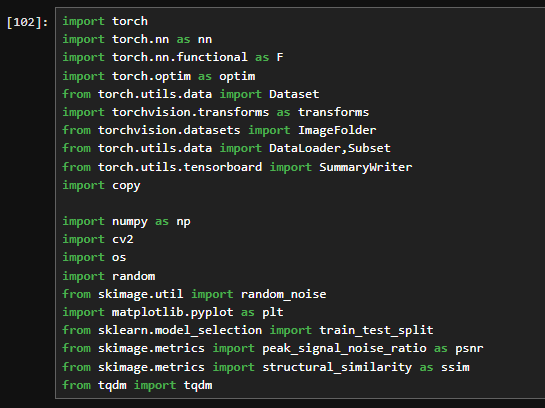
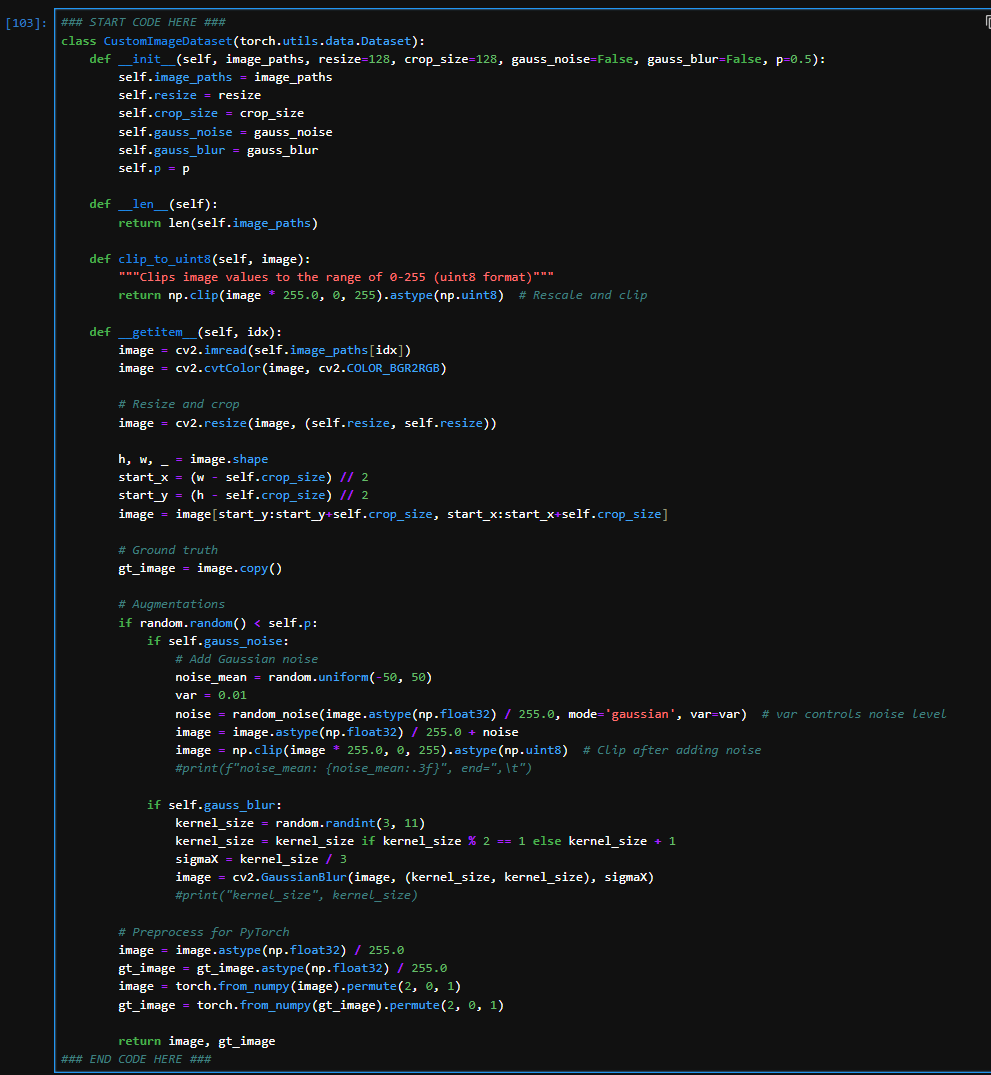
**Lab 5.2**

**Denoise & Deblur image with Autoencoder**

**Data Preparation**



[1] Import Library ที่จำเป็นที่ต้องใช้ใน Lab 5.1 นี้



[2] สร้าง class CustomImageDataset ซึ่งสืบทอดมาจากคลาส Dataset ของไลบรารี torch.utils.data ใน PyTorch

โดยที่ **init**:

* + image\_paths (รายการของ path): รายการของ path ไปยังไฟล์ภาพ
  + resize (int, optional): ขนาดที่จะ resize ภาพ (default: 128)
  + crop\_size (int, optional): ขนาดที่จะ crop ส่วนกลางของภาพที่ resize แล้ว (default: 128)
  + gauss\_noise (bool, optional): เปิด/ปิด การสุ่มเพิ่มสัญญาณรบกวนแบบ Gaussian (default: False)
  + gauss\_blur (bool, optional): เปิด/ปิด การสุ่มเบลอภาพแบบ Gaussian (default: False)
  + p (float, optional): ความน่าจะเป็นในการประยุกต์ใช้การสุ่ม (default: 0.5)

- designed to load images, apply transformations, and prepare them for training or evaluation in a neural network.

**def \_\_len\_\_:**

Returns the total number of images in the dataset. (จำนวนภาพ)

**def clip\_to\_uint8:**

ensure they are in the range of 0-255, which is standard for image data stored in uint8 format.

**def \_\_getitem\_\_:**

1. Load image & Converts the image from BGR to RGB format
2. Resize and Crop

* The image is resized to the specified dimensions (resize x resize).
* The center of the resized image is cropped to the specified crop\_size.

1. Ground Truth : สร้างสำเนาของภาพต้นฉบับ (gt\_image) สำหรับเปรียบเทียบผลหลังการประยุกต์ใช้การสุ่ม

* The original image is copied and stored as gt\_image, which serves as the ground truth for training. This is useful in tasks like denoising or image restoration, where the network is expected to learn to reconstruct the original image

1. Augmentations

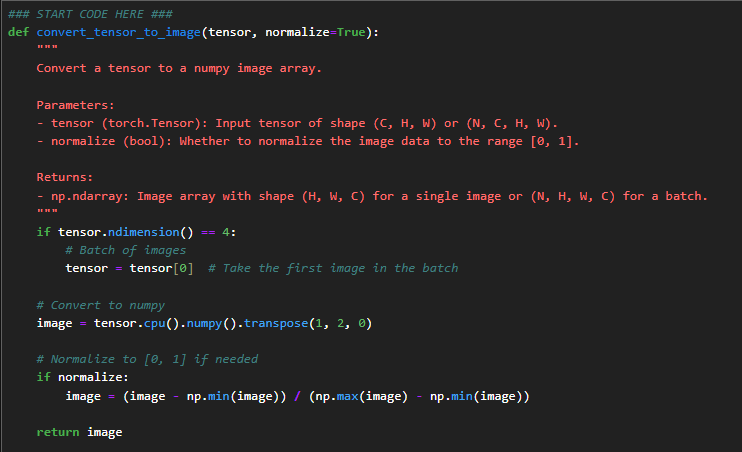
* Gaussian Noise: Adds random noise to the image (ถ้า gauss\_noise เป็นจริง)
* Gaussian Blur: Applies a Gaussian blur to the image (ถ้า gauss\_blur เป็นจริง)

1. Preprocessing

* normalized to the [0, 1] range
* converted from NumPy arrays to PyTorch tensors, and their dimensions are permuted to [C, H, W] format, which is the format expected by PyTorch.

1. Return

* returns the processed image and its corresponding ground truth image as a tuple. These will be used for training or evaluation.

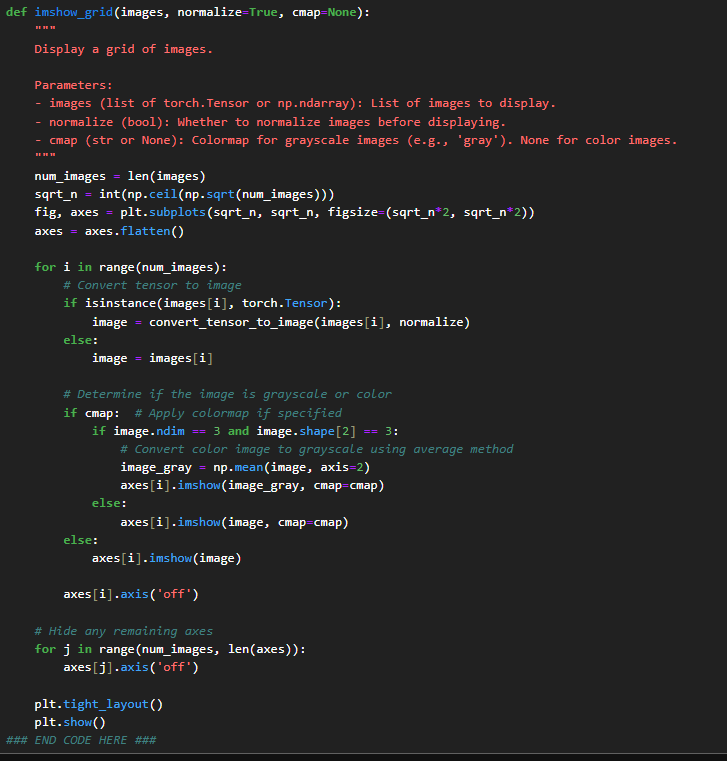


[3] **convert\_tensor\_to\_image(tensor, normalize=True)**

Convert a tensor to a numpy image array.

โดยที่ tensor สามารถมีมิติได้ 3 (C, H, W) สำหรับภาพเดี่ยว หรือ 4 (N, C, H, W) สำหรับชุดภาพ (Batch)

* **Handle Batch of Images:**
* If the input tensor has 4 dimensions (N, C, H, W), it means it's a batch of images. The function selects the first image in the batch for conversion.
* **Convert to NumPy**
* The tensor is moved to the CPU (if it's not already there) and converted to a NumPy array.
* **‘transpose(1, 2, 0)’:** reorders the dimensions of the array from (C, H, W) to (H, W, C), which is the standard format for image arrays.
* **Normalize the Image**
* normalizes the image data to the range [0, 1]
* **Return Image**



[3] **imshow\_grid(images, normalize=True, cmap=None)**

ฟังก์ชันนี้คำนวณจำนวนภาพทั้งหมด (num\_images) และสร้างตารางแสดงผลแบบกริด (grid) โดยแบ่งตามจำนวนภาพที่เหมาะสมCalculate Grid Size

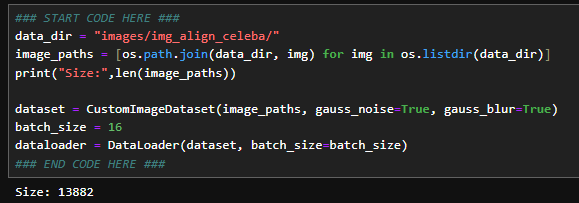
* **num\_images = len(images)**
* **sqrt\_n = int(np.ceil(np.sqrt(num\_images)))**
* **fig, axes = plt.subplots(sqrt\_n, sqrt\_n, figsize=(sqrt\_n\*2, sqrt\_n\*2))**
* **axes = axes.flatten()**

เลือกขนาดที่จะ plot

* **Loop Through and Display Each Image**

iterates over the list of images

* If an image is a PyTorch tensor, it is converted to a NumPy array using the ‘convert\_tensor\_to\_image’ function.
* **Determine If the Image is Grayscale or Color**
* Grayscale Conversion: If the image has 3 channels, it is converted to grayscale by averaging across the color channels.
* **Show the Grid**



[4] **กำหนด Path ไปยังโฟลเดอร์ข้อมูลภาพ:**

data\_dir กำหนด Path ไปยังโฟลเดอร์ images/img\_align\_celeba/ ซึ่งเป็นโฟลเดอร์ที่เก็บภาพ

**สร้างรายการ Path ของภาพ:**

* **image\_paths:** สร้าง list ที่เก็บ Path ของทุกไฟล์ภาพในโฟลเดอร์ data\_dir
* **os.listdir(data\_dir):** สร้าง list ชื่อไฟล์ทั้งหมดในโฟลเดอร์
* **os.path.join(data\_dir, img):** รวม Path ของโฟลเดอร์กับชื่อไฟล์แต่ละอัน

**แสดงจำนวนภาพทั้งหมด:**

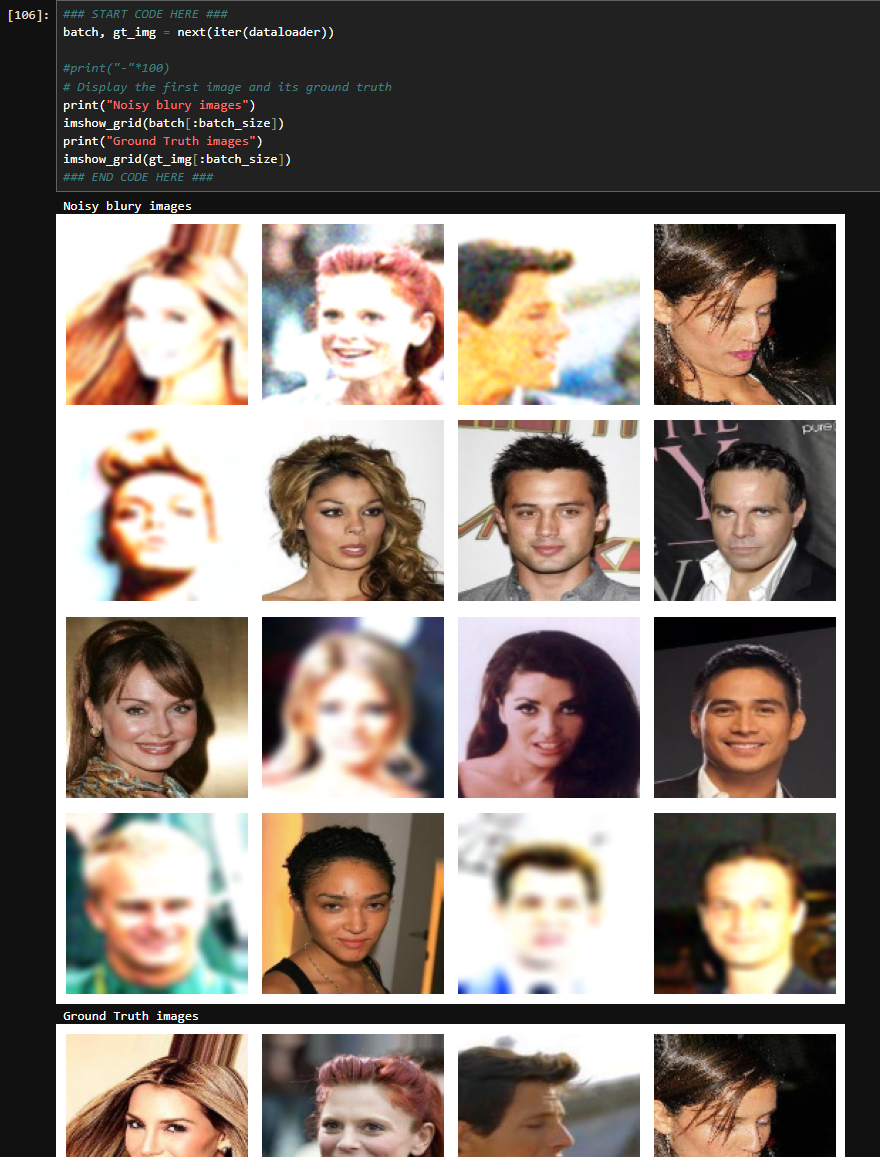
* print("Size:", len(image\_paths)) แสดงจำนวนภาพทั้งหมดในรายการ image\_paths ซึ่งจะเห็นว่ารูปภาพมีทั้งหมด 13,882 รูปภาพ

**สร้าง Dataset:**

* dataset = CustomImageDataset(image\_paths, gauss\_noise=True, gauss\_blur=True) สร้างวัตถุ CustomImageDataset โดยใช้ list: image\_paths และตั้งค่าให้ใช้งานการเพิ่มสัญญาณรบกวนแบบ Gaussian (gauss\_noise) และเบลอภาพแบบ Gaussian (gauss\_blur)เป็น True

**สร้าง DataLoader:**

* dataloader = DataLoader(dataset, batch\_size=16) สร้างวัตถุ DataLoader เพื่อแบ่งข้อมูลใน dataset ออกเป็น Batch ขนาด 16 (wrapped in a DataLoader, which allows for easy batching, shuffling, and parallel loading of the data during model training or evaluation.)



[5] **ดึง Batch แรกจาก DataLoader:**

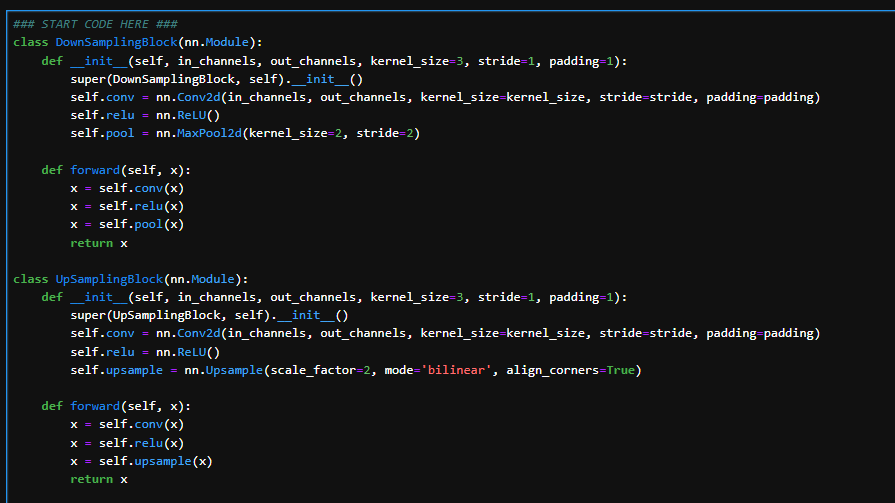
* batch, gt\_img = next(iter(dataloader)) ดึง Batch แรกจากวัตถุ dataloader ซึ่งประกอบด้วยภาพต้นฉบับ (gt\_img) และภาพที่ผ่านการเพิ่มสัญญาณรบกวนและเบลอภาพ (batch)

**แสดงภาพที่ผ่านการเพิ่มสัญญาณรบกวนและเบลอภาพ:**

* print("Noisy blury images") พิมพ์ข้อความ "Noisy blury images"
* imshow\_grid(batch[:batch\_size]) แสดงภาพแรกใน Batch โดยใช้ฟังก์ชัน imshow\_grid

**แสดงภาพต้นฉบับ:**

* print("Ground Truth images") พิมพ์ข้อความ "Ground Truth images"
* imshow\_grid(gt\_img[:batch\_size]) แสดงภาพต้นฉบับแรกใน Batch โดยใช้ฟังก์ชัน imshow\_grid



[6] **class DownSamplingBlock**

The DownSamplingBlock reduces the spatial dimensions of the input while increasing the depth (number of channels). This helps in capturing more complex features in the image.

**องค์ประกอบ:**

* **nn.Conv2d:** ชั้น Convolutional layer ใช้สำหรับการ Convolution โดยมีขนาดเคอร์เนล kernel\_size และ stride stride
* **nn.ReLU:** ชั้น Activation function ช่วยในการเพิ่มความไม่เชิงเส้นให้กับผลลัพธ์
* **nn.MaxPool2d:** ชั้น Max pooling ใช้สำหรับลดขนาดของภาพหรือฟีเจอร์แมปโดยเลือกค่าสูงสุดในพื้นที่ย่อย (region)

**ฟังก์ชัน forward:**

* รับข้อมูลเข้า x
* ส่งข้อมูลเข้าชั้น Convolutional layer
* ใช้ Activation function ReLU
* ส่งข้อมูลเข้าชั้น Max pooling เพื่อลดขนาดภาพ
* ส่งกลับผลลัพธ์

**class UpSamplingBlock**

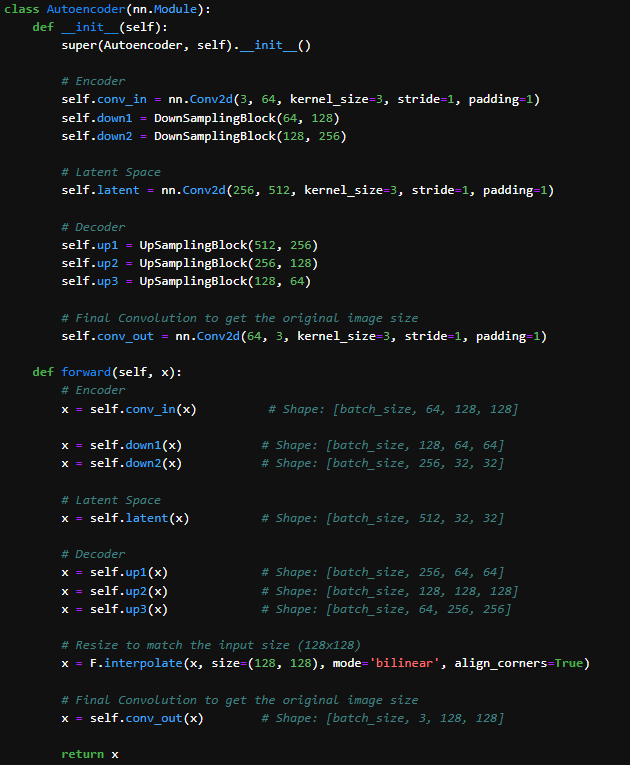
The UpSamplingBlock increases the spatial dimensions, typically to reconstruct the image back to its original size after it has been reduced by downsampling.

**องค์ประกอบ:**

* **nn.Conv2d:** ชั้น Convolutional layer ใช้สำหรับการ Convolution โดยมีขนาดเคอร์เนล kernel\_size และ stride stride
* **nn.ReLU:** ชั้น Activation function ช่วยในการเพิ่มความไม่เชิงเส้นให้กับผลลัพธ์
* **nn.Upsample:** ชั้น Upsampling ใช้สำหรับเพิ่มขนาดของภาพหรือฟีเจอร์แมปโดยใช้วิธี bilinear interpolation

**ฟังก์ชัน forward:**

* รับข้อมูลเข้า x
* ส่งข้อมูลเข้าชั้น Convolutional layer
* ใช้ Activation function ReLU
* ส่งข้อมูลเข้าชั้น Upsampling เพื่อเพิ่มขนาดภาพ
* ส่งกลับผลลัพธ์



[6] **class Autoencoder**

* reduce input image size through convolution and downsampling, capturing essential features in a compressed latent representation.
* then reconstructs the image by upsampling and applying a final convolution to match the original dimensions and color channels.

**หลักการ:**

1. **Encoder:**
   1. **self.conv\_in:**

* **ขนาดของภาพเริ่มต้น:** [batch\_size, 3, 128, 128]
* **Output:** [batch\_size, 64, 128, 128]
* **คำอธิบาย:** ใช้ convolutional layer เพื่อเพิ่มจำนวนช่องสัญญาณ (จาก 3 เป็น 64) แต่ขนาดของภาพไม่เปลี่ยนแปลง (ยังคง 128x128)
  1. **self.down1:**
* **Input:** [batch\_size, 64, 128, 128]
* **Output:** [batch\_size, 128, 64, 64]
* **คำอธิบาย:** ใช้ down-sampling block ที่ลดขนาดของภาพลงครึ่งหนึ่ง (128x128 → 64x64) และเพิ่มจำนวนช่องสัญญาณ (จาก 64 เป็น 128)

**1.3 self.down2:**

* **Input:** [batch\_size, 128, 64, 64]
* **Output:** [batch\_size, 256, 32, 32]
* **คำอธิบาย:** ใช้ down-sampling block อีกครั้งเพื่อลดขนาดของภาพลงครึ่งหนึ่ง (64x64 → 32x32) และเพิ่มจำนวนช่องสัญญาณ (จาก 128 เป็น 256)

1. **Latent Space:**

**2.1 self.latent:**

* **Input:** [batch\_size, 256, 32, 32]
* **Output:** [batch\_size, 512, 32, 32]
* **คำอธิบาย:** ใช้ convolutional layer ที่เพิ่มจำนวนช่องสัญญาณ (จาก 256 เป็น 512) แต่ขนาดของภาพยังคงที่ (32x32)

1. **Decoder:**

**3.1 self.up1:**

* **Input:** [batch\_size, 512, 32, 32]
* **Output:** [batch\_size, 256, 64, 64]
* **คำอธิบาย:** ใช้ up-sampling block เพื่อเพิ่มขนาดของภาพกลับ (32x32 → 64x64) และลดจำนวนช่องสัญญาณ (จาก 512 เป็น 256)

**3.2 self.up2:**

* **Input:** [batch\_size, 256, 64, 64]
* **Output:** [batch\_size, 128, 128, 128]
* **คำอธิบาย:** ใช้ up-sampling block อีกครั้งเพื่อเพิ่มขนาดของภาพ (64x64 → 128x128) และลดจำนวนช่องสัญญาณ (จาก 256 เป็น 128)

**3.3 self.up3:**

* **Input:** [batch\_size, 128, 128, 128]
* **Output:** [batch\_size, 64, 128, 128]
* **คำอธิบาย:** ใช้ up-sampling block เพื่อเพิ่มขนาดของภาพ (128x128 → 128x128) แต่ลดจำนวนช่องสัญญาณ (จาก 128 เป็น 64)

**3.4 F.interpolate:**

* **Input:** [batch\_size, 64, 128, 128]
* **Output:** [batch\_size, 64, 128, 128]
* **คำอธิบาย:** ใช้ interpolation (การปรับขนาด) เพื่อให้ขนาดของภาพตรงตามที่ต้องการ (128x128)

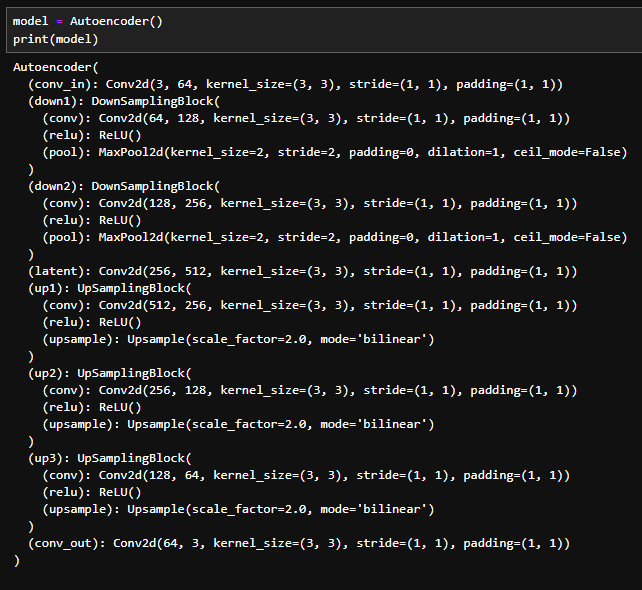
1. **Final Convolution:**

**4.1 self.conv\_out:**

* **Input:** [batch\_size, 64, 128, 128]
* **Output:** [batch\_size, 3, 128, 128]
* **คำอธิบาย:** ใช้ convolutional layer สุดท้ายเพื่อลดจำนวนช่องสัญญาณลงเหลือ 3 ช่อง (สำหรับภาพ RGB) ขนาดของภาพยังคงที่ (128x128)

**การทำงานของ forward Method:**

* ผ่านเลเยอร์ conv\_in เพื่อแปลงภาพเริ่มต้น
* ใช้ down1 และ down2 เพื่อลดขนาดของภาพและเพิ่มช่องสัญญาณ
* ผ่านเลเยอร์ latent เพื่อให้ได้ latent representation
* ใช้ up1, up2, และ up3 เพื่อเพิ่มขนาดของภาพกลับ
* ใช้ F.interpolate ปรับขนาดให้เป็นขนาดสุดท้ายที่ต้องการ
* ใช้ conv\_out เพื่อให้ได้ภาพสุดท้ายที่มีช่องสัญญาณ 3 ช่อง (RGB)



[7] prints out a detailed summary of the autoencoder model architecture

**Train Autoencoder**



[8] **Train โมเดล Autoencoder โดยแบ่งเป็น 3 ส่วนหลัก:**

**1. การเตรียมการ Train:**

* กำหนดอุปกรณ์ที่ใช้ Train (CPU หรือ GPU) โดยใช้ device
* ย้ายโมเดลไปยังอุปกรณ์ที่เลือก (model.to(device))
* วนลูปผ่านจำนวน Epoch ที่กำหนด (epochs) ซึ่งในที่นี้ใช้ epochs = 1 รอบนั้นเอง

**2. Train Loop สำหรับแต่ละ Epoch:**

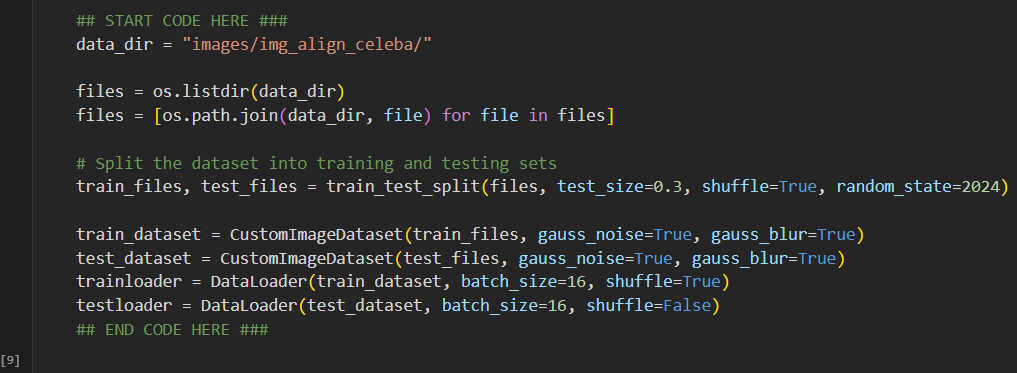
* ตั้งค่าโมเดลให้เป็นโหมด Train (model.train())
* สร้าง Progress bar (tqdm) สำหรับแสดงความคืบหน้าของการ Train
* วนลูปผ่าน Batch ของข้อมูล Train (train\_loader)
  + ย้ายข้อมูลภาพต้นฉบับ (images) และข้อมูลภาพเป้าหมาย (gt) ไปยังอุปกรณ์ที่เลือก
  + ล้างค่า Gradient ของ Optimizer (opt.zero\_grad()) (Clears previous gradients to prevent accumulation)
  + การคำนวณ output: output = model(images) ใช้โมเดลเพื่อคำนวณผลลัพธ์
  + การตรวจสอบขนาด: ตรวจสอบว่า output มีขนาดตรงกับ ground truth หรือไม่
  + การคำนวณ loss: คำนวณค่า loss ด้วย loss\_fn และทำการ backpropagation เพื่อคำนวณ gradient
    - loss = loss\_fn(output, gt)
    - Computes the loss using the specified loss function.
    - loss.backward()
    - Computes the gradient of the loss with respect to the model's parameters.
  + การปรับค่า weight: opt.step() อัพเดตพารามิเตอร์ของโมเดล
  + การบันทึกค่า loss: เก็บค่า loss และอัปเดตความก้าวหน้าใน train\_bar
* คำนวณและพิมพ์ค่า average training loss สำหรับ epoch ปัจจุบัน

**3. ประเมินผลลัพธ์ (Evaluation) หลัง Train:**

* ตั้งค่าโมเดลให้เป็นโหมด Evaluate (model.eval()) ในโหมดการทดสอบ
* เริ่มต้นตัวแปร test\_loss เพื่อเก็บผลรวมของ Loss
* สร้าง list ค่า PSNR และ SSIM สำหรับประเมินคุณภาพของภาพ
* สร้าง Progress bar (tqdm) สำหรับแสดงความคืบหน้าของการประเมินผล
* ปิดการคำนวณ Gradient ด้วย torch.no\_grad() เพื่อประหยัดทรัพยากร
* วนลูปผ่าน Batch ของข้อมูล Test (test\_loader)
  + ย้ายข้อมูลภาพต้นฉบับ (images) และข้อมูลภาพเป้าหมาย (gt) ไปยังอุปกรณ์ที่เลือก
  + ส่งข้อมูลภาพต้นฉบับ (images) ผ่านโมเดล Autoencoder (output = model(images))
  + ตรวจสอบขนาดของข้อมูล Output กับข้อมูลเป้าหมาย (gt)
  + คำนวณค่า Loss (loss\_fn(output, gt))
  + เก็บผลรวมของ Loss
  + แปลงข้อมูลภาพ Output และ เป้าหมาย จาก Tensor เป็น NumPy array
  + คำนวณค่า PSNR และ SSIM สำหรับแต่ละภาพใน Batch
  + เก็บค่า PSNR และ SSIM ไว้ในรายการ
  + อัพเดท Progress bar ด้วยค่า Loss, PSNR เฉลี่ย และ SSIM เฉลี่ย

**สรุปผลลัพธ์:**

* แสดงค่า Loss เฉลี่ยของ Train และ Test
* แสดงค่า PSNR และ SSIM เฉลี่ยของ Test
* บันทึกโมเดล (ถ้ามีการกำหนด checkpoint\_path)



[9] แบ่งชุดข้อมูลภาพออกเป็นสองส่วน: ชุดข้อมูลสำหรับ Train (training set) และชุดข้อมูลสำหรับ Test (testing set)

**กำหนด Path ไปยังโฟลเดอร์ข้อมูลภาพ:**

data\_dir กำหนด Path ไปยังโฟลเดอร์ images/img\_align\_celeba/ ซึ่งเป็นโฟลเดอร์ที่เก็บภาพ

**สร้าง list Path ของภาพ:**

files สร้างรายการที่เก็บ Path ของทุกไฟล์ภาพในโฟลเดอร์ data\_dir

**แบ่งชุดข้อมูลออกเป็น Train และ Test:**

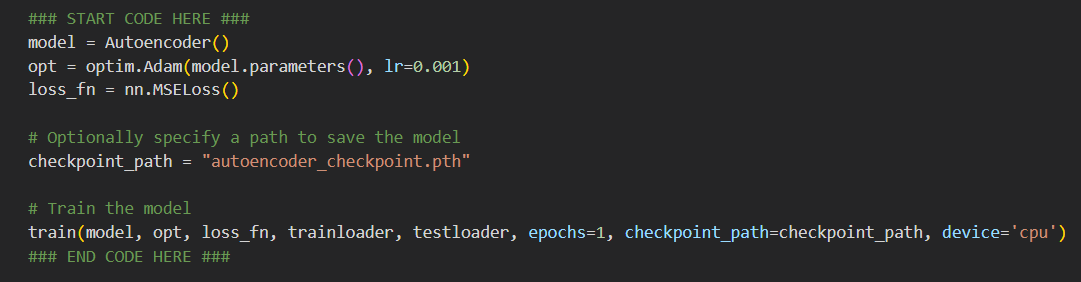
train\_test\_split(files, test\_size=0.3, shuffle=True, random\_state=2024): แบ่งรายการ files ออกเป็นสองส่วน โดยชุด Test มีขนาด 30% ของชุดข้อมูลทั้งหมด และมีการสุ่มการแบ่ง (shuffle) โดยใช้ค่า Random State เป็น 2024 เพื่อให้การแบ่งสามารถทำซ้ำได้

**สร้าง Dataset สำหรับ Train และ Test:**

* **train\_dataset = CustomImageDataset(train\_files, gauss\_noise=True, gauss\_blur=True)** สร้าง Dataset สำหรับ Train โดยใช้รายการ Path ของภาพในชุด Train และเปิดใช้งานการเพิ่มสัญญาณรบกวนและเบลอภาพแบบ Gaussian
* **test\_dataset = CustomImageDataset(test\_files, gauss\_noise=True, gauss\_blur=True)** สร้าง Dataset สำหรับ Test โดยใช้รายการ Path ของภาพในชุด Test และเปิดใช้งานการเพิ่มสัญญาณรบกวนและเบลอภาพแบบ Gaussian

**สร้าง DataLoader สำหรับ Train และ Test:**

* **trainloader = DataLoader(train\_dataset, batch\_size=16, shuffle=True)** สร้าง DataLoader สำหรับ Train โดยใช้ Dataset สำหรับ Train และแบ่งข้อมูลเป็น Batch ขนาด 16 และสุ่มการเรียงลำดับของ Batch
* **testloader = DataLoader(test\_dataset, batch\_size=16, shuffle=False)** สร้าง DataLoader สำหรับ Test โดยใช้ Dataset สำหรับ Test และแบ่งข้อมูลเป็น Batch ขนาด 16 โดยไม่สุ่มการเรียงลำดับของ Batch



[11] **Model Initialization**

**สร้างโมเดล Autoencoder:**

* **model = Autoencoder():** สร้างวัตถุ Autoencoder ซึ่งเป็นโมเดล Autoencoder

**สร้าง Optimizer:**

* **opt = optim.Adam(model.parameters(), lr=0.001):** สร้าง Optimizer แบบ Adam โดยใช้ Learning Rate เป็น 0.001

**สร้าง Loss Function:**

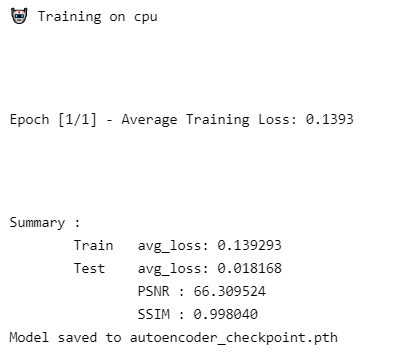
* **loss\_fn = nn.MSELoss()**: สร้าง Loss Function แบบ Mean Squared Error (MSE)

**กำหนด Path สำหรับบันทึกโมเดล (Optional):**

* **checkpoint\_path = "autoencoder\_checkpoint.pth"**: กำหนด Path สำหรับบันทึกโมเดล (ถ้าต้องการ)

**Train โมเดล:**

* **train(model, opt, loss\_fn, trainloader, testloader, epochs=1, checkpoint\_path=checkpoint\_path, device='cpu')** เรียกใช้ฟังก์ชัน train เพื่อ Train โมเดลโดยใช้โมเดลที่สร้าง, Optimizer, Loss Function, DataLoader สำหรับ Train และ Test, จำนวน Epoch เป็น 1, Path สำหรับบันทึกโมเดล (ถ้ามี) และใช้ CPU สำหรับการ Train

**ผลลัพธ์ที่ได้จากการทดลอง:**

* **ค่าเฉลี่ยของการสูญเสียในช่วงการฝึก (Average Training Loss): 0.1393**

ค่าเฉลี่ยของการสูญเสีย (loss) ระหว่างการฝึกฝนโมเดลในช่วงนี้คือ 0.1393 ซึ่งมีค่าต่ำ หมายถึงว่าโมเดลมีการเรียนรู้ได้ดีในชุดข้อมูลฝึกอบรม

* **ค่าเฉลี่ยของการสูญเสียในการทดสอบ (Average Test Loss): 0.0182**

ค่าเฉลี่ยของการสูญเสียในชุดข้อมูลทดสอบมีค่าน้อยมากที่ 0.0182 ซึ่งมีค่าต่ำ บ่งบอกว่าโมเดลสามารถทำงานได้ดีในชุดข้อมูลที่ไม่ได้ใช้ในการฝึกอบรม

* **PSNR (Peak Signal-to-Noise Ratio): 66.31 dB**

PSNR เป็นตัวชี้วัดคุณภาพของภาพ โดยค่า PSNR ที่สูงบ่งบอกถึงคุณภาพของภาพที่ดี ค่า PSNR ที่ 66.31 dB หมายถึงโมเดลสามารถสร้างภาพที่มีคุณภาพสูงมากโดยมีการบิดเบือนน้อย

* **SSIM (Structural Similarity Index): 0.9980**

SSIM ใช้เพื่อวัดความคล้ายคลึงกันเชิงโครงสร้างของภาพ ค่า SSIM ที่ 0.9980 เป็นค่าที่สูงมาก ซึ่งหมายความว่าโมเดลสามารถสร้างภาพที่มีความคล้ายคลึงกับภาพต้นฉบับได้ดีมาก



[12] ฟังก์ชัน **display\_sample\_images** นี้มีวัตถุประสงค์ในการแสดงตัวอย่างภาพที่ผ่านการ Train ในโมเดล Autoencoder

* **ตั้งค่าโมเดลเป็นโหมด Evaluate:**

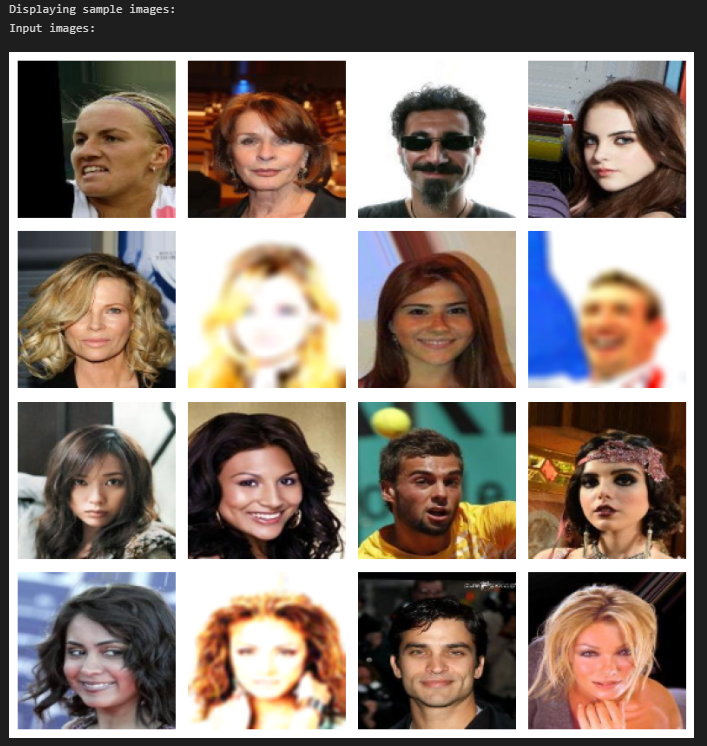
**model.eval():** ตั้งค่าโมเดลให้เป็นโหมด Evaluate เพื่อปิดการคำนวณ Gradient

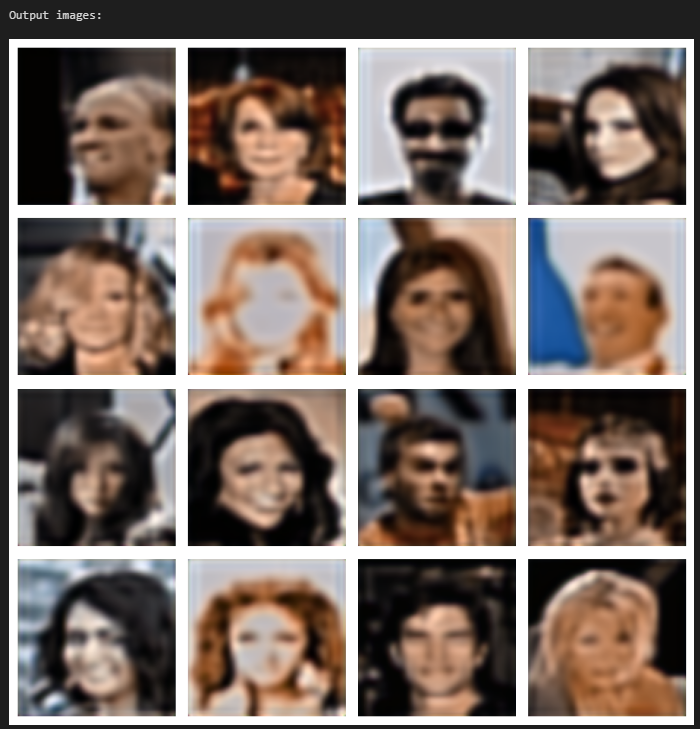
* **ปิดการคำนวณ Gradient:**

**with torch.no\_grad():** บล็อกโค้ดนี้จะปิดการคำนวณ Gradient เพื่อประหยัดทรัพยากร

* **วนลูปผ่าน Batch ของข้อมูล Test:**
  + วนลูปผ่าน Batch ของข้อมูล Test ที่อยู่ใน dataloader
  + ย้ายข้อมูลภาพต้นฉบับ (images) และข้อมูลภาพเป้าหมาย (gt) ไปยังอุปกรณ์ที่เลือก (CPU ในกรณีนี้)
* **ส่งข้อมูลภาพต้นเข้าผ่านโมเดลเพื่อเอา Output:**
* **outputs = model(images)**: ส่งข้อมูลภาพต้นฉบับ (images) ผ่านโมเดล Autoencoder เพื่อสร้างข้อมูล Output
* **แปลงข้อมูลเป็น NumPy array:**
* **images = images.cpu() , gt = gt.cpu() , outputs = outputs.cpu():** แปลงข้อมูลภาพ Tensor จากทั้งภาพต้นฉบับ, ภาพ Output และภาพเป้าหมาย เป็น NumPy array เพื่อเตรียมสำหรับการแสดงผล
* **เตรียมตัวอย่างภาพสำหรับแสดงผล:**
* **num\_samples = min(len(images), 16)**: กำหนดจำนวนภาพตัวอย่างสูงสุดที่จะแสดง โดยจำกัดที่ 16 ภาพ
* เลือกตัวอย่างภาพต้นฉบับ, Output และภาพเป้าหมาย จำนวน num\_samples ภาพ
* **แสดงตัวอย่างภาพ:**
* **print("Displaying sample images:"):** พิมพ์ข้อความแสดงหัวข้อ
* **print("Input images:")**: พิมพ์ข้อความระบุภาพต้นฉบับ
* **imshow\_grid(sample\_images):** แสดงภาพต้นฉบับตัวอย่าง (ฟังก์ชัน imshow\_grid อาจเป็นฟังก์ชันที่คุณมีอยู่แล้วสำหรับการแสดง Grid ของภาพ)
* ทำขั้นตอนเดียวกันสำหรับภาพ Output และภาพเป้าหมาย
* break หยุดการวนลูปหลังแสดงผล Batch แรก (เพื่อประหยัดเวลา)
* **เรียกใช้ฟังก์ชัน:**
* **display\_sample\_images(model, testloader, device='cpu'):** เรียกใช้ฟังก์ชันเพื่อแสดงตัวอย่างภาพ โดยใช้โมเดลที่ Train แล้ว, DataLoader สำหรับข้อมูล Test และกำหนดอุปกรณ์เป็น CPU

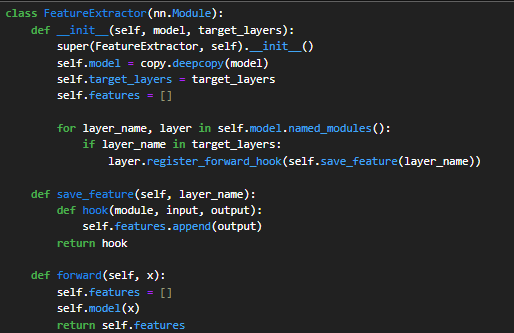
**ผลลัพธ์การทดลอง:**

****

****

****

**Explore feature map**



[14] **class FeatureExtractor**

คลาส FeatureExtractor ออกแบบมาเพื่อดึงข้อมูล Feature จากชั้นที่กำหนดในโครงข่ายประสาทเทียม (CNN) โดยสามารถใช้เพื่อวิเคราะห์การทำงานของโครงข่ายประสาทเทียมหรือเพื่อสร้างข้อมูลสำหรับงานอื่น ๆ (class allows you to specify layers from which to capture outputs. During a forward pass, it captures these outputs using forward hooks and returns them.)

**องค์ประกอบ:**

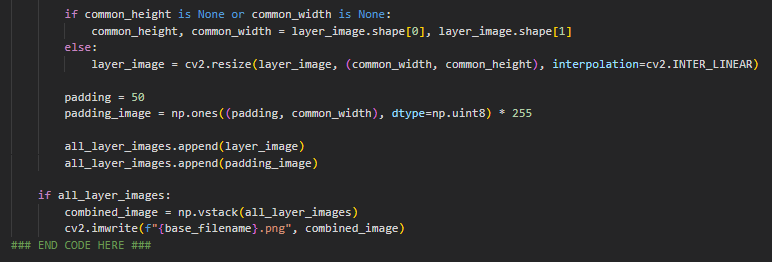
* **\_\_init\_\_:** ฟังก์ชันสร้างวัตถุ
* model: โมเดล CNN ที่ต้องการดึงข้อมูล Feature
* target\_layers: รายการชื่อชั้นในโมเดลที่ต้องการดึงข้อมูล Feature
* **save\_feature:** ฟังก์ชันสร้าง Hook สำหรับบันทึกข้อมูล Feature
* layer\_name: ชื่อของชั้นที่ต้องการบันทึกข้อมูล Feature
* ฟังก์ชันภายใน hook จะถูกเรียกทุกครั้งที่ชั้น module ถูกเรียกใช้งาน
* hook จะบันทึกข้อมูล Output ของชั้น module ไว้ในรายการ self.features
* **forward:** ฟังก์ชัน forward ของคลาส
* ทำการเคลียร์รายการ self.features
* ส่งข้อมูล x ผ่านโมเดล CNN

ส่งกลับรายการ self.features ซึ่งประกอบด้วยข้อมูล Feature จากชั้นที่กำหนด

**วิธีการทำงาน:**

* เมื่อสร้างวัตถุ FeatureExtractor จะทำการคัดลอกโมเดล CNN ที่กำหนด (copy.deepcopy(model)) เพื่อไม่ให้แก้ไขโมเดลต้นฉบับ
* สำหรับแต่ละชั้นที่กำหนดใน target\_layers จะลงทะเบียน Hook (register\_forward\_hook) เพื่อบันทึกข้อมูล Feature
* เมื่อเรียกใช้งาน forward จะส่งข้อมูล x ผ่านโมเดล CNN และ Hook ที่ลงทะเบียนจะบันทึกข้อมูล Feature จากชั้นที่กำหนดไว้ในรายการ self.features
* สุดท้ายจะส่งกลับรายการ self.features ซึ่งประกอบด้วยข้อมูล Feature จากชั้นที่กำหนด



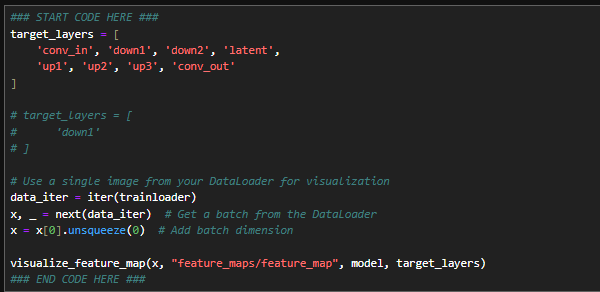


[15] **def visualize\_feature\_map**

ฟังก์ชัน **visualize\_feature\_map** ใช้ในการแสดงภาพ Feature Map จากชั้นที่กำหนดในโมเดล CNN (ช่วยในการวิเคราะห์การทำงานของโมเดล CNN โดยแสดงภาพ Feature Map จากชั้นที่กำหนด ซึ่งจะช่วยให้เข้าใจได้ว่าแต่ละชั้นเรียนรู้ลักษณะอะไรของข้อมูล)

**รายละเอียด:**

* **Input:**
* x: ข้อมูล Input ของโมเดล (Tensor)
* base\_filename: ชื่อไฟล์พื้นฐานสำหรับการบันทึกภาพ
* model: โมเดล CNN
* target\_layers: รายการชื่อชั้นในโมเดลที่ต้องการแสดงภาพ Feature Map
* **สร้าง Feature Extractor:**
* **feature\_extractor = FeatureExtractor(model, target\_layers):** สร้างวัตถุ FeatureExtractor เพื่อดึงข้อมูล Feature Map จากชั้นที่กำหนดในโมเดล
* **ดึงข้อมูล Feature Map:**
* **feature\_maps = feature\_extractor(x):** เรียกใช้ forward ของ FeatureExtractor เพื่อดึงข้อมูล Feature Map จากชั้นที่กำหนด
* **สร้างโฟลเดอร์สำหรับบันทึกภาพ (ถ้ายังไม่มี):**
* ตรวจสอบว่าโฟลเดอร์สำหรับบันทึกภาพมีอยู่แล้วหรือไม่ (โดยใช้ os.path.exists) ถ้ายังไม่มีจะสร้างโฟลเดอร์ด้วย os.makedirs
* **ตรวจสอบข้อมูล Feature Map:**
* if not feature\_maps: ตรวจสอบว่ามีข้อมูล Feature Map ที่ดึงมาหรือไม่ ถ้าไม่มีจะแสดงข้อความและหยุดการทำงาน
* **เตรียมการสำหรับการแสดงภาพ:**
* เริ่มต้นตัวแปร common\_height และ common\_width เพื่อกำหนดขนาดภาพแบบรวม
* สร้างรายการ all\_layer\_images เพื่อเก็บภาพ Feature Map แต่ละชั้น
* **วนลูปแสดงภาพ Feature Map แต่ละชั้น:**
* สำหรับแต่ละ Feature Map ในรายการ feature\_maps
  + num\_feature\_maps จำนวน Feature Map ในชั้นนั้น
  + grid\_size คำนวณขนาด Grid สำหรับการแสดงภาพ Feature Map ย่อย ๆ
  + สร้าง Figure และ Axes สำหรับการ Plot ภาพ Feature Map ย่อย ๆ ด้วย plt.subplots
  + วนลูปแสดงภาพ Feature Map ย่อย ๆ แต่ละภาพใน Grid
    - แปลงข้อมูล Feature Map เป็น NumPy array และ Normalize ค่า
    - แสดงภาพ Feature Map ย่อย ๆ ด้วย imshow
    - ปิดการแสดงแกนของ Plot (axis('off'))
  + ปิดการแสดงแกนของ Plot ที่เหลือ (axis('off')) ในกรณีที่มี Feature Map น้อยกว่าขนาด Grid
  + ตั้งชื่อ Title ให้กับ Figure ตามชื่อชั้น
  + ปรับแต่ง Layout ของ Plot
  + บันทึกภาพ Feature Map ของชั้นนั้นด้วย plt.savefig
  + แสดงภาพ Feature Map ของชั้นนั้นด้วย plt.show
  + ปิด Figure ที่สร้างด้วย plt.close
* **เตรียมภาพสำหรับการรวมภาพ:**
* ตรวจสอบขนาดภาพ Feature Map แต่ละชั้น (common\_height และ common\_width)
  + ถ้ายังไม่มีการกำหนด จะใช้ขนาดของภาพแรกเป็นพื้นฐาน
  + ถ้ามีการกำหนดแล้ว จะ Resize ภาพ Feature Map ชั้นอื่น ๆ ให้มีขนาดเดียวกัน
* สร้าง Padding ภาพสีขาวสำหรับคั่นระหว่างภาพ Feature Map
* **สร้างภาพรวม:**
* all\_layer\_images จะประกอบด้วยภาพ Feature Map แต่ละชั้นสลับกับ Padding ภาพ
* สร้างภาพรวม (combined\_image) โดยการเรียงต่อกันตามแนวดิ่ง (np.vstack)
* บันทึกภาพรวมด้วย cv2.imwrite



[16] **กำหนดรายการชั้นที่ต้องการแสดงภาพ Feature Map:**

target\_layers กำหนดรายการชื่อชั้นในโมเดล Autoencoder ที่ต้องการแสดงภาพ Feature Map

**เลือกภาพตัวอย่าง:**

* data\_iter = iter(trainloader) สร้าง Iterator สำหรับวนลูปผ่านข้อมูล Train
* x, \_ = next(data\_iter) ดึง Batch แรกจาก DataLoader
* x = x[0].unsqueeze(0) เลือกภาพแรกใน Batch และเพิ่มมิติ Batch เพื่อให้เข้ากับรูปแบบที่ฟังก์ชัน visualize\_feature\_map ต้องการ

**เรียกใช้ฟังก์ชัน visualize\_feature\_map:**

**visualize\_feature\_map(x, "feature\_maps/feature\_map", model, target\_layers):** เรียกใช้ฟังก์ชันเพื่อแสดงภาพ Feature Map โดยใช้ภาพตัวอย่าง x, ชื่อไฟล์พื้นฐานสำหรับบันทึกภาพเป็น "feature\_maps/feature\_map", โมเดล Autoencoder และรายการชั้นที่กำหนด

**ผลลัพธ์การทดลอง:**



