Colorization Black & White image using Generative Adversarial Network

Member

Kunuch Terdtinwitid 59070501005

Manchuporn Pungtippimanchai 59070501060

Sarin Anuttranon 59070501073

---

1. Abstract

ในหลายปีที่ผ่านมา มีการคิดค้นขั้นตอนและวิธีการในการแปลงภาพขาวดำให้เป็นภาพสีแบบอัตโนมัติที่เป็นที่น่าสนใจมากมาย และถูกนำไปประยุกต์ใช้ในการกู้คืนภาพเก่าที่มีสภาพเสื่อมโทรม หรือภาพขาวดำในอดีต ซึ่งวิธีการในการแปลงภาพขาวดำเหล่านี้ จะประสบปัญหากับการเลือกสีที่จะใส่ไว้ในตำแหน่งต่าง ๆ ที่มีความอิสระสูง จึงจำเป็นต้องมีการพัฒนาโมเดลในการแปลงภาพขาวดำเป็นภาพสีแบบอัตโนมัติ ที่สามารถจดจำรูปแบบของสีทั่ว ๆ ไปของสิ่งต่าง ๆ และที่ผ่านมาก็มีการคิดค้นวิธีการทำหลากหลายวิธีการ ซึ่งภายในรายงานนี้เล่มจะกล่าวถึงวิธีการในการแปลงภาพขาวดำเป็นภาพสีโดยใช้ conditional Deep Convolutional Generative Adversarial Network (DCGAN) และใช้ CIFAR-10 เป็นชุดข้อมูลสำหรับการสร้างโมเดล

2. Introduction

ในการทำ Automatic Colorization ในการเปลี่ยนภาพขาวดำเป็นภาพสี ซึ่งทำการค้นคว้าด้วยการใช้ Machine Learning ได้ถูกนำมาประยุกต์ใช้ในการกู้คืนภาพสี กู้คืนภาพเก่าที่มีสภาพเสื่อมโทรมให้กลับคืนมา หรือใช้ในการทำแอนิเมชัน ซึ่งภายในรายงานนี้เราจะทำการค้นคว้า และทดลองวิธีการในการลงสีภาพโดยใช้ Generative adversarial network (GANs) โดยเราจะใช้ชุดข้อมูล CIFAR-10 ในการสร้างโมเดลขึ้นมา โดยในการทดลองเราจะใช้การเปรียบเทียบความแตกต่างของโมเดล 2 แบบ โดยแบบแรกคือ GAN model ที่เรากำลังศึกษา เทียบผลลัพธ์กับ Convolution Neuron Network แบบธรรมดา

Model สำหรับการลงสีภาพขาวดำเริ่มต้นประมาณช่วงต้น ๆ ของปีค.ศ. 2000 โดยในปี 2002 มีการคิด algorithm ที่สามารถลงสีภาพขาวดำผ่านการวิเคราะห์ลักษณะของพื้นผิว และใช้การจับคู่ความสว่างกับข้อมูลเกี่ยวกับพื้นผิวเหล่านั้น ผ่านภาพสีที่มีอยู่แล้วและภาพขาวดำที่ถูกลงสี แต่อย่างไรก็ตาม algorithm ดังกล่าว

เป็น forward problem กล่าวคือเริ่มต้นจาก model แล้วนำไปสู่การทำนายผลที่คาดว่าจะได้รับ ฉะนั้นทุกทางออกจะได้รับคำตอบแบบเดียวเท่านั้น ต่อมาในปี 2004 ได้เกิดสมการใหม่ที่ใช้วิธีการของ inverse problem แทน กล่าวคือ การเริ่มจากผลที่ได้รับย้อนกลับไปสร้างเป็น Model ซึ่ง cost function ถูกออกแบบโดยใช้ penalizing ที่แตกต่างกันในแต่ละ pixel และน้ำหนักเฉลี่ยของ pixel รอบข้าง โดยวิธีที่กล่าวมาข้างต้นยังคงต้องมีผู้ใช้เข้ามาแทรกแซงวิธีการเหล่านั้นมากกว่าให้ทำงานแบบในอุดมคติ

ในวิธีการของเรา เราจะทำการเปรียบเทียบการลงสีระหว่าง การสร้างรูปสีจาก CNN กับ GAN และโมเดลที่เรากำลังทำการศึกษา ไม่ได้เรียนรู้แค่วิธีในการ mapping ระหว่าง input และ output เท่านั้น แต่ยังเรียนรู้ loss function ในการ train เพื่อ mapping ซึ่งวิธีการนี้ให้ผลลัพธ์ที่มีประสิทธิภาพมากกับปัญหาในการสังเคราะห์รูปภาพจากแผนผังของ label สร้างรูปขึ้นมาใหม่จากแผนผังของขอบ และการลงสีรูปภาพ

3. Theory

3.1 Generative Adversarial Network(GAN)

ถูกนำเสนอขึ้นในปี 2014 โดย Goodfellow [xx] ซึ่งเป็นหนึ่งในประเภทของ Generative model โดย Generative model เป็นโมเดลประเภทหาวิธีในการสร้างข้อมูลขึ้นมาในรูปแบบของความน่าจะเป็น แต่ GAN จะประกอบไปตัวโครงข่ายเล็ก ๆ สองตัวประกอบกันคือ

3.1.1 Generator Model ทำหน้าที่ในการสร้างผลลัพธ์ของรูปภาพขาวดำที่ทำการวิเคราะห์ โดยต้องสร้างสีออกมาให้แยกไม่ออกจากข้อมูลต้นฉบับ

3.1.2 Discriminator Model ทำหน้าที่ในการแยกผลลัพธ์ที่ได้ออกมาจากตัว Generator model เทียบกับ ตัวอย่างที่ได้มาจากต้นฉบับ ว่ามีความเหมือนกับต้นฉบับหรือไม่

โดย Generative Adversarial Network(GAN) จะเป็นการทำงานร่วมกันของโมเดลทั้งสองแบบ ซึ่งจะถูกเทรนไปเรื่อย ๆ จนกว่า Generator Model จะสามารถสร้างผลลัพธ์ที่ Discriminator ไม่สามารถจำแนกออกได้ หรือคิดว่าภาพที่สร้างสีออกมาคือภาพจริง และเนื่องจากว่าปัญหาการแปลงภาพขาวดำเป็นภาพสี ถือว่าเป็นหนึ่งในปัญหาการแปลงภาพ(ทำงานเกี่ยวกับรูปภาพ) โมเดลทั้งสองตัวจึงมีโครงสร้างเป็น Convolution Neural Network (CNNs)

Generator Model จะถูกอธิบายด้วยสมการ $G(z; \theta_G)$ โดย z แทนด้วย noise variable (การกระจายตัวแบบ uniform) ซึ่งทำตัวเสมือนเป็น input ของตัว Generator ส่วน Discriminator model จะถูกอธิบายด้วย

สมการ D(x; θ_D) ซึ่งจะให้ค่าออกมาอยู่ระหว่าง 0 – 1 เป็นค่าความน่าจะเป็นที่รูปภาพจะเป็นต้นฉบับ โดยที่ x แทนด้วยสีของรูปภาพ โดยโครงสร้างของโมเดลทั้งสองตัวนี้เป็นหนึ่งในปัญหา optimization เมื่อต้องทำการ train โมเดล กล่าวคือ Generator model จะถูก train เพื่อลดความน่าจะเป็นที่ Discriminator model จะทำนายรูปภาพที่สร้างขึ้นโดย Generator ถูก แต่ในขณะเดียวกัน Discriminator model จะถูก train ขึ้นเพื่อเพิ่มความน่าจะเป็นที่จะสามารถแยกได้ว่ารูปใดไม่ใช่รูปต้นฉบับ ซึ่งตัว Generator model สามารถอธิบายสมการได้ดังนี้

$$\min_{\theta_G} J^{(G)}(\theta_D, \theta_G) = \min_{\theta_G} \mathbb{E}_z \left[\log(1 - D(G(z)))\right],$$

ภาพที่ 1 สมการ cost function ของ Generator model

จากสมการ cost function ของ Generator model คือการหาค่าความน่าจะเป็นที่ Discriminator model จะทำนายถูกที่น้อยที่สุด ฉะนั้นหาก D(G(z)) ให้ค่าเข้าใกล้ 1 นั่นหมายความว่า Discriminator model เชื่อว่ารูปที่เราสร้างขึ้นมาเป็นรูปจริง และเมื่อนำมาผ่านฟังก์ชัน log(1-D(G(z))) ยิ่งค่า D(G(z)) เข้าใกล้ 1 ยิ่งจะทำให้ log(1-D(G(z))) มีค่าลู่ออกทางลบนั่นเอง แต่ในขณะเดียวกันหากค่า D(G(z)) เข้าใกล้ 0 จะทำให้ค่า log(1-D(G(z))) มีค่าเข้าใกล้ 0 ซึ่งไม่ตรงตามจุดประสงค์ของสมการที่ต้องการค่าที่น้อยที่สุด ดังนั้นการที่ D(G(z)) ทำให้มีค่าเข้าใกล้ 1 จะเป็นผลดีต่อ Generator Model เพราะมีความหมายว่าสามารถสร้างรูปที่ทำให้ Discriminator model ไม่สามารถแยกออกได้

$$\max_{\theta_D} J^{(D)}(\theta_D, \theta_G) = \max_{\theta_D} \left(\mathbb{E}_x \left[\log(D(x))\right] + \mathbb{E}_z \left[\log(1 - D(G(z)))\right]\right)$$

ภาพที่ 2 สมการ cost function ของ Discriminator model

จากสมการ cost function ของ Discriminator model ซึ่งทำหน้าที่เป็นตัวจำแนกผลลัพธ์ที่ได้มาจากการสร้างรูปภาพของ Generator model กับรูปต้นฉบับ ซึ่งจะเปรียบเทียบว่าสิ่งใดถูกสร้างด้วย Generator model และข้อมูลใดเป็นรูปภาพต้นฉบับ เมื่อเปรียบเทียบแล้วจะส่งค่าไปให้ตัว Generator model ใช้ในการปรับปรุงโมเดลให้ดีขึ้น โดยสมการนี้ประกอบไปด้วยสองส่วนด้วยกัน คือ

- E_x [log D(x)] สำหรับรูปที่มาจากข้อมูลจริง โดย ถ้า D(x) = 1 หมายความว่า Discriminator บอกว่ารูป x เป็นรูปจริง และทำหน้าที่ว่าเป็นรูปจากข้อมูลจริงได้ดีขนาดไหน ฉะนั้นเมื่อทำการผ่าน log function จะได้ค่าเท่ากับ 0 ถ้าเกิดมีส่วนนี้เยอะจะมีผลลัพธ์ที่ดี

- E_z[log (1 - D(G(z)))] เป็นส่วนที่บอกว่า Discriminator สามารถจับรูปที่ถูกสร้างขึ้นได้ดีแค่ไหน ยิ่ง D(G(z)) ให้ค่าเข้าใกล้ 0 จะทำให้ log (1 - D(G(z))) มีค่าเข้าใกล้ 0 แต่ในทางกลับกันถ้า D(G(z)) ให้ค่าเข้าใกล้ 1 จะทำให้ log (1 - D(G(z))) จะมีค่าลู่ออกทางลบ

โดยสมการ Cost function ข้างต้นทั้งสองจะถูกใช้ในการเทรนโมเดล GAN ซึ่งสมการทั้งสองนี้ สามารถนำมารวมกันได้ในรูปแบบของ Single minimax game problem ด้วย function V(G, D)

$$\min_G \max_D V(G,D) = \mathbb{E}_x \left[\log D(x)\right] + \mathbb{E}_z \left[\log(1 - D(G(z)))\right].$$

ภาพที่ 3 สมการ cost function ที่อยู่ในรูป Single minimax

เนื่องจากสมการที่ 1 เป็นสามารถหาค่าที่น้อยที่สุด แต่ค่าคำตอบที่ได้อยู่ในช่วง 0 ถึง -∞ ทำให้พบปัญหา คือ

1) ถ้า Discriminator สามารถทำงานได้ดีมากในขณะที่ทำการ train ตัว Generator จะทำให้เจอปัญหา near-zero gradient เมื่อทำการ Back-propagation (cost function ของ Generator, D(x) สามารถจับได้ทุกครั้งว่าเป็นภาพที่สร้างขึ้นเอง ทำให้ได้ค่าเข้าใกล้ 0 เมื่อผ่านฟังก์ชัน log (1 - D(G(z))) ซึ่งปัญหานี้ทำให้ generator จะมีการพัฒนาที่ช้ามาก ๆ เพราะให้ผลลัพธ์ที่ไม่ต่างกันเมื่อทำการ training

2) จากการที่ cost function สามารถลู่ออก -∞ ได้ทำให้ cost function ของ generator พยายามทำให้เกิดการลู่ออก

เพื่อแก้ไขปัญหาดังกล่าว จำเป็นต้องกำหนด cost function ใหม่โดยการเปลี่ยน objective function จากการหา minimize เป็น maximize แทน จากการการความน่าจะเป็นที่น้อยที่สุดที่ discriminator จะถูก เป็นหาความจะเป็นที่มากที่สุดที่ discriminator จะผิดพลาดแทน

$$\max_{\theta_G} J^{(G)^*}(\theta_D, \theta_G) = \max_{\theta_G} \mathbb{E}_z \left[\log(D(G(z)))\right],$$

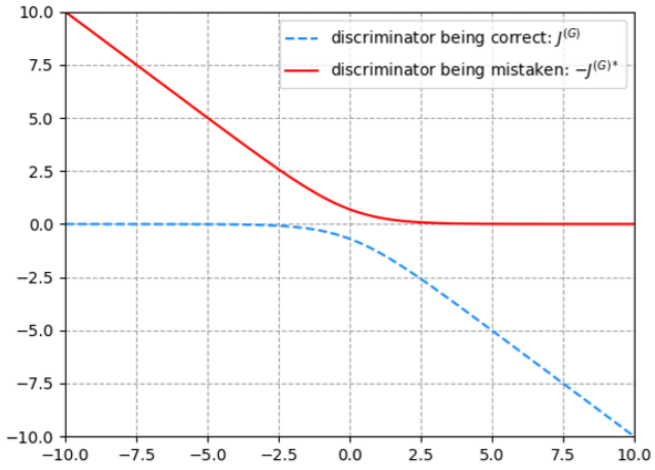
ภาพที่ 4 สมการ cost function ของ generator ที่เปลี่ยน objective function ใหม่

ซึ่งสามารถเขียนในรูปแบบของ minimize problem เป็น

$$\min_{\theta_G} -J^{(G)^*}(\theta_D, \theta_G) = \min_{\theta_G} -\mathbb{E}_z \left[\log(D(G(z)))\right]$$

ภาพที่ 5 สมการของ cost function ของ generator ที่อยู่ในรูปของ minimize problem

เมื่อเปรียบเทียบ cost function ระหว่างสมการที่ 1 และสมการที่ 5 ช่วงของคำตอบจะแสดงได้เป็น graph ดังนี้



ภาพที่ 6 แสดงข้อเปรียบเทียบของ cost function $J^{(G)}$ (เส้นประสีฟ้า) และ $-J^{(G)*}$ (เส้นสีแดง)

นอกจากนี้เพื่อให้ Generator สามารถสร้างผลลัพธ์ที่ใกล้เคียงกับ ground truth image กล่าวคือข้อมูลของรูปสัมพันธ์กับ feature จริง ๆ บนพื้นที่ จึงมีการเพิ่มส่วนของ regularization term ลงไปในสมการ cost function ซึ่งตามทฤษฎีแล้วมันจะให้โครงสร้างของรูปต้นฉบับและป้องกัน Generator กำหนดสีไปที่ pixel ที่หลอก discriminator โดยอธิบายด้วยสมการของ cost function ดังนี้

$$\min_{\theta_G} J^{(G)^*}(\theta_D, \theta_G) = \min_{\theta_G} -\mathbb{E}_z \left[\log(D(G(z)))\right] + \lambda \|G(z) - y\|_1$$

ภาพที่ 7 สมการ cost function ที่มีการใส่ regularization term

โดยแทน λ เป็น regularization parameter และ y คือ ground truth color label

3.2 Conditional GAN

เดิมที input ของตัว Generator จะถูกสร้างขึ้นแบบสุ่ม เรียกว่า Data z แต่วิธีการดังกล่าวไม่สามารถนำมาประยุกต์กับปัญหาการทำ automatic colorization ได้เพราะว่าภาพขาวดำ ไม่ได้เกิดจากการสุ่ม noise ปัญหาดังกล่าวจึงถูกแก้ด้วย GAN แบบอื่นที่ถูกเรียกว่า conditional generative adversarial networks และเนื่องจากเราไม่มี noise ฉะนั้น input ของ generator จะมีค่าเป็น 0 noise พร้อมกับภาพขาวดำ ซึ่งสามารถอธิบายได้ด้วยสมการทางคณิตศาสตร์คือ $G(0_x|x)$ นอกจากนี้ input ของ discriminator จะถูกเปลี่ยนแปลงตามไปด้วยเช่นกัน และสามารถเขียนเป็นสมการได้ดังนี้

$$\min_{\theta_G} J^{(G)}(\theta_D, \theta_G) = \min_{\theta_G} -\mathbb{E}_z \left[\log(D(G(\mathbf{0}_z|x)))\right] + \lambda \|G(\mathbf{0}_z|x) - y\|_1$$

ภาพที่ 8 สมการของ Generator ของ Condition GAN

$$\max_{\theta_D} J^{(D)}(\theta_D, \theta_G) = \max_{\theta_D} \left(\mathbb{E}_y \left[\log(D(y|x))\right] + \mathbb{E}_z \left[\log(1 - D(G(\mathbf{0}_z|x)|x))\right]\right)$$

ภาพที่ 9 สมการของ Discriminator ของ Condition GAN

ตัว Discriminator จะรับสีมาจากทั้งตัว Generator และ ข้อมูลต้นฉบับ โดยมีภาพขาวดำเป็น input เป็นเงื่อนไข และพยายามจะตัดสินใจว่าสีไหนคือสีที่แท้จริง

3.3 วิธีการ

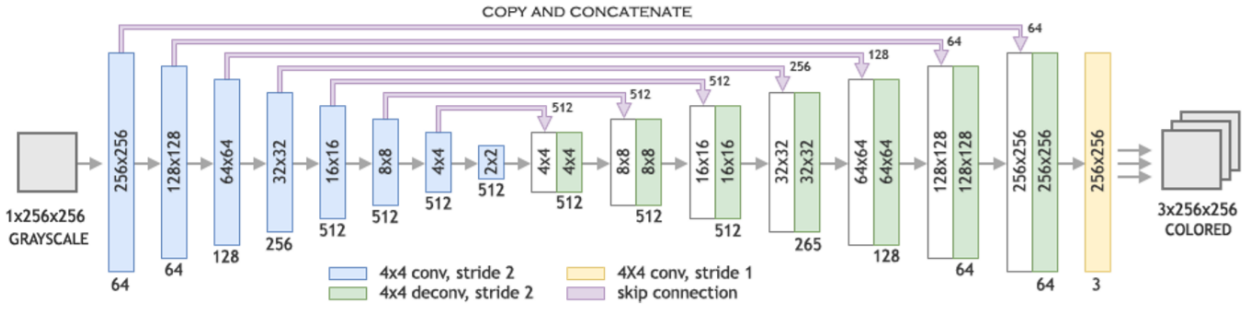
การทำ Image Colorization เป็นการทำ image-image translation problem ซึ่งเป็นการจับคู่ระหว่างภาพ input ที่มีความละเอียดสูง กับ ภาพ output ที่มีความละเอียดสูงเช่นกัน นั่นหมายความว่าโครงข่ายของเราไม่ใช่แค่สร้างผลลัพธ์ออกมาจาก Input แต่จะต้องใส่สีลงไปในแต่ละ pixel ของภาพขาวดำที่เป็น input และใช้โครงข่ายทั้งหมดเป็นโครงสร้างของ convolutional model พร้อมทั้งใช้ regression loss เช่นเดียวกับ baseline และนำมาใช้กับโมเดลของเรา

นอกจากนี้ผลลัพธ์ที่ได้จากโมเดลจะเป็นค่า L*a*b สำหรับการลงสี เพราะมีส่วนที่อธิบายความสว่างของรูปภาพได้ คือ L (Light) และอีก 2 ส่วน a และ b เป็นส่วนที่อธิบายสี ซึ่งผลลัพธ์ที่ได้จะทำให้เราสามารถป้องกันปัญหาความเข้มข้นของสีแบบที่พบเจอในค่า RGB เมื่อมีปัจจัยของความสว่างมาเกี่ยวข้อง

3.4 Baseline Network

สำหรับ baseline model เราทำตาม Fully convolution network model โดยส่วนของ Fully connected layers จะถูกแทนที่ด้วย convolutional layers ซึ่งประกอบไปด้วย up-sampling แทนที่จะเป็นการทำ pooling โดยความคิดนี้มาจากการทำ encoder-decoder networks ที่จะทำการ down-sampling input ต่อเนื่องไปเรื่อย ๆ เรียกว่า contractive encoding layer และจะถูกสร้างขึ้นมาใหม่โดยการทำกลับกัน สร้าง input ขึ้นมาใหม่ เรียกว่า expansive decoding layer ซึ่งข้อดีของการทำโมเดลแบบ end-to-end คือเราจะใช้ memory ไม่มาก แต่การทำ down-sampling จะทำให้เกิดการบีบอัดของ feature ในชั้นตรงกลาง ซึ่งเป็นส่วนสำคัญมาก แต่ในขณะเดียวกัน ความละเอียดของตรงส่วนนี้จะถูกจำกัดด้วย GPU memory

เนื่องจาก baseline model จำเป็นต้องทำการแปลงเป็นสีจากภาพขาวดำ แต่อย่างไรก็ตามยังมีข้อมูลบางส่วนที่เป็น bottleneck ซึ่งป้องกันข้อมูลที่ low level ไม่ให้ผ่านไปได้ในโครงสร้าง encoder-decoder เพื่อแก้ไขปัญหาดังกล่าว Feature จาก contracting path ในโครงข่ายจะต้องถูกนำมารวมเข้าด้วยกันเมื่อต้องทำการ up-sampled output ในช่วงของ expansive path ของโครงข่าย ซึ่งการทำเช่นนี้ยังทำให้ input และ output มีการแลกเปลี่ยนตำแหน่งที่เป็นขอบที่เด่นชัดของรูปในรูปภาพขาวดำและภาพสี ซึ่งโครงสร้างนี้เราเรียกว่า U-Net และทำการกระโดดข้ามเพื่อเชื่อมต่อ feature แบบเช่นเดียวกับกระจก คือ ที่ layer ตำแหน่ง I และ layer ตำแหน่ง n-i ทำให้โครงสร้างของ U-Net มีลักษณะเป็นสมมาตร



ภาพที่ 8 โครงสร้างของ U-Net

จากรูป เราจะมี n encoding unit และ n decoding unit และในแต่ละชั้นจะมีใช้ Batch normalization กับ LeakyReLU เป็น Activation function ด้วย slope เท่ากับ 0.2 สำหรับส่วนของ expansive path จะเป็นการ

ทำ up sampling และ ใช้ activation map ของ layer ที่อยู่ตรงข้ามแบบสมมาตร สำหรับ layer สุดท้ายของโครงข่ายจะใช้ 1x1 convolution ซึ่งเทียบเท่ากับ cross-channel parametric pooling layer และใช้ tanh เป็น activation function และ channel ของ output layer จะประกอบไปด้วย 3 channel คือ L*a*b

นอกจากนี้เรา train baseline model เพื่อลด Euclidean distance ระหว่างค่าที่ทำนายได้กับค่าเฉลี่ย ground truth ของทุก pixel

$$J(x;\theta) = \frac{1}{3n}\sum_{\ell=1}^{3}\sum_{p=1}^{n}\|h(x;\theta)^{(p,\ell)} - y^{(p,\ell)}\|_2^2$$

ภาพที่ 9 สมการหา Euclidean distance

โดยที่ x คือ input ที่เป็นภาพขาวดำ, y คือภาพสี, p และ ℓ คือตำแหน่งของ pixel และ channel ของสีตามลำดับ n คือจำนวน pixel ทั้งหมด และ h คือฟังก์ชันที่ mapping ระหว่างภาพขาวดำและภาพสี

3.5 Convolutional GAN

สำหรับ Generator model และ Discriminator model เราจะทำตามแนวทางของ Deep Convolutional GANs (DCGAN) แต่โครงสร้างจำเป็นต้องถูกดัดแปลงเป็น Conditional GAN แทนที่จะเป็น DCGAN แบบเดิม ซึ่งเราจะมี noise แค่อยู่ในรูปแบบของ dropout เท่านั้น สำหรับโครงสร้างของ Generator model จะมีโครงสร้างเดียวกับ Baseline (U-Net)

สำหรับ Discriminator model จะมี input 2 ตัวคือ 1.รูปภาพที่ Generator model สร้างมา(ภาพสี) และ 2.รูปภาพต้นฉบับ เพื่อใช้ในการเปรียบเทียบ โดยโครงสร้างที่ใช้จะเหมือนกับ baseline(U-Net) ตรงส่วนของ Contractive path โดยมีการใช้ stride 2 กับจำนวนของ Channel เพื่อทำ downsampling จากนั้นทุก Convolution layer จะตามด้วย BatchNormalization และ LeakyReLU เป็น Activation Function หลังจากนั้น layer สุดท้ายจะเป็น Convolution layer อีกครั้งเพื่อให้ได้ dimension เหลือแค่ 1 เป็นผลลัพธ์ออกมา โดยใช้ sigmoid เป็น Activation Function ซึ่งจะให้ค่าผลลัพธ์เป็นความน่าจะเป็นของ input ว่าเป็นรูปจริงหรือรูปที่ถูกสร้างขึ้นมา

4. Experimental Design and Results

ในการทดลองของเราจะทำการทดลองกับโมเดล 4 รูปแบบ คือ

1.CNN(U-net)

2.CNN(U-net) ใช้ร่วมกับ LeakyRelu และ BatchNormaliztion

3.GAN

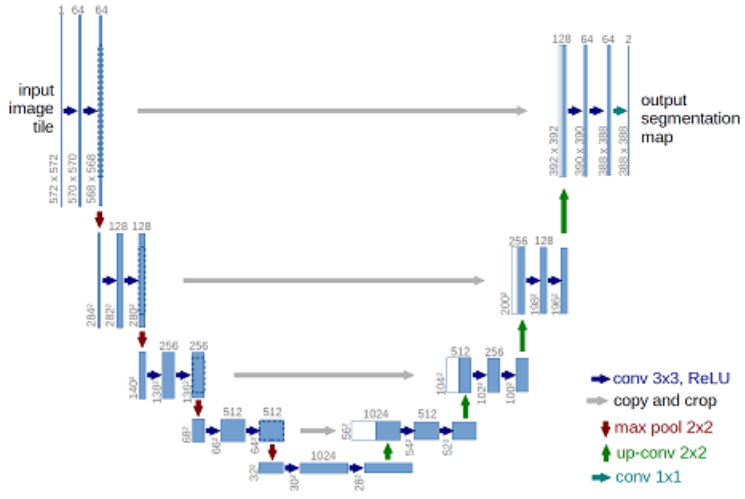
4.GAN ใช้ร่วมกับ LeakyRelu และ BatchNormalization

เพื่อเปรียบเทียบผลลัพธ์ในการสร้างภาพสี จากภาพขาวดำ ระหว่าง CNN(U-net) กับ GAN และเพื่อทดสอบ LeakyRelu และ BatchNormalization ว่ามีผลต่อโมเดลมากเพียงใด

Dataset: สำหรับการทดลองของเรานั้นจะใช้ CIFAR-10 [1] มีขนาด 32x32 pixels จำนวน 60,000 ภาพ โดยแบ่งเป็นภาพสำหรับ training data 50,000 ภาพ และ testing data 10,000 ภาพ

4.1 Implementation details

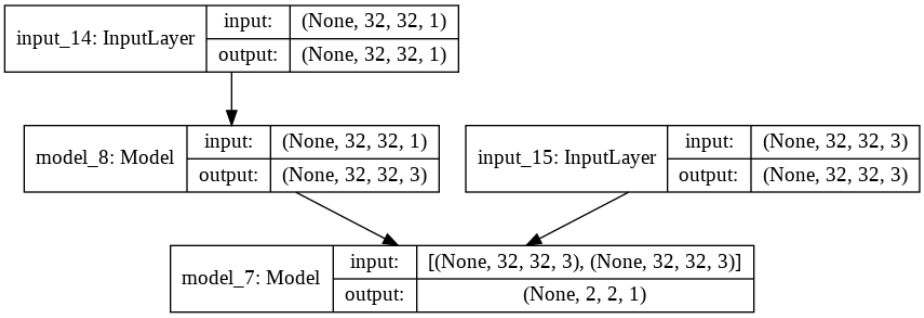
1. CNN (U-net) รูปแบบที่ 1: ในการทดลองของเรานั้นได้นำ CNN(U-net) รูปแบบที่ 1 อ้างอิงจาก [2] ซึ่งเป็นการออกแบบ U-net แบบทั่วไป มาทดลองแปลงภาพขาว-ดำ เป็นภาพสี โดยมีโครงสร้างของโมเดลดังภาพที่ 10



ภาพที่ 10 U-net

2. CNN (U-net) รูปแบบที่ 2: เราได้ดัดแปลง U-net รูปแบบที่ 1 โดยในทุกๆ Convolution layer เราได้เพิ่ม Batch normalization และใช้ Activation function เป็น LeakyReLU อ้างอิงจาก [3] ซึ่งคาดหวังว่าจะทำให้การทำนายขอองโมเดลมีประสิทธิภาพดีขึ้น

3. GAN รูปแบบที่ 1: โมเดลของเราประกอบไปด้วย Generator model(U-Net) และ Discriminator model โดยใน Discriminator model จะมีการใช้ Batch Normalization และใช้ Activation function เป็น LeakyReLU ทุก Convolution layer อ้างอิงจาก [4] ยกเว้นตัว Generator model ที่ไม่ได้ใช้ BatchNormalization และ LeakyRelu ในการเทรนโมเดลนี้จะแบ่งการเทรนออกเป็น 2 ส่วน ซึ่งจะเทรนตัว Discriminator model ก่อนแล้วจึงเทรนตัว GAN model รวมทีหลัง ซึ่งในการเทรน GAN model จะไม่มีการเทรน Discriminator model ร่วมด้วย ซึ่งจะใช้มาคำนวณในการปรับ weight ของ Generator model เท่านั้น จะได้รูปแบบของโมเดล ดังภาพที่ 11



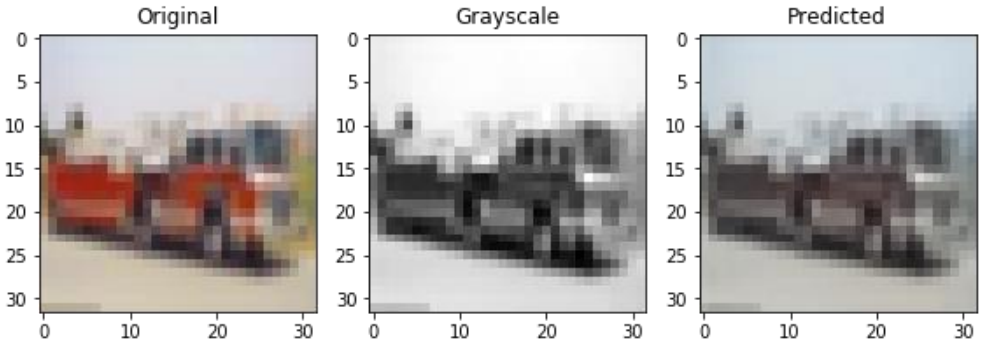
ภาพที่ 11 Generative Adversarial Network

4. GAN รูปแบบที่ 2: โมเดลของเราประกอบไปด้วย Generator model(U-Net รูปแบบที่ 2) ที่มี BatchNormalization และ LeakyRelu ในส่วนของ Discriminator model จะใช้เหมือนกันกับ GAN รูปแบบที่ 1 ซึ่งอ้างอิงจาก [3] ที่กล่าวว่าจะทำให้การทำนายผลลัพธ์จะมีประสิทธิภาพดีขึ้น

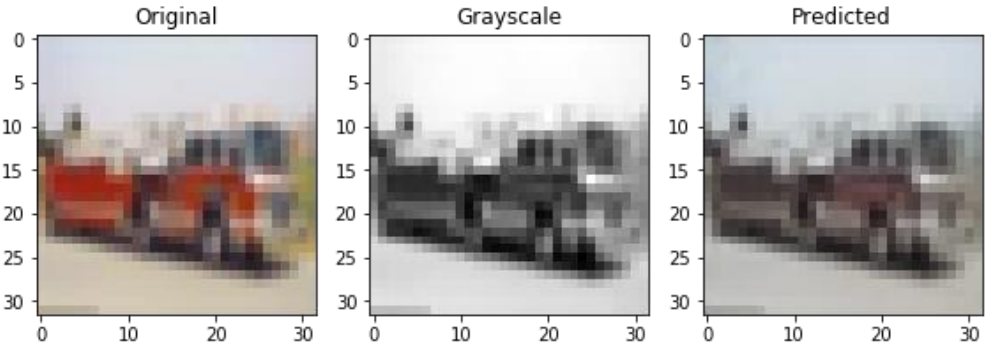
4.2 Evaluation

Dataset	Network	Epoch	Accuracy	Loss
CIFAR-10	CNN(U-net)	15	0.012071	0.556810
CIFAR-10	CNN(U-net) (LeakyRelu)	15	0.012075	0.556328
CIFAR-10	GAN	15	0.012059	0.559959
CIFAR-10	GAN (LeakyRelu)	15	0.012037	0.567164

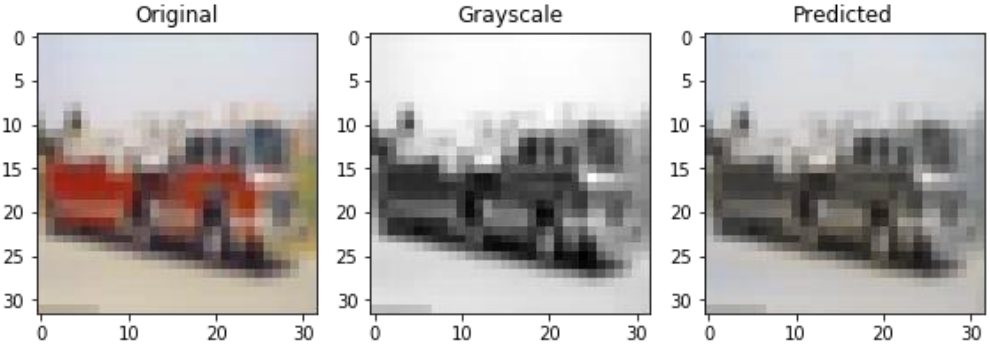
4.3 Results



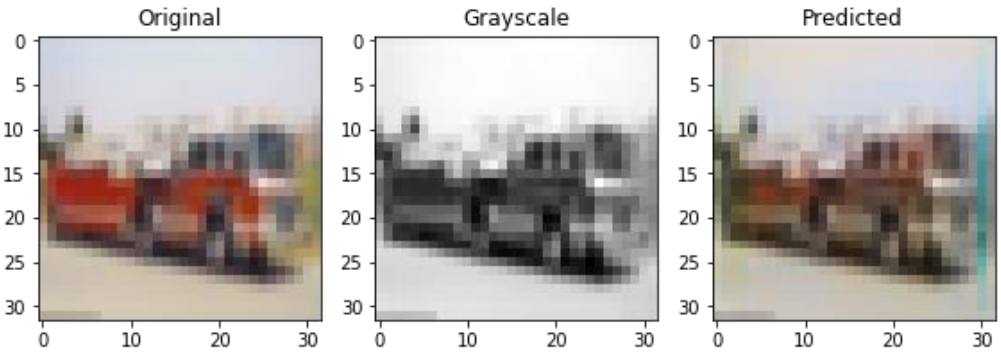
ภาพที่ 12 ผลลัพธ์จากการทำนายด้วย U-net รูปแบบที่ 1



ภาพที่ 13 ผลลัพธ์จากการทำนายด้วย U-net รูปแบบที่ 2



ภาพที่ 14 ผลลัพธ์จากการทำนายด้วย GAN, U-net 1



ภาพที่ 15 ผลลัพธ์จากการทำนายด้วย GAN, U-net 2

จากการทดลองพบว่า ในการใช้ U-Net model จะให้การ Generate สีออกมาเป็นสีโทนเดียว เช่น สีน้ำตาลก็จะน้ำตาลทั้งรูป ทำให้ทั้งภาพไม่ว่าจะเป็นสิ่งของอะไรก็ตามจะเป็นสีในโทนเดียวกัน ส่วนการใช้ LeakyRelu ทำให้การทำนายได้ผลลัพธ์ที่ดีขึ้น

ในส่วนของ GAN model จะสามารถ Generate สีออกมาได้มากกว่า คือจะมีหลายสีในภาพ ซึ่งในกรณีการเพิ่ม LeakyRelu จะทำให้สีของภาพถูก generate ออกมาชัดมากขึ้น แต่ยังคงมีข้อจำกัดในการเทรนโมเดลอยู่ คือ โมเดลทั้ง 4 แบบ ยังเทรนมาได้น้อยเกินไป ทำให้ยังมีข้อผิดพลาดในการใส่สีในแต่ละจุด และสีบางสียัง generate ออกมาได้ไม่ชัดเจนพอ เช่น สีเขียว

5. Conclusion

จากผลการทดลอง เราจะได้ว่า GAN model จะสามารถ generate สีออกมาได้มากกว่า และการใช้ LeakyRelu กับ Batch normalization ทำให้ผลลัพธ์การทำนายดีขึ้น

ในการเทรนโมเดล เราสามารถที่จะเทรนได้แค่เพียง โมเดลละ 15 Epochs เนื่องจาก ตัว GAN model ใช้เวลาในการเทรนนาน จากการศึกษา และคำนวณเวลาออกมาแล้ว ต้องใช้เวลาในการเทรน GAN model อย่างน้อย 8 วัน

ในเรื่องของ dataset เราใช้รูปภาพจาก CIFAR-10 ซึ่งมีขนาด 32*32 ซึ่งเป็นรูปที่มีขนาดเล็ก อาจจะต้องมีการเปลี่ยน dataset ไปใช้ตัวอื่นที่มีขนาดใหญ่กว่า เช่น Place 365 เพื่อดูผลลัพธ์การเทรนที่ชัดเจนกว่า เพราะมีองค์ประกอบของรูปภาพที่มากกว่า แต่ก็จะใช้เวลาในการเทรนมากกว่าเช่นกัน

และจากการทดลอง พบว่าต้องมีการทดลองในแบบต่าง ๆ เพิ่มมากขึ้น คือ ในการทดลองที่ผ่านมาใช้การวัดความถูกต้องของโมเดลเป็นค่า loss และ accuracy ซึ่งพบว่าไม่สามารถนำมาวัดกับการทำ Image Colorization ได้ ซึ่งต้องมีการทดลองเพิ่มด้วยการใช้ MAE ในการวัดค่าแทน

และในการทดลองนี้ เราใช้รูปในรูปแบบของ RGB ในการเทรน ซึ่งพบว่ามีรูปแบบอื่น เช่น Lab ในการเทรนแทนได้

6. References

[1] Wikipedia.: CIFAR-10. สืบค้นเมื่อ 20 พฤศจิกายน 2562 จาก https://en.wikipedia.org/wiki/CIFAR-10

[2] zhixuhao.: unet for image segmentation. สืบค้นเมื่อ 20 พฤศจิกายน 2562 จาก https://github.com/zhixuhao/unet

[3] Kamyar N., Eric Ng., Mehran E.: Image Colorization using Generative Adversarial Networks. สืบค้นเมื่อ 18 พฤศจิกายน 2562 จาก https://arxiv.org/pdf/1803.05400.pdf

[4] Jason Brownlee.: How to Develop a Pix2Pix GAN for Image-to-Image Translation. สืบค้นเมื่อ 21 พฤศจิกายน 2562 จาก https://machinelearningmastery.com/how-to-develop-a-pix2pix-gan-for-image-to-image-translation