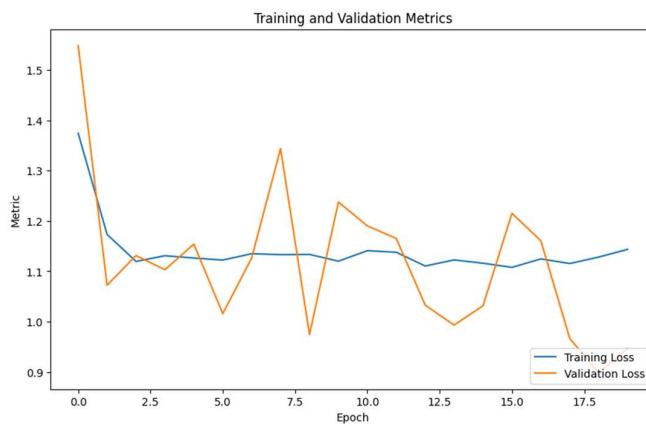
รูปที่ 3.93 ตัวอย่างคำสั่งการฝึกฝนโมเดล Retraction ด้วย ConvNext Tiny

```
Epoch 1/10
[0/1280 steps] - 1280/2s/step - loss: 1.9817 - accuracy: 0.1625 - F1Score: 0.0000 - SpecificityAtSensitivity: 0.4958 - SensitivityAtSpecificity: 0.4983 - AUC: 0.4991 - val_loss: 1.96
Epoch 2/10
[0/1280 steps] - 1280/2s/step - loss: 1.8954 - accuracy: 0.1817 - F1Score: 0.0000e+00 - SpecificityAtSensitivity: 0.4698 - SensitivityAtSpecificity: 0.4983 - AUC: 0.5064 - val_loss: 1.96
Epoch 3/10
[0/1280 steps] - 1280/2s/step - loss: 1.8521 - accuracy: 0.1700 - F1Score: 0.0000e+00 - SpecificityAtSensitivity: 0.5037 - SensitivityAtSpecificity: 0.5183 - AUC: 0.5123 - val_loss: 1.96
Epoch 4/10
[0/1280 steps] - 1280/2s/step - loss: 1.8370 - accuracy: 0.1780 - F1Score: 0.0000e+00 - SpecificityAtSensitivity: 0.4925 - SensitivityAtSpecificity: 0.4692 - AUC: 0.4979 - val_loss: 1.96
Epoch 5/10
[0/1280 steps] - 1280/2s/step - loss: 1.8227 - accuracy: 0.1840 - F1Score: 0.0000e+00 - SpecificityAtSensitivity: 0.4577 - SensitivityAtSpecificity: 0.4942 - AUC: 0.4938 - val_loss: 1.96
Epoch 6/10
[0/1280 steps] - 1280/2s/step - loss: 1.8156 - accuracy: 0.1883 - F1Score: 0.0000e+00 - SpecificityAtSensitivity: 0.4835 - SensitivityAtSpecificity: 0.4967 - AUC: 0.5059 - val_loss: 1.96
Epoch 7/10
[0/1280 steps] - 1280/2s/step - loss: 1.8091 - accuracy: 0.1875 - F1Score: 0.0000e+00 - SpecificityAtSensitivity: 0.4827 - SensitivityAtSpecificity: 0.4925 - AUC: 0.5025 - val_loss: 1.96
Epoch 8/10
[0/1280 steps] - 1280/2s/step - loss: 1.8025 - accuracy: 0.1773 - F1Score: 0.0000e+00 - SpecificityAtSensitivity: 0.4015 - SensitivityAtSpecificity: 0.4750 - AUC: 0.5082 - val_loss: 1.96
Epoch 9/10
[0/1280 steps] - 1280/2s/step - loss: 1.8027 - accuracy: 0.1700 - F1Score: 0.0000e+00 - SpecificityAtSensitivity: 0.4012 - SensitivityAtSpecificity: 0.4840 - AUC: 0.5068 - val_loss: 1.96
Epoch 10/10
[0/1280 steps] - 1280/2s/step - loss: 1.8188 - accuracy: 0.1775 - F1Score: 0.0000e+00 - SpecificityAtSensitivity: 0.4545 - SensitivityAtSpecificity: 0.4942 - AUC: 0.5109 - val_loss: 1.96
```

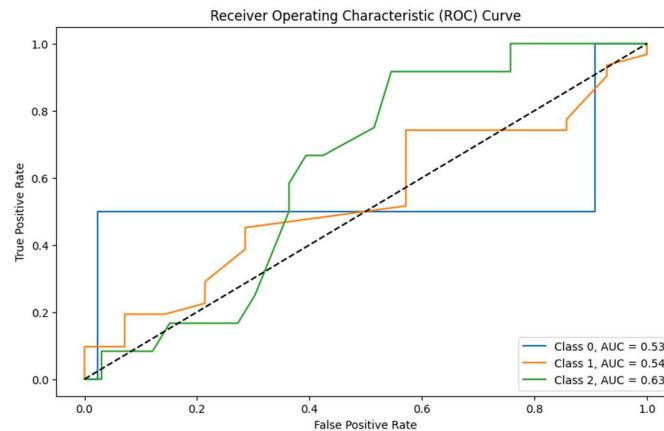
รูปที่ 3.94 ตัวอย่างผลลัพธ์ระหว่างการฝึกฝนโมเดล Retraction ด้วย ConvNext Tiny



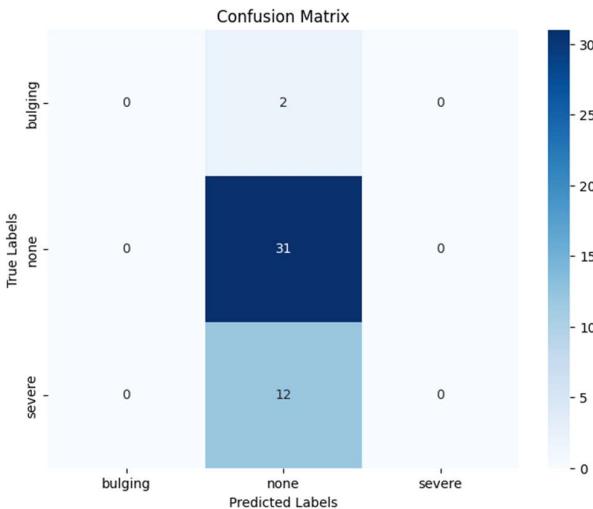
รูปที่ 3.95 ตัวอย่าง Accuracy ของโมเดล Retraction ที่ฝึกฝนโดยใช้ ConvNext Tiny



รูปที่ 3.96 ตัวอย่าง Loss ของโมเดล Retraction ที่ฝึกฝนโดยใช้ ConvNext Tiny



รูปที่ 3.97 ตัวอย่าง AUC ของโมเดล Retraction ที่ฝึกฝนโดยใช้ ConvNext Tiny



รูปที่ 3.98 ตัวอย่าง Confusion Matrix ของโมเดล Retraction ที่ฝึกฝนโดยใช้ ConvNext Tiny

3.2.2.3.2 การฝึกฝนโมเดลด้วย Traditional Machine Learning

■ SVM

ตัวอย่างการฝึกฝนโมเดลของ Retraction ด้วยสถาบัตยกรรม SVM โดยจะเรียกคำสั่ง `classifier.fit` เพื่อทำการฝึกฝนโมเดลดังแสดงดังรูปที่ 3.99 ส่วนในระหว่างการฝึกฝนโมเดลจะแสดงดังรูปที่ 3.100 และเมื่อสิ้นสุดการฝึกฝนจะแสดงผลลัพธ์ดังรูปที่ 3.101

```
# Train the classifier
classifier = SVC(kernel='linear', probability=True)
classifier.fit(X_pca, y_train)
```

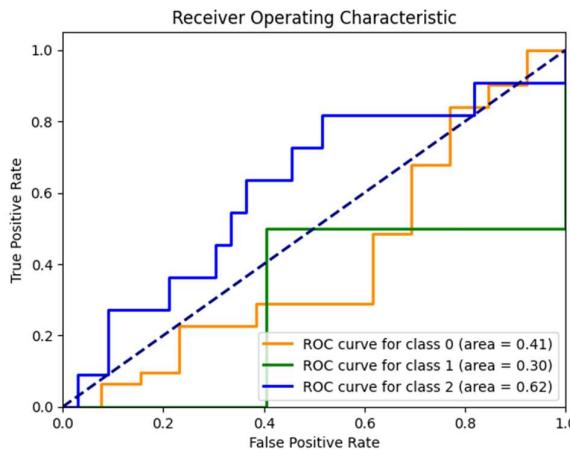
รูปที่ 3.99 ตัวอย่างคำสั่งการฝึกฝนโมเดล Retraction ด้วย SVM

```
Classification report for -
SVC(degree=2, kernel='poly', probability=True):
precision    recall   f1-score   support
          0       0.71      0.65      0.68      31
          1       0.00      0.00      0.00       2
          2       0.21      0.27      0.24      11

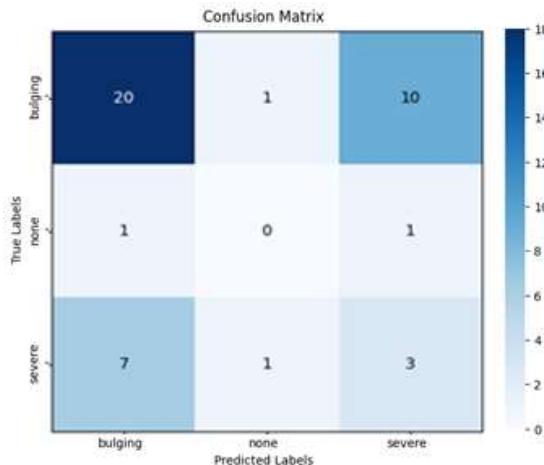
accuracy                           0.44
macro avg       0.31      0.31      0.31      44
weighted avg    0.56      0.52      0.54      44

specifity macro avg {.2f} 0.6678876678876678
specifity micro avg {.2f} 0.7613636363636364
specifity weighted avg {.2f} 0.48093573093573094
```

รูปที่ 3.100 ตัวอย่างผลลัพธ์โดยรวมของการฝึกฝนโมเดล Retraction ด้วย SVM



รูปที่ 3.10 ตัวอย่างผลลัพธ์ AUC ของการฝึกฝนโมเดล Retraction ด้วย SVM



รูปที่ 3.10 ตัวอย่าง Confusion Matrix ของการฝึกฝนโมเดล Retraction ด้วย SVM

■ KNN

ตัวอย่างการฝึกฝนโมเดลของ Retraction ด้วยสถาปัตยกรรม KNN โดยจะเรียกคำสั่ง `classifier.fit` เพื่อทำการฝึกฝนโมเดลโดยจะแสดงดังรูปที่ 3.10 ส่วนในระหว่างการฝึกฝนโมเดลจะแสดงดังรูปที่ 3.104 และเมื่อสิ้นสุดการฝึกฝนจะแสดงผลลัพธ์ดังรูปที่ 3.10

```
# Train the classifier
classifier = KNeighborsClassifier(n_neighbors=2)
classifier.fit(X_pca, y_train)
```

รูปที่ 3.103 ตัวอย่างคำสั่งการฝึกฝนโมเดล Retraction ด้วย KNN

```

Classification report for -
KNeighborsClassifier(n_neighbors=2):
      precision    recall  f1-score   support

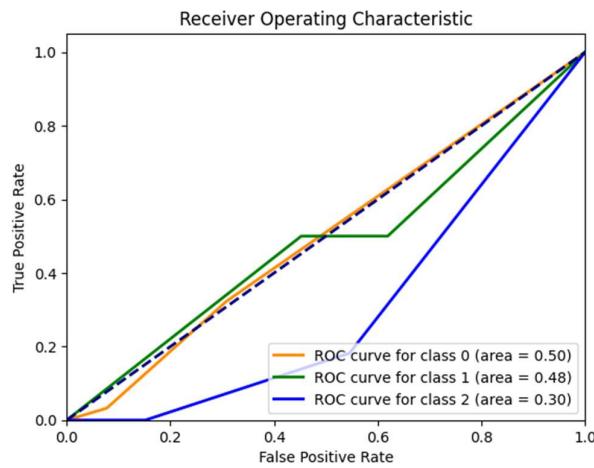
          0       0.71     0.32     0.44     31
          1       0.04     0.50     0.07      2
          2       0.00     0.00     0.00     11

   accuracy                           0.25     44
  macro avg       0.25     0.27     0.17     44
weighted avg       0.51     0.25     0.32     44

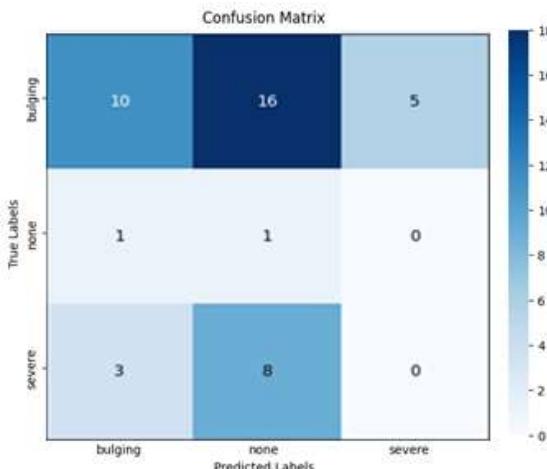
specificity macro avg {.2f} 0.6564546564546565
specificity micro avg {.2f} 0.625
specificity weighted avg {.2f} 0.7193639693639693

```

รูปที่ 3.104 ตัวอย่างผลลัพธ์โดยรวมของการฝึกฝนโมเดล Colot ด้วย KNN



รูปที่ 3.105 ตัวอย่าง AUC ของการฝึกฝนโมเดล Retraction ด้วย KNN



รูปที่ 3.106 ตัวอย่าง Confusion Matrix ของการฝึกฝนโมเดล Retraction ด้วย KNN

■ Decision Tree

ตัวอย่างการฝึกฝนโมเดลของ Retraction ด้วยสถาปัตยกรรม Decision Tree โดยจะเรียกคำสั่ง classifier.fit เพื่อทำการฝึกฝนโมเดลดังแสดงดังรูปที่ 3.107 ส่วนในระหว่างการฝึกฝนโมเดลจะแสดงดังรูปที่ 3.10 และเมื่อสิ้นสุดการฝึกฝนจะแสดงผลลัพธ์ดังรูปที่ 3.109

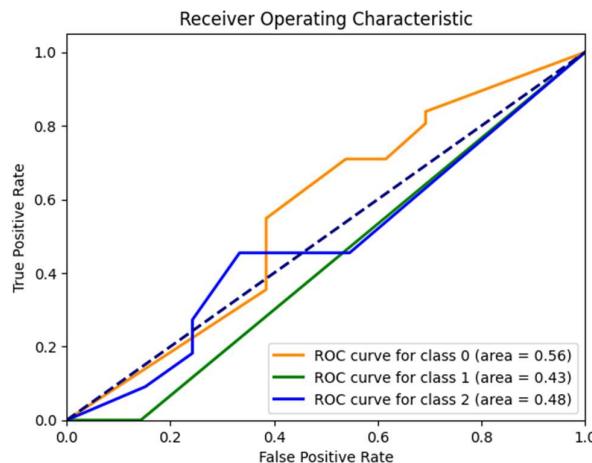
```
# Train the classifier
classifier = tree.DecisionTreeClassifier()
classifier.fit(X_pca, y_train)
```

รูปที่ 3.107 ตัวอย่างคำสั่งการฝึกฝนโมเดล Retraction ด้วย Decision Tree

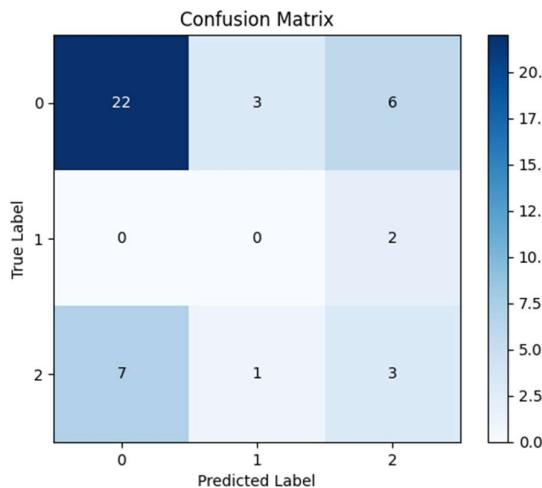
```
Classification report for -
DecisionTreeClassifier(max_depth=30, min_samples_leaf=6
precision    recall   f1-score   support
          0       0.76      0.71      0.73     31
          1       0.00      0.00      0.00      2
          2       0.27      0.27      0.27     11
accuracy                           0.57     44
macro avg       0.34      0.33      0.34     44
weighted avg    0.60      0.57      0.58     44

specifity macro avg {.2f} 0.7079587079587079
specifity micro avg {.2f} 0.7840909090909091
specifity weighted avg {.2f} 0.5556943056943058
```

รูปที่ 3.10 ตัวอย่างผลลัพธ์โดยรวมของการฝึกฝน Retraction ด้วย Decision Tree



รูปที่ 3.109 ตัวอย่าง AUC ของการฝึกฝนโมเดล Retraction ด้วย Decision Tree



รูปที่ 3.110 ตัวอย่าง Confusion Matrix ของการฝึกฝนโมเดล Retraction ด้วย Decision Tree

■ XGBoost

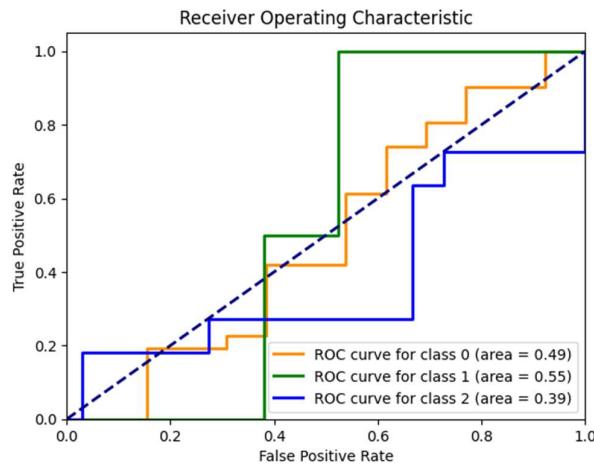
ตัวอย่างการฝึกฝนโมเดลของ Retraction ด้วยสถาปัตยกรรม Decision Tree โดยจะเรียกคำสั่ง classifier.fit เพื่อทำการฝึกฝนโมเดลดังแสดงดังรูปที่ 3.111 ส่วนในระหว่างการฝึกฝนโมเดลจะแสดงดังรูปที่ 3.112 และเมื่อสิ้นสุดการฝึกฝนจะแสดงผลลัพธ์ดังรูปที่ 3.113

```
# Train the classifier
classifier = XGBClassifier(
    n_estimators=50, # Number of boosting rounds or decision trees
    learning_rate=0.01, # Step size shrinkage
    max_depth=4, # Maximum depth of each decision tree
    subsample=1.0, # Fraction of samples used for training each tree
    colsample_bytree=1.0, # Fraction of features used for training each tree
    reg_alpha=0, # L1 regularization term
    reg_lambda=1, # L2 regularization term
    gamma=0.5, # Minimum loss reduction required to split a node
)
classifier.fit(X_pca, y_train)
```

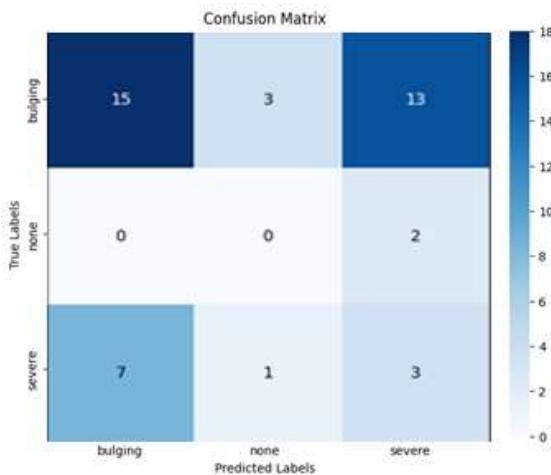
รูปที่ 3.111 ตัวอย่างคำสั่งการฝึกฝน Retraction ด้วย XGBoost

```
Classification report for :
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bytree=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=0.85, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.01, max_delta_step=None,
              max_leaves=7, max_leaves=None, max_depth=20, max_leaves=7,
              min_child_weight=None, missing='nan', monotone_constraints=None,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              objective='multi:softprob', predictor=None, ...):
precision    recall f1-score support
          0       0.68      0.48     0.57      31
          1       0.00      0.00     0.00       2
          2       0.17      0.27     0.21      11
accuracy                           0.41      44
macro avg       0.28      0.25     0.26      44
weighted avg    0.52      0.41     0.45      44
specificity macro avg (.2f) 0.6372516372516372
specificity micro avg (.2f) 0.7045454545454546
specificity weighted avg (.2f) 0.502654026640027
```

รูปที่ 3.112 ตัวอย่างผลลัพธ์โดยรวมของการฝึกฝน Retraction ด้วย XGBoost



รูปที่ 3.113 ตัวอย่าง AUC ของการฝึกฝนโมเดล Retraction ด้วย XGBoost



รูปที่ 3.114 ตัวอย่าง Confusion Matrix ของการฝึกฝนโมเดล Retraction ด้วย XGBoost

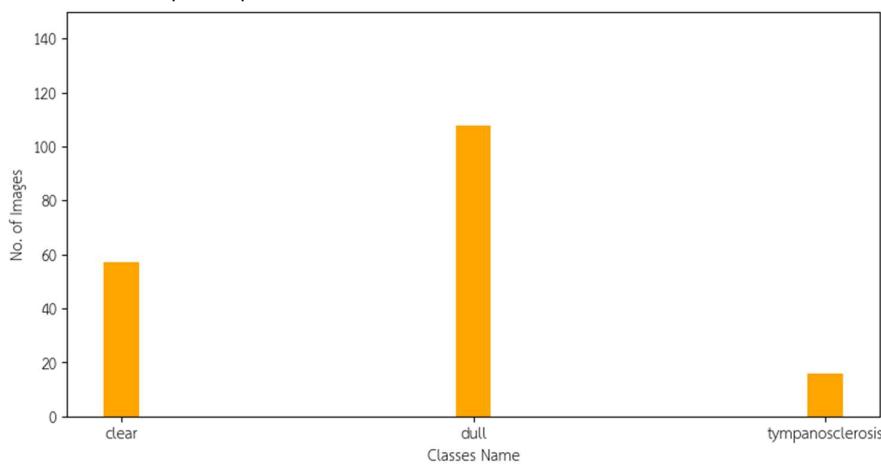
3.2.2.4 Transparency

สำหรับ Transparency จะประกอบไปด้วย 4 คุณลักษณะ ได้แก่ Clear, Dull, Opaque และ Tympanosclerosis โดยที่ Clear หมายถึงหูชั้นกลางที่มีลักษณะใส Dull หมายถึงหูชั้นกลางที่มีลักษณะขุ่น Opaque หมายถึงหูชั้นกลางที่มีลักษณะขุ่น และ Tympanosclerosis หมายถึงหูชั้นกลางที่มีคราบหินปูนเกาะ โดยจะแสดงดังรูปที่ 3.115



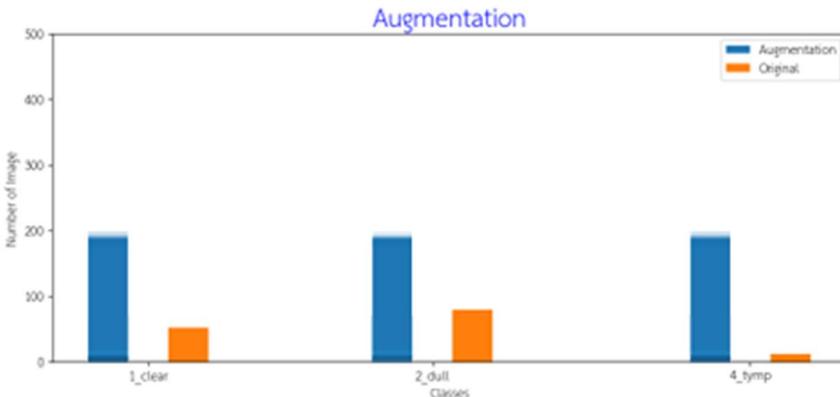
รูปที่ 3.115 ตัวอย่างคุณลักษณะ Clear, Dull, Opaque และ Tympanosclerosis

จากรูปที่ 3.116 จะเห็นว่า Transparency เป็น Imbalanced Class ซึ่งจำเป็นต้องมีการเพิ่มข้อมูลของคลาสให้สมดุลกันทุกคลาส



รูปที่ 3.116 จำนวนข้อมูลในแต่ละ Class ของ Transparency

เพิ่มตัวอย่างของ class ให้สมดุลกันด้วยวิธี Augmentation ด้วยการปรับความสว่างและหมุนภาพ



รูปที่ 3.117 จำนวนข้อมูลในแต่ละ Class ของ Transparency หลังทำการ Augmentation

3.2.2.4.1 การฝึกฝนโมเดลด้วย Neural Networks

■ Xception

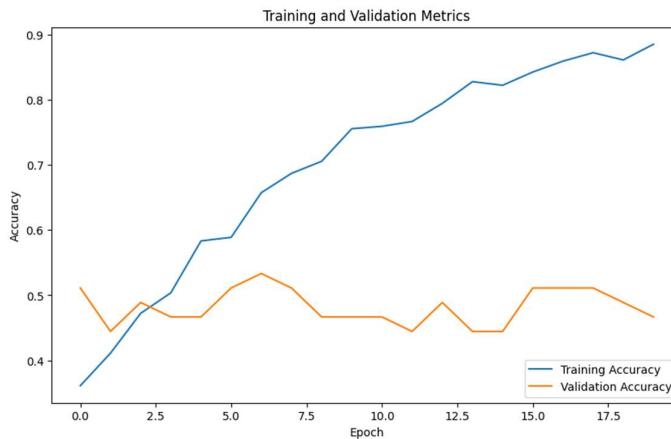
ตัวอย่างการฝึกฝนโมเดลของ Transparency ด้วยสถาปัตยกรรม Xception โดยจะเรียกคำสั่ง `model.fit` เพื่อทำการฝึกฝนโมเดลดังรูปที่ 3.118 ส่วนในระหว่างการฝึกฝนโมเดลจะแสดงดังรูปที่ 3.119 และเมื่อสิ้นสุดการฝึกฝนจะแสดงผลลัพธ์ดังรูปที่ 3.120

```
# Train the model
history = model.fit(train_ds, validation_data=val_ds, epochs=10)
```

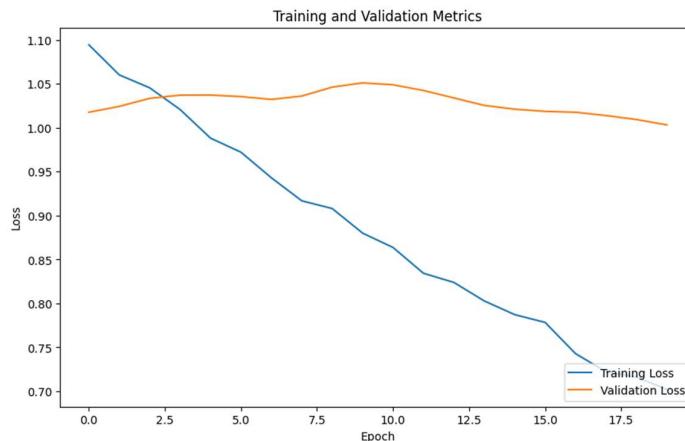
รูปที่ 3.118 ตัวอย่างคำสั่งการฝึกฝนโมเดล Transparency ด้วย Xception

```
Epoch 1/10
[0/1000000000 steps] - 100s/1s/step - loss: 146.1381 - accuracy: 0.1900 - F1Score: 0.0000+0.000 - Specificity@Specificity: 0.4465 - Sensitivity@Specificity: 0.4979 - AUC: 0.5000 - val_loss: 146.1381 - val_accuracy: 0.1900 - val_F1Score: 0.0000+0.000 - val_Specificity@Specificity: 0.4465 - val_Sensitivity@Specificity: 0.4979 - val_AUC: 0.5000
Epoch 2/10
[0/1000000000 steps] - 100s/1s/step - loss: 106.199381 - accuracy: 0.2058 - F1Score: 0.0000+0.000 - Specificity@Specificity: 0.5007 - Sensitivity@Specificity: 0.5545 - AUC: 0.5000 - val_loss: 106.199381 - val_accuracy: 0.2058 - val_F1Score: 0.0000+0.000 - val_Specificity@Specificity: 0.5007 - val_Sensitivity@Specificity: 0.5545 - val_AUC: 0.5000
Epoch 3/10
[0/1000000000 steps] - 100s/1s/step - loss: 106.159599 - accuracy: 0.2042 - F1Score: 0.0000+0.000 - Specificity@Specificity: 0.4484 - Sensitivity@Specificity: 0.4917 - AUC: 0.5010 - val_loss: 106.159599 - val_accuracy: 0.2042 - val_F1Score: 0.0000+0.000 - val_Specificity@Specificity: 0.4484 - val_Sensitivity@Specificity: 0.4917 - val_AUC: 0.5010
Epoch 4/10
[0/1000000000 steps] - 100s/1s/step - loss: 109.3423 - accuracy: 0.2142 - F1Score: 0.0000+0.000 - Specificity@Specificity: 0.5027 - Sensitivity@Specificity: 0.5409 - AUC: 0.5047 - val_loss: 109.3423 - val_accuracy: 0.2142 - val_F1Score: 0.0000+0.000 - val_Specificity@Specificity: 0.5027 - val_Sensitivity@Specificity: 0.5409 - val_AUC: 0.5047
Epoch 5/10
[0/1000000000 steps] - 100s/1s/step - loss: 109.0683 - accuracy: 0.2142 - F1Score: 0.0000+0.000 - Specificity@Specificity: 0.5027 - Sensitivity@Specificity: 0.5409 - AUC: 0.5047 - val_loss: 109.0683 - val_accuracy: 0.2142 - val_F1Score: 0.0000+0.000 - val_Specificity@Specificity: 0.5027 - val_Sensitivity@Specificity: 0.5409 - val_AUC: 0.5047
Epoch 6/10
[0/1000000000 steps] - 100s/1s/step - loss: 108.8434 - accuracy: 0.2153 - F1Score: 0.0000+0.000 - Specificity@Specificity: 0.5038 - Sensitivity@Specificity: 0.5420 - AUC: 0.5058 - val_loss: 108.8434 - val_accuracy: 0.2153 - val_F1Score: 0.0000+0.000 - val_Specificity@Specificity: 0.5038 - val_Sensitivity@Specificity: 0.5420 - val_AUC: 0.5058
Epoch 7/10
[0/1000000000 steps] - 100s/1s/step - loss: 108.7374 - accuracy: 0.2158 - F1Score: 0.0000+0.000 - Specificity@Specificity: 0.5043 - Sensitivity@Specificity: 0.5425 - AUC: 0.5063 - val_loss: 108.7374 - val_accuracy: 0.2158 - val_F1Score: 0.0000+0.000 - val_Specificity@Specificity: 0.5043 - val_Sensitivity@Specificity: 0.5425 - val_AUC: 0.5063
Epoch 8/10
[0/1000000000 steps] - 100s/1s/step - loss: 108.7338 - accuracy: 0.2158 - F1Score: 0.0000+0.000 - Specificity@Specificity: 0.5043 - Sensitivity@Specificity: 0.5425 - AUC: 0.5063 - val_loss: 108.7338 - val_accuracy: 0.2158 - val_F1Score: 0.0000+0.000 - val_Specificity@Specificity: 0.5043 - val_Sensitivity@Specificity: 0.5425 - val_AUC: 0.5063
Epoch 9/10
[0/1000000000 steps] - 100s/1s/step - loss: 108.6973 - accuracy: 0.2172 - F1Score: 0.0000+0.000 - Specificity@Specificity: 0.5058 - Sensitivity@Specificity: 0.5438 - AUC: 0.5078 - val_loss: 108.6973 - val_accuracy: 0.2172 - val_F1Score: 0.0000+0.000 - val_Specificity@Specificity: 0.5058 - val_Sensitivity@Specificity: 0.5438 - val_AUC: 0.5078
Epoch 10/10
[0/1000000000 steps] - 100s/1s/step - loss: 108.6729 - accuracy: 0.2180 - F1Score: 0.0000+0.000 - Specificity@Specificity: 0.5063 - Sensitivity@Specificity: 0.5445 - AUC: 0.5084 - val_loss: 108.6729 - val_accuracy: 0.2180 - val_F1Score: 0.0000+0.000 - val_Specificity@Specificity: 0.5063 - val_Sensitivity@Specificity: 0.5445 - val_AUC: 0.5084
```

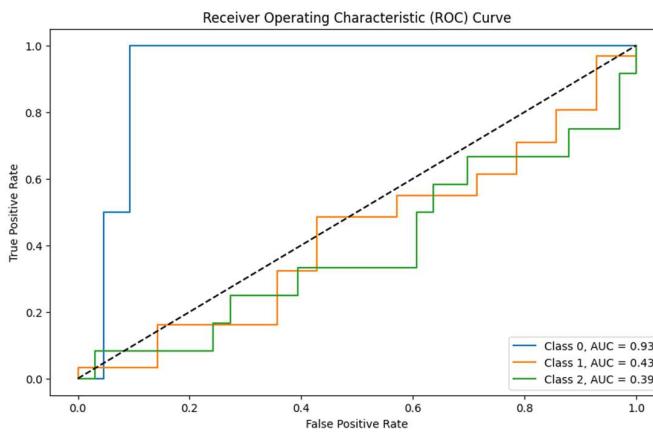
รูปที่ 3.119 ตัวอย่างผลลัพธ์ระหว่างการฝึกฝนโมเดล Transparency ด้วย Xception



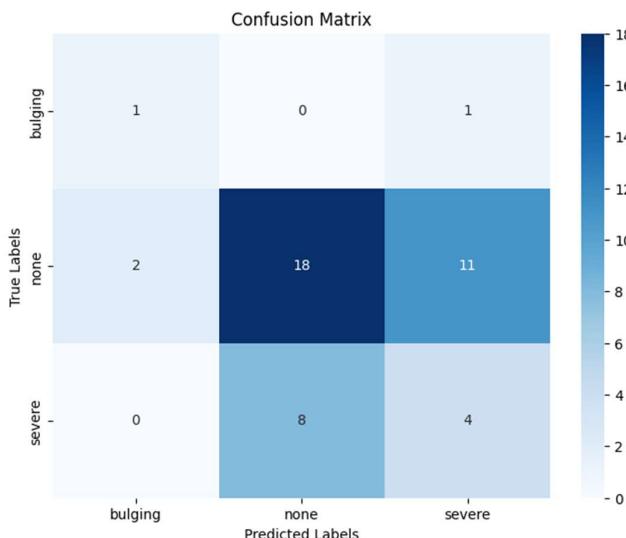
รูปที่ 3.120 ตัวอย่าง Accuracy ของโมเดล Transparency ที่ฝึกฝนโดยใช้ Xception



รูปที่ 3.121 ตัวอย่าง Loss ของโมเดล Transparency ที่ฝึกฝนโดยใช้ Xception



รูปที่ 3.122 ตัวอย่าง AUC ของการฝึกฝน Transparency ด้วย Xception



รูปที่ 3.123 ตัวอย่าง Confusion Matrix ของการฝึกฝน Transparency ด้วย Xception

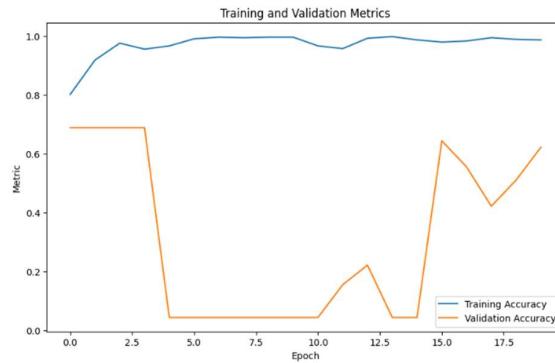
■ MobileNet V2

ตัวอย่างการฝึกฝนโมเดลของ Transparency ด้วยสถาปัตยกรรม MobileNet V2 โดยจะเรียกคำสั่ง `model.fit` เพื่อทำการฝึกฝนโมเดลดังรูปที่ 3.124 ส่วนในระหว่าง การฝึกฝนโมเดลจะแสดงดังรูปที่ 3.125 และเมื่อสิ้นสุดการฝึกฝนจะแสดงผลลัพธ์ดังรูปที่ 3.126

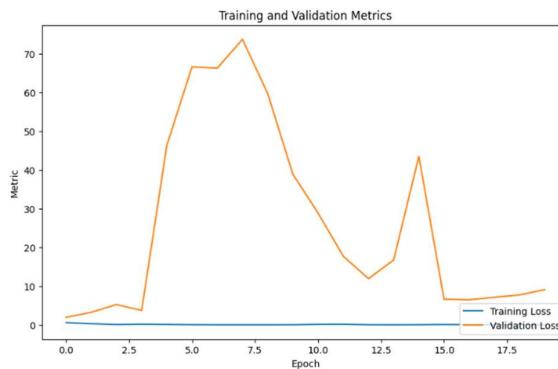
```
# Train the model  
history = model.fit(train_ds, validation_data=val_ds, epochs=10)
```

รูปที่ 3.124 ตัวอย่างคำสั่งการฝึกฝนโมเดล Transparency ด้วย MobileNet V2

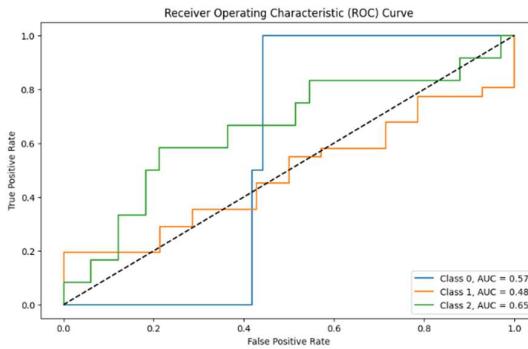
รูปที่ 3.125 ตัวอย่างผลลัพธ์ระหว่างการฝึกฝนโมเดล Transparency ด้วย MobileNet V2



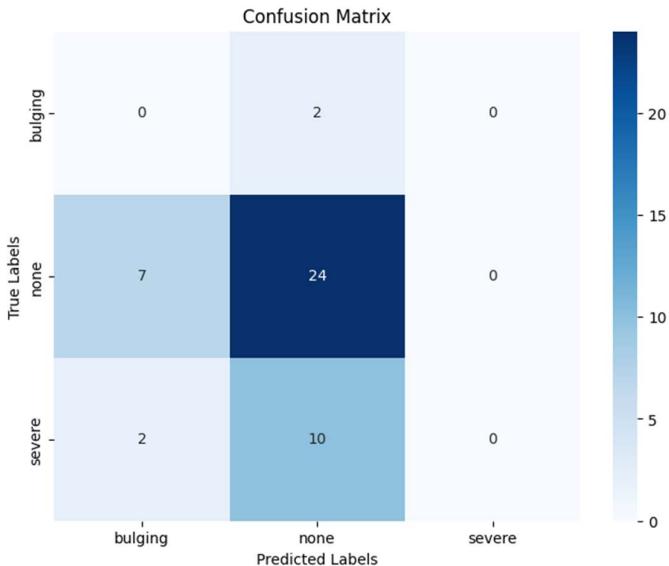
รูปที่ 3.126 ตัวอย่าง Accuracy ของโมเดล Transparency ที่ผึ้งฝนโดยใช้ MobileNet V2



รูปที่ 3.127 ตัวอย่าง Loss ของโมเดล Transparency ที่ฝึกฝนโดยใช้ MobileNet V2



รูปที่ 3.128 ตัวอย่าง AUC ของโมเดล Transparency ที่ฝึกฝนโดยใช้ MobileNet V2



รูปที่ 3.129 ตัวอย่าง Confusion Matrix ของโมเดล Transparency ที่ฝึกฝนโดยใช้ MobileNet V2

▪ ConvNext Tiny

ตัวอย่างการฝึกฝนโมเดลของ Transparency ด้วยสถาปัตยกรรม ConvNext Tiny โดยจะเรียกคำสั่ง `model.fit` เพื่อทำการฝึกฝนโมเดลดังแสดงดังรูปที่ 3.130 ส่วนในระหว่างการฝึกฝนโมเดลจะแสดงดังรูปที่ 3.131 และเมื่อสิ้นสุดการฝึกฝนจะแสดงผลลัพธ์ดังรูปที่ 3.132

```
# Train the model
history = model.fit(train_ds, validation_data=val_ds, epochs=10)
```

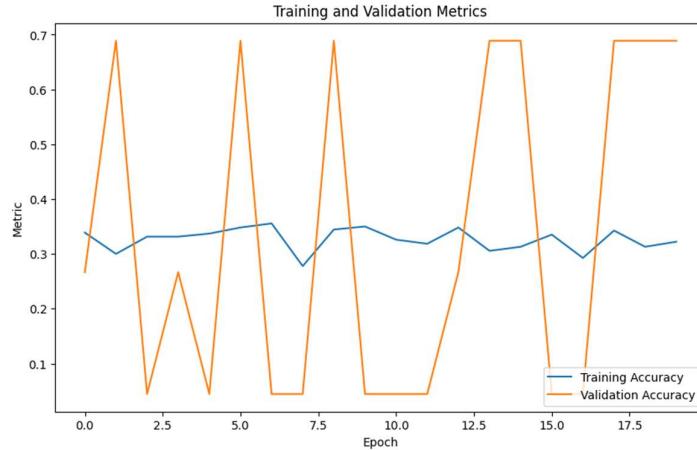
รูปที่ 3.130 ตัวอย่างคำสั่งการฝึกฝนโมเดล Transparency ด้วย ConvNext Tiny

```

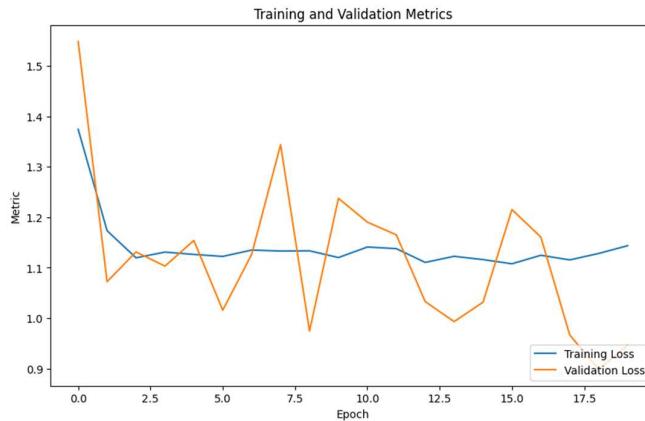
Epoch 1/10
Epoch 2/10 [=====] - 128s 2s/step - loss: 1.0887 - accuracy: 0.4625 - SpecificityAtSensitivity: 0.4958 - SensitivityAtSpecificity: 0.4583 - AUC: 0.4991 - val_loss: 1.86
Epoch 3/10 [=====] - 58s 1s/step - loss: 1.0354 - accuracy: 0.1817 - F1Score: 0.0000e+00 - SpecificityAtSensitivity: 0.4608 - SensitivityAtSpecificity: 0.4983 - AUC: 0.5064 - val_loss: 1.
Epoch 4/10 [=====] - 57s 1s/step - loss: 1.0512 - accuracy: 0.2700 - F1Score: 0.0000e+00 - SpecificityAtSensitivity: 0.5037 - SensitivityAtSpecificity: 0.5153 - AUC: 0.5123 - val_loss: 1.
Epoch 5/10 [=====] - 56s 1s/step - loss: 1.0370 - accuracy: 0.2700 - F1Score: 0.0000e+00 - SpecificityAtSensitivity: 0.4925 - SensitivityAtSpecificity: 0.4992 - AUC: 0.4979 - val_loss: 1.
Epoch 6/10 [=====] - 56s 1s/step - loss: 1.0297 - accuracy: 0.2700 - F1Score: 0.0000e+00 - SpecificityAtSensitivity: 0.4917 - SensitivityAtSpecificity: 0.4942 - AUC: 0.4934 - val_loss: 1.
Epoch 7/10 [=====] - 56s 1s/step - loss: 1.0316 - accuracy: 0.2700 - F1Score: 0.0000e+00 - SpecificityAtSensitivity: 0.4905 - SensitivityAtSpecificity: 0.4967 - AUC: 0.4959 - val_loss: 1.
Epoch 8/10 [=====] - 56s 1s/step - loss: 1.0356 - accuracy: 0.2700 - F1Score: 0.0000e+00 - SpecificityAtSensitivity: 0.4893 - SensitivityAtSpecificity: 0.4978 - AUC: 0.5023 - val_loss: 1.
Epoch 9/10 [=====] - 56s 1s/step - loss: 1.0215 - accuracy: 0.2725 - F1Score: 0.0000e+00 - SpecificityAtSensitivity: 0.4918 - SensitivityAtSpecificity: 0.4750 - AUC: 0.5023 - val_loss: 1.
Epoch 10/10 [=====] - 55s 1s/step - loss: 1.0257 - accuracy: 0.2700 - F1Score: 0.0000e+00 - SpecificityAtSensitivity: 0.4932 - SensitivityAtSpecificity: 0.4902 - AUC: 0.5000 - val_loss: 1.
Epoch 11/10 [=====] - 55s 1s/step - loss: 1.0188 - accuracy: 0.1775 - F1Score: 0.0000e+00 - SpecificityAtSensitivity: 0.4545 - SensitivityAtSpecificity: 0.4942 - AUC: 0.5109 - val_loss: 1.

```

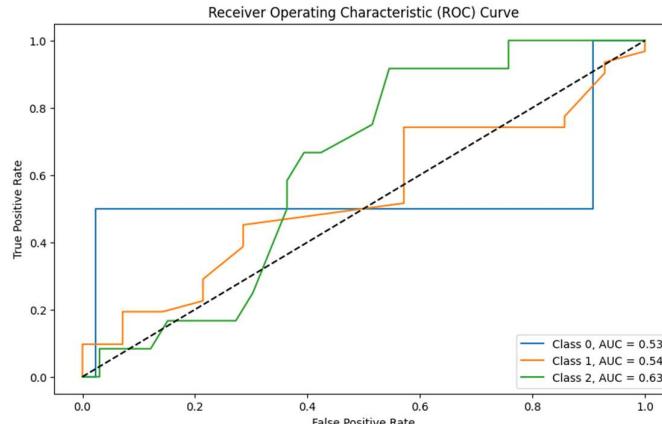
รูปที่ 3.131 ตัวอย่างผลลัพธ์ระหว่างการฝึกฝนโมเดล Transparency ด้วย ConvNext Tiny



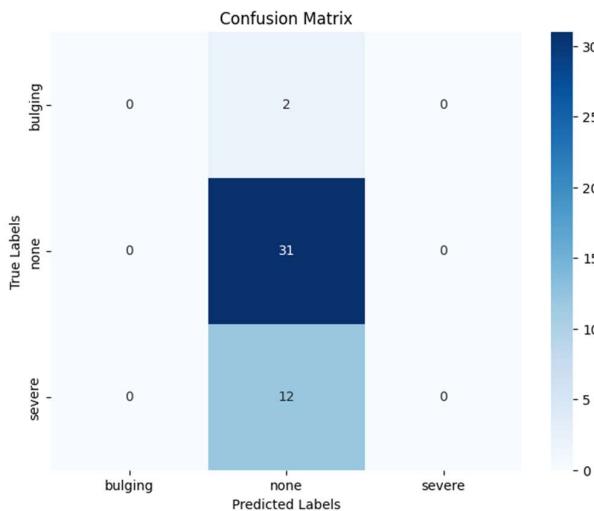
รูปที่ 3.132 ตัวอย่าง Accuracy ของโมเดล Transparency ที่ฝึกฝนโดยใช้ ConvNext Tiny



รูปที่ 3.133 ตัวอย่าง Loss ของโมเดล Transparency ที่ฝึกฝนโดยใช้ ConvNext Tiny



รูปที่ 3.134 ตัวอย่าง AUC ของโมเดล Transparency ที่ฝึกฝนโดยใช้ ConvNext Tiny



รูปที่ 3.135 ตัวอย่าง Confusion Matrix ของโมเดล Transparency ที่ฝึกฝนโดยใช้ ConvNext Tiny

3.2.2.4.2 การฝึกฝนโมเดลด้วย Traditional Machine Learning

■ SVM

ตัวอย่างการฝึกฝนโมเดลของ Transparency ด้วยสถาปัตยกรรม SVM โดยจะเรียกคำสั่ง classifier.fit เพื่อทำการฝึกฝนโมเดลดังแสดงดังรูปที่ 3.136 ส่วนในระหว่างการฝึกฝนโมเดลจะแสดงดังรูปที่ 3.137 และเมื่อสิ้นสุดการฝึกฝนจะแสดงผลลัพธ์ดังรูปที่ 3.138

```
# Train the classifier
classifier = SVC(kernel='linear', probability=True)
classifier.fit(X_pca, y_train)
```

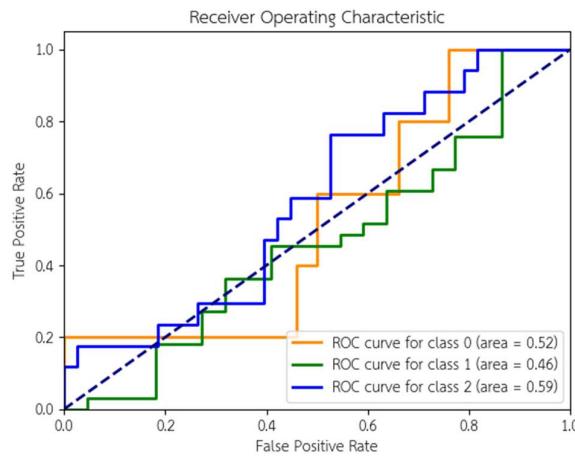
รูปที่ 3.136 ตัวอย่างคำสั่งการฝึกฝนโมเดล Transparency ด้วย SVM

```
Classification report for -
SVC(degree=2, kernel='poly', probability=True):
precision    recall   f1-score   support
          0       1.00     0.20     0.33      5
          1       0.50     0.06     0.11     33
          2       0.32     0.94     0.48     17

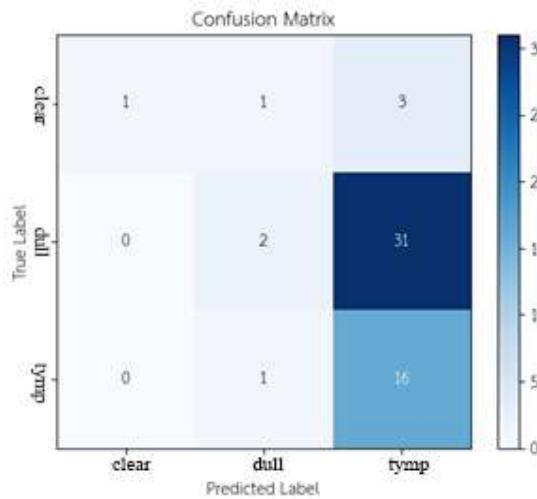
accuracy                           0.35      55
macro avg       0.61     0.40     0.31      55
weighted avg    0.49     0.35     0.24      55

specificity macro avg {.2f} 0.671451355661882
specificity micro avg {.2f} 0.6727272727272727
specificity weighted avg {.2f} 0.6688995215311005
```

รูปที่ 3.137 ตัวอย่างผลลัพธ์โดยรวมของการฝึกฝนโมเดล Transparency ด้วย SVM



รูปที่ 3.138 ตัวอย่างผลลัพธ์ AUC ของการฝึกฝนโมเดล Transparency ด้วย SVM



รูปที่ 3.139 ตัวอย่าง Confusion Matrix ของการฝึกฝนโมเดล Transparency ด้วย SVM

■ KNN

ตัวอย่างการฝึกฝนโมเดลของ Transparency ด้วยสถาปัตยกรรม KNN โดยจะเรียกคำสั่ง `classifier.fit` เพื่อทำการฝึกฝนโมเดลดังแสดงดังรูปที่ 3.140 ส่วนในระหว่างการฝึกฝนโมเดลจะแสดงดังรูปที่ 3.141 และเมื่อสิ้นสุดการฝึกฝนจะแสดงผลลัพธ์ดังรูปที่ 3.142

```
# Train the classifier
classifier = KNeighborsClassifier(n_neighbors=2)
classifier.fit(X_pca, y_train)
```

รูปที่ 3.140 ตัวอย่างคำสั่งการฝึกฝนโมเดล Transparency ด้วย KNN

```

Classification report for -
KNeighborsClassifier(n_neighbors=2):
      precision    recall  f1-score   support

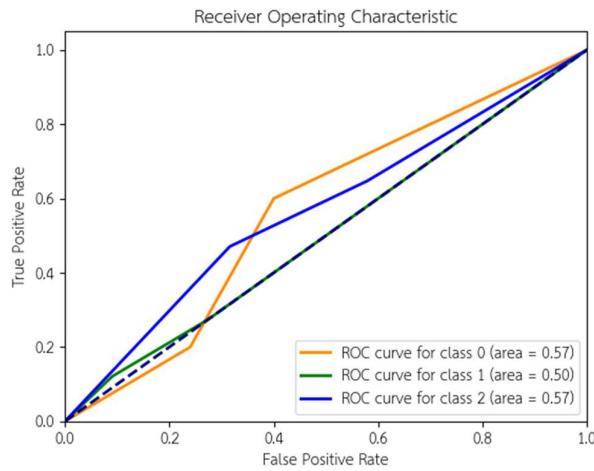
          0       0.13     0.60    0.21      5
          1       0.67     0.24    0.36     33
          2       0.40     0.47    0.43     17

   accuracy                           0.35      55
  macro avg       0.40     0.44    0.33      55
weighted avg       0.54     0.35    0.37      55

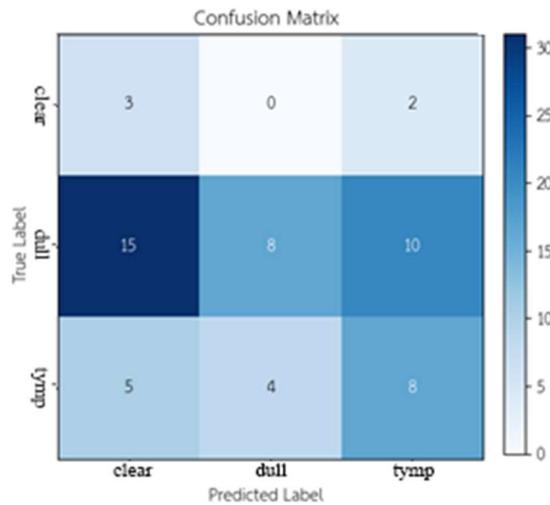
specifity macro avg {.2f} 0.7007974481658693
specifity micro avg {.2f} 0.6727272727272727
specifity weighted avg {.2f} 0.7569377990430622

```

รูปที่ 3.141 ตัวอย่างผลลัพธ์โดยรวมของการฝึกฝนโมเดล Colot ด้วย KNN



รูปที่ 3.142 ตัวอย่าง AUC ของการฝึกฝนโมเดล Transparency ด้วย KNN



รูปที่ 3.143 ตัวอย่าง Confusion Matrix ของการฝึกฝนโมเดล Transparency ด้วย KNN

■ Decision Tree

ตัวอย่างการฝึกฝนโมเดลของ Transparency ด้วยสถาปัตยกรรม Decision Tree โดยจะเรียกคำสั่ง classifier.fit เพื่อทำการฝึกฝนโมเดลดังแสดงดังรูปที่ 3.144 ส่วนในระหว่างการฝึกฝนโมเดลจะแสดงดังรูปที่ 3.145 และเมื่อสิ้นสุดการฝึกฝนจะแสดงผลลัพธ์ดังรูปที่ 3.146

```
# Train the classifier
classifier = tree.DecisionTreeClassifier()
classifier.fit(X_pca, y_train)
```

รูปที่ 3.144 ตัวอย่างคำสั่งการฝึกฝนโมเดล Transparency ด้วย Decision Tree

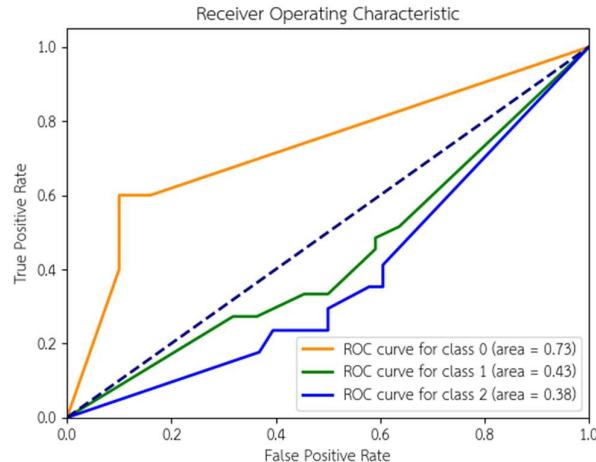
```
Classification report for -
DecisionTreeClassifier(max_depth=26, min_samples_leaf=4):
              precision    recall  f1-score   support

             0       0.33    0.60    0.43     5
             1       0.56    0.45    0.50    33
             2       0.21    0.24    0.22    17

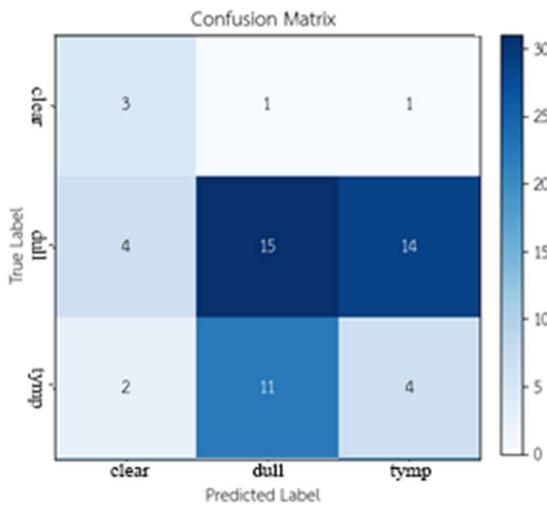
      accuracy                           0.40    55
   macro avg       0.37    0.43    0.38    55
weighted avg       0.43    0.40    0.41    55

specifity macro avg {.2f} 0.6466028708133971
specifity micro avg {.2f} 0.7
specifity weighted avg {.2f} 0.5398086124401914
```

รูปที่ 3.145 ตัวอย่างผลลัพธ์โดยรวมของการฝึกฝน Transparency ด้วย Decision Tree



รูปที่ 3.146 ตัวอย่าง AUC ของการฝึกฝนโมเดล Transparency ด้วย Decision Tree



รูปที่ 3.147 ตัวอย่าง Confusion Matrix ของการฝึกฝนโมเดล Transparency ด้วย Decision Tree

■ XGBoost

ตัวอย่างการฝึกฝนโมเดลของ Transparency ด้วยสถาปัตยกรรม Decision Tree โดยจะเรียกคำสั่ง classifier.fit เพื่อทำการฝึกฝนโมเดลดังแสดงดังรูปที่ 3.148 ส่วนในระหว่างการฝึกฝนโมเดลจะแสดงดังรูปที่ 3.149 และเมื่อสิ้นสุดการฝึกฝนจะแสดงผลลัพธ์ดังรูปที่ 3.150

```
# Train the classifier
classifier = XGBClassifier(
    n_estimators=50, # Number of boosting rounds or decision trees
    learning_rate=0.01, # Step size shrinkage
    max_depth=4, # Maximum depth of each decision tree
    subsample=1.0, # Fraction of samples used for training each tree
    colsample_bytree=1.0, # Fraction of features used for training each tree
    reg_alpha=0, # L1 regularization term
    reg_lambda=1, # L2 regularization term
    gamma=0.5, # Minimum loss reduction required to split a node
)
classifier.fit(X_pca, y_train)
```

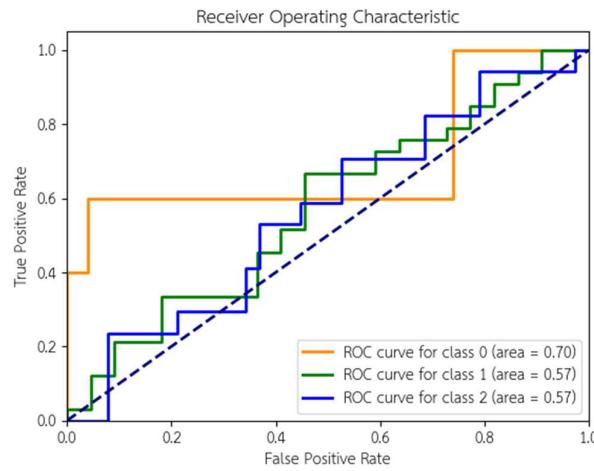
รูปที่ 3.148 ตัวอย่างคำสั่งการฝึกฝน Transparency ด้วย XGBoost

```

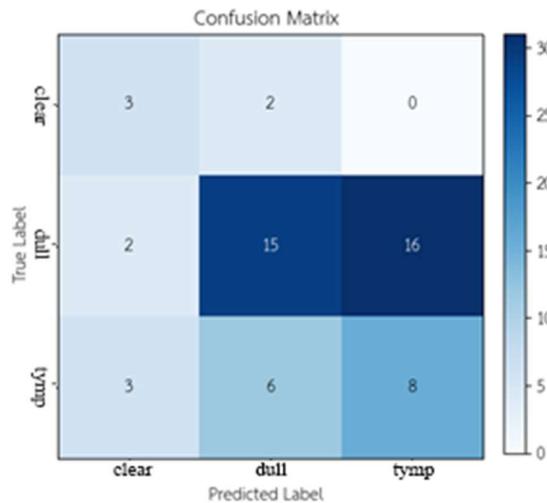
Classification report for -
XGBClassifier(base_score=None, booster=None, callbacks=None,
             colsample_bytree=None, colsample_bynode=None,
             colsample_bylevel=None, enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=0.25, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.01, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=5, max_leaves=3,
             min_child_weight=None, missing='nan', monotone_constraints=None,
             n_estimators=10, n_jobs=None, num_parallel_tree=None,
             objective='multi:softmax', predictor=None, ...):
precision    recall f1-score   support
          0       0.38      0.60      0.46      5
          1       0.65      0.45      0.54     33
          2       0.33      0.47      0.39     55
   accuracy                           0.47      55
  macro avg       0.45      0.51      0.46      55
weighted avg       0.53      0.47      0.48      55
specifity macro avg (.2f) 0.7051036682615631
specifity micro avg (.2f) 0.7363636363636363
specifity weighted avg (.2f) 0.6425837320574164

```

รูปที่ 3.149 ตัวอย่างผลลัพธ์โดยรวมของการฝึกฝน Transparency ด้วย XGBoost



รูปที่ 3.150 ตัวอย่าง AUC ของการฝึกฝนโมเดล Transparency ด้วย XGBoost



รูปที่ 3.151 ตัวอย่าง Confusion Matrix ของการฝึกฝนโมเดล Transparency ด้วย XGBoost

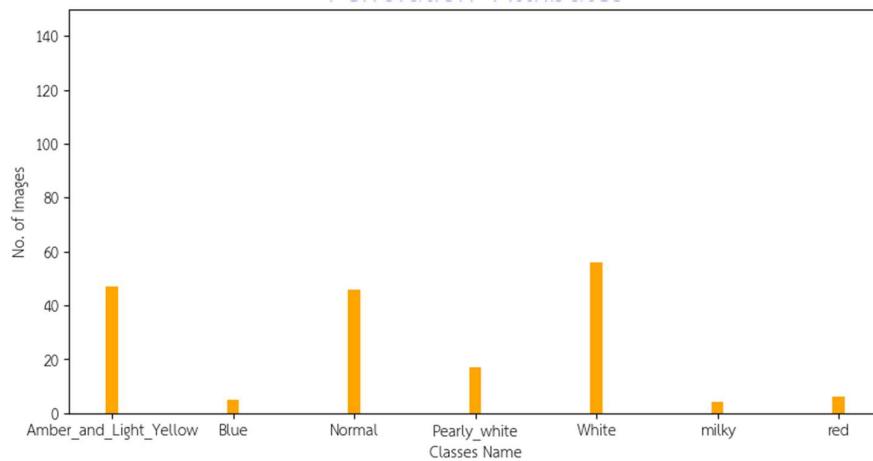
3.2.2.5 Color

สำหรับ Color จะประกอบไปด้วย 8 คุณลักษณะ ได้แก่ Normal, White, Amber, Blue, Pearly White, Milky, Light Yellow และ Red โดยที่ Normal หมายถึงหูชั้นกลางที่มีลักษณะสีเทา White หมายถึงหูชั้นกลางที่มีลักษณะสีขาว Amber หมายถึงหูชั้นกลางที่มีลักษณะสีเหลืองอันพัน Blue หมายถึงหูชั้นกลางที่มีลักษณะสีเหลืองน้ำเงินหรือสีเข้มของหูชั้นกลาง Pearly White หมายถึงหูชั้นกลางที่มีลักษณะสีขาวไข่มุกซึ่งเป็นลักษณะของหนองในหูชั้นกลาง Milky หมายถึงหูชั้นกลางที่มีลักษณะสีขาวขุ่นซึ่งเป็นลักษณะของหนองในหูชั้นกลาง Light Yellow หมายถึงหูชั้นกลางที่มีลักษณะสีเหลืองอ่อน และ Red หมายถึงหูชั้นกลางที่มีสีแดง โดยจะแสดงดังรูปที่ 3.152



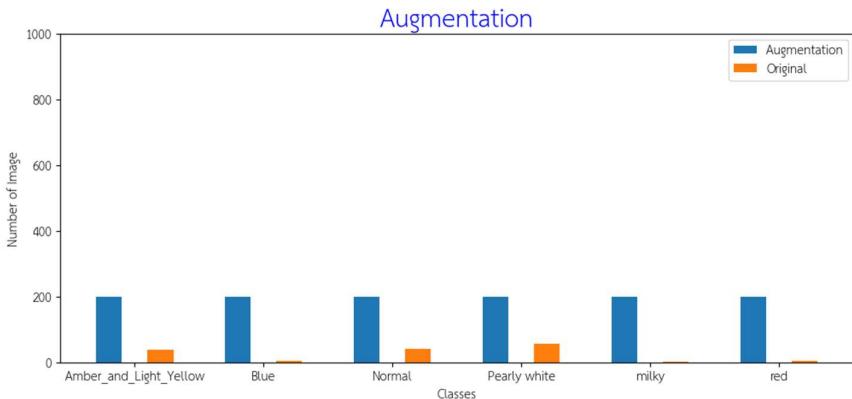
รูปที่ 3.152 ตัวอย่างคุณลักษณะ Normal, White, Amber, Blue, Pearly White, Milky, Light Yellow และ Red

จากรูปที่ 3.153 จะเห็นว่า Color เป็น Imbalanced Class ซึ่งจำเป็นต้องมีการเพิ่มข้อมูลของคลาสให้สมดุลกันทุกคลาส



รูปที่ 3.153 จำนวนข้อมูลในแต่ละ Class ของ Color

เพิ่มตัวอย่างของ Class ให้สมดุลกันด้วยวิธี Augmentation ด้วยการปรับความสว่างและหมุนภาพ



รูปที่ 3.154 จำนวนข้อมูลในแต่ละ Class ของ Color หลังทำการ Augmentation

3.2.2.5.1 การฝึกฟันโน้มเดลด้วย Neural Networks

▪ Xception

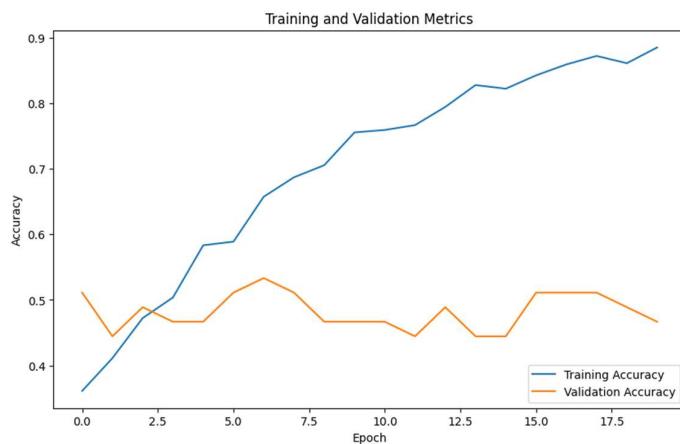
ตัวอย่างการฝึกฟันโน้มเดลของ Color ด้วยสถาปัตยกรรม Xception โดยจะเรียกคำสั่ง `model.fit` เพื่อทำการฝึกฟันโน้มเดลดังแสดงดังรูปที่ 3.155 ส่วนในระหว่างการฝึกฟันโน้มเดลจะแสดงดังรูปที่ 3.156 และเมื่อสิ้นสุดการฝึกฟันจะแสดงผลลัพธ์ดังรูปที่ 3.157

```
# Train the model
history = model.fit(train_ds, validation_data=val_ds, epochs=10)
```

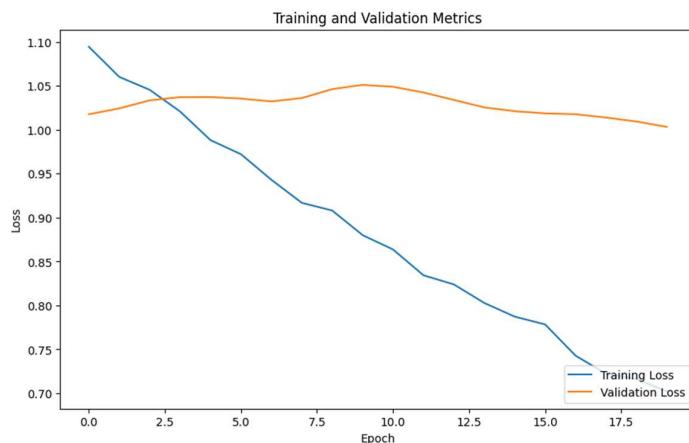
รูปที่ 3.155 ตัวอย่างคำสั่งการฝึกฟันโน้มเดล Color ด้วย Xception

```
Epoch 1/10
38/38 [=====] - 102s 1s/step - loss: 140.1381 - accuracy: 0.1908 - F1Score: 0.0000e+00 - SpecificityAtSensitivity: 0.4975 - SensitivityAtSpecificity: 0.4975 - AUC: 0.5083 - val_loss: 139.0000e+00 - SpecificityAtSensitivity: 0.5000 - SensitivityAtSpecificity: 0.5000 - AUC: 0.5000 - val_accuracy: 0.1908 - val_F1Score: 0.0000e+00 - val_SpecificityAtSensitivity: 0.5000 - val_SensitivityAtSpecificity: 0.5000 - val_AUC: 0.5000
Epoch 2/10
38/38 [=====] - 96s 1s/step - loss: 139.8581 - accuracy: 0.2658 - F1Score: 0.0000e+00 - SpecificityAtSensitivity: 0.5987 - SensitivityAtSpecificity: 0.5548 - AUC: 0.5983 - val_loss: 139.0000e+00 - SpecificityAtSensitivity: 0.6000 - SensitivityAtSpecificity: 0.5548 - AUC: 0.5983 - val_accuracy: 0.2658 - val_F1Score: 0.0000e+00 - val_SpecificityAtSensitivity: 0.6000 - val_SensitivityAtSpecificity: 0.5548 - val_AUC: 0.5983
Epoch 3/10
38/38 [=====] - 62s 1s/step - loss: 139.5990 - accuracy: 0.3042 - F1Score: 0.0189 - SpecificityAtSensitivity: 0.6162 - SensitivityAtSpecificity: 0.6117 - AUC: 0.6138 - val_loss: 139.0000e+00 - SpecificityAtSensitivity: 0.6180 - SensitivityAtSpecificity: 0.6117 - AUC: 0.6147 - val_accuracy: 0.3042 - val_F1Score: 0.0189 - val_SpecificityAtSensitivity: 0.6180 - val_SensitivityAtSpecificity: 0.6117 - val_AUC: 0.6147
Epoch 4/10
38/38 [=====] - 53s 1s/step - loss: 139.3423 - accuracy: 0.3742 - F1Score: 0.0472 - SpecificityAtSensitivity: 0.7027 - SensitivityAtSpecificity: 0.6450 - AUC: 0.6747 - val_loss: 139.0000e+00 - SpecificityAtSensitivity: 0.7040 - SensitivityAtSpecificity: 0.6450 - AUC: 0.6747 - val_accuracy: 0.3742 - val_F1Score: 0.0472 - val_SpecificityAtSensitivity: 0.7040 - val_SensitivityAtSpecificity: 0.6450 - val_AUC: 0.6747
Epoch 5/10
38/38 [=====] - 49s 1s/step - loss: 139.0863 - accuracy: 0.4233 - F1Score: 0.0894 - SpecificityAtSensitivity: 0.7268 - SensitivityAtSpecificity: 0.7750 - AUC: 0.7198 - val_loss: 139.0000e+00 - SpecificityAtSensitivity: 0.7280 - SensitivityAtSpecificity: 0.7750 - AUC: 0.7198 - val_accuracy: 0.4233 - val_F1Score: 0.0894 - val_SpecificityAtSensitivity: 0.7280 - val_SensitivityAtSpecificity: 0.7750 - val_AUC: 0.7198
Epoch 6/10
38/38 [=====] - 54s 1s/step - loss: 139.0343 - accuracy: 0.4533 - F1Score: 0.1069 - SpecificityAtSensitivity: 0.7842 - SensitivityAtSpecificity: 0.7758 - AUC: 0.7538 - val_loss: 139.0000e+00 - SpecificityAtSensitivity: 0.7855 - SensitivityAtSpecificity: 0.7758 - AUC: 0.7538 - val_accuracy: 0.4533 - val_F1Score: 0.1069 - val_SpecificityAtSensitivity: 0.7855 - val_SensitivityAtSpecificity: 0.7758 - val_AUC: 0.7538
Epoch 7/10
38/38 [=====] - 54s 1s/step - loss: 139.0574 - accuracy: 0.4658 - F1Score: 0.1510 - SpecificityAtSensitivity: 0.8795 - SensitivityAtSpecificity: 0.8168 - AUC: 0.7888 - val_loss: 139.0000e+00 - SpecificityAtSensitivity: 0.8810 - SensitivityAtSpecificity: 0.8168 - AUC: 0.7888 - val_accuracy: 0.4658 - val_F1Score: 0.1510 - val_SpecificityAtSensitivity: 0.8810 - val_SensitivityAtSpecificity: 0.8168 - val_AUC: 0.7888
Epoch 8/10
38/38 [=====] - 54s 1s/step - loss: 139.0336 - accuracy: 0.5325 - F1Score: 0.1723 - SpecificityAtSensitivity: 0.9103 - SensitivityAtSpecificity: 0.8650 - AUC: 0.8096 - val_loss: 139.0000e+00 - SpecificityAtSensitivity: 0.9125 - SensitivityAtSpecificity: 0.8650 - AUC: 0.8096 - val_accuracy: 0.5325 - val_F1Score: 0.1723 - val_SpecificityAtSensitivity: 0.9125 - val_SensitivityAtSpecificity: 0.8650 - val_AUC: 0.8096
Epoch 9/10
38/38 [=====] - 57s 1s/step - loss: 139.0791 - accuracy: 0.5608 - F1Score: 0.2042 - SpecificityAtSensitivity: 0.9267 - SensitivityAtSpecificity: 0.8817 - AUC: 0.8344 - val_loss: 139.0000e+00 - SpecificityAtSensitivity: 0.9287 - SensitivityAtSpecificity: 0.8817 - AUC: 0.8344 - val_accuracy: 0.5608 - val_F1Score: 0.2042 - val_SpecificityAtSensitivity: 0.9287 - val_SensitivityAtSpecificity: 0.8817 - val_AUC: 0.8344
Epoch 10/10
38/38 [=====] - 54s 1s/step - loss: 137.8279 - accuracy: 0.6800 - F1Score: 0.2323 - SpecificityAtSensitivity: 0.9557 - SensitivityAtSpecificity: 0.9175 - AUC: 0.9527 - val_loss: 139.0000e+00 - SpecificityAtSensitivity: 0.9577 - SensitivityAtSpecificity: 0.9175 - AUC: 0.9527 - val_accuracy: 0.6800 - val_F1Score: 0.2323 - val_SpecificityAtSensitivity: 0.9577 - val_SensitivityAtSpecificity: 0.9175 - val_AUC: 0.9527
```

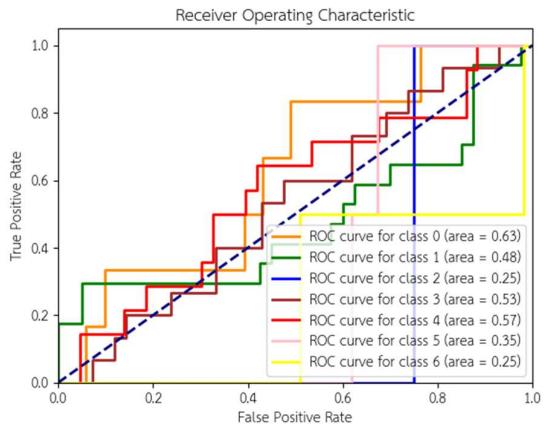
รูปที่ 3.156 ตัวอย่างผลลัพธ์ระหว่างการฝึกฟันโน้มเดล Color ด้วย Xception



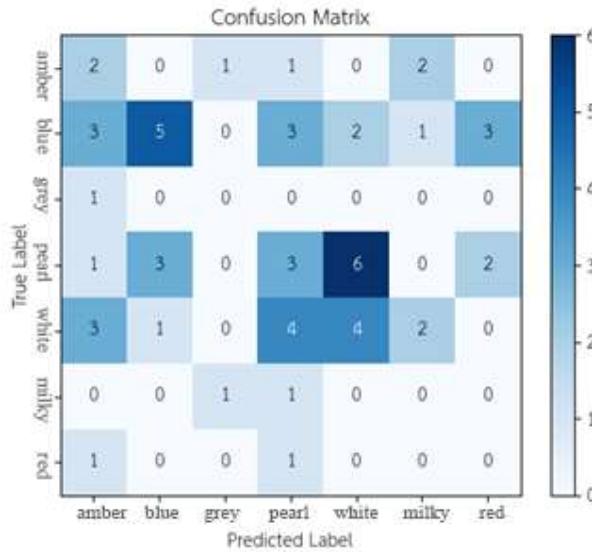
รูปที่ 3.157 ตัวอย่าง Accuracy ของโมเดล Color ที่ฝึกโดยใช้ Xception



รูปที่ 3.158 ตัวอย่าง Loss ของโมเดล Color ที่ฝึกฝนโดยใช้ Xception



รูปที่ 3.159 ตัวอย่าง AUC ของการฝึกฝน Color ด้วย Xception



รูปที่ 3.160 ตัวอย่าง Confusion Matrix ของการฝึกฝน Color ด้วย Xception

■ MobileNet V2

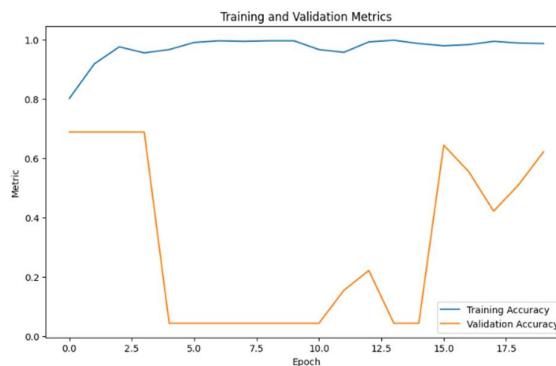
ตัวอย่างการฝึกฝนโมเดลของ Color ด้วยสถาปัตยกรรม MobileNet V2 โดยจะเรียกคำสั่ง model.fit เพื่อทำการฝึกฝนโมเดลดังแสดงดังรูปที่ 3.161 ส่วนในระหว่างการฝึกฝนโมเดลจะแสดงดังรูปที่ 3.162 และเมื่อสิ้นสุดการฝึกฝนจะแสดงผลลัพธ์ดังรูปที่ 3.163

```
# Train the model
history = model.fit(train_ds, validation_data=val_ds, epochs=10)
```

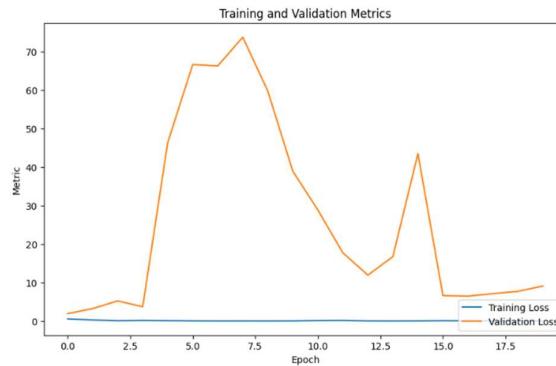
รูปที่ 3.161 ตัวอย่างคำสั่งการฝึกฝนโมเดล Color ด้วย MobileNet V2

```
Epoch 1/10
  104s/step - loss: 0.5584 - accuracy: 0.8083 - F1Score: 0.8162 - SpecificityAtSensitivity: 0.9963 - SensitivityAtSpecificity: 0.9992 - AUC: 0.9993 - val_loss:
Epoch 2/10
  104s/step - loss: 0.3218 - accuracy: 0.9558 - F1Score: 0.9581 - SpecificityAtSensitivity: 0.9993 - SensitivityAtSpecificity: 0.9997 - AUC: 0.9997 - val_loss:
Epoch 3/10
  104s/step - loss: 0.1863 - accuracy: 0.9558 - F1Score: 0.9581 - SpecificityAtSensitivity: 0.9992 - SensitivityAtSpecificity: 0.9998 - AUC: 0.9998 - val_loss:
Epoch 4/10
  104s/step - loss: 0.1373 - accuracy: 0.9481 - F1Score: 0.9478 - SpecificityAtSensitivity: 0.9997 - SensitivityAtSpecificity: 0.9925 - AUC: 0.9995 - val_loss:
Epoch 5/10
  104s/step - loss: 0.1273 - accuracy: 0.9481 - F1Score: 0.9478 - SpecificityAtSensitivity: 0.9997 - SensitivityAtSpecificity: 0.9925 - AUC: 0.9995 - val_loss:
Epoch 6/10
  104s/step - loss: 0.1158 - accuracy: 0.9592 - F1Score: 0.9585 - SpecificityAtSensitivity: 0.9998 - SensitivityAtSpecificity: 0.9992 - AUC: 0.9998 - val_loss:
Epoch 7/10
  104s/step - loss: 0.1273 - accuracy: 0.9573 - F1Score: 0.9538 - SpecificityAtSensitivity: 0.9995 - SensitivityAtSpecificity: 0.9997 - AUC: 0.9995 - val_loss:
Epoch 8/10
  104s/step - loss: 0.2127 - accuracy: 0.9517 - F1Score: 0.9597 - SpecificityAtSensitivity: 0.9995 - SensitivityAtSpecificity: 0.9937 - AUC: 0.9993 - val_loss:
Epoch 9/10
  104s/step - loss: 0.1390 - accuracy: 0.9633 - F1Score: 0.9611 - SpecificityAtSensitivity: 0.9995 - SensitivityAtSpecificity: 0.9958 - AUC: 0.9981 - val_loss:
Epoch 10/10
  104s/step - loss: 0.1464 - accuracy: 0.9653 - F1Score: 0.9639 - SpecificityAtSensitivity: 0.9991 - SensitivityAtSpecificity: 0.9942 - AUC: 0.9995 - val_loss:
Epoch 11/10
  104s/step - loss: 0.0681 - accuracy: 0.9800 - F1Score: 0.9801 - SpecificityAtSensitivity: 0.9997 - SensitivityAtSpecificity: 0.9983 - AUC: 0.9997 - val_loss:
```

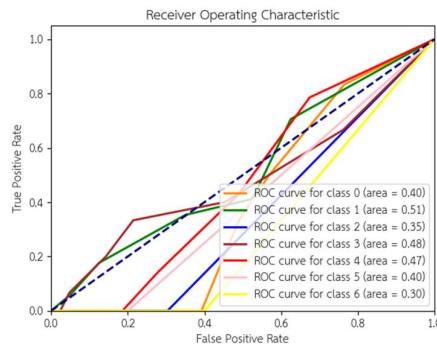
รูปที่ 3.162 ตัวอย่างผลลัพธ์ระหว่างการฝึกฝนโมเดล Color ด้วย MobileNet V2



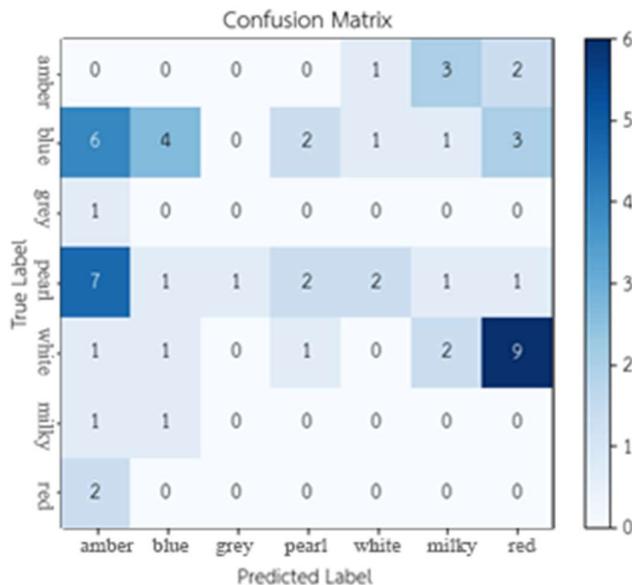
รูปที่ 3.163 ตัวอย่าง Accuracy ของโมเดล Color ที่ฝึกฝนโดยใช้ MobileNet V2



รูปที่ 3.164 ตัวอย่าง Loss ของโมเดล Color ที่ฝึกฝนโดยใช้ MobileNet V2



รูปที่ 3.165 ตัวอย่าง AUC ของโมเดล Color ที่ฝึกฝนโดยใช้ MobileNet V2



รูปที่ 3.166 ตัวอย่าง Confusion Matrix ของโมเดล Color ที่ฝึกฝนโดยใช้ MobileNet V2

▪ ConvNext Tiny

ตัวอย่างการฝึกฝนโมเดลของ Color ด้วยสถาปัตยกรรม ConvNext Tiny โดยจะเรียกคำสั่ง `model.fit` เพื่อทำการฝึกฝนโมเดลดังแสดงดังรูปที่ 3.167 ส่วนในระหว่างการฝึกฝนโมเดลจะแสดงดังรูปที่ 3.168 และเมื่อสิ้นสุดการฝึกฝนจะแสดงผลลัพธ์ดังรูปที่ 3.169

```
# Train the model
history = model.fit(train_ds, validation_data=val_ds, epochs=10)
```

รูปที่ 3.167 ตัวอย่างคำสั่งการฝึกฝนโมเดล Color ด้วย ConvNext Tiny

```

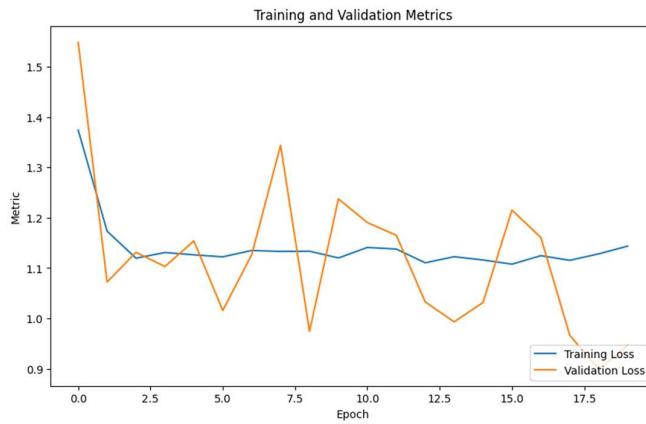
Epoch 1/10
Epoch 2/10 [=====] - 128s 2s/step - loss: 1.0887 - accuracy: 0.1625 - SpecificityAtSensitivity: 0.4958 - SensitivityAtSpecificity: 0.4083 - AUC: 0.4991 - val_loss: 1.86
Epoch 3/10 [=====] - 58s 1s/step - loss: 1.0354 - accuracy: 0.1817 - F1Score: 0.0000e+00 - SpecificityAtSensitivity: 0.4608 - SensitivityAtSpecificity: 0.4983 - AUC: 0.5064 - val_loss: 1.
Epoch 4/10 [=====] - 57s 1s/step - loss: 1.0512 - accuracy: 0.1700 - F1Score: 0.0000e+00 - SpecificityAtSensitivity: 0.5035 - SensitivityAtSpecificity: 0.4523 - AUC: 0.5123 - val_loss: 1.
Epoch 5/10 [=====] - 56s 1s/step - loss: 1.0370 - accuracy: 0.1700 - F1Score: 0.0000e+00 - SpecificityAtSensitivity: 0.4992 - SensitivityAtSpecificity: 0.4577 - AUC: 0.5073 - val_loss: 1.
Epoch 6/10 [=====] - 56s 1s/step - loss: 1.0297 - accuracy: 0.1840 - F1Score: 0.0000e+00 - SpecificityAtSensitivity: 0.4917 - SensitivityAtSpecificity: 0.4942 - AUC: 0.4934 - val_loss: 1.
Epoch 7/10 [=====] - 56s 1s/step - loss: 1.0316 - accuracy: 0.1865 - F1Score: 0.0000e+00 - SpecificityAtSensitivity: 0.4903 - SensitivityAtSpecificity: 0.4567 - AUC: 0.5059 - val_loss: 1.
Epoch 8/10 [=====] - 56s 1s/step - loss: 1.0315 - accuracy: 0.1855 - F1Score: 0.0000e+00 - SpecificityAtSensitivity: 0.4925 - SensitivityAtSpecificity: 0.4525 - AUC: 0.5023 - val_loss: 1.
Epoch 9/10 [=====] - 55s 1s/step - loss: 1.0257 - accuracy: 0.1700 - F1Score: 0.0000e+00 - SpecificityAtSensitivity: 0.4918 - SensitivityAtSpecificity: 0.4750 - AUC: 0.5023 - val_loss: 1.
Epoch 10/10 [=====] - 55s 1s/step - loss: 1.0188 - accuracy: 0.1775 - F1Score: 0.0000e+00 - SpecificityAtSensitivity: 0.4945 - SensitivityAtSpecificity: 0.4942 - AUC: 0.5100 - val_loss: 1.

```

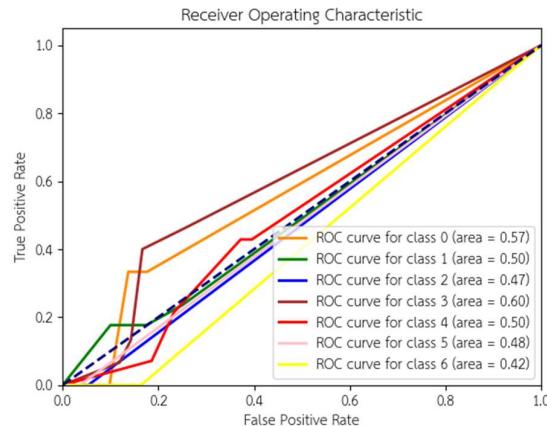
รูปที่ 3.168 ตัวอย่างผลลัพธ์ระหว่างการฝึกฝนโมเดล Color ด้วย ConvNext Tiny



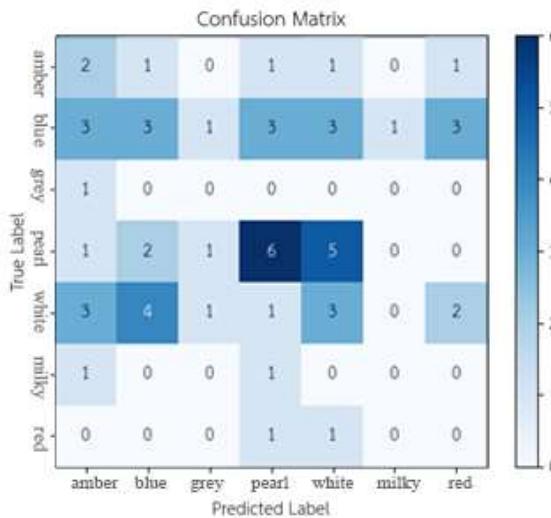
รูปที่ 3.169 ตัวอย่าง Accuracy ของโมเดล Color ที่ฝึกฝนโดยใช้ ConvNext Tiny



รูปที่ 3.170 ตัวอย่าง Loss ของโมเดล Color ที่ฝึกฝนโดยใช้ ConvNext Tiny



รูปที่ 3.171 ตัวอย่าง AUC ของโมเดล Color ที่ฝึกฝนโดยใช้ ConvNext Tiny



รูปที่ 3.172 ตัวอย่าง Confusion Matrix ของโมเดล Color ที่ฝึกฝนโดยใช้ ConvNext Tiny

3.2.2.5.2 การฝึกฝนโมเดลด้วย Traditional Machine Learning

■ SVM

ตัวอย่างการฝึกฝนโมเดลของ Color ด้วยสถาบันตยกรรม SVM โดยจะเรียกคำสั่ง classifier.fit เพื่อทำการฝึกฝนโมเดลดังแสดงดังรูปที่ 3.173 ส่วนในระหว่างการฝึกฝนโมเดลจะแสดงดังรูปที่ 3.174 และเมื่อสิ้นสุดการฝึกฝนจะแสดงผลลัพธ์ดังรูปที่ 3.175

```
# Train the classifier
classifier = SVC(kernel='linear', probability=True)
classifier.fit(X_pca, y_train)
```

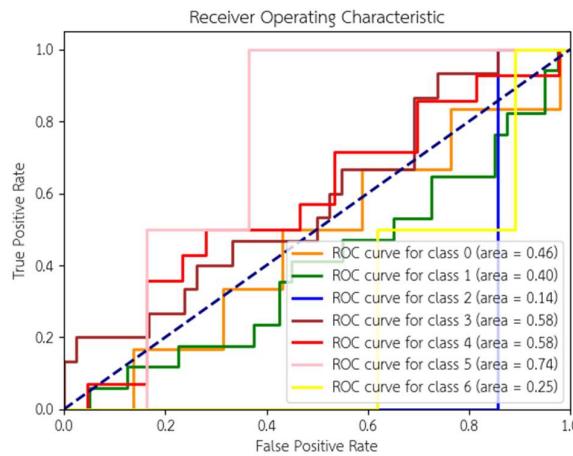
รูปที่ 3.173 ตัวอย่างคำสั่งการฝึกฝนโมเดล Color ด้วย SVM

```
Classification report for -
SVC(degree=2, kernel='poly', probability=True):
precision    recall   f1-score   support
          0       0.00     0.00     0.00      6
          1       0.00     0.00     0.00     17
          2       0.00     0.00     0.00      1
          3       0.28     1.00     0.44     15
          4       0.00     0.00     0.00     14
          5       0.00     0.00     0.00      2
          6       0.00     0.00     0.00      2

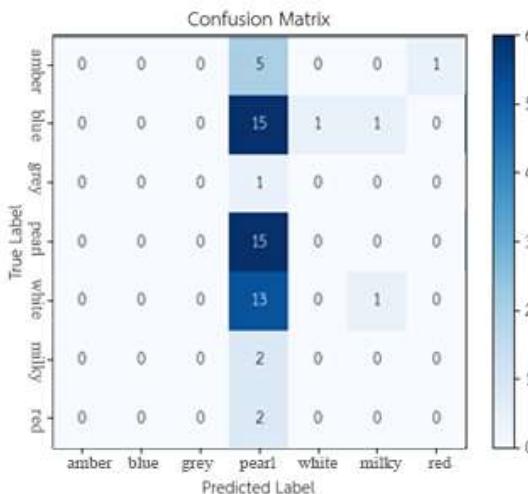
accuracy                           0.26      57
macro avg       0.04    0.14    0.06      57
weighted avg    0.07    0.26    0.12      57

specifity macro avg {.2f} 0.8596338323913074
specifity micro avg {.2f} 0.8771929824561403
specifity weighted avg {.2f} 0.7542789320023102
```

รูปที่ 3.174 ตัวอย่างผลลัพธ์โดยรวมของการฝึกฝนโมเดล Color ด้วย SVM



รูปที่ 3.175 ตัวอย่างผลลัพธ์ AUC ของการฝึกฝนโมเดล Color ด้วย SVM



รูปที่ 3.176 ตัวอย่าง Confusion Matrix ของการฝึกฝนโมเดล Color ด้วย SVM

■ KNN

ตัวอย่างการฝึกฝนโมเดลของ Color ด้วยสถาปัตยกรรม KNN โดยจะเรียกคำสั่ง `classifier.fit` เพื่อทำการฝึกฝนโมเดลตังแต่ดังรูปที่ 3.177 ส่วนในระหว่างการฝึกฝนโมเดลจะแสดงดังรูปที่ 3.178 และเมื่อสิ้นสุดการฝึกฝนจะแสดงผลลัพธ์ดังรูปที่ 3.179

```
# Train the classifier
classifier = KNeighborsClassifier(n_neighbors=2)
classifier.fit(X_pca, y_train)
```

รูปที่ 3.177 ตัวอย่างคำสั่งการฝึกฝนโมเดล Color ด้วย KNN

```

Classification report for -
KNeighborsClassifier(n_neighbors=12):
      precision    recall  f1-score   support

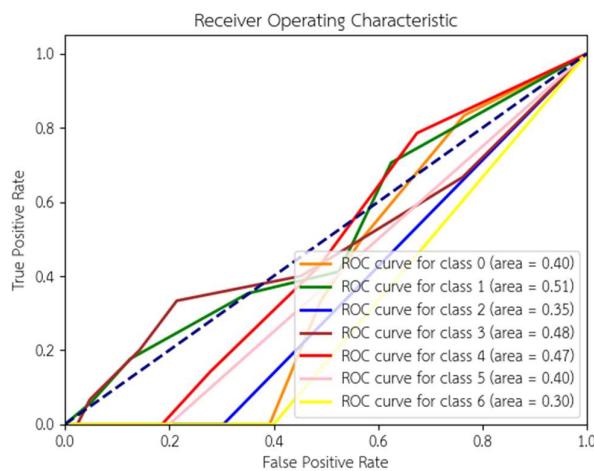
          0       0.00    0.00    0.00     6
          1       0.57    0.24    0.33    17
          2       0.00    0.00    0.00     1
          3       0.40    0.13    0.20    15
          4       0.00    0.00    0.00    14
          5       0.00    0.00    0.00     2
          6       0.00    0.00    0.00     2

   accuracy                           0.11    57
  macro avg       0.14    0.05    0.08    57
weighted avg       0.28    0.11    0.15    57

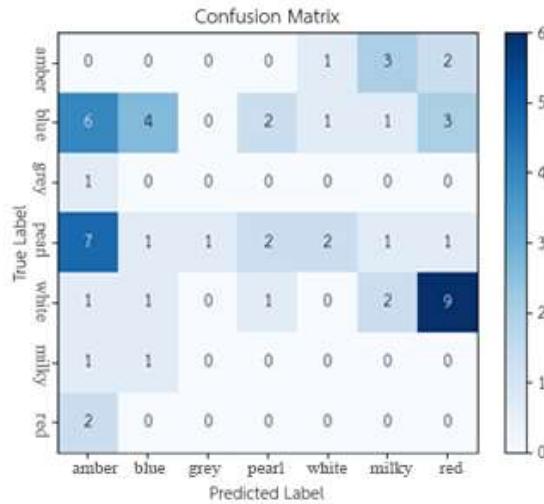
specificity macro avg {.2f} 0.8556785504899634
specificity micro avg {.2f} 0.8508771929824561
specificity weighted avg {.2f} 0.884486695535007

```

รูปที่ 3.178 ตัวอย่างผลลัพธ์โดยรวมของการฝึกฝนโมเดล Colot ด้วย KNN



รูปที่ 3.179 ตัวอย่าง AUC ของการฝึกฝนโมเดล Color ด้วย KNN



รูปที่ 3.18 ตัวอย่าง Confusion Matrix ของการฝึกฝนโมเดล Color ด้วย KNN

■ Decision Tree

ตัวอย่างการฝึกฝนโมเดลของ Color ด้วยสถาปัตยกรรม Decision Tree โดยจะเรียกคำสั่ง `classifier.fit` เพื่อทำการฝึกฝนโมเดลดังรูปที่ 3.180 ส่วนในระหว่างการฝึกฝนโมเดลจะแสดงดังรูปที่ 3.181 และเมื่อสิ้นสุดการฝึกฝนจะแสดงผลลัพธ์ดังรูปที่ 3.182

```
# Train the classifier
classifier = tree.DecisionTreeClassifier()
classifier.fit(X_pca, y_train)
```

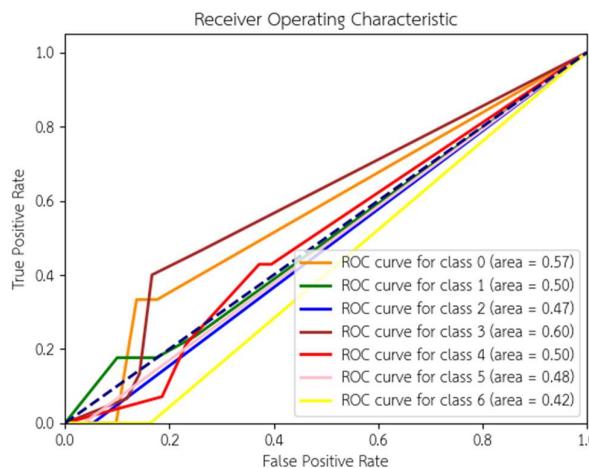
รูปที่ 3.180 ตัวอย่างคำสั่งการฝึกฝนโมเดล Color ด้วย Decision Tree

```
Classification report for -
DecisionTreeClassifier(max_depth=12, min_samples_leaf=2):
precision    recall   f1-score   support
          0       0.18      0.33      0.24      6
          1       0.30      0.18      0.22     17
          2       0.00      0.00      0.00      1
          3       0.46      0.40      0.43     15
          4       0.23      0.21      0.22     14
          5       0.00      0.00      0.00      2
          6       0.00      0.00      0.00      2

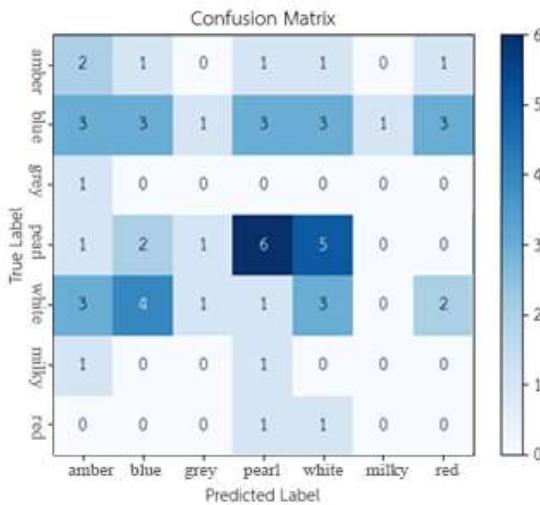
accuracy                           0.25      57
macro avg       0.17      0.16      0.16      57
weighted avg    0.29      0.25      0.26      57

specifity macro avg {.2f} 0.8669229213884284
specifity micro avg {.2f} 0.8742690058479532
specifity weighted avg {.2f} 0.8228464146312803
```

รูปที่ 3.181 ตัวอย่างผลลัพธ์โดยรวมของการฝึกฝน Color ด้วย Decision Tree



รูปที่ 3.182 ตัวอย่าง AUC ของการฝึกฝนโมเดล Color ด้วย Decision Tree



รูปที่ 3.183 ตัวอย่าง Confusion Matrix ของการฝึกฝนโมเดล Color ด้วย Decision Tree

■ XGBoost

ตัวอย่างการฝึกฝนโมเดลของ Color ด้วยสถาบัตยกรรม Decision Tree โดยจะเรียกคำสั่ง classifier.fit เพื่อทำการฝึกฝนโมเดลดังรูปที่ 3.184 ส่วนในระหว่างการฝึกฝนโมเดลจะแสดงดังรูปที่ 3.185 และเมื่อสิ้นสุดการฝึกฝนจะแสดงผลลัพธ์ดังรูปที่ 3.186

```
# Train the classifier
classifier = XGBClassifier(
    n_estimators=50, # Number of boosting rounds or decision trees
    learning_rate=0.01, # Step size shrinkage
    max_depth=4, # Maximum depth of each decision tree
    subsample=1.0, # Fraction of samples used for training each tree
    colsample_bytree=1.0, # Fraction of features used for training each tree
    reg_alpha=0, # L1 regularization term
    reg_lambda=1, # L2 regularization term
    gamma=0.5, # Minimum loss reduction required to split a node
)
classifier.fit(X_pca, y_train)
```

รูปที่ 3.184 ตัวอย่างคำสั่งการฝึกฝน Color ด้วย XGBoost

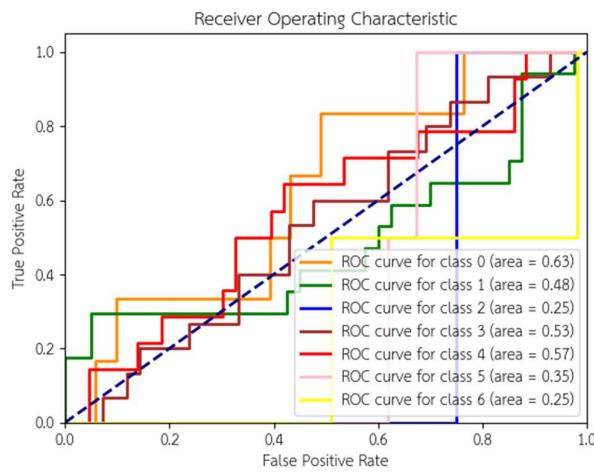
```

Classification report for
XGBClassifier(base_score=0.5, booster='None', callbacks=None,
              colsample_bytree=None, colsample_bynode=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=0.0, gpu_id=None, grow_policy='None', importance_type='None',
              interaction_constraints=None, keep_rank=True, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=5, max_leaves=17,
              max_child_weight=None, max_leaves=None, monotone_constraints=None,
              n_estimators=50, n_jobs=None, n_parallel=None, tree=None, ...)
objectives: multi:softmax, predictor='None', ...
precision recall f1-score support
          0       0.18   0.33   0.24    6
          1       0.56   0.29   0.38   17
          2       0.00   0.00   0.00    1
          3       0.00   0.00   0.21   15
          4       0.33   0.29   0.31    4
          5       0.00   0.00   0.00    2
          6       0.00   0.00   0.00    2
accuracy                           0.25    57
macro avg       0.19   0.16   0.16    57
weighted avg    0.33   0.25   0.27    57

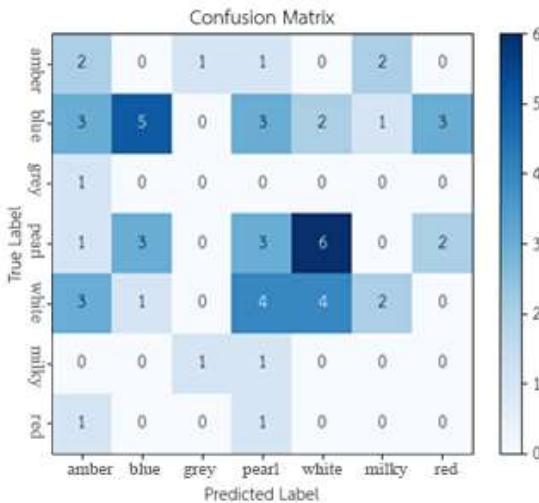
specify macro avg f1: 0.8688364963584491
specify micro avg f1: 0.8742690058419532
specify weighted avg f1: 0.8362411594213741

```

รูปที่ 3.185 ตัวอย่างผลลัพธ์โดยรวมของการฝึกฝน Color ด้วย XGBoost



รูปที่ 3.186 ตัวอย่าง AUC ของการฝึกฝนโมเดล Color ด้วย XGBoost



รูปที่ 3.187 ตัวอย่าง Confusion Matrix ของการฝึกฝนโมเดล Color ด้วย XGBoost

บทที่ 4

ผลการดำเนินงานและการอภิปราย

ในบทนี้จะกล่าวถึงผลการดำเนินงานผลการทดลองสร้างโมเดลแบบ Neural Networks และแบบ Traditional Machine Learning บนชุดข้อมูล Perforation, Fluid, Retraction, Transparency และ Color นอกจากนี้ในบทที่ 4 จะมีการอภิปรายองค์ความรู้ที่ได้ในการทดลองสร้างโมเดล

4.1 ผลการทดลอง

4.1.1 ประสิทธิภาพโมเดลแบบ Neural Networks และ Traditional Machine Learning สำหรับการวิเคราะห์องค์ประกอบ Perforation

การวิเคราะห์ Perforation มีการวิเคราะห์ทั้งหมดคุณลักษณะ 2 แบบได้แก่ No และ Yes จากตารางที่ 4.1 และ 4.2 พบว่าสำหรับการวิเคราะห์ภาพด้วย Neural Networks ด้วยโมเดล MobileNet V2 มี Accuracy สูงที่สุด 68.9% และสำหรับการวิเคราะห์ภาพด้วย Traditional Machine Learning พบว่า การวิเคราะห์ด้วยโมเดล SVM ด้วย Kernel แบบ Sigmoid มี Accuracy สูงที่สุดอยู่ที่ 95%

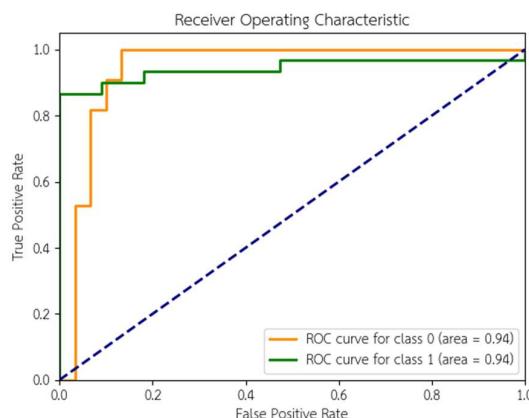
Neural Networks Model							
Model	Architecture	Accuracy	Sensitivity	Specificity	Precision	F1 Score	AUC
XceptionNet	Layers <ul style="list-style-type: none">▪ XceptionNet▪ GlobalAveragePooling2D▪ Dropout(0.2)▪ Sigmoid Optimizer = Adam Loss = Categorical Cross Entropy	0.644	0.500	0.500	0.500	0.602	0.674
ConvNext Tiny	Layers <ul style="list-style-type: none">▪ ConvNext Tiny▪ GlobalAveragePooling2D▪ Dropout(0.2)▪ Sigmoid Optimizer = Adam Loss = Categorical Cross Entropy	0.683	0.500	0.500	0.500	0.650	0.784

MobileNet V2	Layers	0.689	0.500	0.500	0.500	0.553	0.663
	Optimizer = Adam Loss = Categorical Cross Entropy						

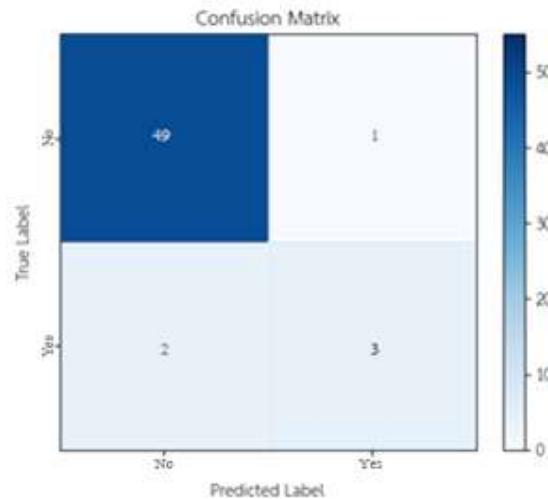
ตารางที่ 4.1 ตารางแสดงประสิทธิภาพโมเดลแบบ Neural Networks สำหรับ Perforation

Traditional Machine Learning Model							
Model	Architecture	Accuracy	Sensitivity	Specificity	Precision	F1 Score	AUC
SVM	Kernel = Sigmoid	0.950	0.950	0.930	0.96	0.950	0.933
Decision Tree	Max Depth = 3 Min Leaf = 10	0.800	0.800	0.798	0.800	0.800	0.798
KNN	Neighbor = 3	0.860	0.860	0.860	0.860	0.860	0.860
XGBoost	Gamma = 0.5 Learning Rate = 0.1 Max Depth = 2 Max Leaves = 3 Estimators = 100	0.790	0.800	0.796	0.800	0.790	0.796

ตารางที่ 4.2 ตารางแสดงประสิทธิภาพโมเดลแบบ Traditional Machine Learning สำหรับ Perforation



รูปที่ 4.2 ตัวอย่างผลลัพธ์ AUC ของโมเดล Perforation ที่ฝึกฝนโดย SVM



รูปที่ 4.3 ตัวอย่างผลลัพธ์ Confusion Matrix ของโมเดล Perforation ที่ฝึกฝนโดย SVM

4.1.2 ประสิทธิภาพโมเดลแบบ Neural Networks และ Traditional Machine Learning สำหรับการวิเคราะห์ห้องค์ประกอบ Fluid

การวิเคราะห์ Fluid มีการวิเคราะห์ทั้งหมดคุณลักษณะ 2 แบบได้แก่ None และ Full จากตารางที่ 4.3 และ 4.4 พบร่วมกันว่า สำหรับการวิเคราะห์ภาพด้วย Neural Networks ด้วยโมเดล Xception แบบที่ 1 มี Accuracy สูงที่สุด 60% และสำหรับการวิเคราะห์ภาพด้วย Traditional Machine Learning พบร่วมกับการวิเคราะห์ด้วยโมเดล SVM ด้วย Kernel แบบ RBF มี Accuracy สูงที่สุดอยู่ที่ 68%

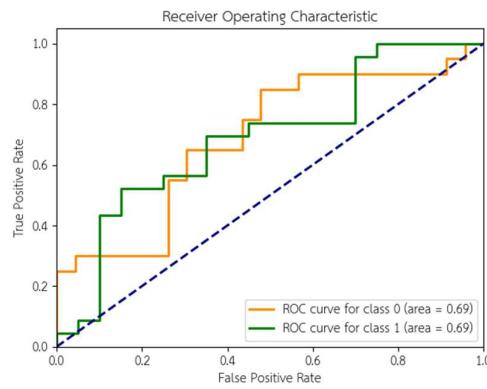
Neural Networks Model							
Model	Architecture	Accuracy	Sensitivity	Specificity	Precision	F1 Score	AUC
Xception	Layers <ul style="list-style-type: none"> ▪ Xception ▪ GlobalAveragePooling2D ▪ Dropout(0.2) ▪ Softmax Optimizer = Adam Loss = Categorical Cross Entropy	0.600	0.600	0.550	0.600	0.700	0.600

ConvNext Tiny Small	Layers <ul style="list-style-type: none">▪ XceptionNet▪ GlobalAveragePooling2D▪ Dropout(0.2)▪ Sigmoid Optimizer = Adam Loss = Categorical Cross Entropy	0.4613	0.534	0.534	0.457	0.355	0.522
MobileNet V2	Layers <ul style="list-style-type: none">▪ MobileNet V2▪ GlobalAveragePooling2D▪ Dropout(0.2)▪ Softmax Optimizer = Adam Loss = Categorical Cross Entropy	0.5324	0.523	0.593	0.461	0.353	0.564

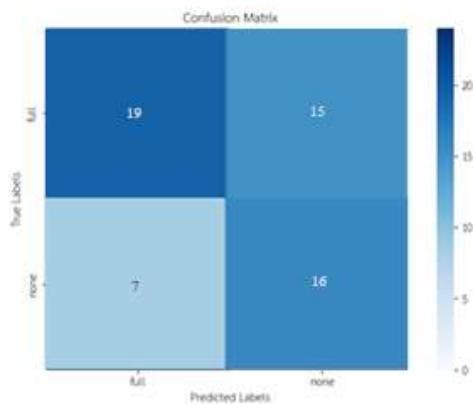
ตารางที่ 4.3 ตารางแสดงประสิทธิภาพโมเดลแบบ Neural Networks สำหรับ Fluid

Traditional Machine Learning Model							
Model	Architecture	Accuracy	Sensitivity	Specificity	Precision	F1 Score	AUC
SVM	Kernel = RBF	0.650	0.680	0.680	0.680	0.680	0.690
Decision Tree	Max Depth = 3 Min Leaf =20	0.640	0.660	0.660	0.660	0.660	0.479
KNN	Neighbor = 11	0.600	0.650	0.650	0.650	0.650	0.650
XGBoost	Gamma = 0.5 Learning Rate = 0.1 Max depth = 10 Max Leaves = 3 Estimators = 100	0.290	0.290	0.820	0.290	0.260	0.534

ตารางที่ 4.4 ตารางแสดงประสิทธิภาพโมเดลแบบ Traditional Machine Learning สำหรับ Fluid



รูปที่ 4.4 ตัวอย่างผลลัพธ์ AUC ของโมเดล Fluid ที่ฝึกฝนโดย SVM



รูปที่ 4.5 ตัวอย่างผลลัพธ์ Confusion Matrix ของโมเดล Fluid ที่ฝึกฝนโดย SVM

4.1.3 ประสิทธิภาพโมเดลแบบ Neural Networks และ Traditional Machine Learning สำหรับการวิเคราะห์ห้องค์ประกอบ Retraction

การวิเคราะห์ Retraction มีการวิเคราะห์ทั้งหมดคุณลักษณะ 3 แบบได้แก่ None, Severe และ Bulging จากตารางที่ 4.5 และ 4.6 พบร่วมกันว่า สำหรับการวิเคราะห์ภาพด้วย Neural Networks ด้วยโมเดล ConvNext Tiny มี Accuracy สูงที่สุด 51.1% และสำหรับการวิเคราะห์ภาพด้วย Traditional Machine Learning พบร่วมกันว่า การวิเคราะห์ด้วยโมเดล Decision Tree ด้วย Max Depth เท่ากับ 10 มี Accuracy สูงที่สุดอยู่ที่ 57%

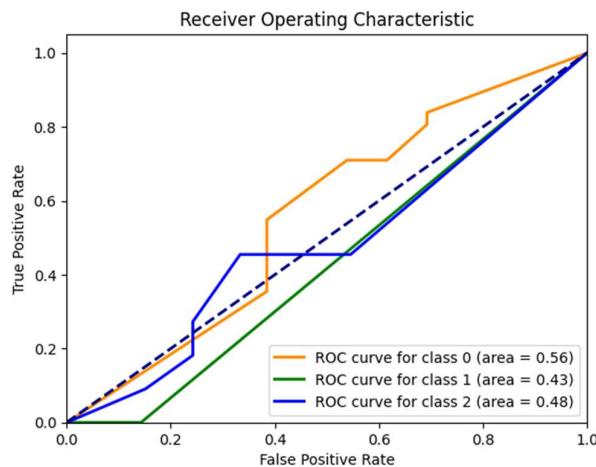
Neural Networks Model							
Model	Architecture	Accuracy	Sensitivity	Specificity	Precision	F1 Score	AUC
Xception	Layers <ul style="list-style-type: none"> ▪ Xception ▪ GlobalAveragePooling2D ▪ Dropout(0.2) ▪ Softmax Optimizer = Adam Loss = Categorical Cross Entropy	0.467	0.711	0.700	0.500	0.139	0.510
ConvNext Tiny	Layers <ul style="list-style-type: none"> ▪ ConvNext Tiny ▪ GlobalAveragePooling2D ▪ Dropout(0.2) ▪ Softmax Optimizer = Adam Loss = Categorical Cross Entropy	0.511	0.700	0.600	0.500	0.260	0.550
MobileNet V2	Layers <ul style="list-style-type: none"> ▪ MobileNet V2 ▪ GlobalAveragePooling2D ▪ Dropout(0.2) ▪ Softmax Optimizer = Adam Loss = Categorical Cross Entropy	0.400	0.600	0.400	0.400	0.140	0.500

ตารางที่ 4.5 ตารางแสดงประสิทธิภาพโมเดลแบบ Neural Networks สำหรับ Retraction

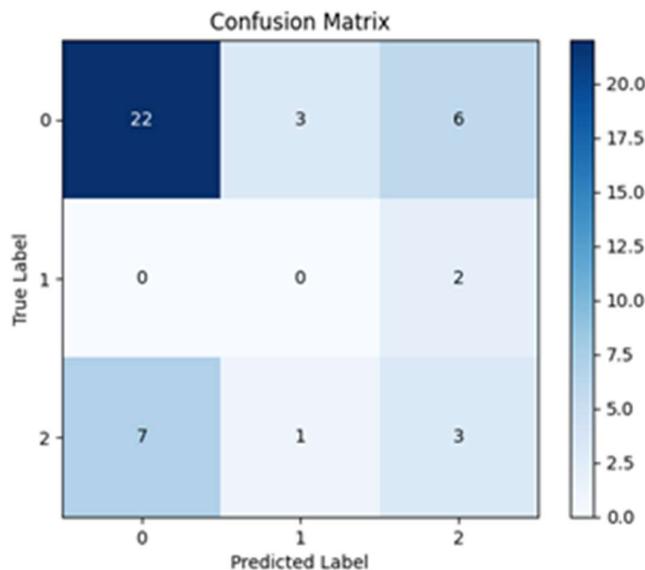
Traditional Machine Learning Model							
Model	Architecture	Accuracy	Sensitivity	Specificity	Precision	F1 Score	AUC
SVM	Kernel = Poly Degree = 2	0.520	0.520	0.480	0.560	0.540	0.400
Decision Tree	Max Depth = 10 Min Leaf = 1	0.570	0.570	0.550	0.600	0.580	0.440
KNN	Neighbor = 3	0.250	0.250	0.270	0.286	0.170	0.486
XGBoost	Gamma = 0.85 Learning Rate = 0.1	0.360	0.360	0.510	0.500	0.410	0.534

	Max depth = 20						
	Max Leaves = 7						
	Estimators = 100						

ตารางที่ 4.6 ตารางแสดงประสิทธิภาพโมเดลแบบ Traditional Machine Learning สำหรับ Retraction



รูปที่ 4.6 ตัวอย่างผลลัพธ์ AUC ของโมเดล Retraction ที่ฝึกฝนโดย Decision Tree



รูปที่ 4.7 ตัวอย่างผลลัพธ์ Confusion Matrix ของโมเดล Retraction ที่ฝึกฝนโดย Decision Tree

4.1.4 ประสิทธิภาพโมเดลแบบ Neural Networks และ Traditional Machine Learning สำหรับการวิเคราะห์องค์ประกอบ Transparency

การวิเคราะห์ Transparency มีการวิเคราะห์ทั้งหมดคุณลักษณะ 3 แบบได้แก่ Clear, Dull, และ Tympanosclerosis จากตารางที่ 4.7 และ 4.8 พบร่วมกันการวิเคราะห์ภาพด้วย Neural Networks ด้วยโมเดล Xception แบบที่ 1 มี Accuracy สูงที่สุด 45.3% และสำหรับการวิเคราะห์ภาพด้วย Traditional Machine Learning พบร่วมกันการวิเคราะห์ด้วยโมเดล XGBoost ซึ่งมี Accuracy สูงที่สุดอยู่ที่ 47%

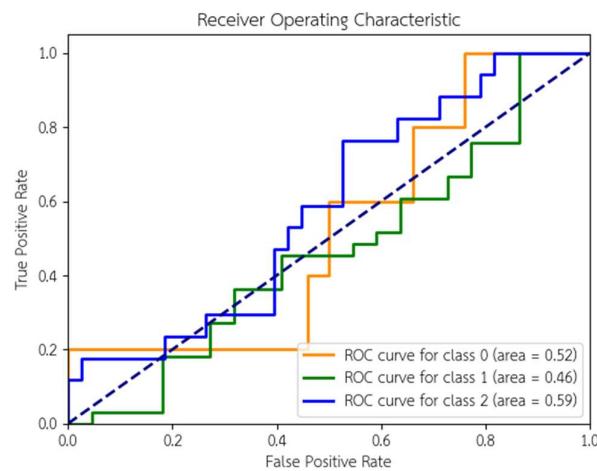
Neural Networks Model							
Model	Architecture	Accuracy	Sensitivity	Specificity	Precision	F1 Score	AUC
Xception	Layers <ul style="list-style-type: none"> ▪ Xception ▪ GlobalAveragePooling2D ▪ Dropout(0.2) ▪ Softmax Optimizer = Adam Loss = Categorical Cross Entropy	0.453	0.652	0.562	0.444	0.433	0.535
ConvNext Tiny	Layers <ul style="list-style-type: none"> ▪ ConvNext Tiny ▪ GlobalAveragePooling2D ▪ Dropout(0.2) ▪ Sigmoid Optimizer = Adam Loss = Categorical Cross Entropy	0.414	0.648	0.504	0.428	0.343	0.600
MobileNet V2	Layers <ul style="list-style-type: none"> ▪ MobileNet V2 ▪ GlobalAveragePooling2D ▪ Dropout(0.2) ▪ Softmax Optimizer = Adam Loss = Categorical Cross Entropy	0.440	0.650	0.500	0.500	0.450	0.570

ตารางที่ 4.7 ตารางแสดงประสิทธิภาพโมเดลแบบ Neural Networks สำหรับ Transparency

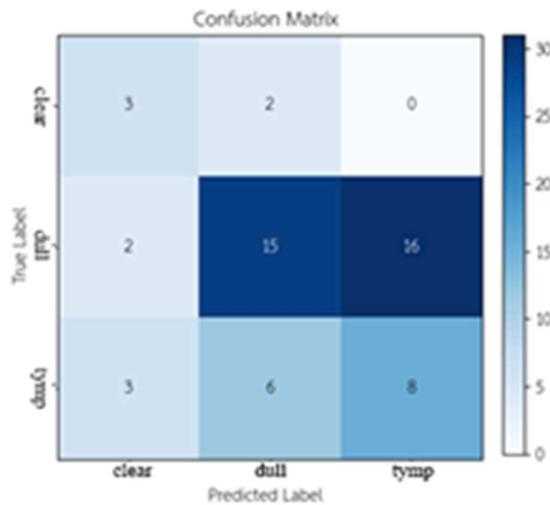
Traditional Machine Learning Model							
Model	Architecture	Accuracy	Sensitivity	Specificity	Precision	F1 Score	AUC
SVM	Kernel = Poly Degree = 2	0.460	0.470	0.500	0.500	0.450	0.600
Decision Tree	Max Depth = 26 Min Leaf = 4	0.449	0.300	0.400	0.400	0.430	0.798
KNN	Neighbor = 2	0.360	0.300	0.500	0.400	0.410	0.860
XGBoost	Gamma = 0.25 Learning Rate = 0.01 Max depth = 5 Max Leaves = 3 Estimators = 10	0.470	0.470	0.640	0.530	0.480	0.600

ตารางที่ 4.8 ตารางแสดงประสิทธิภาพโมเดลแบบ Traditional Machine Learning สำหรับ

Transparency



รูปที่ 4.8 ตัวอย่างผลลัพธ์ AUC ของโมเดล Transparency ที่ฝึกฝนโดย SVM



รูปที่ 4.9 ตัวอย่างผลลัพธ์ Confusion Matrix ของโมเดล Transparency ที่ฝึกฝนโดย XGBoost

4.1.5 ประสิทธิภาพโมเดลแบบ Neural Networks และ Traditional Machine Learning สำหรับการวิเคราะห์องค์ประกอบ Color

การวิเคราะห์ Color มีการวิเคราะห์ทั้งหมดคุณลักษณะ 7 แบบ ได้แก่ Normal, White, Amber, Blue, Pearly White, Milky, และ Red จากตารางที่ 4.9 และ 4.10 พบว่าสำหรับการวิเคราะห์ภาพด้วย Neural Networks ด้วยโมเดล Xception โดยมี Accuracy สูงที่สุด 25.9% และสำหรับการวิเคราะห์ภาพด้วย Traditional Machine Learning พบร่วมกันว่าการวิเคราะห์ด้วยโมเดล SVM ด้วย Kernel Linear มี Accuracy สูงที่สุดอยู่ที่ 26%

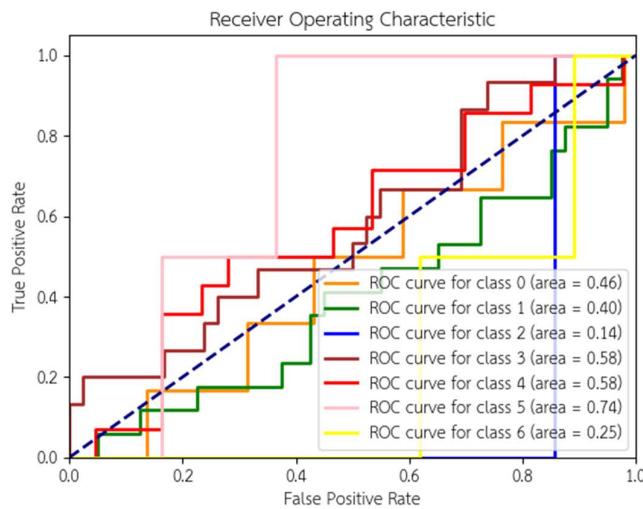
Neural Networks Model							
Model	Architecture	Accuracy	Sensitivity	Specificity	Precision	F1 Score	AUC
Xception	Layers <ul style="list-style-type: none"> ▪ Xception ▪ GlobalAveragePooling2D ▪ Dropout(0.2) ▪ Softmax Optimizer = Adam Loss = Categorical Cross Entropy	0.259	0.230	0.293	0.103	0.200	0.230

ConvNext Tiny	Layers <ul style="list-style-type: none">▪ ConvNext Tiny▪ GlobalAveragePooling2D▪ Dropout(0.2)▪ Sigmoid Optimizer = Adam Loss = Categorical Cross Entropy	0.214	0.240	0.210	0.240	0.210	0.500
MobileNet V2	Layers <ul style="list-style-type: none">▪ MobileNet V2▪ GlobalAveragePooling2D▪ Dropout(0.2)▪ Softmax Optimizer = Adam Loss = Categorical Cross Entropy	0.234	0.230	0.200	0.200	0.234	0.230

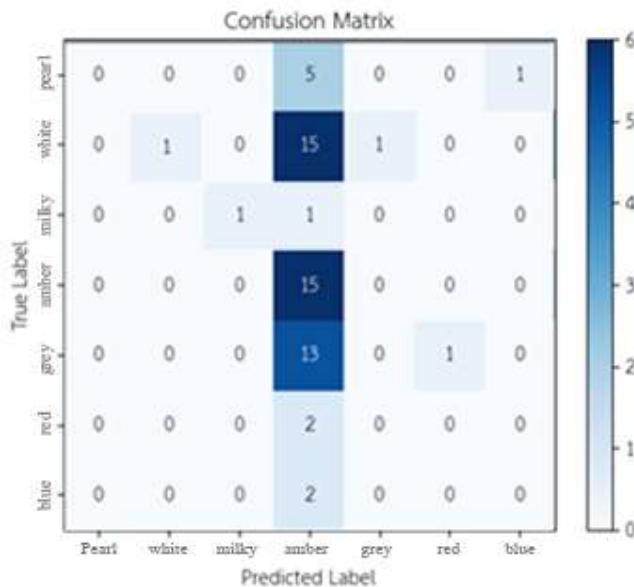
ตารางที่ 4.9 ตารางแสดงประสิทธิภาพโมเดลแบบ Neural Networks สำหรับ Color

Traditional Machine Learning Model							
Model	Architecture	Accuracy	Sensitivity	Specificity	Precision	F1 Score	AUC
SVM	Kernel = Sigmoid	0.260	0.240	0.210	0.240	0.210	0.500
Decision Tree	Max Depth = 10 Min Leaf = 4	0.230	0.240	0.203	0.239	0.200	0.450
KNN	Neighbor = 3 Min Leaf = 4	0.234	0.230	0.293	0.103	0.200	0.440
XGBoost	Gamma = 0.25 Learning Rate = 0.1 Max depth = 5 Max Leaves = 17 Estimators = 50	0.300	0.300	0.210	0.380	0.330	0.510

ตารางที่ 4.10 ตารางแสดงประสิทธิภาพโมเดลแบบ Traditional Machine Learning สำหรับ Color



รูปที่ 4.10 ตัวอย่างผลลัพธ์ AUC ของโมเดล Color ที่ฝึกฝนด้วย XGBoost



รูปที่ 4.11 ตัวอย่างผลลัพธ์ Confusion Matrix ของโมเดล Color ที่ฝึกฝนด้วย XGBoost

4.2 การอภิปรายผล

4.2.1 องค์ความรู้จากการทดลองของโมเดลแบบ Neural Networks

จากผลการทดลองของโมเดลแบบ Neural Networks ได้แก่ Xception, ConvNext Tiny และ MobileNet V2 ผู้วิจัยคาดว่าจำนวนข้อมูลที่มีในการฝึกฝนจำนวนน้อยเกินไป จึงไม่เพียงพอ สำหรับโมเดลแบบ Neural Network เนื่องจาก Neural Network มีจำนวน Parameter จำนวนมาก จึงอาจทำให้โมเดลมีการเรียนรู้ที่ไม่ครอบคลุมขนาดของ Parameter จึงอาจเป็นสาเหตุสำคัญที่ทำให้โมเดลที่ทำการทดลองมีประสิทธิภาพที่ต่ำกว่าที่คาดหวัง

4.2.2 องค์ความรู้จากการทดลองโมเดลแบบ Traditional Machine Learning

จากการทดลองโมเดลแบบ Traditional Machine Learning พบว่ามีประสิทธิภาพโดยรวมที่ดีกว่าโมเดลแบบ Neural Networks เนื่องจากมีจำนวน Parameter น้อยกว่า อีกทั้งยังหากมีการใช้เทคนิค PCA ซึ่งมีหน้าที่ในการคัดเลือกเฉพาะฟีเจอร์ที่มีผลต่อประสิทธิภาพของโมเดลจะช่วยให้การฝึกฝนโมเดลมีประสิทธิภาพสูงกว่าแบบที่ไม่ใช้เทคนิค PCA ช่วยให้ข้อมูลฝึกฝนได้รับเฉพาะคุณลักษณะที่สำคัญสำหรับการจำแนก

4.2.3 องค์ความรู้จากการเพิ่มจำนวนข้อมูล

เนื่องจากข้อมูลภาพที่ใช้ในการฝึกฝนโมเดลในการทดลองครั้งนี้มีจำนวนที่ค่อนข้างน้อยซึ่งเมื่อทดลองฝึกฝนโมเดลด้วยข้อมูลที่จำนวนน้อยแล้วพบว่าประสิทธิภาพของโมเดลต่ำกว่าที่คาดหวังจึงจำเป็นต้องใช้การเพิ่มจำนวนข้อมูล ซึ่งการเพิ่มจำนวนข้อมูลนั้นมีส่วนจำเป็นอย่างมีนัยยะสำคัญ เพราะช่วยให้ประสิทธิภาพของโมเดลเพิ่มขึ้นทั้งโมเดลแบบ Neural Networks และ Traditional Machine Learning ซึ่งการเพิ่มจำนวนภาพที่ใช้ในการทดลองครั้งนี้ เช่น การหมุนภาพและการปรับแต่งความสว่างในสัดส่วนที่แตกต่างกันก็ทำให้ประสิทธิภาพโมเดลมีความแตกต่างกัน

บทที่ 5

สรุปผลการดำเนินงานและข้อเสนอแนะ

5.1 สรุปผลการดำเนินงาน

ปัญหาพิเศษนี้ได้นำเสนอโมเดลจำแนกผู้ป่วยโรคหูชั้นกลางอักเสบซึ่งใช้ข้อมูลจากกล้อง Otoscope โดยจำแนกของค์ประกอบเป็น 5 องค์ประกอบ การศึกษาครั้งนี้ได้ทำการปรับสมดุลข้อมูลของข้อมูลด้วยการหมุนและการเพิ่มและลดแสงสว่าง ซึ่งยังคงความหมายของภาพได้ครบถ้วนคงเดิม โดยในแต่ละองค์ประกอบจะทดลองด้วยโมเดล 2 กลุ่ม ได้แก่กลุ่มของ Neural Networks และกลุ่มของ Traditional Machine Learning โดยกลุ่มของ Neural Networks ประกอบไปด้วยสถาปัตยกรรม Xception, ConvNext Tiny และ MobileNet V2 ส่วนกลุ่มของ Traditional Machine Learning ประกอบไปด้วย SVM KNN, Decision Tree และ XGBoost ซึ่งฝึกฝนโดย Google Colab สำหรับการทดลองนี้ได้ทดลองเพิ่มข้อมูล 2 เท่า 3 เท่า 4 เท่า และ 5 เท่าจากข้อมูลเดิม โดยการเพิ่มข้อมูล 2 เท่าให้ประสิทธิภาพที่สูงสุด โดยโมเดลที่ดีที่สุดของ Perforation คือ MobileNetV2 มี Accuracy อยู่ที่ 68.9%, Fluid คือ Xception มี Accuracy อยู่ที่ 60%, Retraction คือ ConvNext มี Accuracy อยู่ที่ 51.1%, Transparency คือ Xception มี Accuracy อยู่ที่ 45.3%, และ Color คือ Xception มี Accuracy อยู่ที่ 25.9% ส่วนโมเดล Traditional Machine Learning มีความแม่นยำที่ดีที่สุดในแต่ละองค์ประกอบดังนี้ Perforation คือ SVM มี Accuracy อยู่ที่ 95%, Fluid คือ SVM มี Accuracy อยู่ที่ 68%, Retraction คือ Decision Tree มี Accuracy อยู่ที่ 57%, Transparency คือ XGBoost มี Accuracy อยู่ที่ 47% และ Color คือ XGBoost มี Accuracy อยู่ที่ 31%

5.2 ข้อเสนอแนะ

5.2.1 เพิ่มจำนวนข้อมูล

เนื่องจากข้อมูลที่ใช้ในการฝึกฝนมีจำนวนน้อยเกินจึงมีผลกระทบกับประสิทธิภาพของโมเดลอย่างชัดเจนโดยเฉพาะสำหรับการฝึกฝนโดย Neural Networks ซึ่งมีประสิทธิภาพที่ต่ำเกินความคาดหวังในทุก ๆ สถาปัตยกรรม โดยในอนาคตหากสามารถเพิ่มจำนวนข้อมูลสำหรับฝึกฝนให้มากขึ้นและมีความสมดุลในแต่ละคุณลักษณะอาจจะสามารถเพิ่มความแม่นยำของโมเดลได้

5.2.2 เพิ่มการวิเคราะห์แบบระบุตำแหน่งขององค์ประกอบ

เนื่องจากข้อมูลโรคหูชั้นกลางอักเสบนั้นมีตำแหน่งที่จำเพาะของแต่ละองค์ประกอบบนหูชั้นกลาง เช่น บริเวณที่หูทะลุและบริเวณที่หูมีน้ำสีขาวขุ่น หากสามารถนำข้อมูลที่ผ่านการระบุบริเวณต่างๆ ในหู

ชั้นกลางมาใช้ในการฝึกฝนโมเดลแบบ Object Detection หรือ Instance Segmentation อาจสามารถเพิ่มประโยชน์ในการวินิจฉัยโรคเพิ่มขึ้นได้ เพราะช่วยให้ระบุตำแหน่งที่อาจมองเห็นได้ยาก

5.2.3 เพิ่มการวิเคราะห์แบบภาพเคลื่อนไหว

เนื่องจากข้อมูลโรคชั้นกลางอักเสบมีการวิเคราะห์ในทางการแพทย์ด้วยองค์ประกอบ 6 แบบ ได้แก่ 1. ลักษณะของเหลวในหูชั้นกลาง (Fluid) 2. ลักษณะการหดตัวของเยื่อหูชั้นกลาง (Position Retraction) 3. สีของเหลวในหูชั้นกลาง (Color) 4. การทะลุของเยื่อหูชั้นกลาง (Perforation) 5. ความโปร่งแสงของเยื่อหูชั้นกลาง (Transparency) 6. ลักษณะของการเคลื่อนไหวของเยื่อหูชั้นกลาง (Mobility) แต่ในงานวิจัยชนิดนี้มุ่งเน้นไปที่การวิเคราะห์เฉพาะข้อมูลภาพนิ่ง หากในอนาคตมีการวิเคราะห์ภาพเคลื่อนไหวอาจทำให้มีประโยชน์แก่การวินิจฉัยมากยิ่งขึ้นและครอบคลุมการใช้งานในทางการแพทย์ได้มากขึ้น

บรรณานุกรม

- [1] Nature Scientific Reports. (2022). **Analysing wideband absorbance immittance in normal and ears with otitis media with effusion using machine learning.** [online]. Available: <https://www.nature.com/articles/s41598-021-89588-4.pdf>
- [2] Stanford University. (2022). **Machine Learning tips and tricks cheatsheet.** [online]. Available: <https://stanford.edu/~shervine/teaching/cs-229/cheatsheet-machine-learning-tips-and-tricks>
- [3] โรงพยาบาลเมดพาร์ค. (2022). **หูชั้นกลางอักเสบ.** Available: <https://www.medparkhospital.com/content/ear-infection-middle-ear>
- [4] National Library of Medicine, National Center for Biotechnology Information. (2022). **Acute Otitis Media.** [online]. Available: <https://www.ncbi.nlm.nih.gov/books/NBK470332/>
- [5] Analyticsvidhya.(2022). **Support Vector Machine(SVM): A Complete guide for beginners.** [online]. Available: <https://www.analyticsvidhya.com/blog/2021/10/support-vector-machinessvm-a-complete-guide-for-beginners/>
- [6] Paarth Bir. Medium.com. (2022). **Image Classification with K Nearest Neighbours.** [online]. Available: <https://medium.com/swlh/image-classification-with-k-nearest-neighbours-51b3a289280>
- [7] Humboldt-Universität zu Berlin, Department of Geography. (2022). **Image classification - Decision Trees.** [online]. Available: https://pages.cms.hu-berlin.de/EOL/geo_rs/S08_Image_classification1.html
- [8] Humboldt-Universität zu Berlin, Department of Geography. (2022). **Image classification - Random Forest.** [online]. Available: https://pages.cms.hu-berlin.de/EOL/geo_rs/S09_Image_classification2.html
- [9] Humboldt-Universität zu Berlin, Department of Geography. (2022).

Accuracy- Assessment. [online]. Available: https://pages.cms.hu-berlin.de/EOL/geo_rs/S10_Accuracy_assessment.html

[10] Sumit Saha Medium.com. (2022). **A Comprehensive Guide to Convolutional Neural Networks - the ELI5 way.** [online]. Available: <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>

[11] ชิตพงษ์ กิตติราดร.(2022).**Convolutional Neural Network.** [online]. Available: <https://guopai.github.io/ml-blog19.html>

[12] MyPathologyReport.ca. (2022). Mucin . [online]. Available: <https://www.mypathologyreport.ca/th/%E0%B8%84%E0%B8%B3%E0%B8%99%E0%B8%B4%E0%B8%A2%E0%B8%B2%E0%B8%A1-mucin/>

[13] ภาควิชาโสต ศอ นาสิก และลาริงซ์วิทยา คณะแพทยศาสตร์ มหาวิทยาลัยขอนแก่น. (2022). Review Article. [online]. Available: <https://www.thaiscience.info/Journals/Article/SRMJ/10463355.pdf>

[14] วารสารกรมการแพทย์. (2022). แนวโน้มความซุกของหูชั้นกลางอักเสบเรื้อรังในโรงพยาบาลสังกัดสำนักงานปลัดกระทรวงสาธารณสุขในแต่ละปีและภาวะแทรกซ้อนในโรงพยาบาลติดภูมิ ตั้งแต่ปี พศ 2557-2561. [online]. Available: <https://he02.tci-thaijo.org/index.php/JDMS/article/view/253533/172347?fbclid=IwAR1ma2UEpPt1Mvm4tsliYjdjC9fiyeBXxjYrQT8Z6XUq6nbBz5zrPe-ukhY>

[17] M. Karthi, V. Muthulakshmi, R. Priscilla, P. Praveen and K. Vanisri, "Evolution of YOLO-V5 Algorithm for Object Detection: Automated Detection of Library Books and Performance validation of Dataset," (2021) **International Conference on Innovative Computing, Intelligent Communication and Smart Electrical Systems (ICSES)**, Chennai, India, 2021, pp. 1-6, doi: 10.1109/ICSES52305.2021.9633834.

[18] Ashutosh Varma.(2022). **Image classification using SVM (92% accuracy).** Available: <https://www.kaggle.com/code/ashutoshvarma/image-classification-using-svm-92-accuracy>

- [19] https://en.wikipedia.org/wiki/Precision_and_recall
- [20] K. He, G. Gkioxari, P. Dollár and R. Girshick, "Mask R-CNN," 2017 IEEE International Conference on Computer Vision (ICCV), Venice, Italy, 2017, pp. 2980-2988, doi: 10.1109/ICCV.2017.322.
- [21] Liu, Z., Mao, H., Wu, C. Y., Feichtenhofer, C., Darrell, T., & Xie, S. (2022). A ConvNet for the 2020s. In Proceedings - 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022 (pp. 11966-11976). (Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition; Vol. 2022-June). IEEE Computer Society. <https://doi.org/10.1109/CVPR52688.2022.01167>
- [22] Chollet, Francois. (2017). Xception: Deep Learning with Depthwise Separable Convolutions. 1800-1807. 10.1109/CVPR.2017.195.
- [23] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov and L. -C. Chen, "MobileNet V2: Inverted Residuals and Linear Bottlenecks," 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 2018, pp. 4510-4520, doi: 10.1109/CVPR.2018.00474.
- [24] Habib AR, Kajbafzadeh M, Hasan Z, Wong E, Gunasekera H, Perry C, Sacks R, Kumar A, Singh N. Artificial intelligence to classify ear disease from otoscopy: A systematic review and meta-analysis. Clin Otolaryngol. 2022 May;47(3):401-413. doi: 10.1111/coa.13925. Epub 2022 Mar 15. PMID: 35253378; PMCID: PMC9310803.



งานทะเบียนคณะวิทยาศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
คำรับรองเล่มโครงการพิเศษ/ปัญหาพิเศษ/สหกิจศึกษา

วันที่ 1 เดือน มิถุนายน พ.ศ.

2566

ข้าพเจ้า นาย เมธานนท์ แก้วกระจาง รหัสประจำตัว 62050214

นาย เอกสิทธิ์ บุตรดา รหัสประจำตัว 62050251

นักศึกษาหลักสูตรวิทยาศาสตรบัณฑิต สาขาวิชา วิทยาการคอมพิวเตอร์ ภาควิชา วิทยาการ
คอมพิวเตอร์

ขอรับรองว่า ปัญหาพิเศษ เรื่อง

ชื่อภาษาไทย การจำแนกองค์ประกอบของโรคหูชั้นกลางอักเสบจากภาพออโตสโคป

ชื่อภาษาอังกฤษ Classification of Otitis Media Factors from Otoscope Image

ชื่อภาษาอังกฤษ Medical Condition Classification from Medical Image

ปีการศึกษา 2565 เป็นผลงานวิจัยที่มีได้คัดลอกหรือลงทะเบียนสิทธิ์ของผู้อื่นและได้ผ่านการตรวจสอบ
ความช้าช้อน เรียบร้อยแล้ว และได้แนบเอกสารการตรวจสอบการลอกเลียนงานวรรณกรรมที่ตรวจสอบ
จากเล่ม ปัญหาพิเศษฉบับสมบูรณ์แล้ว

โปรแกรมอักขระวิสุทธริ 0.00 %

ลงชื่อ	ลงชื่อ
(นาย เมธานนท์ แก้วกระจาง)	(นาย เอกสิทธิ์ บุตรดา)
นักศึกษา	นักศึกษา

ข้าพเจ้า ดร. อุดมชัยพร อาจารย์ที่ปรึกษาปัญหาพิเศษ ได้ตรวจสอบปัญหาพิเศษของ นักศึกษา
ข้างต้น แล้ว ขอรับรองว่าเป็นผลงานวิจัยของนักศึกษาจริงและมีเนื้อหาสมบูรณ์ จึงลงชื่อไว้ เป็นหลักฐาน
ลงชื่อ..... อาจารย์ที่ปรึกษา