Lab 5.2

Denoise & Deblur image with Autoencoder

Data Preparation

[102]:
```python
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import Dataset
import torchvision.transforms as transforms
from torchvision.datasets import ImageFolder
from torch.utils.data import DataLoader,Subset
from torch.utils.tensorboard import SummaryWriter
import copy

import numpy as np
import cv2
import os
import random
from skimage.util import random_noise
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from skimage.metrics import peak_signal_noise_ratio as psnr
from skimage.metrics import structural_similarity as ssim
from tqdm import tqdm
```

[1] Import Library ที่จำเป็นที่ต้องใช้ใน Lab 5.1 นี้

[103]:
```python
### START CODE HERE ###
class CustomImageDataset(torch.utils.data.Dataset):
    def __init__(self, image_paths, resize=128, crop_size=128, gauss_noise=False, gauss_blur=False, p=0.5):
        self.image_paths = image_paths
        self.resize = resize
        self.crop_size = crop_size
        self.gauss_noise = gauss_noise
        self.gauss_blur = gauss_blur
        self.p = p

    def __len__(self):
        return len(self.image_paths)

    def clip_to_uint8(self, image):
        """Clips image values to the range of 0-255 (uint8 format)"""
        return np.clip(image * 255.0, 0, 255).astype(np.uint8)  # Rescale and clip

    def __getitem__(self, idx):
        image = cv2.imread(self.image_paths[idx])
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

        # Resize and crop
        image = cv2.resize(image, (self.resize, self.resize))

        h, w, _ = image.shape
        start_x = (w - self.crop_size) // 2
        start_y = (h - self.crop_size) // 2
        image = image[start_y:start_y+self.crop_size, start_x:start_x+self.crop_size]

        # Ground truth
        gt_image = image.copy()

        # Augmentations
        if random.random() < self.p:
            if self.gauss_noise:
                # Add Gaussian noise
                noise_mean = random.uniform(-50, 50)
                var = 0.01
                noise = random_noise(image.astype(np.float32) / 255.0, mode='gaussian', var=var)  # var controls noise level
                image = image.astype(np.float32) / 255.0 + noise
                image = np.clip(image * 255.0, 0, 255).astype(np.uint8)  # Clip after adding noise
                #print(f"noise_mean: {noise_mean:.3f}", end=",\t")

            if self.gauss_blur:
                kernel_size = random.randint(3, 11)
                kernel_size = kernel_size if kernel_size % 2 == 1 else kernel_size + 1
                sigmaX = kernel_size / 3
                image = cv2.GaussianBlur(image, (kernel_size, kernel_size), sigmaX)
                #print("kernel_size", kernel_size)

        # Preprocess for PyTorch
        image = image.astype(np.float32) / 255.0
        gt_image = gt_image.astype(np.float32) / 255.0
        image = torch.from_numpy(image).permute(2, 0, 1)
        gt_image = torch.from_numpy(gt_image).permute(2, 0, 1)

        return image, gt_image
### END CODE HERE ###
```

[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.
3. 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
4. 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)
5. 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.

6. Return

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

```python
### START CODE HERE ###
def convert_tensor_to_image(tensor, normalize=True):
    """
    Convert a tensor to a numpy image array.

    Parameters:
    - tensor (torch.Tensor): Input tensor of shape (C, H, W) or (N, C, H, W).
    - normalize (bool): Whether to normalize the image data to the range [0, 1].

    Returns:
    - np.ndarray: Image array with shape (H, W, C) for a single image or (N, H, W, C) for a batch.
    """
    if tensor.ndimension() == 4:
        # Batch of images
        tensor = tensor[0]  # Take the first image in the batch

    # Convert to numpy
    image = tensor.cpu().numpy().transpose(1, 2, 0)

    # Normalize to [0, 1] if needed
    if normalize:
        image = (image - np.min(image)) / (np.max(image) - np.min(image))

    return image
```

[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**

```python
def imshow_grid(images, normalize=True, cmap=None):
    """
    Display a grid of images.

    Parameters:
    - images (list of torch.Tensor or np.ndarray): List of images to display.
    - normalize (bool): Whether to normalize images before displaying.
    - cmap (str or None): Colormap for grayscale images (e.g., 'gray'). None for color images.
    """
    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()

    for i in range(num_images):
        # Convert tensor to image
        if isinstance(images[i], torch.Tensor):
            image = convert_tensor_to_image(images[i], normalize)
        else:
            image = images[i]

        # Determine if the image is grayscale or color
        if cmap:  # Apply colormap if specified
            if image.ndim == 3 and image.shape[2] == 3:
                # Convert color image to grayscale using average method
                image_gray = np.mean(image, axis=2)
                axes[i].imshow(image_gray, cmap=cmap)
            else:
                axes[i].imshow(image, cmap=cmap)
        else:
            axes[i].imshow(image)

        axes[i].axis('off')

    # Hide any remaining axes
    for j in range(num_images, len(axes)):
        axes[j].axis('off')

    plt.tight_layout()
    plt.show()
### END CODE HERE ###
```

[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**

```
### START CODE HERE ###
data_dir = "images/img_align_celeba/"
image_paths = [os.path.join(data_dir, img) for img in os.listdir(data_dir)]
print("Size:",len(image_paths))

dataset = CustomImageDataset(image_paths, gauss_noise=True, gauss_blur=True)
batch_size = 16
dataloader = DataLoader(dataset, batch_size=batch_size)
### END CODE HERE ###
```

```
Size: 13882
```

[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:

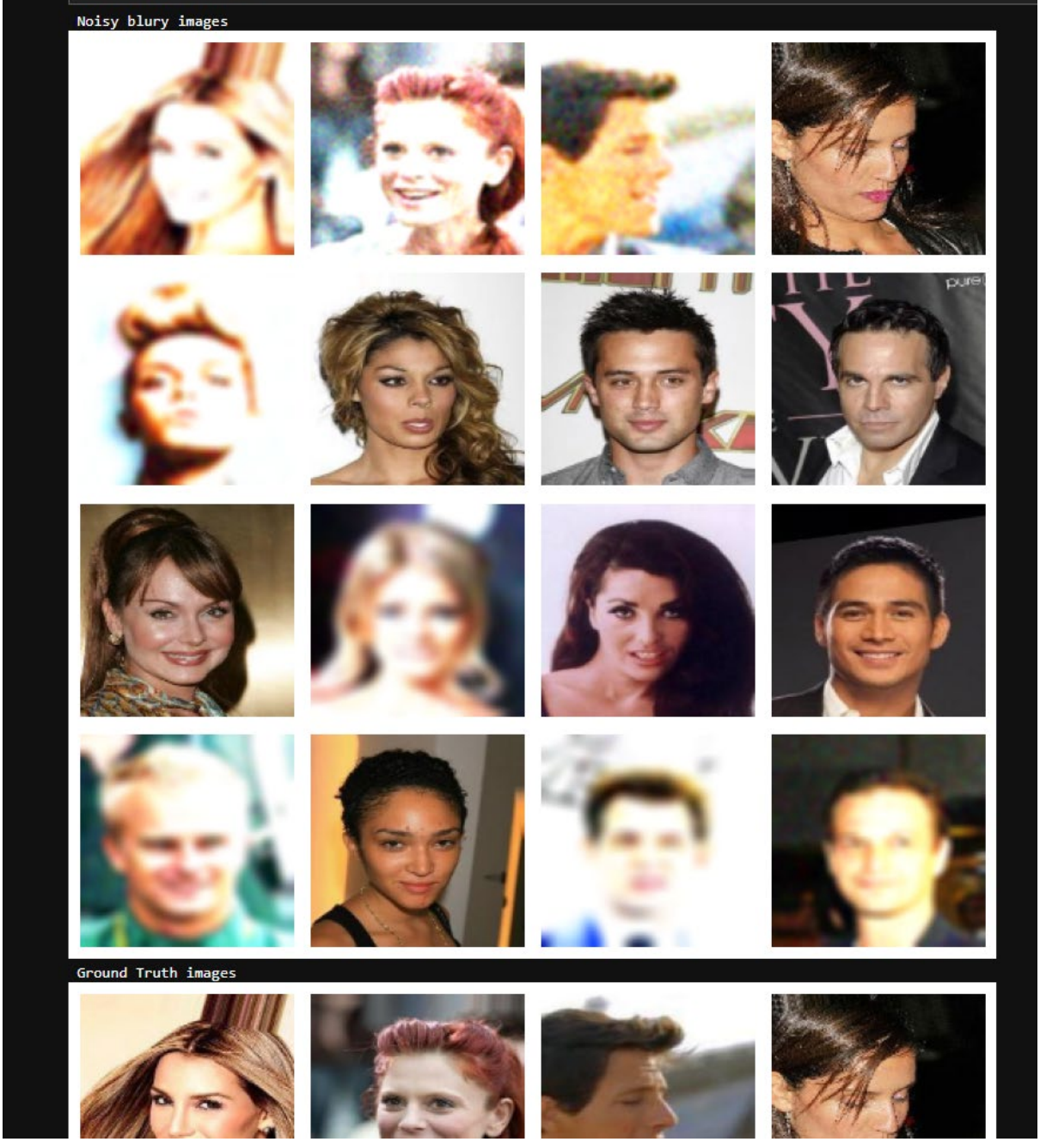
- dataset = CustomImageDataset(image_paths, gauss_noise=True, gauss_blur=True) สร้างวัตถุ CustomImageDataset โดยใช้ list: image_paths และตั้งค่าให้ใช้งานการเพิ่มสัญญาณรบกวนแบบ Gaussian (gauss_noise) และเบลอภาพแบบ Gaussian (gauss_blur) เป็น True

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.)

```
[106]: ### START CODE HERE ###
batch, gt_img = next(iter(dataloader))

#print("-"*100)
# Display the first image and its ground truth
print("Noisy blury images")
imshow_grid(batch[:batch_size])
print("Ground Truth images")
imshow_grid(gt_img[:batch_size])
### END CODE HERE ###
```



[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

```python
### START CODE HERE ###
class DownSamplingBlock(nn.Module):
    def __init__(self, in_channels, out_channels, kernel_size=3, stride=1, padding=1):
        super(DownSamplingBlock, self).__init__()
        self.conv = nn.Conv2d(in_channels, out_channels, kernel_size=kernel_size, stride=stride, padding=padding)
        self.relu = nn.ReLU()
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)

    def forward(self, x):
        x = self.conv(x)
        x = self.relu(x)
        x = self.pool(x)
        return x

class UpSamplingBlock(nn.Module):
    def __init__(self, in_channels, out_channels, kernel_size=3, stride=1, padding=1):
        super(UpSamplingBlock, self).__init__()
        self.conv = nn.Conv2d(in_channels, out_channels, kernel_size=kernel_size, stride=stride, padding=padding)
        self.relu = nn.ReLU()
        self.upsample = nn.Upsample(scale_factor=2, mode='bilinear', align_corners=True)

    def forward(self, x):
        x = self.conv(x)
        x = self.relu(x)
        x = self.upsample(x)
        return x
```

[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

```python
class Autoencoder(nn.Module):
    def __init__(self):
        super(Autoencoder, self).__init__()

        # Encoder
        self.conv_in = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1)
        self.down1 = DownSamplingBlock(64, 128)
        self.down2 = DownSamplingBlock(128, 256)

        # Latent Space
        self.latent = nn.Conv2d(256, 512, kernel_size=3, stride=1, padding=1)

        # Decoder
        self.up1 = UpSamplingBlock(512, 256)
        self.up2 = UpSamplingBlock(256, 128)
        self.up3 = UpSamplingBlock(128, 64)

        # Final Convolution to get the original image size
        self.conv_out = nn.Conv2d(64, 3, kernel_size=3, stride=1, padding=1)

    def forward(self, x):
        # Encoder
        x = self.conv_in(x)          # Shape: [batch_size, 64, 128, 128]

        x = self.down1(x)            # Shape: [batch_size, 128, 64, 64]
        x = self.down2(x)            # Shape: [batch_size, 256, 32, 32]

        # Latent Space
        x = self.latent(x)           # Shape: [batch_size, 512, 32, 32]

        # Decoder
        x = self.up1(x)              # Shape: [batch_size, 256, 64, 64]
        x = self.up2(x)              # Shape: [batch_size, 128, 128, 128]
        x = self.up3(x)              # Shape: [batch_size, 64, 256, 256]

        # Resize to match the input size (128x128)
        x = F.interpolate(x, size=(128, 128), mode='bilinear', align_corners=True)

        # Final Convolution to get the original image size
        x = self.conv_out(x)         # Shape: [batch_size, 3, 128, 128]

        return x
```

[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:**

    1.1 **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)

1.2 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)

2. 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)

3. 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)

4. 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)

```
model = Autoencoder()
print(model)
```

```
Autoencoder(
  (conv_in): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (down1): DownSamplingBlock(
    (conv): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (relu): ReLU()
    (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (down2): DownSamplingBlock(
    (conv): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (relu): ReLU()
    (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (latent): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (up1): UpSamplingBlock(
    (conv): Conv2d(512, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (relu): ReLU()
    (upsample): Upsample(scale_factor=2.0, mode='bilinear')
  )
  (up2): UpSamplingBlock(
    (conv): Conv2d(256, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (relu): ReLU()
    (upsample): Upsample(scale_factor=2.0, mode='bilinear')
  )
  (up3): UpSamplingBlock(
    (conv): Conv2d(128, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (relu): ReLU()
    (upsample): Upsample(scale_factor=2.0, mode='bilinear')
  )
  (conv_out): Conv2d(64, 3, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
)
```

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

Train Autoencoder

```python
def train(model, opt, loss_fn, train_loader, test_loader, epochs=10, checkpoint_path=None, device='cpu'):
    print(f"🖥️ Training on {device}")
    model = model.to(device)

    for epoch in range(epochs):
        model.train()
        train_loss = 0.0
        train_bar = tqdm(train_loader, desc=f'🚀 Training Epoch [{epoch+1}/{epochs}]', unit='batch', leave=False)
        for images, gt in train_bar:
            images, gt = images.to(device), gt.to(device)

            opt.zero_grad()
            output = model(images)

            if output.shape != gt.shape:
                raise ValueError(f"Output shape {output.shape} does not match ground truth shape {gt.shape}")

            loss = loss_fn(output, gt)
            loss.backward()
            opt.step()

            train_loss += loss.item()
            avg_train_loss = train_loss / (train_bar.n + 1)
            train_bar.set_postfix(loss=f"{avg_train_loss:.4f}")

        avg_train_loss = train_loss / len(train_loader)
        print(f"\nEpoch [{epoch+1}/{epochs}] - Average Training Loss: {avg_train_loss:.4f}")

        model.eval()
        test_loss = 0.0
        psnr_values = []
        ssim_values = []
        test_bar = tqdm(test_loader, desc=f'📄 Testing Epoch [{epoch+1}/{epochs}]', unit='batch', leave=False)
        with torch.no_grad():
            for images, gt in test_bar:
                images, gt = images.to(device), gt.to(device)
                output = model(images)

                if output.shape != gt.shape:
                    raise ValueError(f"Output shape {output.shape} does not match ground truth shape {gt.shape}")

                loss = loss_fn(output, gt)
                test_loss += loss.item()

                for i in range(images.size(0)):
                    gt_image = gt[i].cpu().numpy().transpose(1, 2, 0)
                    output_image = output[i].cpu().numpy().transpose(1, 2, 0)

                    data_range=255.0 # Assuming images are normalized in the range [0, 255]

                    psnr_value = psnr(gt_image, output_image, data_range=data_range)
                    ssim_value = ssim(gt_image, output_image, multichannel=True, win_size=3, data_range=data_range)

                    psnr_values.append(psnr_value)
                    ssim_values.append(ssim_value)

                avg_test_loss = test_loss / (test_bar.n + 1)
                test_bar.set_postfix(loss=f"{avg_test_loss:.4f}", psnr=f"{np.mean(psnr_values):.1f}", ssim=f"{np.mean(ssim_values):.3f}")

        avg_test_loss = test_loss / len(test_loader)
        avg_psnr = np.mean(psnr_values)
        avg_ssim = np.mean(ssim_values)

        print(f"\nSummary :")
        print(f"\tTrain\tavg_loss: {avg_train_loss:.6f}")
        print(f"\tTest\tavg_loss: {avg_test_loss:.6f}")
        print(f"\t\tPSNR : {avg_psnr:.6f}")
        print(f"\t\tSSIM : {avg_ssim:.6f}")

        if checkpoint_path:
            torch.save(model.state_dict(), checkpoint_path)
            print(f"Model saved to {checkpoint_path}")
### END CODE HERE ###
```

[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]
## START CODE HERE ###
data_dir = "images/img_align_celeba/"

files = os.listdir(data_dir)
files = [os.path.join(data_dir, file) for file in files]

# Split the dataset into training and testing sets
train_files, test_files = train_test_split(files, test_size=0.3, shuffle=True, random_state=2024)

train_dataset = CustomImageDataset(train_files, gauss_noise=True, gauss_blur=True)
test_dataset = CustomImageDataset(test_files, gauss_noise=True, gauss_blur=True)
trainloader = DataLoader(train_dataset, batch_size=16, shuffle=True)
testloader = DataLoader(test_dataset, batch_size=16, shuffle=False)
## END CODE HERE ###
```

[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

```
### START CODE HERE ###
model = Autoencoder()
opt = optim.Adam(model.parameters(), lr=0.001)
loss_fn = nn.MSELoss()

# Optionally specify a path to save the model
checkpoint_path = "autoencoder_checkpoint.pth"

# Train the model
train(model, opt, loss_fn, trainloader, testloader, epochs=1, checkpoint_path=checkpoint_path, device='cpu')
### END CODE HERE ###
```

[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:

```
Training on cpu



Epoch [1/1] - Average Training Loss: 0.1393



Summary :
        Train   avg_loss: 0.139293
        Test    avg_loss: 0.018168
                PSNR : 66.309524
                SSIM : 0.998040
Model saved to autoencoder_checkpoint.pth
```

- **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

```python
### START CODE HERE ###
# Define a function to display images
def display_sample_images(model, dataloader, device='cpu'):
    model.eval()
    with torch.no_grad():
        for images, gt in dataloader:
            images, gt = images.to(device), gt.to(device)
            outputs = model(images)
            # Convert images and outputs to CPU and numpy arrays
            images = images.cpu()
            gt = gt.cpu()
            outputs = outputs.cpu()

            # Prepare sample images for display
            num_samples = min(len(images), 16)  # Limit to 16 samples for display
            sample_images = [images[i] for i in range(num_samples)]
            sample_gts = [gt[i] for i in range(num_samples)]
            sample_outputs = [outputs[i] for i in range(num_samples)]

            print("Displaying sample images:")
            print("Input images:")
            imshow_grid(sample_images)  # Display input images
            print("Output images:")
            # print(sample_outputs[0].ndim )
            imshow_grid(sample_outputs)
            print("Ground Truth images:")
            imshow_grid(sample_gts)      # Display ground truth images

            break  # Only display the first batch

# Call the function with your model and data loader
display_sample_images(model, testloader, device='cpu')
### END CODE HERE ###
```

[12]

[12] ฟังก์ชัน **display_sample_images** นี้มีวัตถุประสงค์ในการแสดงตัวอย่างภาพที่ผ่านการ Train ในโมเดล Autoencoder

- **ตั้งค่าโมเดลเป็นโหมด Evaluate:**

  model.eval(): ตั้งค่าโมเดลให้เป็นโหมด Evaluate เพื่อปิดการคำนวณ Gradient

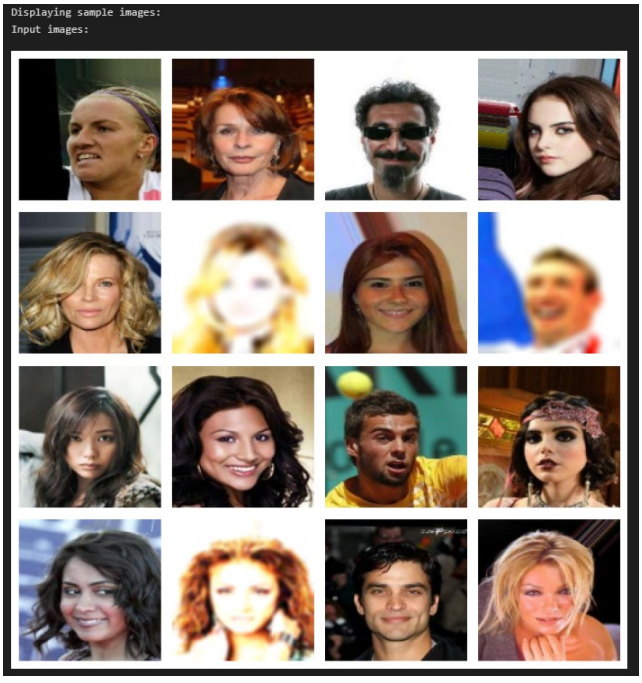
- **ปิดการคำนวณ Gradient:**

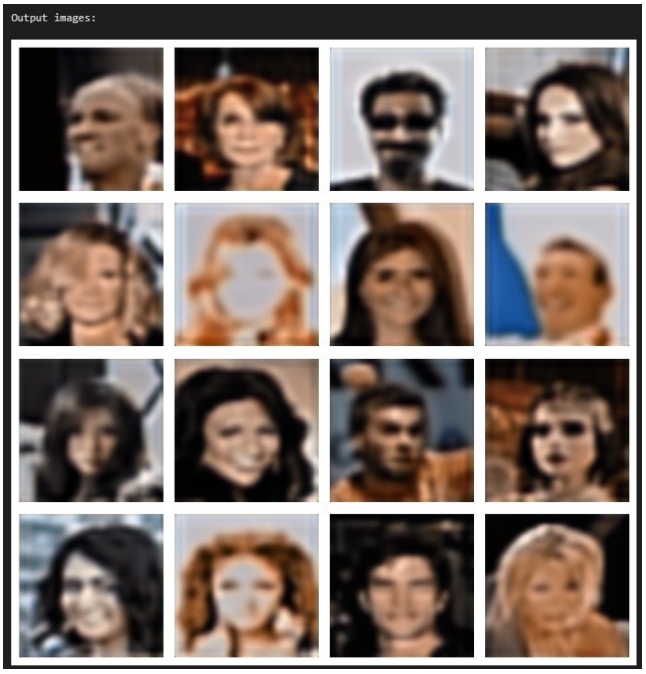
  with torch.no_grad(): บล็อกโค้ดนี้จะปิดการคำนวณ Gradient เพื่อประหยัดทรัพยากร

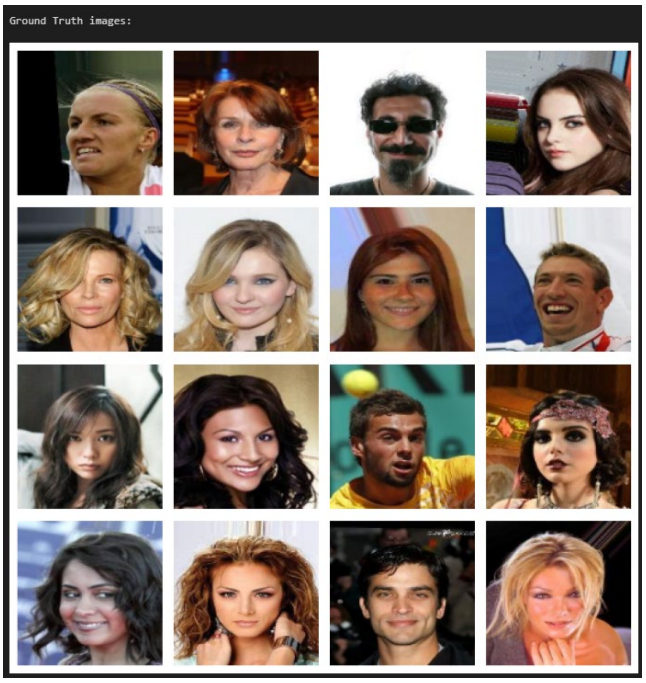
- **วนลูปผ่าน Batch ของข้อมูล Test:**
  - วนลูปผ่าน Batch ของข้อมูล Test ที่อยู่ใน dataloader
  - ย้ายข้อมูลภาพต้นฉบับ (images) และข้อมูลภาพเป้าหมาย (gt) ไปยังอุปกรณ์ที่เลือก (CPU ในกรณีนี้)
- **ส่งข้อมูลภาพต้นเข้าผ่านโมเดลเพื่อเอา Output:**
  - **outputs = model(images)**: ส่งข้อมูลภาพต้นฉบับ (images) 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:




Output images:


Ground Truth images:

Explore feature map

```python
class FeatureExtractor(nn.Module):
    def __init__(self, model, target_layers):
        super(FeatureExtractor, self).__init__()
        self.model = copy.deepcopy(model)
        self.target_layers = target_layers
        self.features = []

        for layer_name, layer in self.model.named_modules():
            if layer_name in target_layers:
                layer.register_forward_hook(self.save_feature(layer_name))

    def save_feature(self, layer_name):
        def hook(module, input, output):
            self.features.append(output)
        return hook

    def forward(self, x):
        self.features = []
        self.model(x)
        return self.features
```

[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

```python
### START CODE HERE ###
def visualize_feature_map(x, base_filename, model, target_layers):
    feature_extractor = FeatureExtractor(model, target_layers)
    feature_maps = feature_extractor(x)

    output_dir = os.path.dirname(base_filename)
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    if not feature_maps:
        print("No feature maps extracted.")
        return

    common_height, common_width = None, None
    all_layer_images = []

    for i, feature_map in enumerate(feature_maps):
        num_feature_maps = feature_map.size(1)
        grid_size = int(np.ceil(np.sqrt(num_feature_maps)))

        fig, axes = plt.subplots(grid_size, grid_size, figsize=(15, 15))
        axes = axes.flatten()

        for j in range(num_feature_maps):
            fmap = feature_map[0, j].cpu().detach().numpy()
            fmap_min = np.min(fmap)
            fmap_max = np.max(fmap)

            if fmap_max > fmap_min:
                fmap = (fmap - fmap_min) / (fmap_max - fmap_min)
            else:
                fmap = np.zeros_like(fmap)

            axes[j].imshow(fmap, cmap='gray')
            axes[j].axis('off')

        for k in range(num_feature_maps, len(axes)):
            axes[k].axis('off')

        plt.suptitle(f"Layer: {target_layers[i]}", fontsize=16, y=0.95)
        plt.subplots_adjust(top=0.9)


        each_layers_filename = f"{base_filename}_{target_layers[i]}.png"
        plt.savefig(each_layers_filename, bbox_inches='tight', pad_inches=0)
        plt.show()
        plt.close(fig)

        layer_image = cv2.imread(each_layers_filename, cv2.IMREAD_GRAYSCALE)
```

```
        if common_height is None or common_width is None:
            common_height, common_width = layer_image.shape[0], layer_image.shape[1]
        else:
            layer_image = cv2.resize(layer_image, (common_width, common_height), interpolation=cv2.INTER_LINEAR)

        padding = 50
        padding_image = np.ones((padding, common_width), dtype=np.uint8) * 255

        all_layer_images.append(layer_image)
        all_layer_images.append(padding_image)

    if all_layer_images:
        combined_image = np.vstack(all_layer_images)
        cv2.imwrite(f"{base_filename}.png", combined_image)
### END CODE HERE ###
```

[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

```
### START CODE HERE ###
target_layers = [
    'conv_in', 'down1', 'down2', 'latent',
    'up1', 'up2', 'up3', 'conv_out'
]

# target_layers = [
#       'down1'
# ]

# Use a single image from your DataLoader for visualization
data_iter = iter(trainloader)
x, _ = next(data_iter)  # Get a batch from the DataLoader
x = x[0].unsqueeze(0)  # Add batch dimension

visualize_feature_map(x, "feature_maps/feature_map", model, target_layers)
### END CODE HERE ###
```

[16] **กำหนดรายการชั้นที่ต้องการแสดงภาพ Feature Map:**

target_layers กำหนดรายการชื่อชั้นในโมเดล Autoencoder ที่ต้องการแสดงภาพ Feature Map

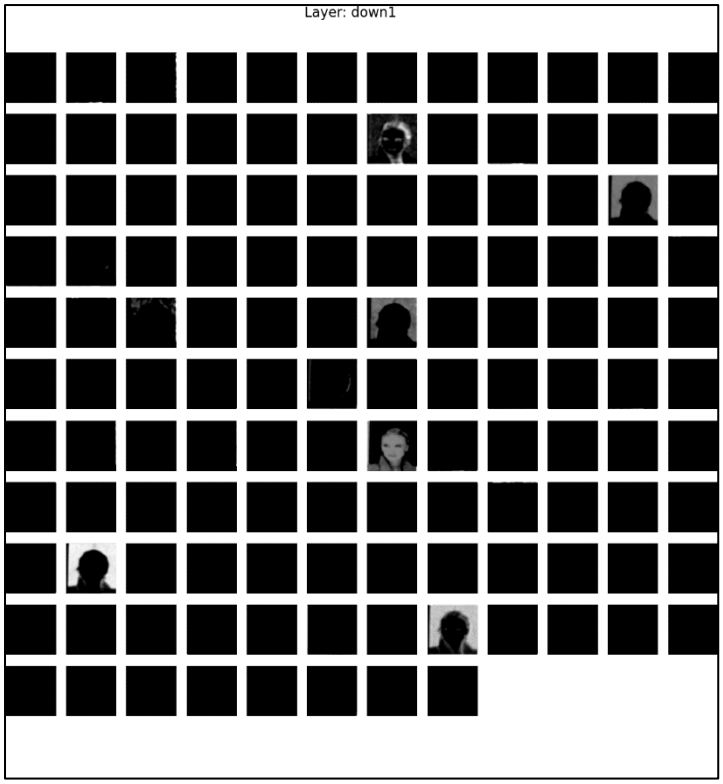
เลือกภาพตัวอย่าง:

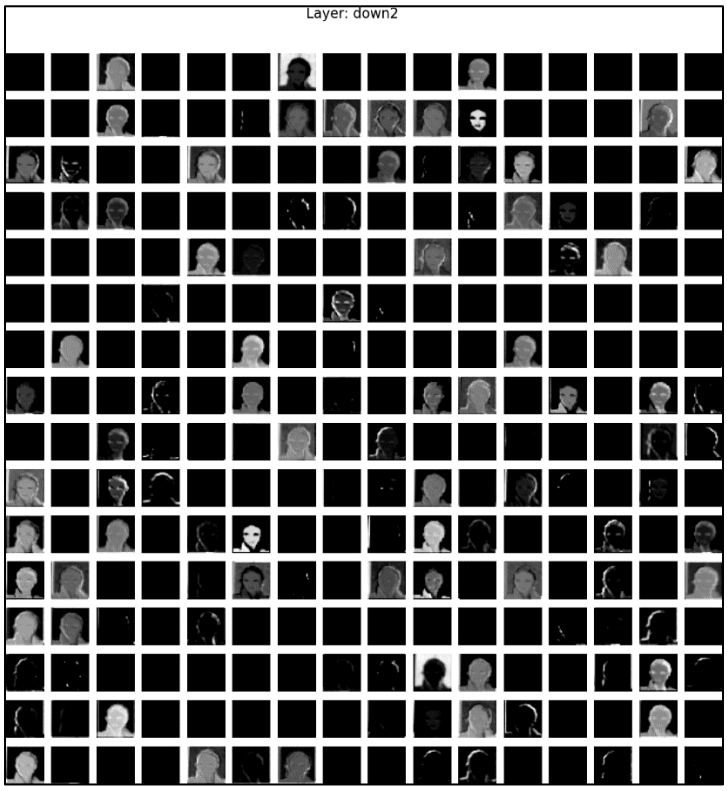
- data_iter = iter(trainloader) สร้าง Iterator สำหรับวนลูปผ่านข้อมูล Train
- x, _ = next(data_iter) ดึง Batch แรกจาก DataLoader
- x = x[0].unsqueeze(0) เลือกภาพแรกใน Batch และเพิ่มมิติ Batch เพื่อให้เข้ากับรูปแบบที่ฟังก์ชัน visualize_feature_map ต้องการ

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

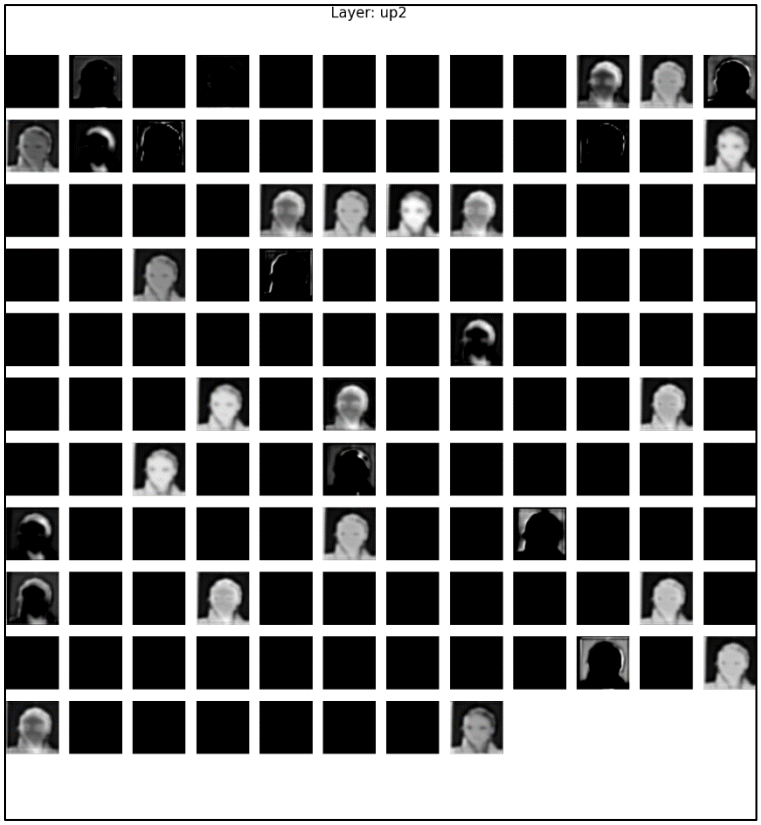
ผลลัพธ์การทดลอง:

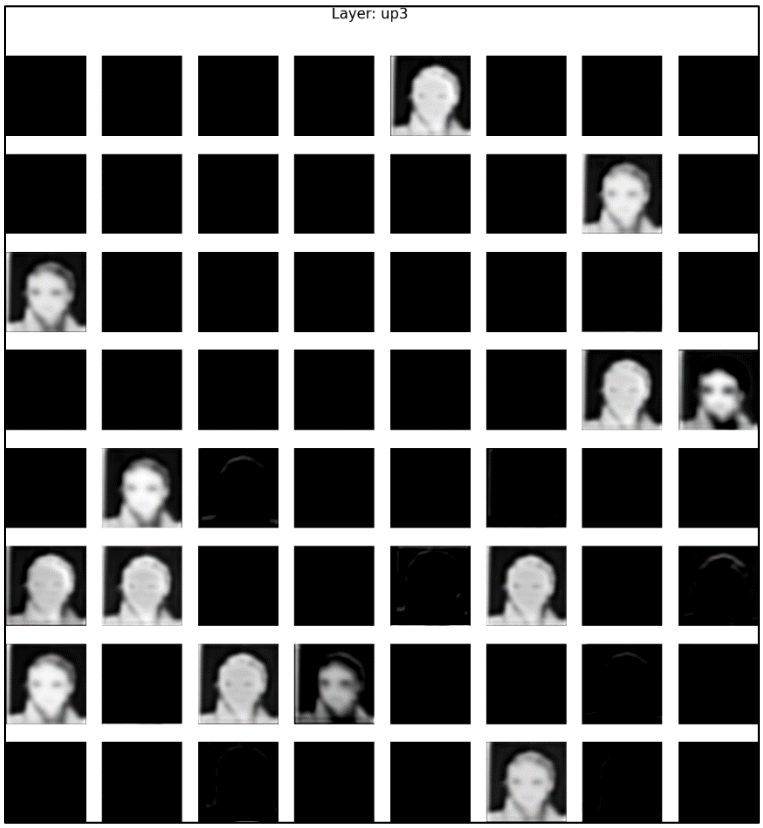

Layer: down1

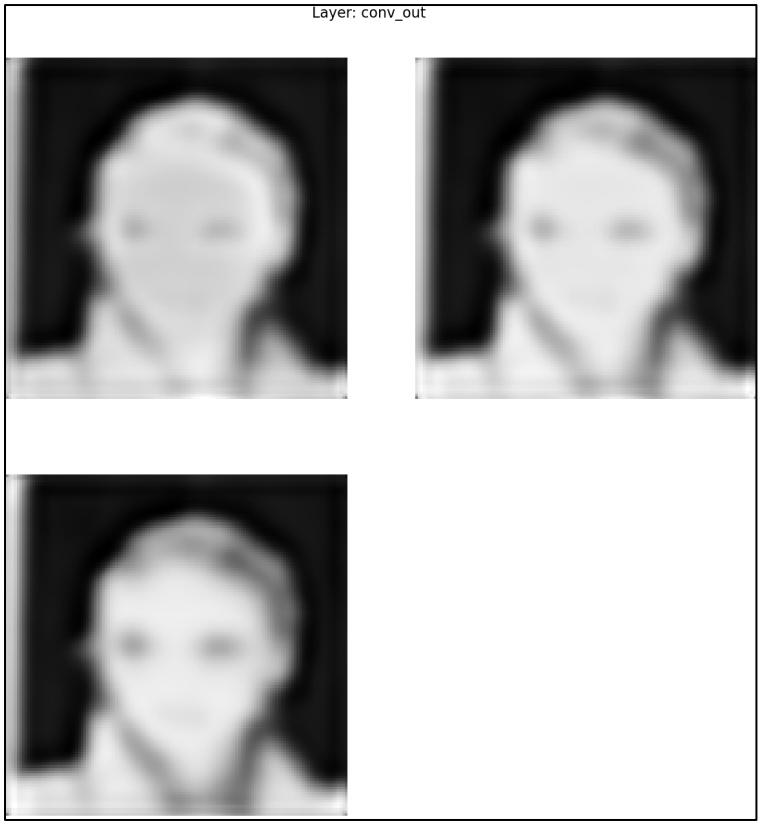

Layer: down2


Layer: latent


Layer: up1


Layer: up2


Layer: up3


Layer: conv_out