การจำแนกโครงสร้างจุลภาคไทเทเนียมด้วยโครงข่ายประสาทคอนโวลูชันเต็มรูป

นายสิโรดม มงคลธนาภรณ์

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2561

CLASSIFICATION OF TITANIUM MICROSTRUCTURE WITH FULLY CONVOLUTIONAL NEURAL NETWORKS

Mr. Sirodom Mongkhonthanaphon

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Engineering in Computer Engineering
Department of Computer Engineering
Faculty of Engineering
Chulalongkorn University
Academic Year 2018

หัวข้อวิทยานิพนธ์	การจำแนกโครงสร้างจุลภาคไทเทเนียมด้วยโครงข่ายประสาทคอนโวลูชันเต็มรูป
โดย	นายสิโรดม มงคลธนาภรณ์
สาขาวิชา	วิศวกรรมคอมพิวเตอร์
อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก	รองศาสตราจารย์ ดร.ญาใจ ลิ่มปิยะกรณ์

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต

..... คณบดีคณะวิศวกรรมศาสตร์
(รองศาสตราจารย์ ดร.สุพจน์ เตชวรสินสกุล)

คณะกรรมการสอบวิทยานิพนธ์

..... ประธานกรรมการ
(ผู้ช่วยศาสตราจารย์ ดร.สุกรี สินธุภิญโญ)

..... อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก
(รองศาสตราจารย์ ดร.ญาใจ ลิ่มปิยะกรณ์)

..... กรรมการภายนอกมหาวิทยาลัย
(อาจารย์ ดร.ภาสกร อภิรักษ์วรพินิต)

สิโรดม มงคลธนาภรณ์ : การจำแนกโครงสร้างจุลภาคไทเทเนียมด้วยโครงข่ายประสาทคอนโวลูชันเต็มรูป. (CLASSIFICATION OF TITANIUM MICROSTRUCTURE WITH FULLY CONVOLUTIONAL NEURAL NETWORKS) อ.ที่ปรึกษาหลัก : รศ. ดร.ญาใจ ลิ่มปิยะกรณ์

ในช่วงทศวรรษที่ผ่านมา ได้มีการใช้การเรียนรู้เชิงลึกอย่างแพร่หลายเพื่อจำแนกภาพที่ให้ความแม่นยำสูง การแบ่งส่วนความหมายเป็นประเภทหนึ่งของการเรียนรู้เชิงลึกที่มุ่งเน้นการจำแนกคลาสในแต่ละพิกเซลของภาพ ทางด้านโลหะศาสตร์ ไทเทเนียมและอัลลอยมีคุณสมบัติที่โดดเด่นสำหรับการใช้งานทางชีวการแพทย์ โดยเฉพาะในการผ่าตัดฝังวัสดุทดแทน การตรวจสอบวัสดุโดยทั่วไปมักกระทำโดยผู้เชี่ยวชาญในการจำแนกโครงสร้างจุลภาคไทเทเนียม งานวิจัยนี้เสนอการประยุกต์ใช้เทคนิคการเรียนรู้เชิงลึกเพื่อเป็นแนวทางการจำแนกระดับพิกเซลโครงสร้างจุลภาคไทเทเนียมที่อาจช่วยลดทรัพยากรและความไม่แน่นอนระหว่างการควบคุมคุณภาพ วิธีการที่ใช้คือเทคนิคการแบ่งส่วนความหมายที่เรียกว่า โครงข่ายประสาทคอนโวลูชันเต็มรูปที่ถูกพัฒนาด้วยสถาปัตยกรรมยู-เน็ต งานวิจัยนี้ได้สำรวจการบูรณาการเทคนิคการปรับจูนเข้ากับสถาปัตยกรรมยู-เน็ตเพื่อปรับปรุงสมรรถนะแบบจำลอง โมเดลที่สร้างขึ้นถูกปรับจูนด้วยค่าน้ำหนักที่เรียนรู้มาแล้วก่อนหน้าจากตัวจำแนกวีจีจี-16 นอกจากนี้ ชุดข้อมูลภาพโครงสร้างจุลภาคไทเทเนียมได้ถูกเพิ่มจำนวนตัวอย่างด้วยวิธีความผิดปกติยืดหยุ่น สำหรับการประเมินสมรรถนะแบบจำลองจะใช้ตัววัด 4 ตัว ประกอบด้วย ความแม่นยำพิกเซล ความแม่นยำเฉลี่ย ไอโอยูเฉลี่ย และ ไอโอยูถ่วงน้ำหนักความถี่ ผลลัพธ์การประเมินพบว่าค่าความแม่นยำเพิ่มขึ้นเล็กน้อย ในขณะที่เวลาเรียนรู้เร็วกว่ามากเทียบกับการเรียนรู้ปกติที่ไม่มีการปรับจูนค่าน้ำหนักเริ่มต้น

สาขาวิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อนิสิต .....

ปีการศึกษา 2561 ลายมือชื่อ อ.ที่ปรึกษาหลัก .....

6070340521 : MAJOR COMPUTER ENGINEERING

KEYWORD: Fully convolutional neural networks, Semantic segmentation, Fine tuning, Titanium microstructure

Sirodom Mongkhonthanaphon : CLASSIFICATION OF TITANIUM MICROSTRUCTURE WITH FULLY CONVOLUTIONAL NEURAL NETWORKS. Advisor: Assoc. Prof. Yachai Limpiyakorn, Ph.D.

In recent decades, deep learning has been widely used for automatically classifying images with high accuracy. Semantic segmentation is a type of deep learning that focuses on classifying every pixel into classes. In Metallurgy, Titanium and its alloy exhibit excellent properties for biomedical applications, especially in implant surgery. Material inspection is generally done by experts to classify Titanium microstructure. This research introduced applying a deep learning technique for pixel-wise classification of Titanium microstructure that would reduce resources and uncertainties during quality control. The method applies a semantic segmentation technique, fully convolutional neural network, implemented with the U-net architecture. The research work has explored the integration of Fine-tuning technique to the U-net architecture for improving the model performance. The constructed model is fine-tuned with the pretrained weights obtained from the VGG-16 classifier. The dataset of Titanium microstructure images is also augmented using elastic deformations. Four metrics are applied for the assessment of model performances, including Pixel accuracy, Mean accuracy, Mean of intersection over union (IoU) and Frequency weighted of IoU. The evaluation results reported slightly increase of accuracy, while the training time is much faster than training from scratch.

Field of Study:	Computer Engineering	Student's Signature .....
Academic Year:	2018	Advisor's Signature .....

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงได้ด้วยความอนุเคราะห์อย่างดียิ่งของรองศาสตราจารย์ ดร.ญาใจ ลิ่มปิยะกรณ์ อาจารที่ปรึกษาวิทยานิพนธ์ ซึ่งท่านได้กรุณาสละเวลาให้ความรู้ ให้คำปรึกษา ตรวจสอบ ให้คำแนะนำแนวทางการวิจัย และสนับสนุนจนทำให้การวิจัยในครั้งนี้สำเร็จลุล่วงด้วยดี ข้าพเจ้าจึงขอกราบระลึกพระคุณรองศาสตราจารย์ ดร.ญาใจ ลิ่มปิยะกรณ์ ไว้ ณ ที่นี้

ข้าพเจ้าขอกราบขอบพระคุณ ผู้ช่วยศาสตราจารย์ ดร.สุกรี สินธุภิญโญ และ ดร.ภาสกร อภิรักษ์วรพินิต กรรมการสอบวิทยานิพนธ์ ที่ได้กรุณาสละเวลา ให้คำแนะนำ ตรวจสอบ และแก้ไขวิทยานิพนธ์ฉบับนี้ให้ถูกต้องสมบูรณ์ยิ่งขึ้น

ข้าพเจ้าขอขอบพระคุณบริษัท Meticuly เป็นอย่างยิ่ง ที่ได้ให้ข้อแนะนำทางวิชาการและชุดข้อมูลที่สำคัญในการวิจัยครั้งนี้

ท้ายนี้ข้าพเจ้าขอกราบขอบพระคุณ คุณพ่อ คุณแม่ และครอบครัวสำหรับกำลังใจที่มีค่ายิ่ง รวมถึงขอขอบพระคุณผู้บังคับบัญชาในสายงาน เพื่อนร่วมงาน และมิตรสหาย ที่คอยติดตามให้กำลังใจ ให้การสนับสนุนและความช่วยเหลือในด้านต่าง ๆ และท่านอื่น ๆ ที่มิได้กล่าวชื่อไว้ ณ ที่นี้ที่มีส่วนช่วยให้วิทยานิพนธ์ของข้าพเจ้าสำเร็จไปได้ด้วยดี

สิโรดม มงคลธนาภรณ์

สารบัญ

หน้า

สารบัญตาราง

หน้า

สารบัญภาพ

หน้า

บทที่ 1
บทนำ

1.1 ที่มาและความสำคัญของปัญหา

ไทเทเนียม (Titanium) เป็นโลหะที่มีแนวโน้มการใช้งานที่หลากหลาย เนื่องด้วยสมบัติทางกล (Mechanical Properties) ที่ดีเยี่ยม ความหนาแน่นต่ำ (Low Density) ไม่เป็นพิษต่อร่างกายมนุษย์ (Bio-Compatibility) รวมถึงมีความต้านทานการกัดกร่อนที่ดี (Corrosion resistance) ดังนั้นไทเทเนียมจึงเป็นที่นิยมในการใช้เป็นวัสดุสมัยใหม่ในงานที่ต้องการคุณสมบัติดังกล่าว อาทิ ใช้เป็นวัสดุทดแทนชิ้นส่วนในร่างกายของสิ่งมีชีวิต ใช้เป็นตัวกรองในระบบเครื่องยนต์ ชิ้นส่วนของเครื่องบิน ฯลฯ เนื่องด้วยการใช้งานไทเทเนียมที่หลากหลาย ทำให้มีการผลิตชิ้นงานไทเทเนียมในหลายรูปทรงที่ซับซ้อน โดยเฉพาะอย่างยิ่งการใช้งานเป็นวัสดุทดแทนชิ้นส่วนในร่างกายของสิ่งมีชีวิต [1] งานวิจัยนี้ได้ให้ความสนใจกับชิ้นงาน Titanium-6Al-4V หรือ Ti-6Al-4V (6% Aluminium, 4% Vanadium) ที่ขึ้นรูปด้วยกรรมวิธี Selective Laser Melting (SLM) ซึ่งเป็นกระบวนการขึ้นรูปสามมิติ (3D Printing Process) ประเภทหนึ่ง โดยขึ้นรูปโลหะจากผง และใช้ลำแสงฉายไปยังบางส่วนของชั้นโลหะผง และหลอมละลายจนเกิดการขึ้นรูป

โครงสร้างด้านในของวัสดุเรียกว่าโครงสร้างจุลภาค (Microstructure) การตรวจสอบโครงสร้างจุลภาคกระทำได้โดยการส่องผ่านด้วยกล้องจุลทรรศน์กำลังขยายสูง ทำให้สามารถทราบคุณลักษณะทางกายภาพและคุณลักษณะทางเคมีของวัสดุนั้นได้ ภายในโครงสร้างจุลภาคของโลหะจะประกอบด้วยเฟส (Phase) อันเป็นลักษณะเด่นที่เป็นแพทเทิร์น (Pattern) แสดงความแตกต่างของโครงสร้างจุลภาค เฟสที่แตกต่างกันเกิดจากส่วนผสมทางเคมีและกระบวนการที่วัสดุนั้นได้ถูกกระทำมา แต่ในการสังเกตโครงสร้างจุลภาคโลหะนั้นต้องอาศัยความรู้ทางโลหการ รวมถึงประสบการณ์ความชำนาญในการจำแนกเฟสภายในโครงสร้างจุลภาค ทำให้การสังเกตโครงสร้างจุลภาคมักกระทำโดยผู้เชี่ยวชาญที่สามารถคัดแยกลักษณะเด่นของโครงสร้างจุลภาค

งานวิจัยนี้จึงมีแนวคิดที่จะนำเสนอวิธีจำแนกเฟสของโครงสร้างจุลภาค Ti-6Al-4V ด้วยเทคนิคการเรียนรู้เชิงลึก (Deep Learning) สำหรับงานควบคุมคุณภาพในกระบวนการผลิตชิ้นงาน ซึ่งต้องการทราบสัดส่วนประเภทเฟสในแต่ละชิ้นงาน ผู้วิจัยได้ศึกษาและสนใจการสร้างแบบจำลองด้วยโครงข่ายประสาทคอนโวลูชันเต็มรูป (Fully Convolutional Neural Networks–FCNN) ซึ่งเป็นวิธีการที่ใช้การแบ่งส่วนตามความหมาย (semantic segmentation) สำหรับตรวจหาวัตถุทั้งหมดที่ต้องการในภาพ เพื่อใช้คำนวณค่าตัววัดที่ใช้ประเมินคุณภาพของชิ้นงาน Ti-6Al-4V

1.2 วัตถุประสงค์ของงานวิจัย

สร้างแบบจำลองที่แม่นยำในการจำแนกเฟสของโครงสร้างจุลภาค Ti-6Al-4V ที่ขึ้นรูปด้วยวิธีการ SLM สำหรับนำไปใช้กับงานควบคุมคุณภาพในกระบวนการผลิตชิ้นงาน เพื่อลดเวลาที่ใช้ในการตรวจสอบคุณภาพโครงสร้างจุลภาคของชิ้นงาน

1.3 ขอบเขตการดำเนินงาน

1) แบบจำลองใช้กับโครงสร้างจุลภาค Ti-6Al-4V ที่ขึ้นรูปด้วยกระบวนการ SLM และผ่าน Heat Treatment
2) จำแนกประเภทระหว่างเฟส alpha และเฟส beta เท่านั้น
3) สร้างแบบจำลองโดยใช้ไลบรารีภาษา Python คือ Tensorflow และ Keras โดยใช้ตัวประมวลผล GPU
4) ประเมินความแม่นยำของแบบจำลอง FCNN เทียบกับภาพ ground-truth ด้วยตัวชี้วัดที่กำหนด

1.4 ขั้นตอนการวิจัย

1) ศึกษาแนวทางการสร้างแบบจำลอง CNN ด้วย Tensorflow และ Keras
2) ศึกษาแนวทางการจัดเตรียม Dataset
3) ศึกษาหลักการทำงานและการเรียนรู้ด้วยวิธี FCNN
4) วิเคราะห์และออกแบบแบบจำลองให้ตรงตามวัตถุประสงค์ของงานวิจัย
5) พัฒนาแบบจำลองและส่วนประกอบต่างๆ
6) ทดสอบแบบจำลอง
7) ประเมินและสรุปผลงานวิจัย
8) ตีพิมพ์ผลงานวิชาการ
9) จัดทำวิทยานิพนธ์

1.5 ประโยชน์ที่คาดว่าจะได้รับ

1) ได้แบบจำลองที่มีประสิทธิภาพเพื่อจำแนกเฟสชิ้นงาน Ti-6Al-4V ที่ขึ้นรูปด้วยวิธี SLM
2) ได้วิธีการตรวจสอบโครงสร้างจุลภาครูปแบบใหม่ที่รวดเร็วกว่าการตรวจสอบด้วยมนุษย์
3) ได้วิธีการควบคุมคุณภาพสำหรับตรวจสอบชิ้นงาน Ti-6Al-4V เพื่อสนับสนุนการผลิตในเชิงอุตสาหกรรม

1.6 โครงสร้างของเนื้อหาในวิทยานิพนธ์

เนื้อหาของวิทยานิพนธ์ฉบับนี้แบ่งเป็น 6 บทดังนี้

บทที่ 1 อธิบายถึงที่มาและความสำคัญของปัญหา รวมถึงขอบเขตและประโยชน์ของงานวิจัย

บทที่ 2 อธิบายถึงทฤษฎีที่เกี่ยวข้องและงานวิจัยที่เกี่ยวข้อง

บทที่ 3 อธิบายถึงแนวคิดและวิธีการดำเนินงานวิจัย

บทที่ 4 อธิบายถึงวิธีการพัฒนาเครื่องมือและซอฟต์แวร์สนับสนุนแนวคิดของงานวิจัย

บทที่ 5 อธิบายถึงวิธีการทดลองและวิเคราะห์ผล

บทที่ 6 สรุปงานวิจัยทั้งหมด รวมถึงงานวิจัยในอนาคต

1.7 ผลงานที่ตีพิมพ์จากวิทยานิพนธ์

ส่วนหนึ่งของวิทยานิพนธ์ได้รับการตีพิมพ์บทความวิชาการ 2 บทความ ประกอบด้วย

1) S. Mongkhonthanaphon and Y. Limpiyakorn, “ Classification of Titanium Microstructure with Fully Convolutional Neural Networks” ในรายงานการประชุมวิชาการนานาชาติสืบเนื่องจาก 11th International Conference on Computer and Electrical Engineering (ICCEE 2018), October 12-14, 2018, Tokyo, Japan.
2) S. Mongkhonthanaphon and Y. Limpiyakorn, “ A Deep Neural Network for Pixel-Wise Classification of Titanium Microstructure” ตีพิมพ์ใน International Journal of Machine Learning and Computing (IJMLC) สืบเนื่องจากการประชุมวิชาการ 2nd International Conference on Computer Science and Artificial Intelligence (CSAI2018), December 8-10, 2018, Shenzhen, China.

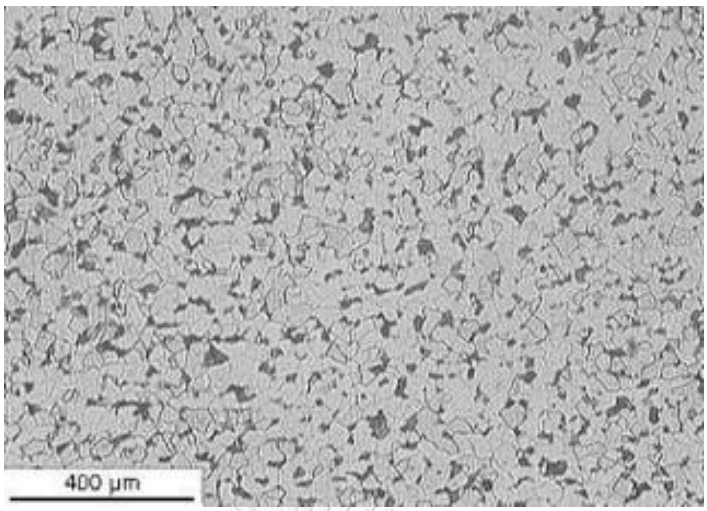
บทที่ 2
ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

2.1 ลักษณะภาพโครงสร้างจุลภาค

โครงสร้างจุลภาค (Mocrostructure) คือโครงสร้างขนาดเล็กภายในวัสดุที่จะมองเห็นได้เมื่อกำลังขยายของกล้องจุลทรรศน์มากกว่า 25 เท่าขึ้นไป [2] ซึ่งมีผลต่อคุณสมบัติทางกลของวัสดุ อาทิ เช่น ความแข็ง ความแกร่ง ความเหนียว ความสามารถทนต่อการกัดกร่อน เป็นต้น โครงสร้างจุลภาคเกิดจากการเรียงตัวของอะตอมที่เรียงตัวกัน การเรียงตัวที่แตกต่างของอะตอมทำให้เกิดเฟสที่แตกต่างขึ้นในวัสดุ โดยสาเหตุที่ทำให้เกิดความแตกต่างของเฟสนั้นมาจากธาตุภายในของวัสดุหรือกระบวนการที่มีผลต่อวัสดุนั้นเช่น ความร้อนหรือแรงทางกล การตรวจสอบโครงสร้างจุลภาคมีความสำคัญและเป็นประโยชน์ต่อการควบคุมคุณภาพ (Quality Control) เนื่องจากข้อมูลที่ได้จากการตรวจสอบนั้นสามารถนำมาวิเคราะห์ได้ถึงความสมบูรณ์หรือบกพร่องของชิ้นงาน โดยกล้องจุลทรรศน์ที่นำมาใช้กับโครงสร้างจุลภาคอาจใช้ตัวกลางฉายภาพที่แตกต่างกันขึ้นอยู่กับชนิดของกล้องจุลทรรศน์ ซึ่งที่นิยมจะมีอยู่ด้วยกัน 2 ชนิดคือ กล้องจุลทรรศน์แบบใช้แสง กับกล้องจุลทรรศน์อิเล็กตรอน โดยที่กล้องจุลทรรศน์แบบแสงจะให้กำลังขยายได้ไม่สูงมากนักแต่ราคาการใช้งานไม่สูง ส่วนกล้องจุลทรรศน์แบบอิเล็กตรอนจะให้กำลังขยายที่สูงแต่ราคาการใช้งานแพง ปัจจุบัน การถ่ายภาพเพื่อนำมาวิเคราะห์โครงสร้างจุลภาคของโลหะมีอยู่ 2 แบบคือ การถ่ายแบบ 2 มิติ และการถ่ายแบบ 3 มิติ

2.1.1 การถ่ายภาพโครงสร้างจุลภาคแบบ 2 มิติ

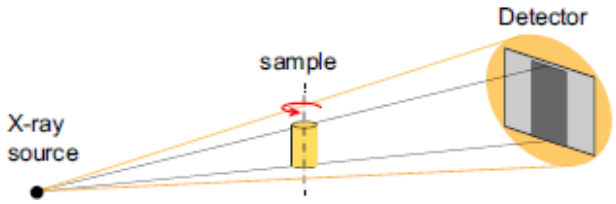
คือ การถ่ายภาพจากผิวชิ้นงานโลหะด้วยกล้องจุลทรรศน์ ทำได้ด้วยการเตรียมชิ้นงานให้มีผิวเรียบ ด้วยการขัดกระดาษทราย โดยไล่ขัดจากกระดาษทรายเบอร์หยาบไปกระดาษทรายเบอร์ละเอียด จากนั้นจึงนำชิ้นงานไปผ่านการกัดกรด เนื่องจากกรดจะมีปฏิกิริยาที่แตกต่างกันในแต่ละเฟสของชิ้นงาน ทำให้ความสึกหรอที่ได้ระหว่างเฟสจะไม่เท่ากัน ช่วยให้การมองเห็นเมื่อนำไปส่องกล้องจะเห็นความแตกต่างของเฟสมากยิ่งขึ้น ช่วยให้การถ่ายภาพชัดมากยิ่งขึ้น ภาพที่ 1 แสดงตัวอย่างของภาพที่ได้จากการถ่ายภาพโครงสร้างจุลภาคแบบ 2 มิติของโลหะเหล็กกล้าคาร์บอนต่ำด้วยกล้องจุลทรรศน์แบบแสง



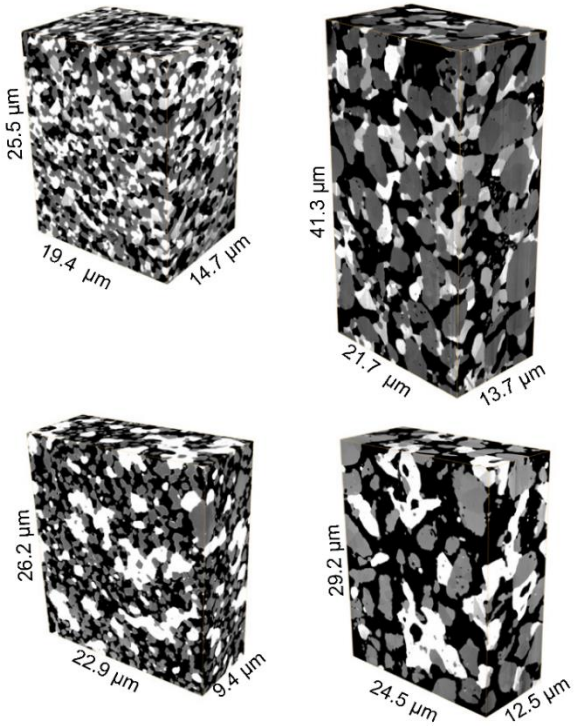
ภาพที่ 1 โครงสร้างจุลภาคของโลหะเหล็กกล้าคาร์บอนต่ำ [3]

2.1.2 **การถ่ายภาพโครงสร้างจุลภาคแบบ 3 มิติ**

คือ การถ่ายภาพทั้งชิ้นงานของโลหะวิธีการหนึ่งที่ได้รับความนิยมคือการใช้เทคนิค X-ray computed microtomography ซึ่งจะใช้ตัวกลางฉายภาพเป็นรังสี X-ray ฉายผ่านชิ้นงานไปยังฉากรับ โดยหมุนชิ้นงานไปด้วยเพื่อให้รังสี X-ray ฉายผ่านภาพได้ครบทุกมุมอย่างในภาพที่ 2 จากนั้นจึงนำภาพจากฉากมาทำการคำนวณเพื่อสร้างภาพโครงสร้างแบบ3มิติ ด้วยเทคนิคนี้จะทำให้ได้มองภาพเฟสของโลหะที่เป็น 3 มิติได้ และยังสามารถมองภาพ 2 มิติที่เป็นภาพตัดขวางของชิ้นงานภายในชิ้นงานได้ แต่เมื่อเปรียบเทียบกับการถ่ายภาพโครงสร้างจุลภาคแบบ2มิติแล้ว วิธีการนี้มีความยุ่งยากในการเตรียมชิ้นงานและอุปกรณ์มากกว่ามาก รวมถึงค่าใช้จ่ายในการถ่ายภาพก็สูงกว่ามาก ภาพที่ 3 แสดงตัวอย่างของภาพที่ได้จากการถ่ายภาพโครงสร้างจุลภาคแบบ 3 มิติหลังจากที่ภาพได้ผ่านกระบวนการสร้างภาพ ซึ่งจะแสดงถึงปริมาตรของเฟสต่างๆและลักษณะการกระจายตัวภายในโครงสร้างจุลภาค



ภาพที่ 2 การถ่ายภาพชิ้นงาน 3 มิติด้วยวิธี X-ray computed microtomography [4]



ภาพที่ 3 ภาพถ่ายโครงสร้างจุลภาค 3 มิติ ที่ได้หลังจากผ่านกระบวนการสร้างภาพจากรูปถ่าย X-ray computed tomography [5]

2.2 ภาพระดับสีเทา (Grayscale Image)

ภาพระดับสีเทาคือภาพที่สร้างมาจากค่าของความเข้มสีจากแต่ละจุด โดยความเป็นไปได้ของภาพระดับสีเทาในภาพทั้งหมดจะขึ้นอยู่กับจำนวนบิตที่ใช้ ซึ่งมักนิยมใช้ระดับสีเทา 8 บิตที่มีระดับสีเทาทั้งหมด 256 ระดับ โดยนิยมระบุเป็นตัวเลขทศนิยมระหว่างช่วง 0-1 หรือจำนวนเต็ม 0-255 โดยให้ 0 แทนสีขาวและ 255 แทนสีดำ ดังภาพที่ 4 การแปลงภาพจากระบบสี RGB เป็นระดับสีเทาสามารถทำได้ด้วยการใช้สมการ (1) ในการแปลง

ภาพที่ 4 ระดับของสีเทา 8 บิตที่ไล่จากสีขาว (0) ไปจนถึงสีดำ (255)

$$Y = 0.3R + 0.59G + 0.11B \quad (1)$$

โดย Y แทนค่าระดับความเข้มสีเทา ณ ตำแหน่งพิกเซลที่ต้องการหา

R แทนระดับความเข้มสีแดง ณ ตำแหน่งพิกเซลที่ต้องการหา

G แทนระดับความเข้มสีเขียว ณ ตำแหน่งพิกเซลที่ต้องการหา

B แทนระดับความเข้มสีน้ำเงิน ณ ตำแหน่งพิกเซลที่ต้องการหา

2.3 ภาพไบนารี (Binary Image)

ภาพไบนารีคือภาพที่ในแต่ละตำแหน่งของภาพมีเพียงแค่ 2 สถานะ คือ 0 กับ 1 กล่าวคือ สีดำเมื่อมีสถานะเป็น 0 และสีขาวเมื่อมีสถานะเป็น 1 โดยการแปลงภาพระดับสีเทาให้กลายเป็นภาพไบนารี จะต้องกำหนดค่าขีดแบ่ง (Threshold) เพื่อกำหนดค่าระดับความเข้มสีเทาที่จะถูกแปลงกลายเป็นสีขาวหรือสีดำ ตัวอย่างดังภาพที่ 5

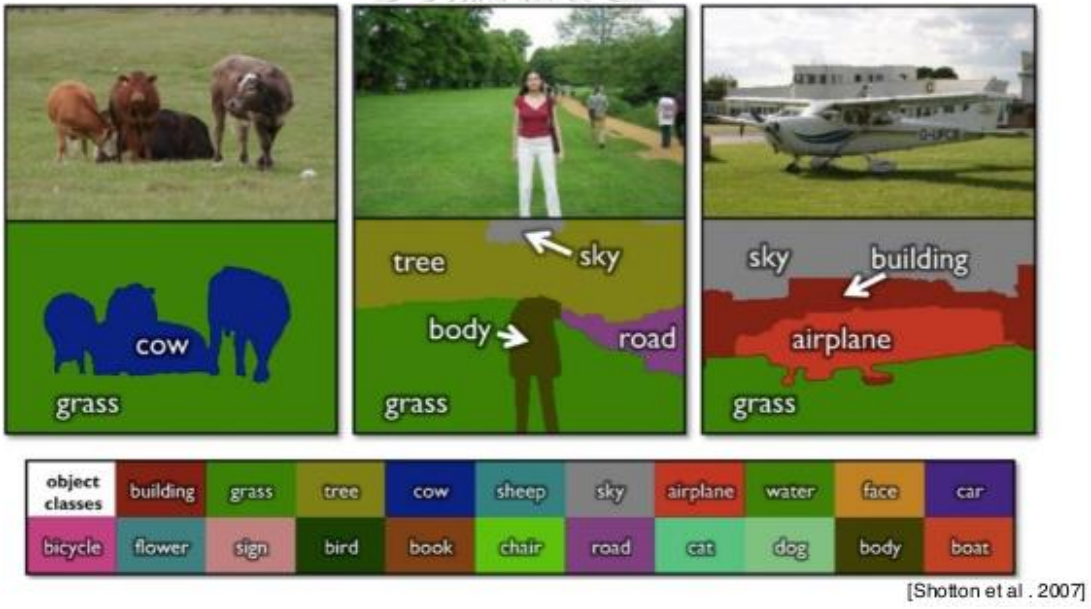
ภาพที่ 5 เปรียบเทียบระหว่างภาพระดับสีเทา (ซ้าย) กับภาพไบนารี (ขวา)

2.4 การเรียนรู้เชิงลึก (Deep Learning)

เป็นรูปแบบการเรียนรู้ของเครื่อง (Machine Learning) รูปแบบหนึ่งที่มีจุดประสงค์ให้คอมพิวเตอร์สามารถเรียนรู้ด้วยตัวเองเหมือนพฤติกรรมการเรียนรู้ของมนุษย์ โดยเรียนรู้จากชุดข้อมูลจำนวนหนึ่ง เพื่อให้คอมพิวเตอร์เข้าใจแพทเทิร์น (Pattern) ของข้อมูล อัลกอริทึมการเรียนรู้เชิงลึกจะมุ่งเน้นการเรียนรู้อย่างต่อเนื่องของข้อมูลผ่านลำดับชั้น (layer) ในการขยายความหมายของความเข้าใจแพทเทิร์นข้อมูล คำว่าลึกของการเรียนรู้เชิงลึกจึงหมายถึงจำนวนชั้นที่ต่อเนื่องกันในแบบจำลอง ลักษณะแบบจำลองการเรียนรู้เชิงลึกสมัยใหม่มักมีจำนวนชั้นตั้งแต่หลักสิบไปจนถึงหลักร้อย

2.5 การแบ่งส่วนตามความหมาย (Semantic segmentation)

การแบ่งส่วนตามความหมาย เป็นปัญหาหนึ่งในการเรียนรู้เชิงลึกที่สนใจการจำแนกคลาสในทุกๆ พิกเซล (pixel) ในภาพที่ 6 ได้แสดงถึงภาพถ่ายที่มีวัตถุหลายชนิดอยู่ในภาพ โดยจะเห็นว่าแต่ละจุดของภาพนั้นจะถูกจัดให้อยู่ในคลาสของวัตถุขึ้นอยู่กับว่าบริเวณนั้นเป็นวัตถุอะไร ซึ่งคลาสที่แตกต่างกันก็จะมีการใช้สีที่ระบุคลาสของวัตถุที่แตกต่างกัน การแบ่งส่วนความหมายสามารถนำไปประยุกต์ใช้กับปัญหาหลากหลายประเภท อาทิ การสร้างรถยนต์ขับเคลื่อนอัตโนมัติ การวิเคราะห์ภาพทางการแพทย์ การวิเคราะห์ภาพในเชิงวัสดุศาสตร์ เป็นต้น สิ่งสำคัญของงานการแบ่งส่วนความหมาย จึงมากไปกว่าการทำนายคลาสของหนึ่งรูปภาพ แต่รวมไปถึงการทราบถึงตำแหน่งของคลาสนั้นในภาพด้วย ซึ่งวิธีการหนึ่งที่เป็นที่รู้จักและใช้กันอย่างกว้างขวางด้วยวิธีการเรียนรู้เชิงลึก คือ โครงข่ายประสาทคอนโวลูชันเต็มรูป (Fully convolutional neural networks)



ภาพที่ 6 การแบ่งส่วนตามความหมายกับรูปภาพที่มีหลายคลาสในภาพ [6]

2.6 Tensor

Tensor เป็นลักษณะโครงสร้างข้อมูลตัวเลขที่นิยมใช้ในการเรียนรู้ของเครื่องและการเรียนรู้เชิงลึก โดยมิติของ Tensor จะขึ้นอยู่กับลักษณะของข้อมูล

2.6.1 Scalars (0D tensor)

คือข้อมูลตัวเลขเพียงแค่ตัวเดียวเรียกว่า tensor 0 มิติ ตัวอย่างดังภาพที่ 7

```
>>> import numpy as np
>>> x = np.array(12)
>>> x
array(12)
>>> x.ndim
0
```

ภาพที่ 7 ข้อมูล Scalars (0D tensor)

2.6.2 Vectors (1D tensor)

คือข้อมูลแถวลำดับ (array) ของตัวเลขหลายตัวเรียกว่า tensor 1 มิติ ตัวอย่างดังภาพที่ 8

```
>>> x = np.array([12, 3, 6, 14])
>>> x
array([12, 3, 6, 14])
>>> x.ndim
1
```

ภาพที่ 8 ข้อมูล Vectors (1D tensor)

2.6.3 Matrices (2D tensor)

คือ ข้อมูลแถวลำดับของ Vectors หลายตัว เรียกว่า tensor 2มิ ติ ตัวอย่างดังภาพที่ 9

```
>>> x = np.array([[5, 78, 2, 34, 0],
                  [6, 79, 3, 35, 1],
                  [7, 80, 4, 36, 2]])
>>> x.ndim
2
```

ภาพที่ 9 ข้อมูล Matrices (2D tensor)

2.6.4 3D tensor และมิติที่สูงขึ้นไป

คือ จำนวนก้อนของเมทริกซ์ที่ซ้อนกัน สำหรับ 3D tensor สามารถมองได้เป็นลูกบาศก์ข้อมูลตัวเลข ตัวอย่างดังภาพที่ 10

```
>>> x = np.array([[[5, 78, 2, 34, 0],
                   [6, 79, 3, 35, 1],
                   [7, 80, 4, 36, 2]],
                  [[5, 78, 2, 34, 0],
                   [6, 79, 3, 35, 1],
                   [7, 80, 4, 36, 2]],
                  [[5, 78, 2, 34, 0],
                   [6, 79, 3, 35, 1],
                   [7, 80, 4, 36, 2]]])
>>> x.ndim
3
```

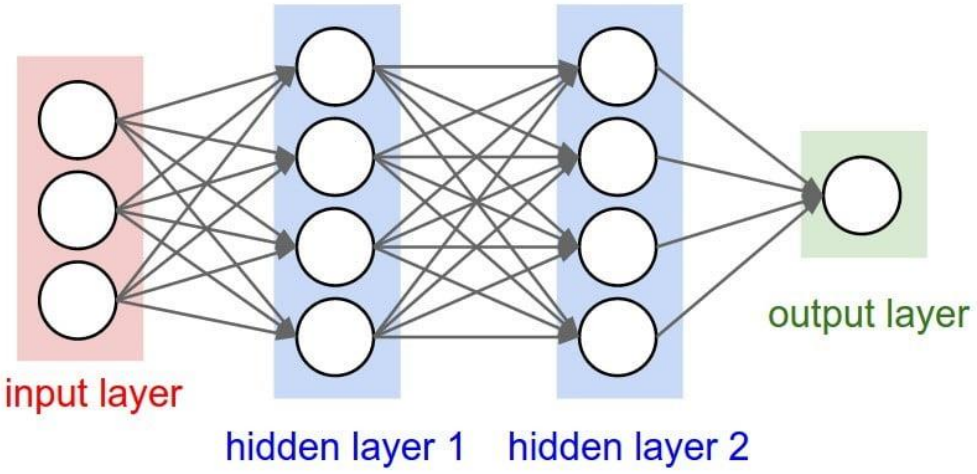
ภาพที่ 10 ข้อมูล Matrices ที่ซ้อนกัน (3D tensor)

2.6.5 ตัวอย่างของข้อมูล tensors ในชีวิตจริง

เช่น Vector data-2D tensor (sample, features), Image-4D tensor (samples, height, width, channels), Video-5D tensor (samples, frames, height, width, channels) เป็นต้น

2.7 โครงข่ายประสาทเทียม (Artificial Neural Network– ANN)

เป็นอัลกอริทึมการเรียนรู้เชิงลึกที่มีลักษณะคล้ายการทำงานของโครงสร้างระบบประสาทของมนุษย์ โดยที่แต่ละเซลล์ประสาทเชื่อมต่อซึ่งกันและกัน มีการเชื่อมต่อส่งข้อมูลระหว่างเซลล์ประสาทแต่ละชั้นในแบบจำลองจะประกอบด้วยจุดเชื่อมต่อ (node) ที่เชื่อมต่อถึงกันหมด (fully connected) ด้วยเส้นเชื่อม (edge) เพื่อรับ-ส่งข้อมูลในระหว่างชั้น ซึ่งแบ่งชั้นของแบบจำลองได้เป็น 3 ชั้นหลักๆคือ ชั้นนำเข้าข้อมูล (input) ชั้นซ่อน (hidden) และชั้นผลลัพธ์ (output) ดังภาพที่ 11



ภาพที่ 11 สถาปัตยกรรมโครงข่ายประสาทเทียม [7]

2.7.1 ฟังก์ชันกระตุ้น (Activation function)

เป็นฟังก์ชันที่ใช้กำหนดขอบเขตค่าของตัวแปรขาออก โดยผลลัพธ์ที่ได้มาจากค่าตัวแปรขาเข้าเมื่อผ่านฟังก์ชัน ซึ่งฟังก์ชั้นกระตุ้นมีอยู่หลากหลายชนิดให้เลือกใช้งาน เช่น ฟังก์ชัน sigmoid ที่จะกำหนดค่าของตัวแปรขาออกให้อยู่ระหว่าง [0,1] ฟังก์ชัน tanh ที่จะกำหนดค่าของตัวแปรขาออกให้อยู่ระหว่าง [-1,1] ฟังก์ชัน relu ที่จะกำหนดค่าของตัวแปรขาออกให้อยู่ระหว่าง [0,∞] เป็นต้น

2.7.2 ค่าน้ำหนัก (weights)

เป็นตัวบ่งบอกถึงความสำคัญของตัวแปรขาเข้า ถูกนำมาใช้เพื่อการถ่วงน้ำหนักค่าของผลรวมเชิงเส้นข้อมูลขาเข้าที่เชื่อมต่อกับเซลล์ประสาทก่อนที่จะนำเข้าไปผ่านฟังก์ชันกระตุ้น ซึ่งค่าน้ำหนักนี้

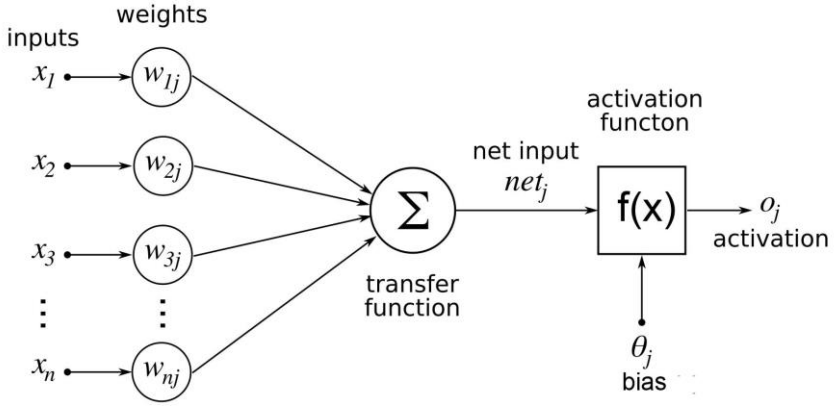
จะถูกปรับเปลี่ยนค่าไปตามการเรียนรู้ของแบบจำลองเพื่อให้ได้ผลลัพธ์สุดท้ายของแบบจำลองมีความใกล้เคียงกับคำตอบที่ต้องการมากที่สุด

2.7.3 ค่าเบี่ยงเบน (bias)

เป็นค่าที่อยู่กับเซลล์ประสาทเทียมแต่ละตัวในแบบจำลอง จะเป็นตัวบ่งบอกถึงความสำคัญของเซลล์ประสาทเทียมตัวนั้น ว่าเป็นจุดเชื่อมต่อที่สำคัญแค่ไหนในแบบจำลอง ซึ่งจะมีความคล้ายคลึงกับค่าน้ำหนักคือ ค่าเบี่ยงเบนจะถูกปรับเปลี่ยนค่าไปตามการเรียนรู้ของแบบจำลองเพื่อให้ได้ผลลัพธ์สุดท้ายของแบบจำลองมีความใกล้เคียงกับคำตอบที่ต้องการมากที่สุดเช่นเดียวกับค่าน้ำหนัก

2.7.4 เพอร์เซปตรอน (Perceptron)

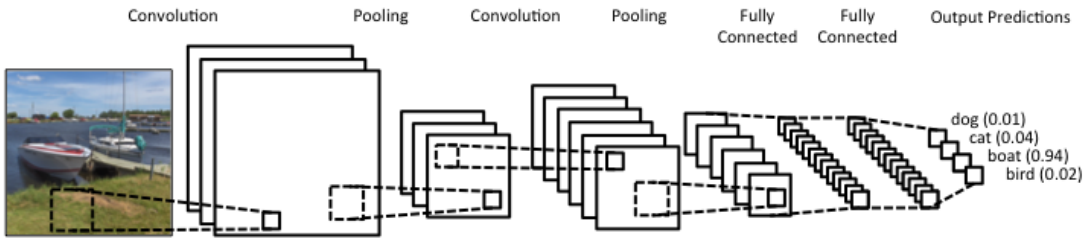
เป็นหน่วยย่อยการเรียนรู้ของโครงข่ายประสาทเทียม ลักษณะดังภาพที่ 12 ประกอบด้วยเวกเตอร์ข้อมูลขาเข้า (x_1, x_2, x_3, ..., x_n) ซึ่งจะถูกนำมาหาผลรวมเชิงเส้นแบบถ่วงน้ำหนัก คูณด้วย w_{1j}, w_{2j}, w_{3j}, ..., w_{nj} ตามลำดับ และเพิ่มด้วยค่าเบี่ยงเบน (θ_j) ก่อนที่จะนำค่าทั้งหมด (net_j) เข้าสู่ฟังก์ชันกระตุ้น f(x) เพื่อสร้างข้อมูลขาออกของเซลล์ประสาท O_j ซึ่งจะถูกนำไปคำนวณหาค่าความคลาดเคลื่อน (error) ในแบบจำลองเมื่อเทียบกับคำตอบที่ต้องการ ค่าความคลาดเคลื่อนที่ได้นี้จะนำมาย้อนคำนวณเพื่อหาค่าน้ำหนักและค่าเบี่ยงเบนของแบบจำลองให้ได้ผลลัพธ์ที่ดีขึ้น ด้วยวิธีที่เรียกว่า backpropagation การทำงานเพื่อปรับค่าน้ำหนักและค่าเบี่ยงเบนของแบบจำลองจะทำงานไปเรื่อยๆจนกว่าค่าความคลาดเคลื่อนจะน้อยมากๆหรือค่าความคลาดเคลื่อนมีการเปลี่ยนแปลงค่าที่น้อยมากๆ แบบจำลองก็จะหยุดการปรับค่าน้ำหนักและค่าเบี่ยงเบน



ภาพที่ 12 โครงสร้างเพอร์เซปตรอน [8]

2.8 Convolutional Neural Network (CNN)

CNN เป็นเทคนิคหนึ่งของการเรียนรู้เชิงลึกที่ถูกพัฒนาขึ้นเพื่อวัตถุประสงค์ในงานการเรียนรู้ด้วยภาพ [9] เนื่องจากลักษณะของภาพในคอมพิวเตอร์ประกอบด้วยพิกเซลหลายๆจุดมาต่อกัน การตีความจากภาพด้วยวิธีการ CNN จึงใช้วิธีเปลี่ยนพิกเซลของรูปภาพให้อยู่ในรูปแบบการเข้ารหัส (encoding) ตัวเลขโดยวิธีการหนึ่งที่นิยมคือใช้ความเข้มสี RGB แทนข้อมูลในแต่ละพิกเซล ซึ่งจะมีค่าได้ตั้งแต่ 0-255 การเข้ารหัสจะประกอบด้วยชั้น convolution layer และชั้น pooling layer ซ้ำๆ กันหลายชั้นซึ่งจะเรียกส่วนนี้ใน CNN ว่า convolutional base แล้วจึงมีชั้นสุดท้ายเป็น Fully Connected Layer ที่รวมข้อมูลทั้งหมดเข้าด้วยกัน ด้วยวิธีการนี้จะสามารถจำแนกประเภท (classify) ภาพได้โดยใช้ข้อมูลขาเข้า (input data) เป็นรูปภาพ และให้การทำนายผลลัพธ์ (output predictions) จากแบบจำลองเป็นความน่าจะเป็นที่รูปภาพจะตรงกับฉลาก (label) กำกับภาพ ดังภาพที่ 13



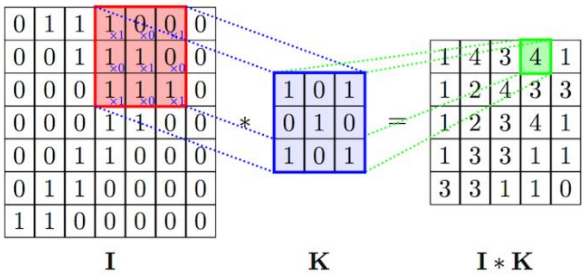
ภาพที่ 13 ตัวอย่างแบบจำลอง Convolutional Neural Network [10]

2.8.1 Convolution Layer

เป็นชั้นที่จะทำการรวมภาพข้อมูลขาเข้ากับ filter ที่มีขนาดเล็กกว่าภาพข้อมูลขาเข้า ตัวอย่างภาพที่ 14 ใช้ filter (K) ขนาด 3x3 ทาบลงไปบนภาพ (I) และทำการเข้ารหัส แล้วจึงขยับ filter ไปที่ตำแหน่งถัดไปซ้ำจนครบทั้งภาพ เพื่อสร้างชุดข้อมูลชุดใหม่ (I*K) ที่มีลักษณะแตกต่างจากชุดข้อมูลเดิม เรียกข้อมูลชุดใหม่ที่ถูกสร้างขึ้นว่า feature map (ตัวอย่างในภาพที่ 15) ในภาพรวมการสร้าง feature map สามารถอธิบายได้จากสมการ (2)

$$(h_k)_{i,j} = (W_k * x)_{i,j} + b_k \tag{2}$$

โดย k แทนจำนวน k-th feature map ใน convolution layer, (i,j) แทนตำแหน่งบน feature map ที่ k-th feature map, x แทนข้อมูลขาเข้า, W_k แทนตัวแปรน้ำหนักของ filter ที่ k-th feature map, b_k แทนค่าเบี่ยงเบนสำหรับที่ k-th feature map

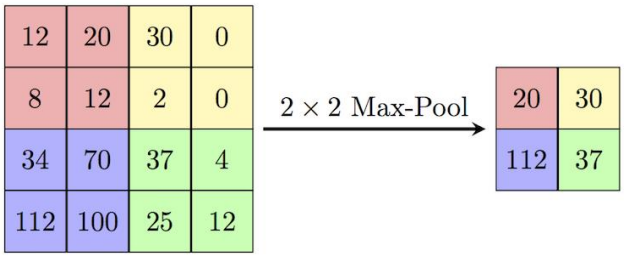


ภาพที่ 14 ตัวอย่างการสร้าง convolution layer โดยกำหนดให้ bias มีค่าเป็น 0 [11]

ภาพที่ 15 ตัวอย่าง feature map ที่สร้างขึ้นในชั้น convolution layer [11]

2.8.2 pooling layer

เป็นชั้นที่ถัดจาก convolution layer ซึ่งจะลดขนาดของ feature map ลงด้วยการหาค่าเฉลี่ย (average pooling) หรือหาค่าที่สูงที่สุด (max pooling) ในบริเวณ sub-region ของข้อมูลที่เข้าชั้นนี้ ภาพที่ 16 แสดงการทำ max pooling โดยให้แต่ละสีใน feature map ขนาด 4x4 แทนบริเวณ sub-region และเลือกค่าที่มากที่สุดในแต่ละ sub-region เพื่อสร้าง feature map ใหม่ขนาด 2x2



ภาพที่ 16 ตัวอย่างการทำ pooling โดยใช้วิธี max pooling [11]

2.8.3 fully connected layer

คือ ชั้นที่ข้อมูลได้มาจากการรวมข้อมูล feature ของชั้นก่อนหน้า ดังสมการ (3)

$$y_k = \sum_l W_{kl} * x_l + b_k \quad (3)$$

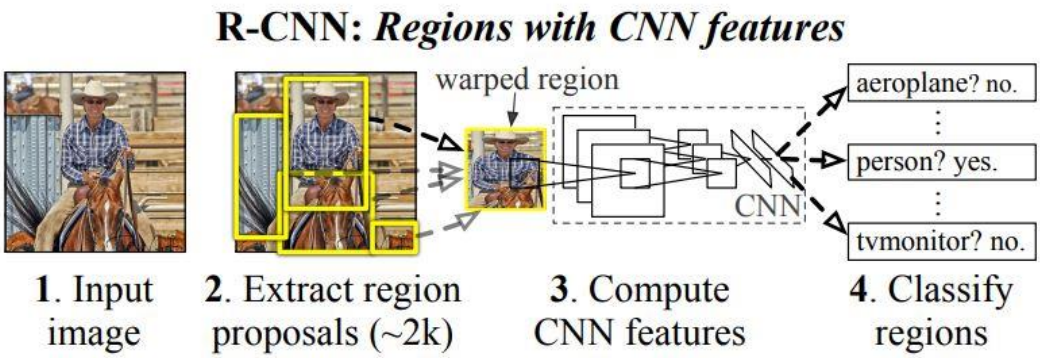
เมื่อ k แทนตำแหน่งข้อมูลขาออกตัวที่ k-th, l แทนตำแหน่งข้อมูลขาเข้า, y_k แทนค่าของข้อมูลขาออก, x_l แทนข้อมูลขาเข้า, b_k แทนเบี่ยงเบนของสมการและ W_{kl} แทนค่าน้ำหนักของข้อมูลระหว่าง x_l กับ y_k ผลลัพธ์ที่ได้จากชั้น fully connected layer จะเป็น output prediction ของภาพที่เรียนรู้ด้วยเทคนิค CNN ซึ่งจะใช้ softmax function เพื่อแปลงค่าการทำนายอยู่ในรูปแบบของความน่าจะเป็น ดังสมการ (4)

$$P(y = j|X; W, b) = \frac{exp^{X_k W_{jk}}}{\sum_{k=1}^{K} exp^{X_k W_{jk}}} \quad (4)$$

เมื่อ P แทนความน่าจะเป็นที่ข้อมูลขาออก y จะถูกคัดแยกเป็น j-th class ด้วยข้อมูลขาเข้า X_k

2.9 Region-based Convolutional Neural Networks (R-CNN)

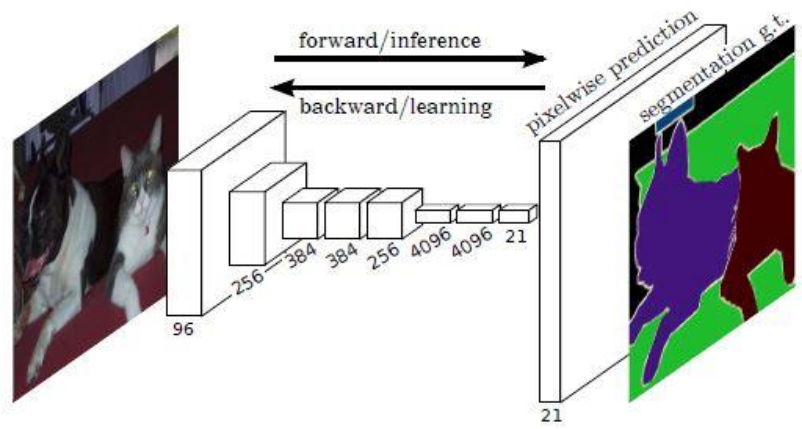
R-CNN เป็นวิธีการประยุกต์แบบจำลอง CNN เพื่อใช้ในการตรวจหาวัตถุ (Object Detection) เมื่อในรูปภาพมีวัตถุที่ต้องการจำแนกมากกว่าหนึ่งวัตถุ โดยจะใช้วิธีสร้างส่วนย่อยในภาพที่เรียกว่า ข้อเสนอภูมิภาค (Region Proposals) ภาพที่ 17 แสดงขั้นตอนการตรวจหาวัตถุหลักๆ ประกอบด้วย 1) นำเข้าภาพ 2) สร้างข้อเสนอภูมิภาคขึ้นประมาณ 2000 ภูมิภาค 3) แต่ละภูมิภาคจะถูกนำเข้าประมวลผลกับลักษณะสำคัญ (features) ของ CNN และ 4) ทำการจำแนกภูมิภาคโดยเลือกเฉพาะข้อเสนอภูมิภาคที่จำแนกประเภทได้ค่าความเชื่อมั่นที่สูง ทำให้เหลือเฉพาะ ข้อเสนอภูมิภาคที่มีวัตถุอยู่ในภูมิภาคเท่านั้น



ภาพที่ 17 การจำแนกภูมิภาคด้วยลักษณะสำคัญของ CNN ในแบบจำลอง R-CNN [12]

2.10 โครงข่ายประสาทคอนโวลูชันเต็มรูป (Fully Convolutional Neural Network–FCNN)

FCNN ถูกสร้างขึ้นด้วยเทคนิค Convolutional Network และ Deconvolutional Network ความแตกต่างระหว่าง CNN กับ FCNN คือ การใช้เทคนิค CNN นั้นจะใช้การเข้ารหัสภาพ โดยคาดหวังให้ขั้นตอนสุดท้ายสามารถจำแนกรูปภาพออกมาได้เป็นชื่อภาพ แต่เป้าหมายของ FCNN คือ การที่จะจำแนกในแต่ละพิกเซล หรือที่เรียกว่าการแบ่งส่วนตามความหมาย จะต้องอาศัยการถอดรหัส (Decoding) ในชั้น Feature Map ของ CNN ซึ่งเป็นเทคนิคที่เรียกว่า Deconvolution ในปี ค.ศ. 2015 Long et al. [13] ได้เผยแพร่ผลงานนำเสนอวิธีการ FCNN โดยได้เพิ่มชั้น Upsampling Layer เข้าไปแทนชั้นของ Fully connected layer และใช้ภาพ Segmentation Ground-Truth เป็นข้อมูลสอนของแบบจำลอง ตัวอย่างดังภาพที่ 18

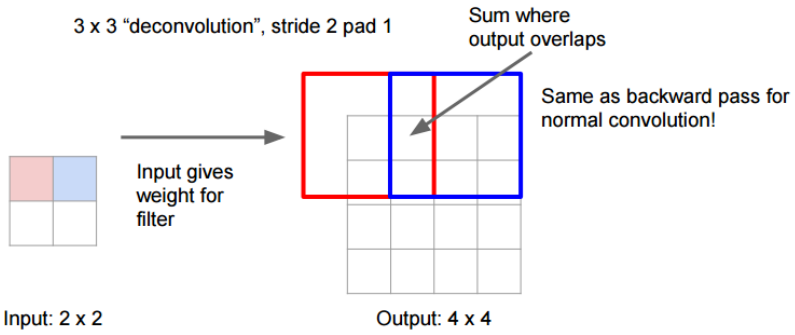


ภาพที่ 18 ตัวอย่างแบบจำลองโครงข่ายประสาทคอนโวลูชันเต็มรูป [13]

2.10.1 Upsampling Layer

คือ ชั้นที่ใช้เทคนิค Deconvolution เพื่อขยายขนาดของ Feature Map ให้ใหญ่ขึ้นเท่ากับรูปภาพดั้งเดิม โดยใช้ข้อมูลขาเข้ามาสร้างเป็น Filter และนำไปทาบกับข้อมูลขาออก โดยให้นำค่าบริเวณส่วนที่ทับซ้อนกันของ Filter ที่ทาบลงไปบนข้อมูลขาออกมารวมกัน ตัวอย่างดังภาพที่ 19

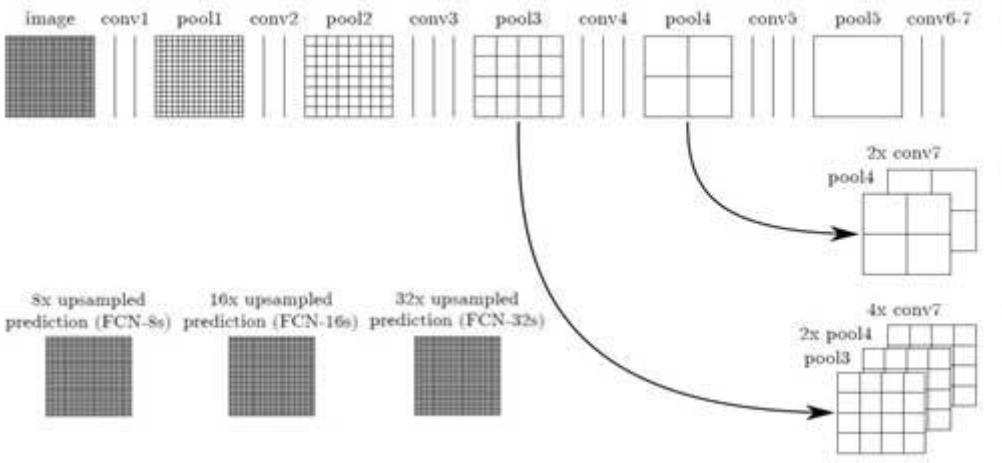
ผลลัพธ์สุดท้ายเมื่อผ่านชั้น Upsampling Layer จะมีขนาดด้านกว้างและด้านยาวเท่ากับรูปภาพดั้งเดิม แต่จะมีจำนวน Channels เท่ากับจำนวนคลาสในการจำแนก ซึ่งจะใช้ Softmax Function ในการจำแนกคลาสในแต่ละพิกเซลภายหลัง



ภาพที่ 19 Deconvolution ในชั้น Upsampling Layer เพื่อขยายขนาดของ Feature Map [14]

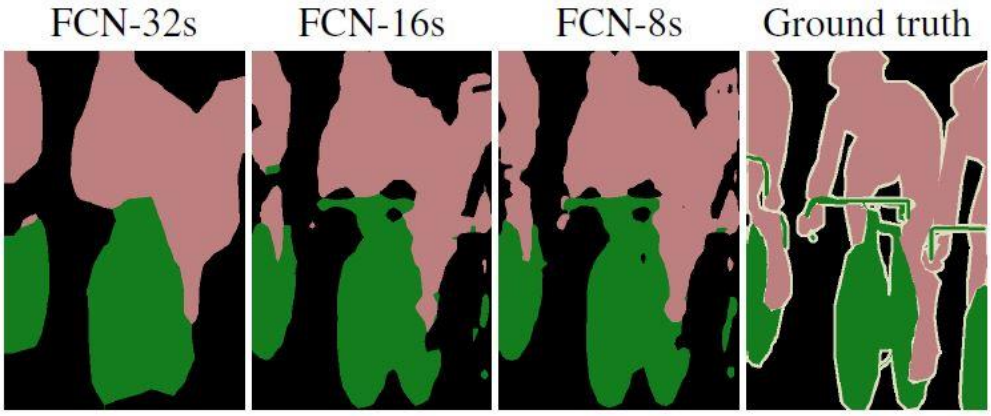
2.10.2 Skip Connections

เสนอโดย Long et al. [13] เพื่อเพิ่มความแม่นยำและแก้ปัญหาการขาดหายของ feature map เมื่อผ่านแบบจำลองในชั้นที่อยู่ลึก เนื่องจากการทำ pooling จะส่งผลให้ข้อมูลนั้นหยาบลง การทำ upsampling ในขั้นตอนสุดท้าย จึงได้ผลลัพธ์ที่ผ่านการเรียนรู้ที่ลึก แต่สามารถถอดรหัสออกมาเป็นพิกเซลได้แบบหยาบ การทำ skip connections คือ การนำเอาชั้น pooling ที่ตื้นกว่าชั้นสุดท้ายมาทำ upsampling ผลลัพธ์ที่ได้จึงสามารถถอดรหัสออกมาได้แบบละเอียดกว่า ช่วยให้การจำแนกขอบในรูปภาพดีขึ้นในบางกรณี ดังภาพที่ 20



ภาพที่ 20 Skip Connections เพื่อ upsampling ในชั้นที่ตื้นกว่าของแบบจำลอง [13]

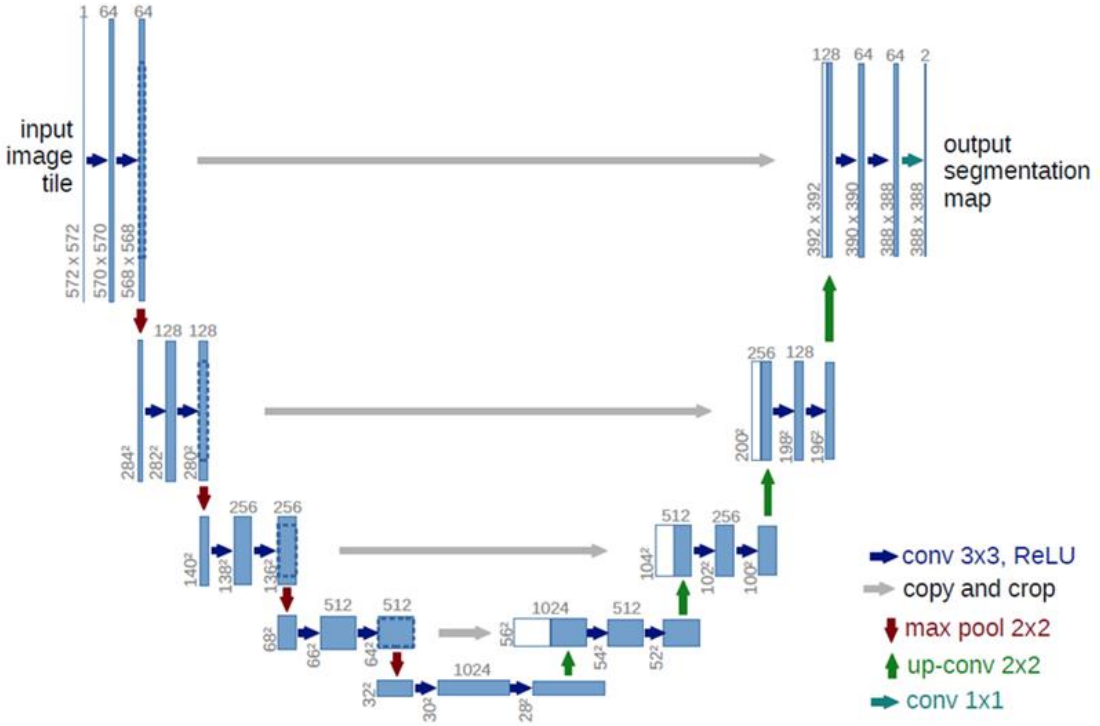
ภาพที่ 21 แสดง FCN-32s, FCN-16s และ FCN-8s ซึ่งได้จากการ upsampling ข้อมูลที่ผ่านชั้น Pool5, Pool4, และ Pool3 ตามลำดับ ผลลัพธ์การจำแนกที่ได้เมื่อเปรียบเทียบกับรูป ground-truth จะเห็นได้ว่าการ upsampling ในชั้นที่ตื้นกว่าของแบบจำลองจะให้รายละเอียดที่มากกว่าเมื่อเทียบกับการ upsampling ในชั้นที่ลึกของแบบจำลอง



ภาพที่ 21 ตัวอย่างผลลัพธ์ที่จำแนกด้วย FCN-32s, FCN-16s และ FCN-8s เมื่อเทียบกับรูปภาพ ground-truth [13]

2.11 สถาปัตยกรรม U-net

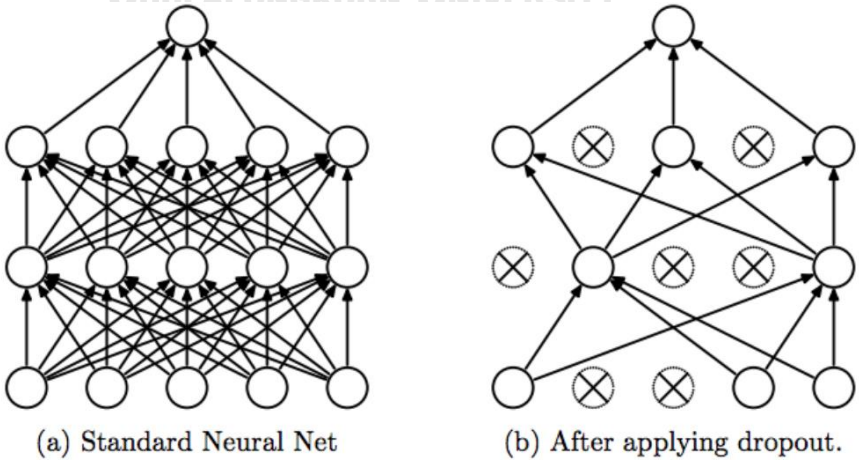
แบบจำลอง FCNN รูปแบบสถาปัตยกรรม U-net เป็นแบบจำลองที่สร้างขึ้นตามมโนทัศน์ FCNN เพื่อใช้ในงานการแบ่งส่วนตามความหมาย นำเสนอโดย Ronneberger et al. [15] แบบจำลองนี้ถูกใช้ครั้งในงานจำแนกภาพทางชีวการแพทย์เพื่อจำแนกเซลล์และเยื่อหุ้มเซลล์ และได้รางวัลชนะเลิศในการแข่งขัน International Symposium on Biomedical Imaging (ISBI) ในปี 2015 ซึ่งเป็นงานแข่งขันจำแนกภาพทางชีวการแพทย์ที่จัดแข่งขึ้นทุกปี ตัวแบบจำลอง U-net นั้นถูกตั้งชื่อตามตัวอักษร U ในภาษาอังกฤษ เนื่องจากตัวแบบจำลองมีลักษณะคล้ายตัว U ดังภาพที่ 22 ตัวสถาปัตยกรรมสามารถมองออกได้เป็นสองส่วนที่แยกกัน ส่วนทางด้านซ้ายเรียกว่า contracting path เป็นส่วนที่แบบจำลองเข้ารหัสรูปภาพขาเข้าให้เป็น feature map ด้วยการวิธีการ convolution และ max-pooling ส่วนทางด้านขวาเรียกว่า expansive path เป็นส่วนที่แบบจำลองถอดรหัสจาก feature map ให้กลับเป็นรูปภาพคำตอบของการจำแนกด้วยวิธีการ upsampling สิ่งที่เพิ่มขึ้นมาของ U-net คือการทำให้ชั้น upsampling ของ U-net จะมีจำนวน channel ของ feature map ที่มากซึ่งแตกต่างกับ FCNN รูปแบบเดิมที่ปกติ upsampling แล้วจำนวน channel จะถูกทำให้ลดลง U-net ได้ใช้วิธีที่เรียกว่า concatenation method เพื่อคัดลอก feature map บางส่วนใน contracting path มารวมกับ feature map ในส่วน expansive path ที่ได้มาจากการ upsampling โดยรวมแล้วแบบจำลองประกอบขึ้นด้วยชั้น convolution จำนวน 23 ชั้น (18 3x3 convolutions, 4 2x2 upsampling convolution, 1 1x1 convolution) และชั้น max-pooling จำนวน 4 ชั้น



ภาพที่ 22 แบบจำลอง FCNN สร้างด้วยสถาปัตยกรรม U-net [15]

2.12 Dropout

ในการเรียนรู้เชิงลึกที่แบบจำลองเรียนรู้ซ้ำหลายๆรอบ มักจะมีความเสี่ยงที่จะเกิดปัญหา over-fitting ซึ่งเป็นปัญหาที่เกิดด้วยเหตุการณ์ที่แบบจำลองสามารถทำงานได้มีความแม่นยำมากกับชุดข้อมูลสอน แต่เมื่อนำไปใช้งานกับชุดข้อมูลสอบแล้ว กลับทำงานได้อย่างไม่มีประสิทธิภาพ การ Dropout จึงเป็นวิธีการหนึ่งที่สามารถนำมาช่วยป้องกัน over-fitting ได้ระดับหนึ่ง



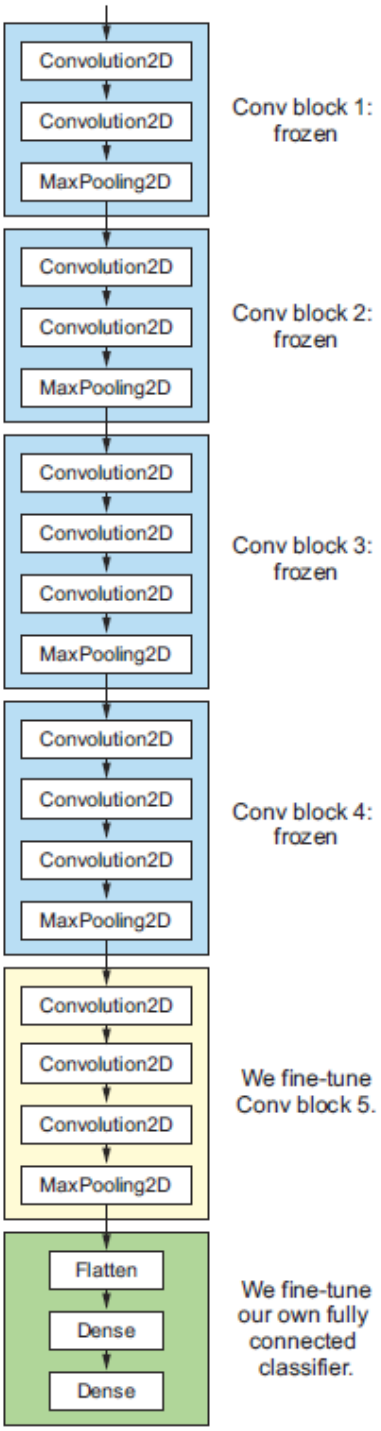
ภาพที่ 23 การ Dropout (a) โครงข่ายประสาทเทียมก่อนการ Dropout (b) โครงข่ายประสาทเทียมหลังการ Dropout [16]

คำว่า Dropout ในที่นี้หมายถึง การมองข้ามซึ่งในเชิงการเรียนรู้เชิงลึกคือการมองข้ามหน่วยประสาทเทียมบางตัว (ทั้งในชั้น hidden layer และชั้นอื่นๆ) ในภาพที่ 23 แบบจำลองที่ถูก Dropout ภาพที่ 23(b) จะต้องเรียนรู้ให้ได้คำตอบเหมือนในแบบจำลองเดิมภาพที่ 23(a) โดยการ Dropout จะทำด้วยการสุ่มหน่วยประสาทเทียมที่จะถูกมองข้ามไปด้วยความน่าจะเป็น 1-p เมื่อ p แทนค่าของความน่าจะเป็นที่จะใช้ในการ Dropout หน่วยประสาทเทียมที่ถูกมองข้ามไปจะถูกลบออกไปทั้งจุดเชื่อมต่อและเส้นเชื่อมต่อดังในภาพที่ 23(b) ทำให้การเชื่อมต่อภายในแบบจำลองขาดหายไปบางส่วน การย้อนคำนวณด้วย back propagation ในแบบจำลองจึงต้องพยายามเปลี่ยนค่าตัวแปรภายในของหน่วยประสาทเทียมที่ยังเหลืออยู่เพื่อให้คำตอบของแบบจำลองยังคงได้คำตอบที่เหมือนเดิม

2.13 Fine-tuning

การเรียนรู้แบบจำลองที่ต้องใช้ชุดข้อมูลสอนที่น้อยเป็นสถานการณ์ที่พบเจอได้ทั่วไปและมักจะเกิดปัญหาเรื่องความแม่นยำของผลลัพธ์การทำนาย ในการเรียนรู้ที่เริ่มต้นจากที่ไม่มีความรู้ก่อนหน้า (learning from scratch) แบบจำลองจะเริ่มต้นการเรียนรู้โดยที่ไม่ทราบอะไรมาก่อนเลย เปรียบเสมือนเด็กทารกไร้เดียงสาที่กำลังหัดเรียนรู้ในสิ่งต่างๆ การเรียนรู้ที่ได้ของแบบจำลองจะมาจากชุดข้อมูลที่ถูกใส่เข้าไปในแบบจำลองเพียงอย่างเดียว ในขณะที่การเรียนรู้แบบ fine-tuning จะใช้ pretrained weights เปรียบได้กับความรู้ที่ได้มาจากการเรียนรู้ของแบบจำลองที่มีสถาปัตยกรรมแบบจำลองเดียวกันแต่อาจจะถูกนำไปใช้เรียนรู้มาแล้วกับชุดข้อมูลอื่น มาช่วยในการเรียนรู้ของแบบจำลองที่ต้องการจะเรียนรู้ใหม่ได้ โดยตัว pretrained weights นี้ มักจะถูกเรียนรู้มากับชุดข้อมูลที่มีขนาดใหญ่มากเช่น ชุดข้อมูล Imagenet ที่เป็นชุดข้อมูลภาพขนาดใหญ่ที่มีจำนวนภาพมากกว่า 14ล้านภาพ และมีจำนวนคลาสมากกว่า 1000 คลาส โดย pretrained weights สามารถสืบค้นได้จากอินเทอร์เน็ตที่ถูกเรียนรู้ด้วยสถาปัตยกรรมของแบบจำลองที่เป็นที่รู้จักทั่วไป อาทิ AlexNet, VGG16, ResNet, Xception เป็นต้น ข้อสงสัยที่มักจะถูกถามถึงเกี่ยวกับวิธีการ fine-tuning คือการนำ weights ของแบบจำลองที่เรียนรู้ด้วยชุดข้อมูลอื่น ทั้งที่เป็นคนละคลาสกัน ทำไมจึงนำมาใช้กับชุดข้อมูลอีกประเภทหนึ่งได้นั้น คำตอบก็คือ pretrained weights ที่นำมาใช้ fine-tuning มักจะใช้แค่บางส่วนที่ตัว weights เอง ยังไม่เกิดการกระตุ้นกับส่วนที่ซับซ้อนในภาพ โดยจะเป็นส่วนที่ตัว weights เกิดการกระตุ้นกับสิ่งที่ไม่ซับซ้อน เช่น ขอบของภาพ หรือส่วนโค้งในภาพ ซึ่งจะเป็นส่วนชั้นแรกๆในแบบจำลองการเรียนรู้เชิงลึก ในส่วนที่ลึกๆเข้าไปจะเป็นการเรียนรู้ที่ซับซ้อนขึ้นเรื่อยๆที่เป็นลักษณะเฉพาะของชุดข้อมูลนั้นๆจะถูกตัดทิ้งออกไประหว่าง fine-tuning และจะถูกนำมาเรียนรู้ใหม่กับชุดข้อมูลใหม่ ภาพที่ 24 แสดงตัวอย่างการทำ fine-tuning กับแบบจำลอง CNN ที่ใช้

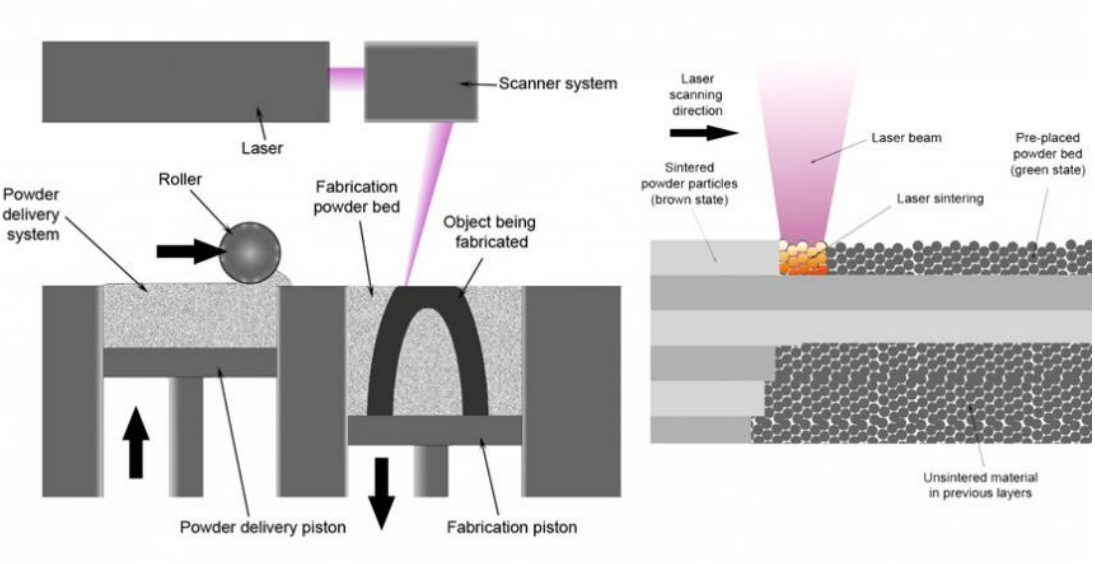
pretrained weights กับ convolution block ที่ 1-4 และทำการคงค่า weights นั้นไว้ ไม่เปลี่ยนแปลงเมื่อนำมาสอนชุดข้อมูลใหม่ ซึ่งจะเกิดการเรียนรู้กับชุดข้อมูลใหม่ๆได้เฉพาะจาก convolution block ที่ 5 เป็นต้นไป



ภาพที่ 24 ตัวอย่างการทำ fine-tuning กับแบบจำลอง CNN [17]

2.14 Selective Laser Melting (SLM)

SLM เป็นกระบวนการขึ้นรูปสามมิติ (ภาพที่ 25) โดยใช้วัตถุดิบที่อยู่ในรูปของผง สามารถขึ้นรูปวัสดุได้หลายประเภท อาทิ โลหะ โพลิเมอร์ และเซรามิก เป็นต้น ซึ่งจะใช้ลำแสงที่มีความหนาแน่นพลังงาน (Energy Density) สูงฉายลงไปบนผิวของวัตถุดิบผงทำให้ส่วนที่ถูกลำแสงเกิดการหลอมละลาย โดยจะทำการฉายลำแสงไล่ทีละชั้นและพ่นผงวัตถุดิบลงไปหลอมละลายจนได้เป็นชิ้นงานที่ต้องการ



ภาพที่ 25 กระบวนการ Selective Laser Melting [18]

2.15 งานวิจัยที่เกี่ยวข้อง

2.15.1 Advanced Steel Microstructural Classification by Deep Learning Methods [19]

Azimi et al. (2018) ได้นำเทคนิคการเรียนรู้เชิงลึกมาใช้กับภาพโครงสร้างจุลภาคของเหล็กกล้าคาร์บอนต่ำโดยต้องการที่จะจำแนกเฟสของโครงสร้างจุลภาคให้ได้ในทุกพิกเซลของภาพ จึงใช้เทคนิค Fully Convolution Neural Networks ซึ่งให้ผลลัพธ์ความแม่นยำของแบบจำลองได้ถึง 93.94%

2.15.2 Selective Laser Melting Produced Ti-6Al-4V:Post-Process Heat Treatments to Achieve Superior Tensile Properties [20]

Terhaar และ Becker (2018) ได้ศึกษากระบวนการให้ความร้อนแก่โลหะ Ti-6Al-4V ที่ขึ้นรูปด้วยวิธี Selective Laser Melting เพื่อปรับปรุงสมบัติทางกลของวัสดุให้รับแรงดึงได้มากขึ้น ด้วย

วิธีทดสอบสมบัติทางกลและศึกษาการเปลี่ยนแปลงของโครงสร้างจุลภาค ค้นพบว่าด้วยกระบวนการให้ความร้อน จะส่งผลให้เฟสของ Ti-6Al-4V เกิดการเปลี่ยนแปลงทั้งสัดส่วนและโครงสร้างของเฟส โดยที่สัดส่วนของเฟสที่เหมาะสมจะส่งผลให้วัสดุมีสมบัติทางกลที่ดีขึ้น

2.15.3 Fully Convolutional Networks for Semantic Segmentation [13]

Long et al. (2015) ได้นำเสนอแบบจำลองโครงข่ายประสาทคอนโวลูชันเต็มรูปเพื่องานด้านการแบ่งส่วนตามความหมาย และประสบความสำเร็จในการนำไปใช้ปรับเปลี่ยนแบบจำลองที่เป็นที่รู้จักแพร่หลาย ได้แก่ AlexNet, VGG16, และ GoogLeNet โดยแทนที่ชั้นสุดท้ายของแบบจำลองด้วยชั้น upsampling layer แต่ก็มีปัญหาเนื่องจากการ upsampling จากชั้นสุดท้ายของแบบจำลองส่งผลให้การจำแนกขาดความแม่นยำ เนื่องจากข้อมูลถูกทำให้หยาบลงเมื่อผ่านชั้นที่ลึกในแบบจำลอง จึงต้องทำ Skip Connection เพื่อนำชั้นที่อยู่ตื้นกว่าในแบบจำลองมาทำ upsampling เพื่อให้ได้ผลลัพธ์ที่แม่นยำขึ้น

2.15.4 U-net: Convolutional Networks for Biomedical Image Segmentation [15]

Ronneberger et al. (2015) นำเสนอสถาปัตยกรรม U-net โดยกระยุกต์จากแนวคิดของแบบจำลอง FCNN โดยแบบจำลอง U-net ถูกใช้ครั้งในงานจำแนกภาพทางชีวการแพทย์เพื่อจำแนกเซลล์และเยื่อหุ้มเซลล์ และได้รางวัลชนะเลิศในการแข่งขัน International Symposium on Biomedical Imaging (ISBI) ในปี 2015

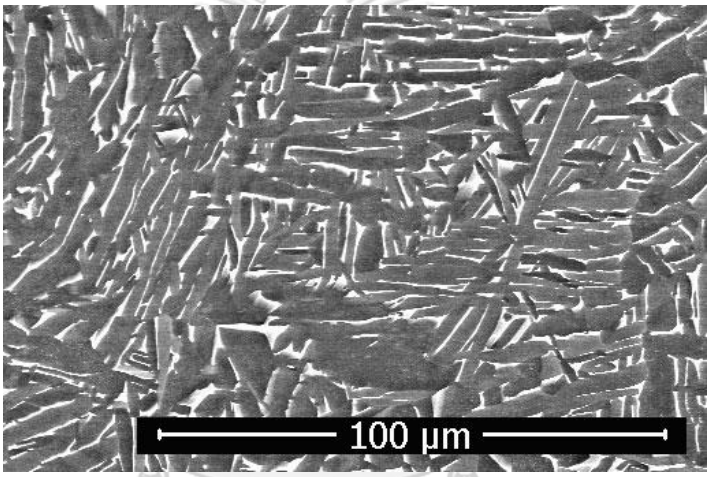
2.15.5 TernausNet: U-net with VGG11 Encoder Pre-trained on ImageNet for Image Segmentation [21]

Iglovikov และ Shvets (2018) ได้นำเสนอการนำเทคนิค fine tuning มาทำบนสถาปัตยกรรม U-net ด้วยจุดประสงค์ที่ต้องการเพิ่มความแม่นยำของแบบจำลองด้วย pretrained weights ที่ผ่านการเรียนรู้มาจากชุดข้อมูลขนาดใหญ่ ด้วยการใช้แบบจำลอง VGG11 เข้ามาทดแทนส่วน contracting path ใน U-net โดยได้นำแบบจำลองนี้ไปทดสอบกับชุดข้อมูล Inria Aerial Image Labeling ซึ่งเป็นชุดข้อมูลที่เป็นภาพถ่ายมุมสูงของอาคารที่มี ground truth เป็นภาพแยกสีของส่วนที่เป็นอาคารและไม่ใช่อาคารในภาพ

บทที่ 3
แนวคิดและวิธีวิจัย

3.1 ภาพรวมแนวคิดงานวิจัย

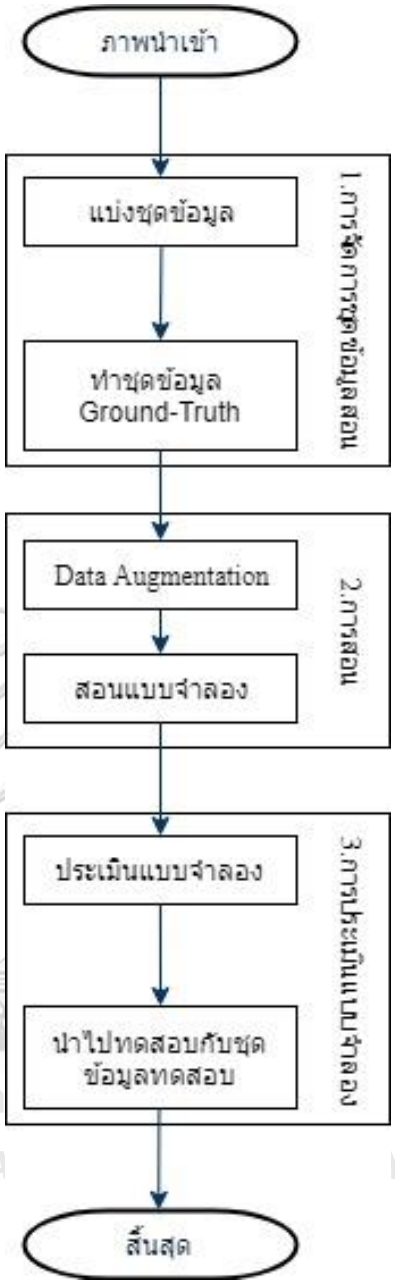
งานวิจัยนี้มุ่งเน้นที่จะศึกษาการเรียนรู้เชิงลึก FCNN สำหรับการจำแนกเฟสในโครงสร้างจุลภาคของโลหะ Ti-6Al-4V ที่ผ่านกระบวนการให้ความร้อนหลังการขึ้นรูปด้วยวิธี Selective Laser Melting โดยจะดำเนินการสร้างแบบจำลองด้วยวิธีดังกล่าวข้างต้น จากชุดข้อมูลภาพโครงสร้างจุลภาค Ti-6Al-4V ภาพที่ 26 แสดงลักษณะข้อมูลสอนของภาพโครงสร้างจุลภาคของ Ti-6Al-4V ซึ่งถ่ายด้วยกล้องจุลทรรศน์อิเล็กตรอนแบบส่องกราด (Scanning Electron Microscope) ประกอบด้วยเฟสที่ต้องการจำแนก 2 ประเภท คือ เฟส alpha (สีเข้ม) และเฟส beta (สีอ่อน)



ภาพที่ 26 โครงสร้างจุลภาคของ Ti-6Al-4V ที่ขึ้นรูปด้วยกระบวนการ SLM

ภาพโครงสร้างจุลภาคได้มาจากบริษัท Meticuly ที่ดำเนินธุรกิจผลิตชิ้นส่วนกระดูกโลหะเทียมจากไทเทเนียม โดยเป็นภาพระดับสีเทาที่มีความละเอียด 512 x 512 พิกเซล จำนวน 96 ภาพ โดยในงานวิจัยนี้ได้เลือกเอาเฉพาะภาพโครงสร้างจุลภาคของไทเทเนียมที่อยู่ในรูปแบบ dual phase มีเพียง 2 เฟสในภาพและถ่ายที่กำลังขยาย 1000 เท่าด้วยกล้องจุลทรรศน์อิเล็กตรอนแบบส่องกราด

กระบวนการสร้างแบบจำลองเพื่อจำแนกโครงสร้างจุลภาคมี 3 ขั้นตอนด้วยกันคือ 1.การจัดการชุดข้อมูลสอน 2.การสอน 3.การประเมินแบบจำลอง ซึ่งจะดำเนินงานเป็นไปตามภาพรวมของกระบวนการในภาพที่ 27



ภาพที่ 27 ภาพรวมกระบวนการสร้างแบบจำลองเพื่อจำแนกโครงสร้างจุลภาค

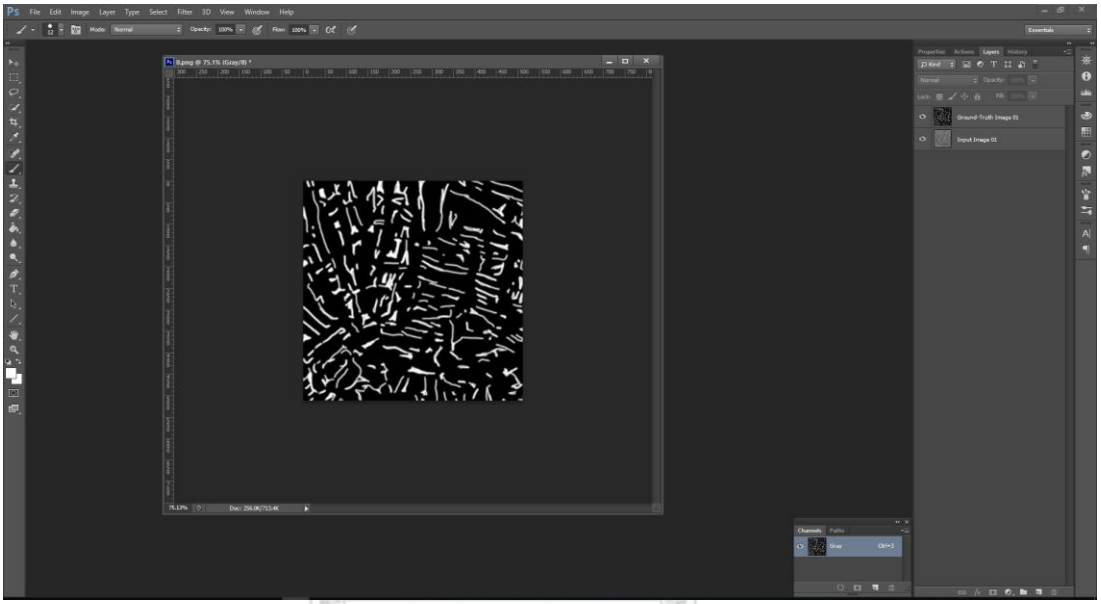
3.2 การจัดการชุดข้อมูลสอน

3.2.1 การแบ่งชุดข้อมูล

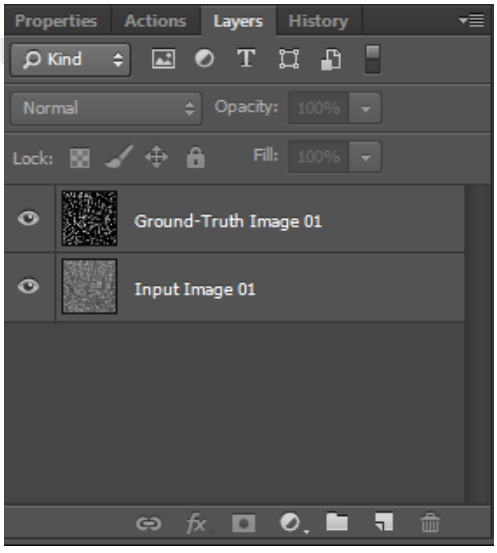
ชุดข้อมูลภาพทั้งหมด 96 ภาพถูกแบ่งออกเป็นชุดข้อมูลสอน 66 ภาพ และชุดข้อมูลทดสอบจำนวน 30 ภาพ โดยที่ภาพข้อมูลสอนจะนำไปทำชุดข้อมูล Ground-truth สำหรับงานการแบ่งส่วนตามความหมาย เพื่อเป็นคำตอบของแบบจำลองในการนำไปสอนแบบจำลองเพื่อจำแนกเฟสของโครงสร้างจุลภาคไทเทเนียม

3.2.2 การทำ Ground-Truth

ภาพที่ 28 แสดงเครื่องมือโปรแกรม photoshop ที่ใช้สร้างภาพ Ground-truth ทีละภาพที่ตรงกับภาพข้อมูลสอน ซึ่งโปรแกรม photoshop มีความสะดวกในการจัดการรูปภาพ เนื่องจากภาพ ground-truth จะถูกสร้างขึ้นมาจากภาพโครงสร้างจุลภาค ตัวโปรแกรมสามารถสร้างชั้น layer ดังภาพที่ 29 ซ้อนขึ้นมาอยู่บนภาพโครงสร้างไทเทเนียมเพื่อใช้ลงสีด้วยเครื่องมือ brush tool ได้ โดยไม่แก้ไขภาพโครงสร้างจุลภาคต้นฉบับ



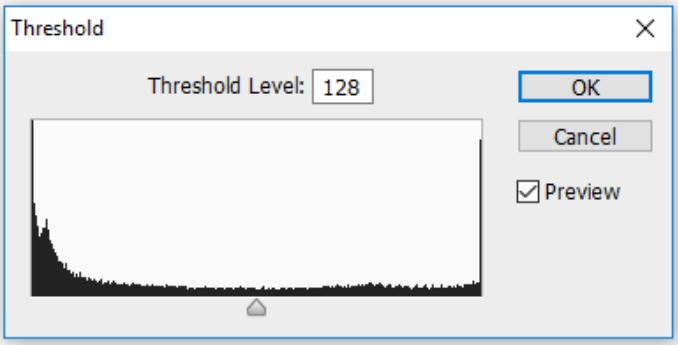
ภาพที่ 28 โปรแกรม photoshop



ภาพที่ 29 หน้าต่างแสดงชั้น layer ที่อยู่ในรูปภาพ

3.2.3 การทำภาพไบนารี ground-truth

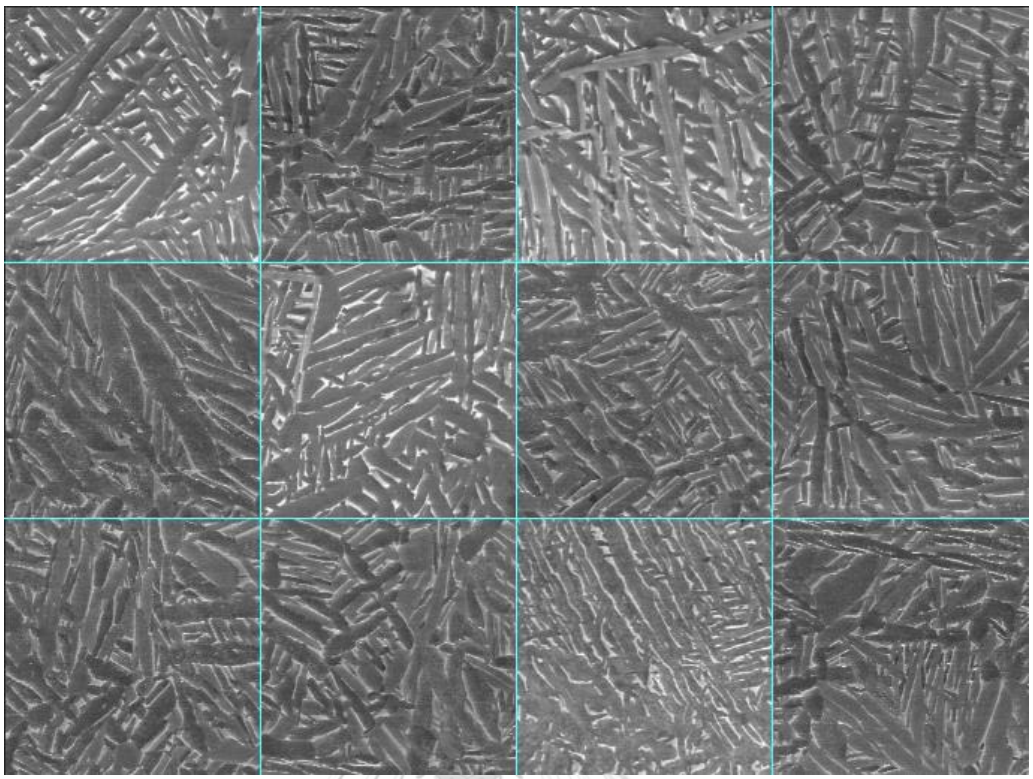
เมื่อลงสีภาพ ground-truth แล้วเสร็จ จะทำการแปลงภาพ ground-truth ให้เป็นภาพไบนารีเพื่อให้เหมาะสำหรับการนำไปใช้ในแบบจำลองที่จะจำแนกเพียง 2 คลาส โดยในโปรแกรม photoshop สามารถเลือกใช้คำสั่ง Threshold ภาพที่ 30 ในการเลือกค่าขีดแบ่ง (Threshold) เพื่อกำหนดค่าระดับความเข้มสีเทาที่จะถูกแปลงกลายเป็นสีขาวหรือสีดำ โดยเลือกที่แถบเมนูด้านบนของโปรแกรม Image -> Adjustments -> Threshold... ซึ่งค่าขีดแบ่งในการแปลงภาพไบนารี ground-truth จะถูกตั้งค่าไว้ที่ 128 ตัวอย่างของภาพไบนารี ground-truth ดังภาพที่ 32 ซึ่งเป็นการระบุคลาสของตัวอย่างภาพโครงสร้างจุลภาคในภาพที่ 31 โดยใช้สีขาวแทนจุดภาพที่พื้นที่ในภาพเป็นเฟส beta และใช้สีดำแทนจุดภาพที่พื้นที่ในภาพเป็นเฟส alpha ตามตารางที่ 1 ซึ่งได้สรุปค่าตัวแปรต่างๆ ที่ใช้สร้างภาพไบนารี ground-truth



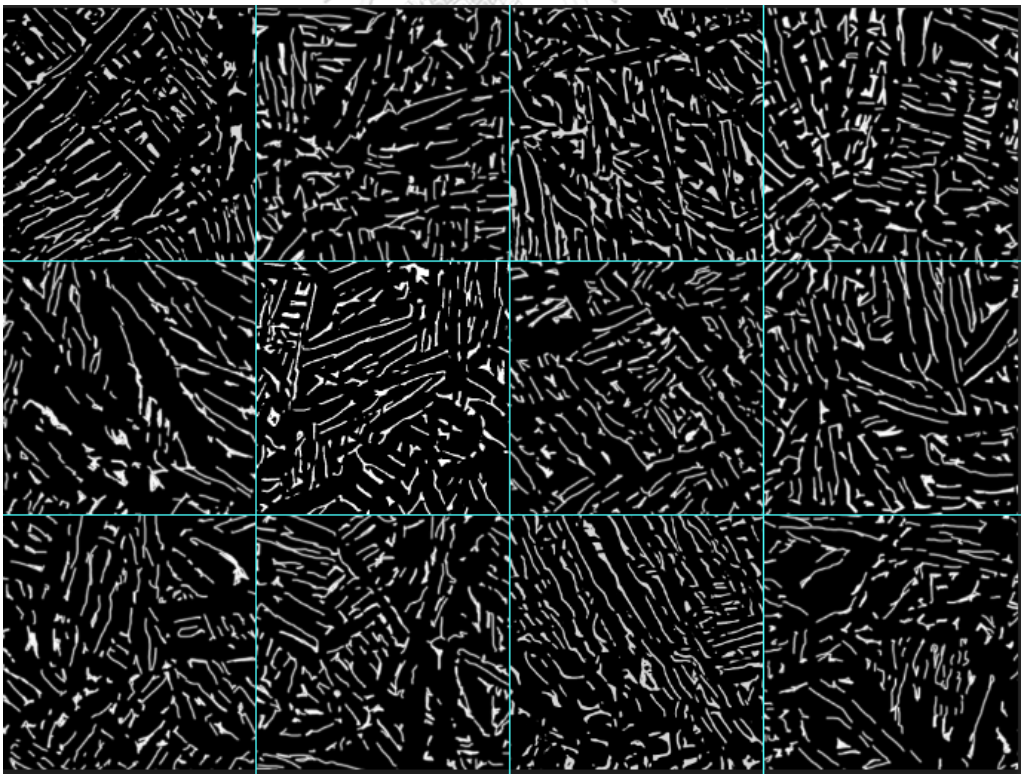
ภาพที่ 30 หน้าต่างคำสั่ง Threshold

ตารางที่ 1 ตัวแปรและค่าที่ใช้ในการสร้างภาพไบนารี ground-truth

ตัวแปร	ค่า
ขนาดรูปภาพ	512 พิกเซล x 512 พิกเซล
จำนวนรูปภาพ	96 รูป
สีของ alpha	(255,255,255)
สีของ beta	(0,0,0)
ค่า threshold ที่ใช้สร้างภาพไบนารี	128



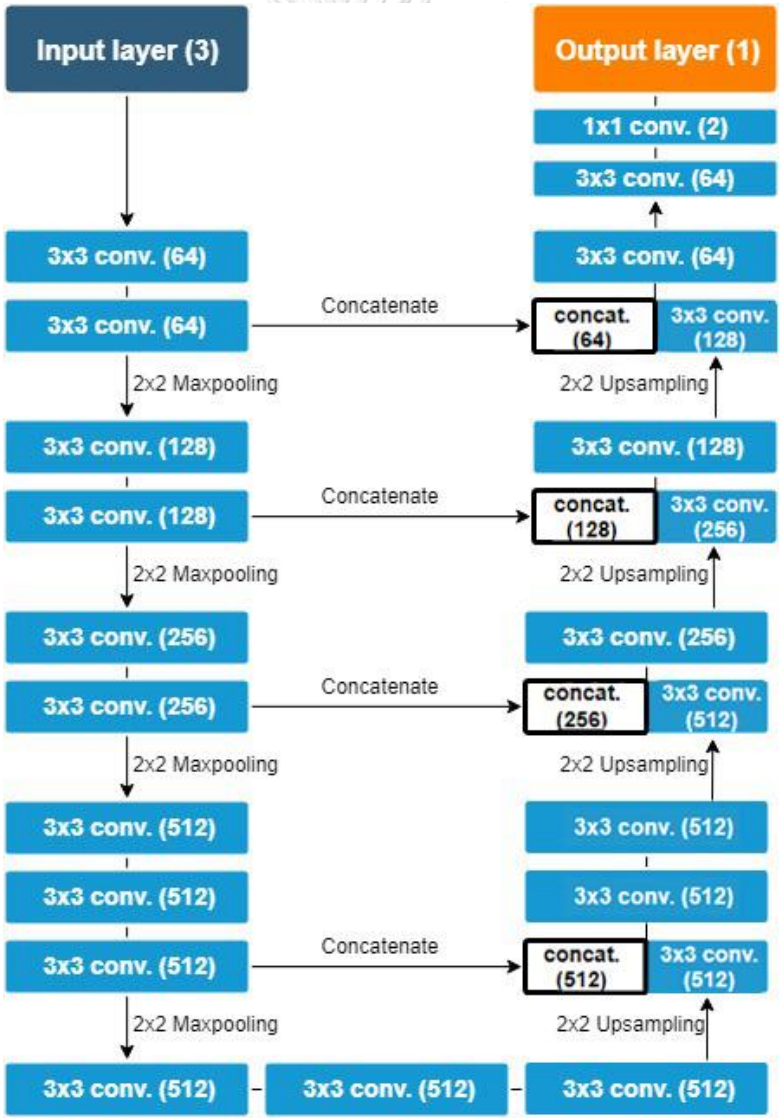
ภาพที่ 31 ตัวอย่างภาพโครงสร้างจุลภาค Ti-6Al-4V ที่มีเฟส alpha และเฟส beta



ภาพที่ 32 ตัวอย่างภาพไบนารี ground-truth ของภาพโครงสร้างจุลภาค

3.3 ภาพรวมแบบจำลอง

สถาปัตยกรรมที่เลือกใช้ คือ การบูรณาการระหว่าง VGG16 และ U-net ดังภาพที่ 33 ซึ่งโดยรวมแบบจำลองจะยังคงเป็นสถาปัตยกรรม U-net แต่จะเลือกใช้ส่วนที่เป็น convolution base ของ VGG16 เข้ามาแทนที่ส่วน contracting path (ด้านซ้าย) ของ U-net และปรับส่วน expansive path ให้สอดคล้องกับส่วน contracting path เพื่อให้สามารถนำเอา pretrained weights ที่ถูกเรียนรู้ด้วยแบบจำลอง VGG16 มาใช้งานได้ ซึ่ง pretrained weights นี้ได้นำมาจาก Github repository ของ Francois Chollet ผู้ที่สร้าง library Keras โดยตัว pretrained weights นี้สามารถดาวน์โหลดได้ทั้งในรูปแบบที่มีส่วน classifier หรือไม่มี classifier



ภาพที่ 33 สถาปัตยกรรมบูรณาการระหว่าง VGG16 และ U-net ที่สร้างขึ้นในงานวิจัย

บทที่ 4
การพัฒนาเครื่องมือและซอฟต์แวร์

4.1 สภาพแวดล้อมและเครื่องมือที่ใช้ในการพัฒนา

4.1.1 สภาพแวดล้อม

1. ระบบปฏิบัติการ Windows 10 Enterprise แบบ 64 บิต
2. หน่วยประมวลผล CPU Intel(R) Processor Core(TM) i5-6400 2.70GHz (4CPUs)
3. หน่วยความจำ 8 กิกะไบต์ (RAM 8 GB)
4. หน่วยประมวลผลกราฟฟิก GPU NVidia GeForce GTX 1060 (6GB)

4.1.2 เครื่องมือที่ใช้ในการพัฒนา

1. Python 3.6.4
2. Anaconda3 (64 บิต) เพื่อใช้ Jupyter Notebook และ Library พื้นฐานต่างๆ ของ Python เช่น Numpy Pip Wheel เป็นต้น
3. Tensorflow-gpu 1.5.0 เพื่อใช้เป็น backend ของ Keras
4. Keras 2.1.4
5. Scikit-Learn 0.14.0
6. OpenCV-python 3.4.2
7. Matplotlib 2.1.2
8. Pillow 5.0.0

4.2 การพัฒนาโปรแกรม

4.2.1 การพัฒนาตัวจัดการข้อมูล

เนื่องจากการนำเข้าข้อมูลจำเป็นที่จะต้องแปลงข้อมูลภาพให้เป็นแถวลำดับ 2 มิติที่พร้อมใช้งานในแบบจำลอง ซึ่งฟังก์ชัน adjustData จะทำการเข้ารหัสแต่ละพิกเซลเข้ากับคลาสของภาพ ground-truth ซึ่งผลลัพธ์ที่ได้ออกมาจะเป็นชุดข้อมูลแถวลำดับ 2 มิติ 2 ชุด โดยเป็นข้อมูลของภาพนำเข้าและภาพ ground-truth ภาพที่ 34 แสดงชุดคำสั่งในการสร้างฟังก์ชัน adjustData การจัดการชุดข้อมูลสอนจะกระทำโดยฟังก์ชัน trainGenerator ซึ่งจะทำการ import library keras ในการทำ Data augmentation โดยที่จะรับข้อมูลตัวแปร aug_dict ในรูปแบบ dictionary ในการเปลี่ยนแปลงชุดข้อมูล ภาพที่ 35 แสดงชุดคำสั่งในการสร้างฟังก์ชัน trainGenerator

```
from __future__ import print_function
from keras.preprocessing.image import ImageDataGenerator
import numpy as np
import os
import glob
import skimage.io as io
import skimage.transform as trans
import cv2
def adjustData(img,mask,flag_multi_class,num_class,numpic):
    if(flag_multi_class):
        img = img / 255
        mask = mask[:,:,:,0] if(len(mask.shape) == numpic) else mask[:,:,0]
        new_mask = np.zeros(mask.shape + (num_class,))
        for i in range(num_class):
            new_mask[mask == i,i] = 1
        new_mask =
np.reshape(new_mask,(new_mask.shape[0],new_mask.shape[1]*new_mask.shape[
2],new_mask.shape[3])) if flag_multi_class else
np.reshape(new_mask,(new_mask.shape[0]*new_mask.shape[1],new_mask.shape[
2]))
        mask = new_mask
    elif(np.max(img) > 1):
        img = img / 255
        mask = mask /255
        mask[mask > 0.5] = 1
mask[mask <= 0.5] = 0
    return (img,mask)
```

ภาพที่ 34 ชุดคำสั่งในการสร้างฟังก์ชัน adjustData

```
def trainGenerator(batch_size,train_path,image_folder,mask_folder,
        aug_dict,image_color_mode="rgb",mask_color_mode="grayscale",
        image_save_prefix  = "image",mask_save_prefix  = "mask",
        flag_multi_class = False,num_class = 2,
        save_to_dir = None,target_size = (512,512),seed = 1):
    image_datagen = ImageDataGenerator(**aug_dict)
    mask_datagen = ImageDataGenerator(**aug_dict)
    image_generator = image_datagen.flow_from_directory(
        train_path,
        classes = [image_folder],
        class_mode = None,
        color_mode = image_color_mode,
        target_size = target_size,
        batch_size = batch_size,
        save_to_dir = save_to_dir,
        save_prefix  = image_save_prefix,
        seed = seed)
    mask_generator = mask_datagen.flow_from_directory(
        train_path,
        classes = [mask_folder],
        class_mode = None,
        color_mode = mask_color_mode,
        target_size = target_size,
        batch_size = batch_size,
        save_to_dir = save_to_dir,
        save_prefix  = mask_save_prefix,
        seed = seed)
    train_generator = zip(image_generator, mask_generator)
    for (img,mask) in train_generator:
        img,mask = adjustData(img,mask,flag_multi_class,num_class)
        yield (img,mask)
```

ภาพที่ 35 ชุดคำสั่งในการสร้างฟังก์ชัน trainGenerator

การจัดการชุดข้อมูลทดสอบจะถูกจัดการโดยฟังก์ชัน testGenerator ซึ่งจะรับค่าเป็น directory ของชุดภาพข้อมูลทดสอบและจำนวนของภาพทดสอบ จะทำการอ่านค่าความเข้มสีของภาพข้อมูลทดสอบทีละภาพและคืนค่าเป็นแถวลำดับ 2 มิติ ตามชุดคำสั่งในภาพที่ 36

```
def testGenerator(test_path,num_image ):
    for i in range(num_image):
        img = cv2.imread(os.path.join(test_path,"%d.png"%i))
        img = img / 255
        img = np.reshape(img,(1,)+img.shape)
        yield img
```

ภาพที่ 36 ชุดคำสั่งในการสร้างฟังก์ชัน testGenerator

ภาพคำตอบที่ได้จากการทำนายจะถูกสร้างขึ้นด้วยแถวลำดับที่สร้างจากด้วยฟังก์ชัน labelVisualize ซึ่งจะรับตัวแปร dictionary เพื่อระบุสีของคลาสที่ต้องการให้ปรากฏในภาพคำตอบ จากนั้นจึงคืนค่าแถวลำดับภาพที่เป็นข้อมูลสีของคลาสในภาพ ตามชุดคำสั่งในภาพที่ 37

```
def labelVisualize(num_class,color_dict,img):
    img = img[:,:,0] if len(img.shape) == 3 else img
    img_out = np.zeros(img.shape + (3,))
    for i in range(num_class):
        img_out[img == i,:] = color_dict[i]
    return img_out / 255
```

ภาพที่ 37 ชุดคำสั่งในการสร้างฟังก์ชัน labelVisualize

ทำการบันทึกภาพที่ทำนายด้วยฟังก์ชัน saveResult โดยรับค่า directory ที่ต้องการเก็บภาพคำตอบ ซึ่งภาพที่ได้จะถูกบันทึกไปยัง directory ที่ระบุในนามสกุล .png ทีละภาพ ตามชุดคำสั่งในภาพที่ 38

```
def saveResult(save_path,npyfile,flag_multi_class = False,num_class):
    for i,item in enumerate(npyfile):
        img = labelVisualize(num_class,COLOR_DICT,item) if flag_multi_class else item[:,:,0]
        io.imsave(os.path.join(save_path,"%d_predict.png"%i),img)
```

ภาพที่ 38 ชุดคำสั่งในการสร้างฟังก์ชัน saveResult

4.2.2 การพัฒนาแบบจำลอง

แบบจำลองที่สร้างขึ้นได้อ้างอิงตามแบบจำลอง FCNN ในรูปแบบสถาปัตยกรรมแบบ U-net ซึ่งผู้วิจัยได้เลือกเอา VGG16 ในส่วนที่เป็น convolution base มาแทนที่ส่วน contracting path ของ U-net เพื่อให้สามารถใช้ pretrained weight ในการ fine tuning แบบจำลองได้ โดยได้เลือกใช้ pretrained weight ที่มีการเรียบรู้มาแล้วกับชุดข้อมูล ImageNet ที่เป็นชุดข้อมูลรูปภาพขนาดใหญ่และมีจำนวนคลาสมากกว่า 1000 คลาส

ภาพที่ 39 แสดงชุดคำสั่งในการเรียกไฟล์ pretrained weights ของแบบจำลอง VGG16 ชื่อ vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5 ซึ่งเป็น pretrained weights ในส่วน convolution base เท่านั้นและยังเป็น pretrained weights ที่ใช้งานได้กับชุดคำสั่ง keras ที่ใช้ tensorflow เป็น backend เท่านั้นด้วย

```
import os
file_path = os.getcwd()
VGG_Weights_path =
file_path+"\\vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5"
```

ภาพที่ 39 ชุดคำสั่งในการเรียกไฟล์ pretrained weights

ชุดคำสั่งในการสร้างแบบจำลอง จะเป็นฟังก์ชันชื่อว่า unet_vgg16_imagenet โดยแบบจำลองจะถูกสร้างอ้างอิงตามรูปแบบที่ได้ออกแบบไว้ในภาพที่ 33 ซึ่งในภาพที่ 40 เป็นชุดคำสั่งในการสร้างส่วนของ contracting path ในแบบจำลอง contracting path ที่ได้จะประกอบด้วยชั้นข้อมูลขาเข้า และชั้น convolution layer กับชั้น max pooling ที่มีลักษณะเป็น convolution base ของ VGG16 โดยแบ่งกลุ่มชั้นของแบบจำลองออกได้เป็น 5 blocks ซึ่งชื่อของแต่ละชั้นนั้นจะมี

การระบุหมายเลข block ตามด้วยกระบวนการที่ทำในชั้นนั้นเช่น block4_conv3 จะเป็นกระบวนการ convolution รอบที่ 3 ในส่วนของ block 4 นอกจากนี้ใน block 4 และ block 5 จะมีการใช้ dropout ที่ตั้งค่าไว้ที่ 0.5 เมื่อสร้างส่วน contracting path ของแบบจำลองเสร็จสิ้นก็จะเรียกใช้ pretrained weights เพื่อใส่เข้าไปในแบบจำลอง ตามชุดคำสั่งในภาพที่ 41

```
import numpy as np
import skimage.io as io
import skimage.transform as trans
import numpy as np
from keras.models import *
from keras.layers import *
from keras.optimizers import *
from keras.callbacks import ModelCheckpoint, LearningRateScheduler
from keras import backend as keras

def unet_vgg16_imagenet():
    inputs = Input(shape=(512,512,3), name='input_layer')
    x = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal', name='block1_conv1')(inputs)
    x = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal', name='block1_conv2')(x)
    conv1 = x
    x = MaxPooling2D(pool_size=(2, 2), name='block1_maxpool')(x)
    x = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal', name='block2_conv1')(x)
    x = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal', name='block2_conv2')(x)
    conv2 = x
    x = MaxPooling2D(pool_size=(2, 2), name='block2_maxpool')(x)
```

```
    x = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal', name='block3_conv1')(x)
    x = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal', name='block3_conv2')(x)
    x = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal', name='block3_conv3')(x)
    conv3 = x
    x = MaxPooling2D(pool_size=(2, 2), name='block3_maxpool')(x)
    x = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal', name='block4_conv1')(x)
    x = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal', name='block4_conv2')(x)
    x = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal', name='block4_conv3')(x)
    conv4 = x
    drop4 = Dropout(0.5)(conv4)
    x = MaxPooling2D(pool_size=(2, 2), name='block4_maxpool')(x)
    x = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal', name='block5_conv1')(x)
    x = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal', name='block5_conv2')(x)
    x = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal', name='block5_conv3')(x)
    conv5 = x
    drop5 = Dropout(0.5)(conv5)
    x = MaxPooling2D(pool_size=(2, 2), name='block5_maxpool')(x)
    vgg16 = x
```

ภาพที่ 40 ชุดคำสั่งสร้างส่วนที่เป็น contracting path ของแบบจำลอง

```
model = Model(input = inputs, output = vgg16)

model.load_weights(VGG_Weights_path)
```

ภาพที่ 41 ชุดคำสั่งเรียกใช้ pretrained weights ของ VGG16

ภาพที่ 42 แสดงชุดคำสั่งที่จะกำหนดส่วนที่จะถูกสอนและไม่ถูกสอนใน contracting path โดยตั้งค่าการสอนตามตัวแปร set_trainable ซึ่งเป็นตัวแปร boolean ที่มีค่าเป็น true หรือ false จากชุดคำสั่ง ส่วนของ block ที่ 1-3 จะถูกตั้งค่า set_trainable ไว้ที่ false เพื่อใช้ pretrained weights และส่วนของ block ที่ 4-5 จะถูกตั้งค่า set_trainable ไว้ที่ true เพื่อเปิดให้แบบจำลองเรียนรู้ ตามลักษณะของการทำ fine tuning

```
model.trainable = True
set_trainable = False
for layer in model.layers:
    if layer.name == 'block4_conv1':
        set_trainable = True
    if set_trainable:
        layer.trainable = True
    else:
        layer.trainable = False
```

ภาพที่ 42 ชุดคำสั่งในการตั้งค่า block ที่จะถูกสอนของส่วน contracting path

ในส่วนของ expansive path จะถูกสร้างขึ้นอ้างอิงตามรูปแบบที่ได้ออกแบบไว้ในภาพที่ 33 เช่นเดียวกับส่วน expansive path ซึ่งในภาพที่ 43 เป็นชุดคำสั่งในการสร้างส่วนของ expansive path ที่จะประกอบด้วยชั้นข้อมูลขาออกและชั้น convolution layer กับชั้น upsampling layer ที่โดยแบ่งกลุ่มชั้นของแบบจำลองออกได้เป็น 5 blocks คือชั้นที่ 6-10 ในชั้นที่ 6-9 จะมีการทำ merge ซึ่งเป็น concatenation method เพื่อคัดลอก feature map บางส่วนใน contracting path มารวมกับ feature map ในส่วน expansive path ที่สร้างมาจากการ upsampling เมื่อแบบจำลองเสร็จสิ้นแล้วก็จะทำการประมวลแบบจำลอง ตามคำสั่ง model.compile โดยใช้การหาค่าที่ดีที่สุดแบบ Adam [22] และใช้ค่า learning rate ที่ 0.0001

```
    up6 = Conv2D(512, 2, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal', name='block6_up')(UpSampling2D(size = (2,2))(drop5))
    merge6 = merge([drop4,up6], mode = 'concat', concat_axis = 3, name='block6_merge')
    x = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal', name='block6_conv1')(merge6)
    x = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal', name='block6_conv2')(x)

    up7 = Conv2D(256, 2, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal', name='block7_up')(UpSampling2D(size = (2,2))(x))
    merge7 = merge([conv3,up7], mode = 'concat', concat_axis = 3, name='block7_merge')
    x = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal', name='block7_conv1')(x)
    x = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal', name='block7_conv2')(x)
    up8 = Conv2D(128, 2, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal', name='block8_up')(UpSampling2D(size = (4,4))(x))
    merge8 = merge([conv2,up8], mode = 'concat', concat_axis = 3, name='block8_merge')
    up8 = Conv2D(128, 2, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal', name='block8_up')(UpSampling2D(size = (4,4))(x))
    merge8 = merge([conv2,up8], mode = 'concat', concat_axis = 3, name='block8_merge')
    x = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal', name='block8_conv1')(x)
    x = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal', name='block8_conv2')(x)
```

```
    up9 = Conv2D(64, 2, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal', name='block9_up')(UpSampling2D(size = (8,8))(x))
    merge9 = merge([conv1,up9], mode = 'concat', concat_axis = 3, name='block9_merge')
    x = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal', name='block9_conv1')(merge9)
    x = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal', name='block9_conv2')(x)
    x = Conv2D(2, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal', name='block9_conv3')(x)

    conv10 = Conv2D(1, 1, activation = 'sigmoid', name='block10_conv2')(x)
    model = Model(input = inputs, output = conv10)

    model.compile(optimizer = Adam(lr = 1e-4), loss = 'binary_crossentropy', metrics = ['accuracy'])

    model.summary()
    return model
```


ภาพที่ 43 ชุดคำสั่งสร้างส่วนที่เป็น expansive path ของแบบจำลอง

บทที่ 5
การทดลองและวิเคราะห์ผล

5.1 การเรียกใช้แบบจำลอง

5.1.1 การเรียกใช้คำสั่ง Data augmentation

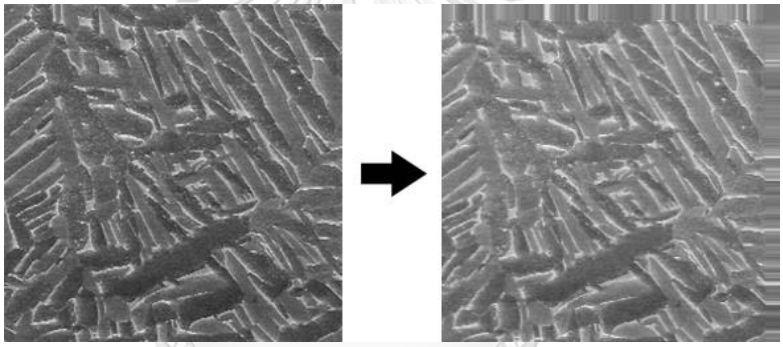
การสร้างข้อมูล Data augmentation เพื่อเพิ่มชุดข้อมูลสอนให้เพิ่มมากขึ้นจากการเรียกใช้ ImageDataGenerator ของ library Keras ซึ่งจะปรับเปลี่ยนภาพที่กำหนดตามตัวแปรที่เราตั้งค่าได้ ดังนี้

- rotation_range คือการสุ่มหมุนภาพตั้งแต่ 0-180 องศา
- width_shift_range, height_shift_range คือ การสุ่มเลื่อนภาพให้ออกจากตำแหน่งเดิม โดยเทียบขนาดในการเลื่อนเป็นสัดส่วนของความกว้างทั้งหมดและความสูงทั้งหมดในภาพ
- shear_range คือ การสุ่มเปลี่ยนแปลงบิดรูปภาพในลักษณะเฉือน
- zoom_range คือ การสุ่มขยายภาพบางส่วนในภาพ
- horizontal_flip คือ การสุ่มกลับด้านภาพตามแนวนอน
- fill_mode คือ วิธีการเติมเม็ดสีในบริเวณที่ภาพถูกเปลี่ยนไปเนื่องจากการบิดหมุนภาพ หรือการเลื่อนภาพ โดยทั่วไปจะใช้โหมด ‘nearest’ เพื่อใช้เม็ดสีบริเวณใกล้เคียงมาเติม

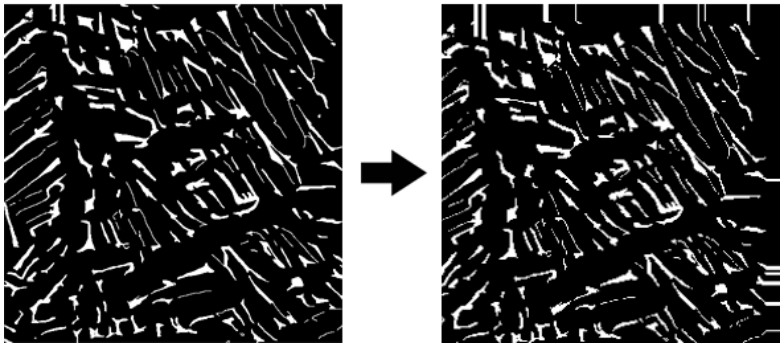
การเพิ่มชุดข้อมูลกระทำโดยกำหนดรูปแบบด้วยตัวแปร data_gen_ans ซึ่งเป็นตัวแปรแบบ dictionary ตัวอย่างชุดคำสั่งในการสร้าง data augmentation เพื่อตั้งค่าการเปลี่ยนแปลงข้อมูลภาพ ได้แก่ rotation_range=0.2, width_shift_range=0.05, height_shift_range=0.05, shear_range=0.05, zoom_range=0.05, horizontal_flip=True, fill_mode='nearest' ดังแสดงในภาพที่ 44 ตัวอย่างภาพที่ได้จากการเปลี่ยนแปลง ดังภาพที่ 45 และภาพที่ 46

```
data_gen_args = dict(rotation_range=0.2,
                     width_shift_range=0.05,
                     height_shift_range=0.05,
                     shear_range=0.05,
                     zoom_range=0.05,
                     horizontal_flip=True,
                     fill_mode='nearest')
myGene = trainGenerator(2,'data/titanium/train','image','label',data_gen_args,
save_to_dir = None)
```

ภาพที่ 44 ชุดคำสั่งการทำ data augmentation



ภาพที่ 45 ตัวอย่างภาพโครงสร้างจุลภาคที่ผ่าน data augmentation



ภาพที่ 46 ตัวอย่างภาพ ground-truth ที่ผ่าน data augmentation

5.1.2 การเรียกใช้งานแบบจำลอง

ภาพที่ 47 ทำการเรียกฟังก์ชัน unet_vgg16_imagenet() เพื่อเรียกใช้งานแบบจำลอง

```
model = unet_vgg16_imagenet()
```

ภาพที่ 47 คำสั่งเรียกใช้งานแบบจำลอง

เมื่อเรียกใช้งานแบบจำลอง โปรแกรมจะแสดงผลลัพธ์ภาพรวมของแบบจำลองตามคำสั่ง model.summary() ที่อยู่ในฟังก์ชัน unet_vgg16_imagenet ซึ่งจะแสดงแต่ละชั้นของแบบจำลอง และจำนวน parameters ทั้งหมดที่ใช้ในแบบจำลอง โดยแสดงจำนวน parameters ในแบบจำลองที่ไม่สามารถสอนได้ (มาจากการใช้ pretrained weights), จำนวน parameters ในแบบจำลองที่สามารถสอนได้ และจำนวน parameters ในแบบจำลองทั้งหมด ดังตารางที่ 2 และภาพที่ 48

ตารางที่ 2 ผลลัพธ์ภาพรวมข้อมูลชั้นของแบบจำลอง unet_vgg16_imagenet

Layer (type)	Output Shape	Parameters	Connected to
input_layer (InputLayer)	(None, 512, 512, 3)	0	-
block1_conv1 (Conv2D)	(None, 512, 512, 64)	1792	Input_layer
block1_conv2 (Conv2D)	(None, 512, 512, 64)	36928	block1_conv1
block1_maxpool (MaxPooling2D)	(None, 256, 256, 64)	0	block1_conv2
block2_conv1 (Conv2D)	(None, 256, 256, 128)	73856	block1_maxpool
block2_conv2 (Conv2D)	(None, 256, 256, 128)	14758	block2_conv1
block2_maxpool (MaxPooling2D)	(None, 128, 128, 256)	0	block2_conv2
block3_conv1 (Conv2D)	(None, 128, 128, 256)	295168	block2_maxpool
block3_conv2 (Conv2D)	(None, 128, 128, 256)	590080	block3_conv1
block3_conv3	(None, 128, 128,	590080	block3_conv2

(Conv2D)	256)		
block3_maxpool (MaxPooling2D)	(None, 64, 64, 256)	0	block3_conv3
block4_conv1 (Conv2D)	(None, 64, 64, 512)	1180160	block3_maxpool
block4_conv2 (Conv2D)	(None, 64, 64, 512)	2359808	block4_conv1
block4_conv3 (Conv2D)	(None, 64, 64, 512)	2359808	block4_conv2
block4_maxpool (MaxPooling2D)	(None, 32, 32, 512)	0	block4_conv3
block5_conv1 (Conv2D)	(None, 32, 32, 512)	2359808	block4_maxpool
block5_conv2 (Conv2D)	(None, 32, 32, 512)	2359808	block5_conv1
block5_conv3 (Conv2D)	(None, 32, 32, 512)	2359808	block5_conv2
dropout_5 (Dropout)	(None, 32, 32, 512)	0	block5_conv3
up_sampling2d_8 (UpSampling2D)	(None, 64, 64, 512)	0	dropout_5
dropout_4 (Dropout)	(None, 64, 64, 512)	0	block4_conv3
block6_up (Conv2D)	(None, 64, 64, 512)	1049088	up_sampling2d_8
block6_merge (Merge)	(None, 64, 64, 1024)	0	dropout_4 block6_up
block6_conv1 (Conv2D)	(None, 64, 64, 512)	4719104	block6_merge
block6_conv2	(None, 64, 64, 512)	2359808	block6_conv1

(Conv2D)			
block7_conv1 (Conv2D)	(None, 64, 64, 256)	1179904	block6_conv2
block7_conv2 (Conv2D)	(None, 64, 64, 256)	590080	block7_conv1
block8_conv1 (Conv2D)	(None, 64, 64, 128)	295040	block7_conv2
block8_conv2 (Conv2D)	(None, 64, 64, 128)	147584	block8_conv1
up_sampling2d_16 (UpSampling2D)	(None, 512, 512, 128)	0	block8_conv2
block9_up (Conv2D)	(None, 512, 512, 64)	32832	up_sampling2d_16
block9_merge (Merge)	(None, 512, 512, 128)	0	block1_conv2 block9_up
block9_conv1 (Conv2D)	(None, 512, 512, 64)	73792	block9_merge
block9_conv2 (Conv2D)	(None, 512, 512, 64)	36928	block9_conv1
block9_conv3 (Conv2D)	(None, 512, 512, 2)	1154	block9_conv2
block10_conv2 (Conv2D)	(None, 512, 512, 1)	3	block9_conv3

Total params: 25,200,005

Trainable params: 23,464,517

Non-trainable params: 1,735,488

ภาพที่ 48 สรุปจำนวน parameters ทั้งหมดของแบบจำลอง แบ่งเป็น parameters ที่ถูกสอนได้ และ parameters ที่ไม่ถูกสอนเนื่องจากใช้ pretrained weights

5.1.3 การสอนแบบจำลอง

การสอนแบบจำลองได้ใช้คำสั่ง fit_generator ใน library Keras โดยจะเก็บค่าผลลัพธ์ในแต่ละรอบของการสอนไว้ในตัวแปร history เพื่อนำไปใช้สร้างกราฟความเปลี่ยนแปลงของค่า accuracy และ loss ของแบบจำลอง โดยได้ตั้งค่าการเรียนรู้ steps_per_epoch ไว้ที่ 100 และจำนวนรอบของการสอนอยู่ที่ 150 รอบ นอกจากนี้ยังได้สร้างตัวแปร model_checkpoint เพื่อใช้งานคำสั่ง callback เผื่อในกรณีที่มีความผิดพลาดเกิดขึ้นทำให้แบบจำลองหยุดเรียนรู้ในระหว่างที่การสอนแบบจำลองยังไม่เสร็จสิ้น เช่น ไฟฟ้าขัดข้อง ก็จะสามารถเรียกคืนค่าของการเรียนรู้เพื่อทำงานต่อจากเดิมได้ ดังภาพที่ 49

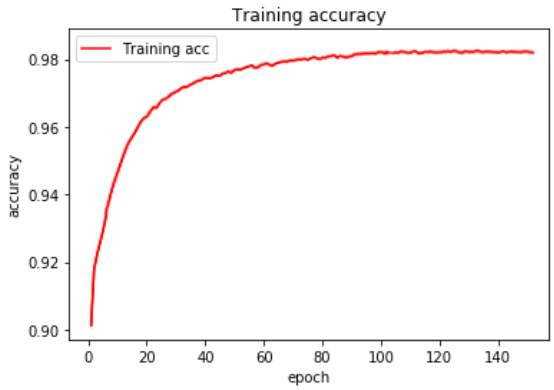
```
model_checkpoint = ModelCheckpoint('unet_vgg_from_imagenet.hdf5',
monitor='loss',verbose=1, save_best_only=True)
history = model.fit_generator(myGene,steps_per_epoch=100, epochs=150,
callbacks=[model_checkpoint])
```

ภาพที่ 49 คำสั่ง model_checkpoint และคำสั่ง fit_generator

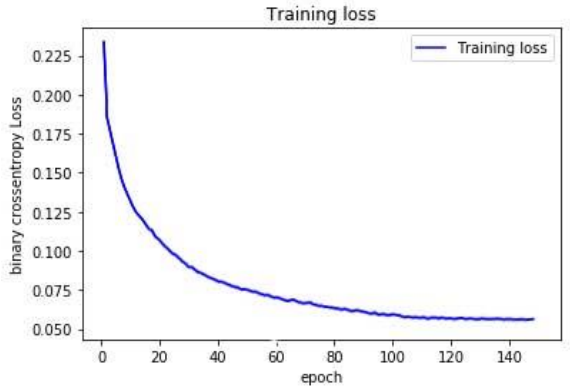
แบบจำลองจะทำการเรียนรู้โดยแสดงการทำงานตามภาพที่ 50 ซึ่งจะถูกเก็บบันทึกค่าความแม่นยำในการเรียนรู้และค่าความสูญเสียของแบบจำลองในแต่ละ epoch เพื่อนำมาสร้างกราฟค่าความแม่นยำในการเรียนรู้ในภาพที่ 51 และกราฟค่าความสูญเสียของแบบจำลองในภาพที่ 52

```
Epoch 00024: loss improved from 0.11104 to 0.11044, saving model to unet_vgg_from_imagenet.hdf5
Epoch 25/150
100/100 [==============================] - 62s 617ms/step - loss: 0.1091 - acc: 0.9658

Epoch 00025: loss improved from 0.11044 to 0.10911, saving model to unet_vgg_from_imagenet.hdf5
Epoch 26/150
100/100 [==============================] - 62s 616ms/step - loss: 0.1078 - acc: 0.9666

Epoch 00026: loss improved from 0.10911 to 0.10779, saving model to unet_vgg_from_imagenet.hdf5
Epoch 27/150
100/100 [==============================] - 61s 609ms/step - loss: 0.1066 - acc: 0.9671

Epoch 00027: loss improved from 0.10779 to 0.10661, saving model to unet_vgg_from_imagenet.hdf5
Epoch 28/150
100/100 [==============================] - 61s 613ms/step - loss: 0.1057 - acc: 0.9675

Epoch 00028: loss improved from 0.10661 to 0.10574, saving model to unet_vgg_from_imagenet.hdf5
Epoch 29/150
100/100 [==============================] - 62s 616ms/step - loss: 0.1045 - acc: 0.9683

Epoch 00029: loss improved from 0.10574 to 0.10450, saving model to unet_vgg_from_imagenet.hdf5
Epoch 30/150
100/100 [==============================] - 62s 617ms/step - loss: 0.1034 - acc: 0.9689

Epoch 00030: loss improved from 0.10450 to 0.10343, saving model to unet_vgg_from_imagenet.hdf5
Epoch 31/150
100/100 [==============================] - 62s 616ms/step - loss: 0.1032 - acc: 0.9689
```

ภาพที่ 50 รายงานผลการทำงานของแบบจำลองในแต่ละ epoch



ภาพที่ 51 กราฟค่าความแม่นยำในการเรียนรู้



ภาพที่ 52 กราฟค่าความสูญเสียของแบบจำลอง

5.2 การทดสอบแบบจำลอง

การทดสอบแบบจำลองจะใช้คำสั่ง testGenerator เพื่อสร้างชุดข้อมูลการทดสอบจาก directory ในเครื่องจากนั้นทำการเรียกใช้งานแบบจำลอง unet_vgg16_imagenet() และเรียกใช้ weights จากไฟล์ unet_vgg_from_imagenet.hdf5 ซึ่งได้ผ่านการเรียนรู้เสร็จสิ้นแล้ว แบบจำลองจะทำการทำนายจากคำสั่ง predict_generator และเก็บผลลัพธ์ภาพที่ได้จากการทำนายด้วยคำสั่ง saveResult ตาม directory ที่ระบุในคำสั่ง ดังภาพที่ 53

```
testGene = testGenerator("data/titanium/test")
model = unet_vgg16_imagenet()
model.load_weights("unet_vgg_from_imagenet.hdf5")
results = model.predict_generator(testGene,30,verbose=1)
saveResult("data/titanium/test",results)
```

ภาพที่ 53 ชุดคำสั่งทดสอบแบบจำลอง

5.3 การประเมินผลลัพธ์ที่ได้จากแบบจำลอง

สมรรถนะของแบบจำลองถูกประเมินด้วยมาตรวัด 4 ตัว (four metrics) ประกอบด้วย 1. Pixel accuracy, 2. Mean accuracy, 3. Mean of intersection over union (IoU), 4. Frequency weighted of IoU ซึ่งเป็นวิธีการที่มักใช้ในการประเมินผลลัพธ์ในงานการแบ่งส่วนความหมาย โดยเป็นการเปรียบเทียบระหว่างภาพที่ทำนายได้กับภาพคำตอบ ground-truth

5.3.1 Pixel accuracy

เป็นตัววัดที่ใช้ในการหาสัดส่วนของจำนวนพิกเซลภาพที่ทำนายได้ถูกต้องเทียบกับจำนวนพิกเซลทั้งหมดในภาพ ค่าตัววัดคำนวณจากสมการ (5) และถูกเขียนเป็นชุดคำสั่งในภาพที่ 54

$$\text{pixel accuracy} = \frac{\sum_i n_{ii}}{\sum_i t_i} \tag{5}$$

โดย n_{ii} แทนจำนวนพิกเซลของคลาส i ที่คำตอบเป็นคลาส i

t_i แทนจำนวนพิกเซลทั้งหมดที่เป็นคลาส i

```
import numpy as np
def pixel_accuracy(eval_segm, gt_segm):
    check_size(eval_segm, gt_segm)
    cl, n_cl = extract_classes(gt_segm)
    eval_mask, gt_mask = extract_both_masks(eval_segm, gt_segm, cl, n_cl)
    sum_n_ii = 0
    sum_t_i  = 0
    for i, c in enumerate(cl):
        curr_eval_mask = eval_mask[i, :, :]
        curr_gt_mask = gt_mask[i, :, :]
        sum_n_ii += np.sum(np.logical_and(curr_eval_mask, curr_gt_mask))
        sum_t_i  += np.sum(curr_gt_mask)
    if (sum_t_i == 0):    pixel_accuracy_ = 0
    else:                 pixel_accuracy_ = sum_n_ii / sum_t_i
    return pixel_accuracy_
```

ภาพที่ 54 ชุดคำสั่งคำนวณค่า pixel accuracy

5.3.2 Mean accuracy

เป็นตัววัดที่ใช้ในการหาสัดส่วนของจำนวนของพิกเซลภาพที่ทำนายได้ถูกต้องเทียบกับจำนวนพิกเซลทั้งหมด โดยค่าความแม่นยำจะถูกคำนวณแยกคลาสกันและนำมาเฉลี่ยกัน สูตรการคำนวณดังสมการ (6) และเขียนเป็นชุดคำสั่งในภาพที่ 55

$$\text{mean accuracy} = \frac{1}{n_{cl}} \frac{\sum_i n_{ii}}{t_i} \quad (6)$$

โดย n_{ii} แทนจำนวนพิกเซลของคลาส i ที่คำตอบเป็นคลาส i

t_i แทนจำนวนพิกเซลทั้งหมดที่เป็นคลาส i

n_{cl} แทนจำนวนคลาสที่แตกต่างกันในภาพ ground-truth

```
def mean_accuracy(eval_segm, gt_segm):
    check_size(eval_segm, gt_segm)
    cl, n_cl = extract_classes(gt_segm)
    eval_mask, gt_mask = extract_both_masks(eval_segm, gt_segm, cl, n_cl)
    accuracy = list([0]) * n_cl
    for i, c in enumerate(cl):
        curr_eval_mask = eval_mask[i, :, :]
        curr_gt_mask = gt_mask[i, :, :]
        n_ii = np.sum(np.logical_and(curr_eval_mask, curr_gt_mask))
        t_i  = np.sum(curr_gt_mask)
        if (t_i != 0):
            accuracy[i] = n_ii / t_i
    mean_accuracy_ = np.mean(accuracy)
    return mean_accuracy_
```

ภาพที่ 55 ชุดคำสั่งคำนวณค่า mean accuracy

5.3.3 Mean intersection over union

เป็นตัววัดที่ใช้ในการหาจำนวนพิกเซลที่ทับซ้อนกันระหว่างภาพที่ทำนายกับภาพ ground-truth ในแต่ละคลาส จากนั้นจึงนำค่าที่ได้มาหารเฉลี่ยกันในทุกคลาส สูตรการคำนวณดังสมการ (7) และเขียนเป็นชุดคำสั่งในภาพที่ 56

$$\text{mean IoU} = \frac{1}{n_{cl}} \sum_i \frac{n_{ii}}{(t_i + \sum_j n_{ji} - n_{ii})} \quad (7)$$

โดย n_{ii} แทนจำนวนพิกเซลของคลาส i ที่คำตอบเป็นคลาส i

t_i แทนจำนวนพิกเซลทั้งหมดที่เป็นคลาส i

n_{cl} แทนจำนวนคลาสที่แตกต่างกันในภาพ ground-truth

```
def mean_IU(eval_segm, gt_segm):
    check_size(eval_segm, gt_segm)
    cl, n_cl   = union_classes(eval_segm, gt_segm)
    _, n_cl_gt = extract_classes(gt_segm)
    eval_mask, gt_mask = extract_both_masks(eval_segm, gt_segm, cl, n_cl)
    IU = list([0]) * n_cl
    for i, c in enumerate(cl):
        curr_eval_mask = eval_mask[i, :, :]
        curr_gt_mask = gt_mask[i, :, :]
        if (np.sum(curr_eval_mask) == 0) or (np.sum(curr_gt_mask) == 0):
            continue
        n_ii = np.sum(np.logical_and(curr_eval_mask, curr_gt_mask))
        t_i  = np.sum(curr_gt_mask)
        n_ij = np.sum(curr_eval_mask)
        IU[i] = n_ii / (t_i + n_ij - n_ii)
    mean_IU_ = np.sum(IU) / n_cl_gt
    return mean_IU_
```

ภาพที่ 56 ชุดคำสั่งคำนวณค่า mean intersection over union

5.3.4 Frequency weighted intersection over union

เป็นตัววัดที่ใช้ในการหาจำนวนพิกเซลที่ทับซ้อนกันระหว่างภาพที่ทำนายกับภาพ ground-truth ในแต่ละคลาส จากนั้นจึงนำค่าที่ได้มาหารเฉลี่ยกันในทุกคลาสแบบเดียวกับ Mean intersection over union แต่จะใช้การหารเฉลี่ยแบบถ่วงน้ำหนักของแต่ละคลาสในภาพ ground-truth สูตรคำนวณดังสมการ (8) และเขียนเป็นชุดคำสั่งในภาพที่ 57

$$\text{Frequency weighted IoU} = \sum_k \frac{1}{t_k} \sum_i \frac{t_i n_{ii}}{(t_i + \sum_j nji - n_{ii})} \quad (8)$$

โดย n_{ii} แทนจำนวนพิกเซลของคลาส i ที่คำตอบเป็นคลาส i

t_i แทนจำนวนพิกเซลทั้งหมดที่เป็นคลาส i

n_{cl} แทนจำนวนคลาสที่แตกต่างกันในภาพ ground-truth

```
def frequency_weighted_IU(eval_segm, gt_segm):
    check_size(eval_segm, gt_segm)
    cl, n_cl = union_classes(eval_segm, gt_segm)
    eval_mask, gt_mask = extract_both_masks(eval_segm, gt_segm, cl, n_cl)
    frequency_weighted_IU_ = list([0]) * n_cl
    for i, c in enumerate(cl):
        curr_eval_mask = eval_mask[i, :, :]
        curr_gt_mask = gt_mask[i, :, :]
        if (np.sum(curr_eval_mask) == 0) or (np.sum(curr_gt_mask) == 0):
            continue
        n_ii = np.sum(np.logical_and(curr_eval_mask, curr_gt_mask))
        t_i  = np.sum(curr_gt_mask)
        n_ij = np.sum(curr_eval_mask)
        frequency_weighted_IU_[i] = (t_i * n_ii) / (t_i + n_ij - n_ii)
    sum_k_t_k = get_pixel_area(eval_segm)
    frequency_weighted_IU_ = np.sum(frequency_weighted_IU_) / sum_k_t_k
    return frequency_weighted_IU_
```

ภาพที่ 57 ชุดคำสั่งคำนวณค่า frequency weighted intersection over union

ทั้งนี้ ยังมีชุดคำสั่งฟังก์ชันอื่นๆที่ใช้ช่วยในการคำนวณ ได้แก่

ฟังก์ชัน get_pixel_area (ภาพที่ 58) คืนค่าเป็นจำนวนพิกเซลในพื้นที่ของภาพ

```
def get_pixel_area(segm):
    return segm.shape[0] * segm.shape[1]
```

ภาพที่ 58 ชุดคำสั่งฟังก์ชัน get_pixel_area

ฟังก์ชัน segm_size (ภาพที่ 59) คืนค่าเป็นขนาดแนวตั้งแนวนอนของภาพ

```
def segm_size(segm):
    try:
        height = segm.shape[0]
        width  = segm.shape[1]
    except IndexError:
        raise
    return height, width
```

ภาพที่ 59 ชุดคำสั่งฟังก์ชัน segm_size

ฟังก์ชัน check_size (ภาพที่ 60) ใช้เพื่อตรวจสอบว่าขนาดภาพทดสอบกับขนาดภาพ ground-truth มีขนาดเท่ากันหรือไม่ ซึ่งถ้าขนาดไม่เท่ากันจะเรียกคลาส EvalSegErr (ภาพที่ 61) เพื่อหยุดการทำงานและระบุว่าเกิดข้อผิดพลาดขนาดไม่เท่ากันขึ้น

```
def check_size(eval_segm, gt_segm):
    h_e, w_e = segm_size(eval_segm)
    h_g, w_g = segm_size(gt_segm)

    if (h_e != h_g) or (w_e != w_g):
        raise EvalSegErr("DiffDim: Different dimensions of matrices!")
```

ภาพที่ 60 ชุดคำสั่งฟังก์ชัน check_size

```
class EvalSegErr(Exception):
    def __init__(self, value):
        self.value = value
    def __str__(self):
        return repr(self.value)
```

ภาพที่ 61 ชุดคำสั่งคลาส EvalSegErr

ฟังก์ชัน extract_mask (ภาพที่ 62) ใช้เพื่อสร้างแถวลำดับที่ระบุคลาสบนรูปภาพ

```
def extract_masks(segm, cl, n_cl):
    h, w  = segm_size(segm)
    masks = np.zeros((n_cl, h, w))
    for i, c in enumerate(cl):
        masks[i, :, :] = segm == c
    return masks
```

ภาพที่ 62 ชุดคำสั่งฟังก์ชัน extract_mask

ฟังก์ชัน extract_mask_both_masks (ภาพที่ 63) เป็นการเรียกใช้ฟังก์ชัน extract_mask เพื่อสร้างแถวลำดับที่ระบุคลาสบนรูปภาพพร้อมกันทั้งภาพที่ทำนายและภาพ groud-truth


```
def extract_both_masks(eval_segm, gt_segm, cl, n_cl):
    eval_mask = extract_masks(eval_segm, cl, n_cl)
    gt_mask   = extract_masks(gt_segm, cl, n_cl)
    return eval_mask, gt_mask
```

ภาพที่ 63 ชุดคำสั่งฟังก์ชัน extract_mask_both_masks

ฟังก์ชัน extract_classes (ภาพที่ 64) ใช้เพื่อหาจำนวนคลาสในรูปภาพ

```
def extract_classes(segm):
    cl = np.unique(segm)
    n_cl = len(cl)
    return cl, n_c
```

ภาพที่ 64 ชุดคำสั่งฟังก์ชัน extract_classes

ฟังก์ชัน union_classes (ภาพที่ 65) ใช้เพื่อหาจำนวนพิกเซลที่ union กันในแต่ละคลาสของภาพผลลัพธ์การทำนายและภาพ ground-truth

```
def union_classes(eval_segm, gt_segm):
    eval_cl, _ = extract_classes(eval_segm)
    gt_cl, _   = extract_classes(gt_segm)
    cl = np.union1d(eval_cl, gt_cl)
    n_cl = len(cl)
    return cl, n_cl
```

ภาพที่ 65 ชุดคำสั่งฟังก์ชัน union_classes

5.4 ผลการทดลอง

ตารางที่ 3 แสดงผลของการทดลองของแบบจำลองใน 5 แผนงาน 1) แบบจำลอง U-net แบบดั้งเดิมที่ถูกสอนด้วยชุดข้อมูลที่ไม่มีการทำ data augmentation 2) แบบจำลอง U-net แบบดั้งเดิมที่ถูกสอนด้วยชุดข้อมูลที่มีการทำ data augmentation 3) แบบจำลอง U-net VGG16ที่ถูกสอนด้วยชุดข้อมูลที่ไม่มีการทำ data augmentation 4) แบบจำลอง U-net VGG16ที่ถูกสอนด้วยชุดข้อมูลที่มีการทำ data augmentation 5) แบบจำลอง U-net VGG16ที่ถูกสอนด้วยชุดข้อมูลที่มีการทำ data augmentation และใช้วิธีการ fine tuning

ตารางที่ 3 เปรียบเทียบผลการประเมินสมรรถนะแบบจำลอง U-net และ U-net+VGG16

Model	Augmented	Training strategy	Pixel acc.	Mean acc.	Mean IoU	f.w. IoU	Training time (hrs.)
U-net	✘	from scratch	0.8021	0.6042	0.5532	0.8019	6
	✓	from scratch	0.8154	0.6208	0.6066	0.8374	6
U-net VGG16	✘	from scratch	0.8309	0.6727	0.6512	0.8203	6
	✓	from scratch	0.9267	0.8039	0.7130	0.8724	6
	✓	fine tuning	**0.9303**	**0.8307**	**0.7317**	**0.8794**	**2**

5.5 การวิเคราะห์ผลการทดลอง

การเปรียบเทียบสมรรถนะของแบบจำลองถูกประเมินด้วยมาตรวัด 4 ตัว ซึ่งเป็นตัววัดที่มักใช้งานในการเทียบสมรรถนะของการแบ่งส่วนตามความหมาย โดยค่า Pixel accuracy สื่อถึงสัดส่วนจำนวนพิกเซลที่ทำนายได้ถูกต้องเมื่อเทียบกับภาพ ground-truth กับพิกเซลทั้งหมดในภาพ โดยไม่สนใจในส่วนที่ทำนายผิด ค่า Mean accuracy สื่อถึงค่าเฉลี่ยของสัดส่วนจำนวนพิกเซลที่ทำนายได้ถูกต้องในแต่ละคลาสเมื่อเทียบกับภาพ ground-truth กับพิกเซลทั้งหมดในแต่ละคลาส ค่า Mean IoU สื่อถึงค่าเฉลี่ยของสัดส่วนพิกเซลที่ภาพทำนายและภาพ ground-truth ทับซ้อนกันกับพิกเซลทั้งหมดในแต่ละคลาส ค่า frequency weighted IoU สื่อถึงค่าเฉลี่ยถ่วงน้ำหนักด้วยคลาสของสัดส่วนพิกเซลที่ภาพทำนายและภาพ ground-truth ทับซ้อนกันกับพิกเซลทั้งหมดในแต่ละคลาส ซึ่งการเปรียบเทียบผลลัพธ์มักนิยมใช้ค่า Mean IoU เป็นตัวแทนในการเปรียบเทียบสมรรถนะของแบบจำลอง เนื่องจากค่าตัวเลขที่สะท้อนออกมาคิดจากส่วนที่ทำนายได้ถูกต้องและส่วนที่ทำนายได้ผิดในแบบจำลอง ในขณะที่ Pixel accuracy และ Mean accuracy จะสนใจแต่ส่วนที่แบบจำลองทำนายได้ถูกต้องอย่างเดียวเท่านั้น ส่วน frequency weighted IoU จะเหมาะสมในการเป็นตัวแทนเปรียบเทียบสมรรถนะของแบบจำลองก็ต่อเมื่อชุดข้อมูลของแบบจำลองที่นำมาเปรียบเทียบมีลักษณะของคลาสใกล้เคียงกัน

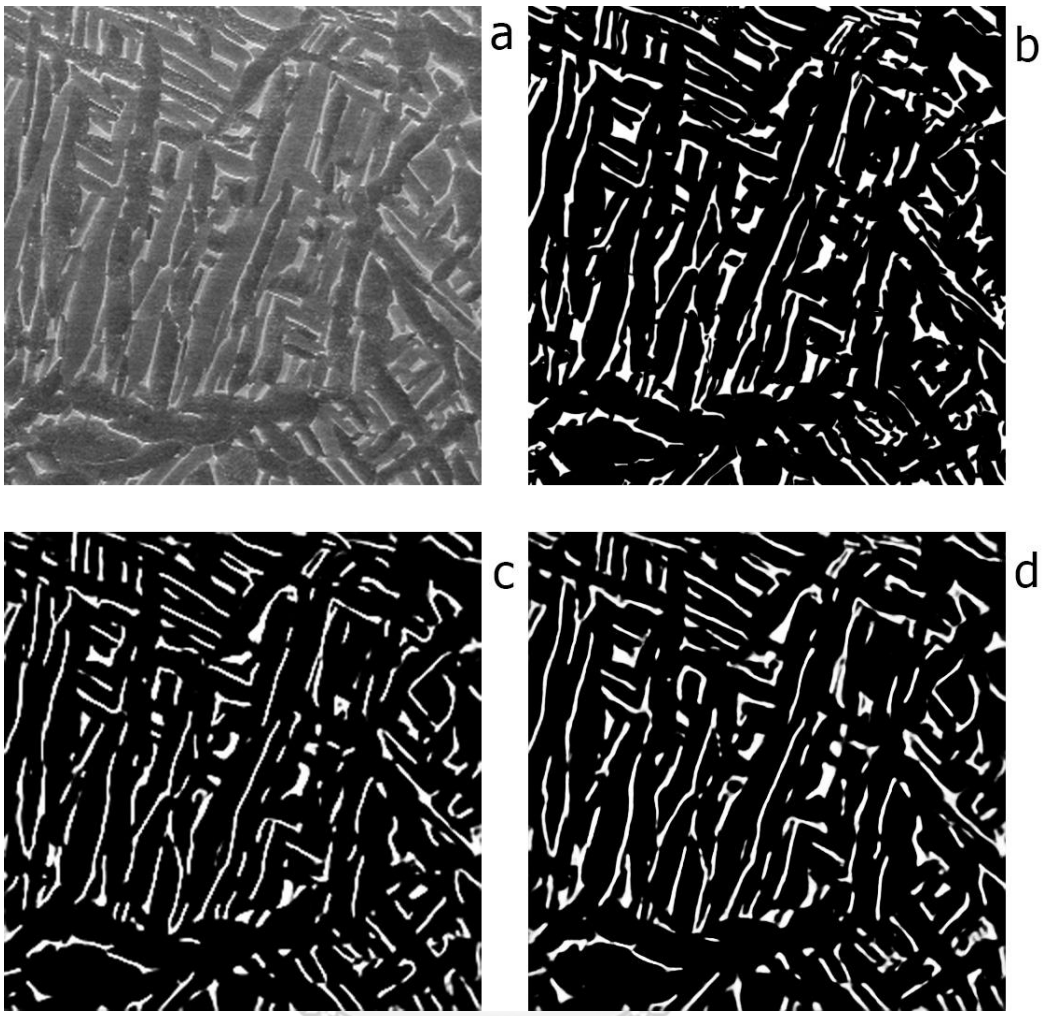
จากตารางที่ 3 พบว่าแบบจำลอง U-net ที่บูรณาการสถาปัตยกรรม VGG16 ใช้วิธี fine tuning และชุดข้อมูลมีการทำ data augmentation ให้ผลการประเมินสมรรถนะที่ดีที่สุดเมื่อเทียบกับวิธีอื่นๆ ด้วยชุดข้อมูลทดสอบจำนวน 30 ภาพ ได้ค่า Pixel accuracy 93.03% ค่า Mean accuracy 83.07% ค่า Mean IoU 73.17% และค่า frequency weighted IoU 87.94% ซึ่ง

มากกว่าวิธีการเรียนรู้ from scratch อยู่เล็กน้อย โดยแบบจำลอง U-net ที่บูรณาการสถาปัตยกรรม VGG16 ที่ชุดข้อมูลมีการทำ data augmentation แต่ใช้วิธีเรียนรู้ from scratch จากชุดข้อมูลโดยไม่ใช้ pretrained weights ได้ค่า Pixel accuracy 92.67% ค่า Mean accuracy 80.39% ค่า Mean IoU 71.30% และค่า frequency weighted IoU 87.24% แต่ว่าการเรียนรู้จะใช้เวลานานกว่า 6 ชั่วโมง ซึ่งมากกว่าการเรียนรู้ด้วยวิธี fine tuning ที่ใช้เวลาเพียง 2 ชั่วโมงเท่านั้น

นอกจากนี้ ยังพบว่าการใช้ data augmentation มีส่วนช่วยให้แบบจำลองทำนายผลลัพธ์ได้แม่นยำขึ้น โดยแบบจำลอง U-net ที่บูรณาการสถาปัตยกรรม VGG16 แต่ชุดข้อมูลไม่มีการทำ data augmentation ได้ผลลัพธ์ค่า Pixel accuracy 83.09% ค่า Mean accuracy 67.27% ค่า Mean IoU 65.12% และค่า frequency weighted IoU 82.03% ซึ่งค่าการเปรียบเทียบวิธีการที่ใช้และไม่ใช้ data augmentation ก็ให้ผลลัพธ์แบบเดียวกันกับแบบจำลอง U-net แบบดั้งเดิมไปในทิศทางเดียวกัน แบบจำลอง U-net แบบดั้งเดิมที่ชุดข้อมูลไม่มีการทำ data augmentation ได้ผลลัพธ์ค่า Pixel accuracy 80.21% ค่า Mean accuracy 60.42% ค่า Mean IoU 55.32% และค่า frequency weighted IoU 80.19% ในขณะที่แบบจำลอง U-net แบบดั้งเดิมที่ชุดข้อมูลมีการมีการทำ data augmentation ได้ผลลัพธ์ค่า Pixel accuracy 81.54% ค่า Mean accuracy 62.08% ค่า Mean IoU 60.66% และค่า frequency weighted IoU 83.74%

เมื่อนำผลลัพธ์ที่ดีที่สุดของแบบจำลองไปเทียบกับงานวิจัย [19] ที่สร้างแบบจำลอง FCNN เพื่อแบ่งส่วนตามความหมายในรูปภาพโครงสร้างจุลภาคเหล็กกล้าคาร์บอนต่ำที่ถ่ายด้วยกล้องจุลทรรศน์อิเล็กตรอน พบว่าผลลัพธ์การทำนายให้ค่าที่ดีที่สุด Pixel accuracy 93.92% ค่า Mean accuracy 76.70% ค่า Mean IoU 67.84% และค่า frequency weighted IoU 88.81% ในการจำแนกเฟส Martensite, Tempered Martensite, Bainite, Pearlite ซึ่งเมื่อเปรียบเทียบโดยใช้ค่า Mean IoU เป็นตัวแทนสมรรถนะของแบบจำลอง งานวิจัยนี้ได้ผลลัพธ์ Mean IoU ที่มากกว่าอยู่ที่ 73.17%

ภาพที่ 66(a) คือ ตัวอย่างภาพโครงสร้างจุลภาค Ti-6Al-4V ที่ใช้ทดสอบ โดยภาพ ground-truth ที่ตรงกันคือภาพ 66(b) การทดสอบกับแบบจำลอง U-net ที่บูรณาการ VGG16 โดยสร้างบนชุดข้อมูลที่มีการเพิ่มจำนวนตัวอย่าง และ 1) ไม่มีการทำ fine tuning จะได้ภาพผลลัพธ์การทำนายดังภาพ 66(c) 2) มีการทำ fine tuning จะได้ภาพผลลัพธ์การทำนายดังภาพ 66(d)



ภาพที่ 66 (a) ตัวอย่างภาพโครงสร้างจุลภาคที่นำมาทดสอบ (b) ภาพ ground-truth (c) ภาพทำนายของแบบจำลอง U-net+VGG16 ที่ทำ data augmentation แต่ไม่ได้ทำ fine tuning (d) ภาพทำนายของแบบจำลอง U-net+VGG16 ที่ทำ data augmentation และทำ fine tuning

บทที่ 6
สรุปผลการวิจัยและข้อเสนอแนะ

6.1 สรุปผลการวิจัย

เนื่องด้วยคุณสมบัติหลายประการที่เหมาะสม โลหะไทเทเนียมอัลลอยจึงถูกใช้งานอย่างแพร่หลายสำหรับใช้เป็นวัสดุกระดูกเทียมทดแทนในร่างกาย และเพื่อเพิ่มความสามารถในการผลิตชิ้นงานโลหะไทเทเนียมอัลลอยที่มีคุณภาพ งานวิจัยนี้จึงนำเสนอวิธีการตรวจสอบเฟสโครงสร้างจุลภาคแบบอัตโนมัติด้วยวิธีการแบ่งส่วนความหมาย ซึ่งได้เลือกใช้เทคนิค FCNN เพื่อการจำแนกภาพแบบจุดต่อจุดในภาพ โดยตัวจำแนก FCNN ได้ถูกสร้างขึ้นโดยใช้สถาปัตยกรรมแบบ U-net บูรณาการกับแบบจำลอง VGG16 ในการจำแนกเฟสของโลหะ 2 เฟส คือ เฟส alpha และเฟส beta ในภาพโครงสร้างจุลภาคของโลหะ Ti-6Al-4V ที่ถ่ายด้วยกล้องจุลทรรศน์อิเล็กตรอนแบบส่องกราด

ชุดข้อมูลรูปภาพทั้งหมด 96 ภาพ ขนาด 512x512 pixelsที่นำมาใช้งาน ได้รับการช่วยเหลือมาจากบริษัท Meticuly ซึ่งทำธุรกิจเกี่ยวกับการผลิตชิ้นส่วนวัสดุกระดูกเทียมทดแทนในร่างกาย โดยแบ่งชุดข้อมูลออกเป็นข้อมูลเพื่อสอนแบบจำลองจำนวน 66 ภาพ และข้อมูลเพื่อทดสอบแบบจำลอง 30 ภาพ นอกจากนี้แบบจำลองยังได้ใช้เทคนิค data augmentation ในการเพิ่มชุดข้อมูลการสอน และใช้เทคนิค fine tuning ด้วย pretrained weights ของ VGG16 ที่เรียนรู้จากชุดข้อมูลภาพ Imagenet เพื่อเพิ่มความแม่นยำของแบบจำลองและลดเวลาในการเรียนรู้ของแบบจำลอง

จากการทดลอง พบว่าสถาปัตยกรรมแบบจำลองที่มีการบูรณาการระหว่าง U-net กับ VGG16 ที่ใช้เทคนิค fine tuning และมีการทำ data augmentation ให้ผลลัพธ์ที่ดีที่สุดที่ mean IoU 73.17% ซึ่งมากกว่าการไม่ใช้ pretrained weights เพื่อ fine tuning อยู่เพียงเล็กน้อย แต่สามารถช่วยลดเวลาในการเรียนรู้ของแบบจำลองได้ถึงประมาณ 3 เท่า

6.2 ข้อจำกัดงานวิจัย

1) แบบจำลองนี้ถูกสร้างขึ้นเพื่อใช้งานกับภาพโครงสร้างจุลภาค Ti6-Al-4V
2) ภาพโครงสร้างที่นำมาใช้นั้นต้องอยู่ในลักษณะ dual phase คือมีเพียงแค่ 2 เฟสในภาพเท่านั้น

6.3 งานวิจัยในอนาคต

1) เปลี่ยนการใช้งาน pretrained weights ด้วยแบบจำลองที่ซับซ้อนมากขึ้นเพื่อเพิ่มสมรรถนะของแบบจำลอง เช่น ResNet หรือ GoogLeNet
2) พัฒนาต่อยอดแบบจำลองเพื่อการใช้งานกับภาพถ่ายโครงสร้างจุลภาค 3 มิติ

บรรณานุกรม

1. Acero, J., et al., *The behaviour of titanium as a biomaterial: microscopy study of plates and surrounding tissues in facial osteosynthesis.* 1999. **27**(2): p. 117-123.
2. Kiepura, R.T. and B.R. Sanders, *ASM handbook: metallography and microstructures*. 1985: ASM International.
3. Aji, W.A. *Microstructure of low carbon steel*. 2012, Februari 21; Available from: http://ajiedogawargnr.blogspot.com/2012/02/normal-0-false-false-false-en-us-x-none.html.
4. Ranut, P., et al., *High resolution X-ray microtomography-based CFD simulation for the characterization of flow permeability and effective thermal conductivity of aluminum metal foams.* 2015. **67**: p. 30-36.
5. Pecho, O.M., et al., *3D microstructure effects in Ni-YSZ anodes: influence of TPB lengths on the electrochemical performance.* 2015. **8**(10): p. 7129-7144.
6. Shotton, J., M. Johnson, and R. Cipolla. *Semantic texton forests for image categorization and segmentation*. in *Computer vision and pattern recognition, 2008. CVPR 2008. IEEE Conference on*. 2008. IEEE.
7. Dormehl, L. *What is an artificial neural network? Here's everything you need to know.* 2018, September 13; Available from: https://www.digitaltrends.com/cool-tech/what-is-an-artificial-neural-network/.
8. Castrounis, A. *Artificial intelligence, deep learning, and neural networks, explained*. 2016, October 14; Available from: https://www.kdnuggets.com/2016/10/artificial-intelligence-deep-learning-neural-networks-explained.html.
9. LeCun, Y., K. Kavukcuoglu, and C. Farabet. *Convolutional networks and applications in vision*. in *ISCAS*. 2010.
10. Juan, C.G. *Use of convolutional neural network for image classification*. 2017, November 20; Available from: https://www.apsl.net/blog/2017/11/20/use-convolutional-neural-network-image-classification/.

11. Peter, V. *Deep learning for complete beginners: convolutional neural networks with keras.* 2017, March 20; Available from: https://cambridgespark.com/content/tutorials/convolutional-neural-networks-with-keras/index.html.
12. Girshick, R., et al. *Rich feature hierarchies for accurate object detection and semantic segmentation*. in *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014.
13. Long, J., E. Shelhamer, and T. Darrell. *Fully convolutional networks for semantic segmentation*. in *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015.
14. Pakhomov, D. *Upsampling and Image Segmentation with Tensorflow and TF-Slim.* 2016, November 22; Available from: http://warmspringwinds.github.io/tensorflow/tf-slim/2016/11/22/upsampling-and-image-segmentation-with-tensorflow-and-tf-slim/.
15. Ronneberger, O., P. Fischer, and T. Brox. *U-net: Convolutional networks for biomedical image segmentation*. in *International Conference on Medical image computing and computer-assisted intervention*. 2015. Springer.
16. Srivastava, N., et al., *Dropout: a simple way to prevent neural networks from overfitting.* 2014. **15**(1): p. 1929-1958.
17. Chollet, F., *Deep learning with python*. 2017: Manning Publications Co.
18. Elizabeth, P. *What is selective laser sintering?* 2013, August 13; Available from: https://www.livescience.com/38862-selective-laser-sintering.html.
19. Azimi, S.M., et al., *Advanced Steel Microstructural Classification by Deep Learning Methods.* 2018. **8**(1): p. 2128.
20. Ter Haar, G.M. and T.H.J.M. Becker, *Selective Laser Melting Produced Ti-6Al-4V: Post-Process Heat Treatments to Achieve Superior Tensile Properties.* 2018. **11**(1): p. 146.
21. Iglovikov, V. and A.J.a.p.a. Shvets, *TernausNet: U-Net with VGG11 Encoder Pre-Trained on ImageNet for Image Segmentation.* 2018.
22. Kingma, D.P. and J.J.a.p.a. Ba, *Adam: A method for stochastic optimization.*

2014.

ประวัติผู้เขียน

ชื่อ-สกุล	สิโรดม มงคลธนาภรณ์
วัน เดือน ปี เกิด	26 กรกฎาคม 2537
สถานที่เกิด	กรุงเทพมหานคร
วุฒิการศึกษา	วิศวกรรมศาสตรบัณฑิต (วศ. บ.)
ที่อยู่ปัจจุบัน	98 ซ.อนามัยงามเจริญ 25 แยก 1 แขวงท่าข้าม เขตบางขุนเทียน กรุงเทพมหานคร 10150
ผลงานตีพิมพ์	1)S. Mongkhonthanaphon and Y. Limpiyakorn, “Classification of Titanium Microstructure with Fully Convolutional Neural Networks” 11th International Conference on Computer and Electrical Engineering (ICCEE 2018), October 12-14, 2018, Tokyo, Japan. 2)S. Mongkhonthanaphon and Y. Limpiyakorn, “A Deep Neural Network for Pixel-Wise Classification of Titanium Microstructure” International Journal of Machine Learning and Computing (IJMLC) 2nd International Conference on Computer Science and Artificial Intelligence (CSAI2018), December 8-10, 2018, Shenzhen, China.
รางวัลที่ได้รับ	-