**รายงาน 1**

หัวข้อ

* ศึกษาการ Fine-tune โมเดล Stable Diffusion ให้เข้ากับ Dataset ที่ต้องการ ในที่นี้เป็นภาพสถานที่
* ทำความเข้าใจโค้ดบางส่วนที่ต้องใช้ประกอบการ Fine-tune

การ Fine-tune โมเดล

คือ การปรับเปลี่ยนบางสิ่งเพียงเล็กน้อยเพื่อให้ได้ประสิทธิภาพตามที่ต้องการ (พจนานุกรม Oxford) ฉะนั้นการ Fine-tune โมเดล Stable Diffusion ในโปรเจกต์นี้ ก็คือ การปรับเปลี่ยนด้วยการให้โมเดลเรียนรู้ภาพประเภทหนึ่ง ๆ เพื่อให้โมเดล ผลิตภาพให้มีลักษณะค่อนไปทางนั้นได้ดีขึ้น อาทิ ถ้า Fine-tune ด้วยภาพตัวการ์ตูน โมเดลนั้นก็จะผลิตภาพออกมาไปในรูปแบบที่เหมือนตัวการ์ตูนมากขึ้น

ภาพที่ใช้ Fine-tune จะต้องมีลักษณะจำเพาะที่เด่นชัด ยกตัวอย่างเช่น ภาพโลโก้ ภาพเสื้อผ้า เพราะถ้า Fine-tune ด้วยภาพที่มีความแตกต่างมากเกินไป อาทิ ภาพสถานที่ท่องเที่ยวที่ไม่ได้เจาะจง โมเดลจะผลิตภาพออกมาได้เละ

ต่อมาเป็นเรื่องโค้ดที่ใช้ประกอบการ Fine-tune

from google.colab import drive

drive.mount('/content/drive/', force\_remount=True)

data\_path = "/content/drive/My Drive/Colab\_Datasets"

โค้ดชุดนี้ ทำหน้าที่เชื่อมต่อ Google Drive ของบัญชีที่ใช้บน Colab โดยการเรียก Path ก็ทำด้วยการอ้างอิงเป็น “/content/drive/My Drive/…” โดยข้อมูลใน Google Drive จะมีทั้งที่เป็นภาพที่ต้องการนำมา Fine-tune และไฟล์ .csv (Microsoft Excel Comma Separated Values File) ที่มีไว้เก็บชื่อภาพที่ไว้ใช้อ้าง path และ Caption ของภาพ

รูปภาพประกอบด้วย โต๊ะ

คำอธิบายที่สร้างโดยอัตโนมัติ

**รายงาน 2**

หัวข้อ

* ทำความเข้าใจโค้ดบางส่วนที่ยังไม่ได้ทำความเข้าใจ เพื่อให้รู้ว่า โค้ดแต่ละส่วน มีที่มาที่ไปอย่างไร

มาทำความเข้าใจโค้ดต่าง ๆ ที่ต้องใช้ในการ Fine-tune กันเถอะ!

from google.colab import drive

drive.mount('/content/drive/', force\_remount=True)

  # เป็นการเชื่อม Google Drive กับ Colab

  # force\_remount คือ บังคับให้เชื่อมทุกครั้งที่โค้ดรันผ่าน

data\_path = "/content/drive/My Drive/Colab\_Datasets/pokemon\_dataset"

  # path ใน Google Drive ที่มีทั้งไฟล์ภาพ และไฟล์ .csv ที่ระบุชื่อไฟล์ภาพพร้อม Caption

data\_frame = pd.read\_csv(os.path.join(data\_path, "data.csv"))

  # os.path.join(a, "b") มันก็เหมือน concat ของ String เพียงแต่เปลี่ยนรูปเป็น path แทน

  # os.path.join(data\_path, "data.csv") จะกลายเป็น

  # /content/drive/My Drive/Colab\_Datasets/pokemon\_dataset/data.csv

  # pd.read\_csv จะเป็นการอ่าน path ที่อยู่ในรูปของ .csv (Microsoft Excel)

  # แล้วแปลงให้อยู่ในรูปของตาราง ดังตัวอย่างข้างล่าง

  #     image\_path    caption

  # 0   image\_0.png   a drawing of a green pokemon with red eyes

  # 1   image\_1.png   a green and yellow toy with a red nose

  # 2   image\_2.png   a red and white ball with an angry look on its...

data\_frame["image\_path"] = data\_frame["image\_path"].apply(

    lambda x: os.path.join(data\_path, x)

)

  # ข้อมูลใน Column "image\_path" แทนที่ด้วย os.path.join(data\_path, x)

  # โดย x คือ path ใน image\_path นั้น ๆ ส่วน data\_path ก็คือ

  # /content/drive/My Drive/Colab\_Datasets/pokemon\_dataset/

data\_frame.head()

  # ไล่แสดงส่วนหัว (คล้าย ๆ กับคำสั่ง head ใน Linux เลย)

PADDING\_TOKEN = 49407

MAX\_PROMPT\_LENGTH = 77

tokenizer = SimpleTokenizer()

def process\_text(caption):

    tokens = tokenizer.encode(caption)

    # แปลงให้ชุดอักขระ กลายเป็น token ในรูป array ตัวเลข

    # เช่น จาก "a" เป็น [49406, 320, 49407]

    # จาก "an" เป็น [49406, 550, 49407]

    # และจาก "and" เป็น [49406, 537, 49407]

    # สังเกตว่า a, an, and เมื่อ encode แล้ว ข้อมูลจะแตกต่างกันอย่างสิ้นเชิง

    # จะทำให้ไม่มองว่า and เกิดจาก a ประกอบกับ n และ d

    # ทำให้สามารถทำให้เครื่องเรียนรู้ความแตกต่างของคำได้อย่างชัดเจน

    tokens = tokens + [PADDING\_TOKEN] \* (MAX\_PROMPT\_LENGTH - len(tokens))

    # ทำให้ array หนึ่ง token มีความยาวสูงสุด 77 ตัว ตาม MAX\_PROMPT\_LENGTH

    # ส่วน PADDING\_TOKEN ก็คือขนาดตัวเลขสูงสุดในแต่ละ index ก็คือ 49407

    return np.array(tokens)

    # ส่งค่ากลับเป็น array ที่อยู่ในรูป token

tokenized\_texts = np.empty((len(data\_frame), MAX\_PROMPT\_LENGTH))

# len(data\_frame) คือ ขนาดของ data\_frame

# np.empty(axb) คือ สร้าง array ขนาด axb ที่ข้อมูลข้างในจะเป็นค่าขยะ

all\_captions = list(data\_frame["caption"].values)

# สร้าง list จากค่าตาม column ชื่อ caption

for i, caption in enumerate(all\_captions):

    tokenized\_texts[i] = process\_text(caption)

    # เติม tokenized\_text ซึ่งเป็น array ที่มีแต่ค่าขยะ

    # ให้กลายเป็น token ที่ผ่านเมธอด process\_text(caption)

**ความแตกต่างระหว่างข้อความ**

**ที่ไม่ได้ทำให้เป็น Token กับที่ทำให้เป็น Token**



รูปภาพประกอบด้วย แผนภาพ

คำอธิบายที่สร้างโดยอัตโนมัติ

RESOLUTION = 256

AUTO = tf.data.AUTOTUNE

POS\_IDS = tf.convert\_to\_tensor([list(range(MAX\_PROMPT\_LENGTH))], dtype=tf.int32)

# สร้างวัตถุที่ถูกแปลงเป็น Tensor (a.k.a. multi-dimensional array)

# เทียบกับ array ที่เป็น 1 มิติ และ matrix ที่เป็น 2 มิติ

augmenter = keras\_cv.layers.Augmenter(

    # ตัวทำหน้าที่ดัดแปลงรูป

    layers=[

        keras\_cv.layers.CenterCrop(RESOLUTION, RESOLUTION),

        # crop ภาพบริเวณตรงกลาง ความละเอียด 256 x 256

        keras\_cv.layers.RandomFlip(),

        # สุ่มพลิกกลับภาพ

        tf.keras.layers.Rescaling(scale=1.0 / 127.5, offset=-1),

        # Normalize ด้วยการปรับช่วงค่าแต่ละ channel ของภาพ จาก 0->255 ให้เป็น 0->2

        # offset คือเลื่อน ในที่นี้คือเลื่อนเพิ่มขึ้นไป -1 จาก 0->2 ให้เป็น (-1)->1

    ]

)

text\_encoder = TextEncoder(MAX\_PROMPT\_LENGTH)

# สร้าง TextEncoder ขนาด MAX\_PROMPT\_LENGTH

ละ channel ของภาพ จาก 0->255 ให้เป็น 0->2

        # offset คือเลื่อน ในที่นี้คือเลื่อนเพิ่มขึ้นไป -1 จาก 0->2 ให้เป็น (-1)->1

    ]

)

text\_encoder = TextEncoder(MAX\_PROMPT\_LENGTH)

# สร้าง TextEncoder ขนาด MAX\_PROMPT\_LENGTH

def process\_image(image\_path, tokenized\_text):

    image = tf.io.read\_file(image\_path)

    # ดึงภาพจาก path

    image = tf.io.decode\_png(image, 3)

    # แกะภาพให้อยู่ในรูปที่มี channel 3 ช่อง (RED, GREEN, BLUE)

    image = tf.image.resize(image, (RESOLUTION, RESOLUTION))

    # ปรับขนาดภาพ ให้มีความชัดเป็น 256x256

    return image, tokenized\_text

    # ส่งค่ากลับเป็นภาพที่ถูกประมวลผล และ tokenized\_text

def apply\_augmentation(image\_batch, token\_batch):

    return augmenter(image\_batch), token\_batch

    # ดัดแปลงเฉพาะแค่ภาพ

def run\_text\_encoder(image\_batch, token\_batch):

    return (

        image\_batch,

        token\_batch,

        text\_encoder([token\_batch, POS\_IDS], training=False),

    )

    # ทำให้อยู๋ในรูปของ กลุ่มภาพ, กลุ่ม token และ ข้อความที่ถูก encode

def prepare\_dict(image\_batch, token\_batch, encoded\_text\_batch):

    return {

        "images": image\_batch,

        "tokens": token\_batch,

        "encoded\_text": encoded\_text\_batch,

    }

    # สร้างพจนานุกรมที่มี ภาพ, token และ ข้อความที่ถูก encode

def prepare\_dataset(image\_paths, tokenized\_texts, batch\_size=4):

# เตรียม dataset ด้วยการ

    dataset = tf.data.Dataset.from\_tensor\_slices((image\_paths, tokenized\_texts))

    # หั่นภาพและข้อความที่เป็น token ออกจากรูป Tensor

    dataset = dataset.shuffle(batch\_size \* 10)

    # สับ dataset แบบสุ่ม ๆ โดยให้แต่ละครั้งมีจำนวนที่ถูกสับอยู่ 4 x 10

    dataset = dataset.map(process\_image, num\_parallel\_calls=AUTO).batch(batch\_size)

    # เรียกใช้เมธอด process\_image โดยจะรันเมธอดแบบ parallel อย่างอัตโนมัติ

    # เสร็จแล้วก็แบ่งชุด Dataset ให้แต่ละชุดมีขนาดเป็น 4 ตาม batch\_size

    dataset = dataset.map(apply\_augmentation, num\_parallel\_calls=AUTO)

    # เรียกใช้เมธอด apply\_augmentation และรันเมธอดแบบ parallel อย่างอัตโนมัติ

    dataset = dataset.map(run\_text\_encoder, num\_parallel\_calls=AUTO)

    # เรียกใช้เมธอด run\_text\_encoder และรันเมธอดแบบ parallel อย่างอัตโนมัติ

    dataset = dataset.map(prepare\_dict, num\_parallel\_calls=AUTO)

    # เรียกใช้เมธอด prepare\_dict และรันเมธอดแบบ parallel อย่างอัตโนมัติ

    return dataset.prefetch(AUTO)

    # ส่งค่ากลับเป็น dataset ที่จะส่งไปให้อย่างอัตโนมัติ

training\_dataset = prepare\_dataset(

    np.array(data\_frame["image\_path"]), tokenized\_texts, batch\_size=4

)

# สร้าง dataset ที่ผ่านการเตรียม โดยใช้ข้อมูลจาก data\_frame คอลัมน์ image\_path

# ข้อความที่ถูกทำเป็น token และให้ batch\_size เป็น 4

sample\_batch = next(iter(training\_dataset))

# ดึงตัวอย่าง training\_dataset

for k in sample\_batch:

    print(k, sample\_batch[k].shape)

    # แสดง sample\_batch แต่ละตัว พร้อมทั้งระบุขนาดด้วย

**รายงาน 3**

หัวข้อ

* รันโค้ดที่หาได้บน Colab เพื่อทดสอบว่า สามารถทำงานได้จริงหรือไม่ และถ้าหากเกิดปัญหา จะต้องหาทางแก้ไขว่าต้องทำอย่างไร

ผลการรัน

* สิ่งที่ค้นพบ (เพิ่งรู้) คือ Colab ไม่รอบรับการป้อนข้อมูลด้วยแป้นพิมพ์ ทำให้โค้ดที่ต้องมีการยืนยันด้วยแป้นพิมพ์ ไม่สามารถรันต่อได้ วิธีแก้ปัญหาคือ ต้องรันด้วยเครื่องคอมพิวเตอร์ของตนเองแทน
* โค้ด Finetune ที่ใช้ไลบรารี่ Keras จะกินทรัพยากรมากกว่า Pytorch มาก ๆ ทำให้ไม่สามารถใช้รันด้วย Colab โดยที่ไม่เสียค่า Compute unit ได้ วิธีแก้ปัญหาคือ อาจต้องจ่ายเงินอย่างน้อยไม่เกิน 350 เพื่อซื้อ Compute unit 100 หน่วย โดยวิธีแก้ปัญหานี้ จะใช้เป็นทางเลือกสุดท้ายเท่านั้น เพราะแผนเดิมของโปรเจกต์ ไม่ได้ต้องการให้เสียเงิน