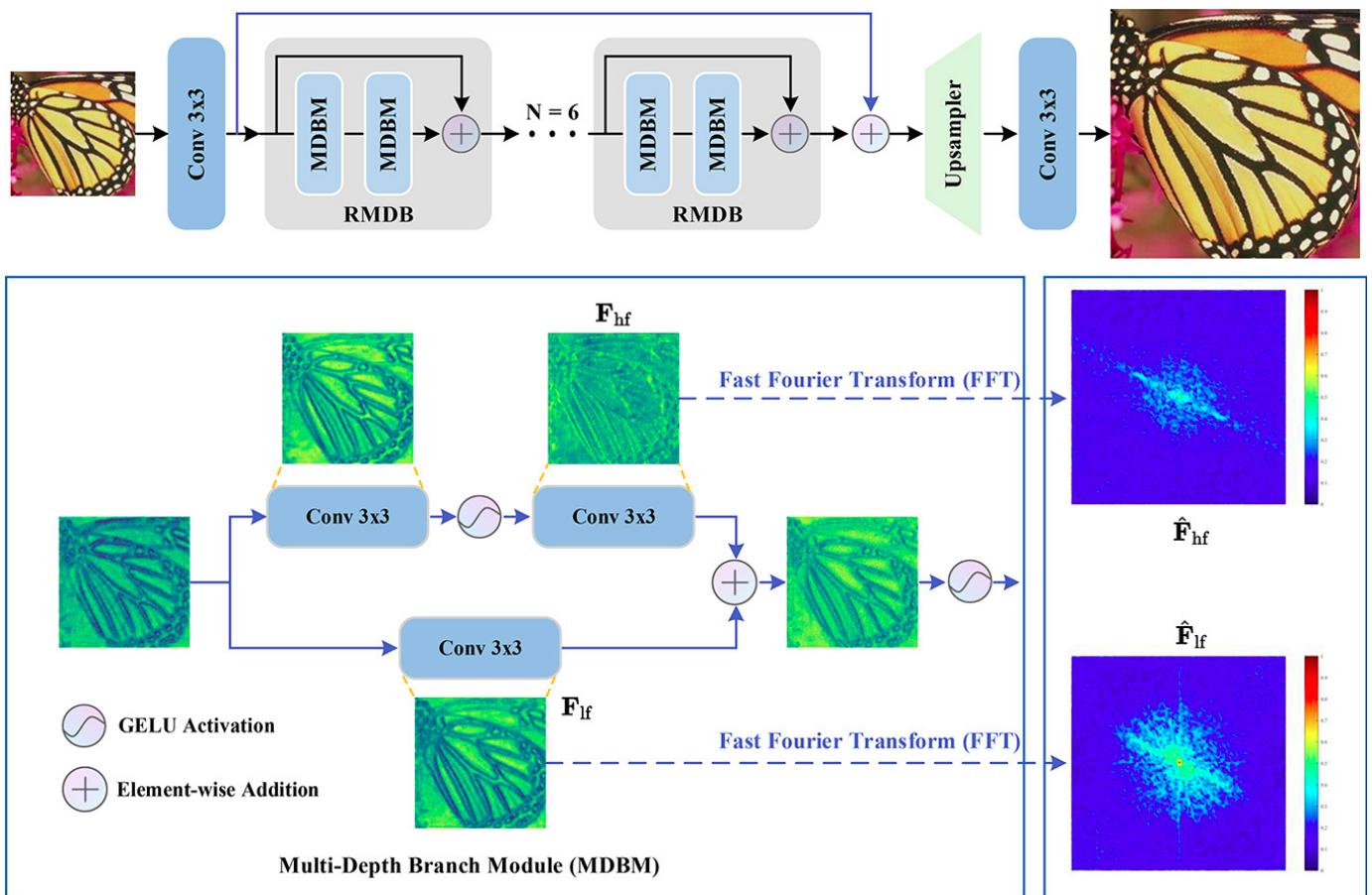


AI VIET NAM – COURSE 2023

## Domain Conversion – Exercise

Ngày 5 tháng 1 năm 2025

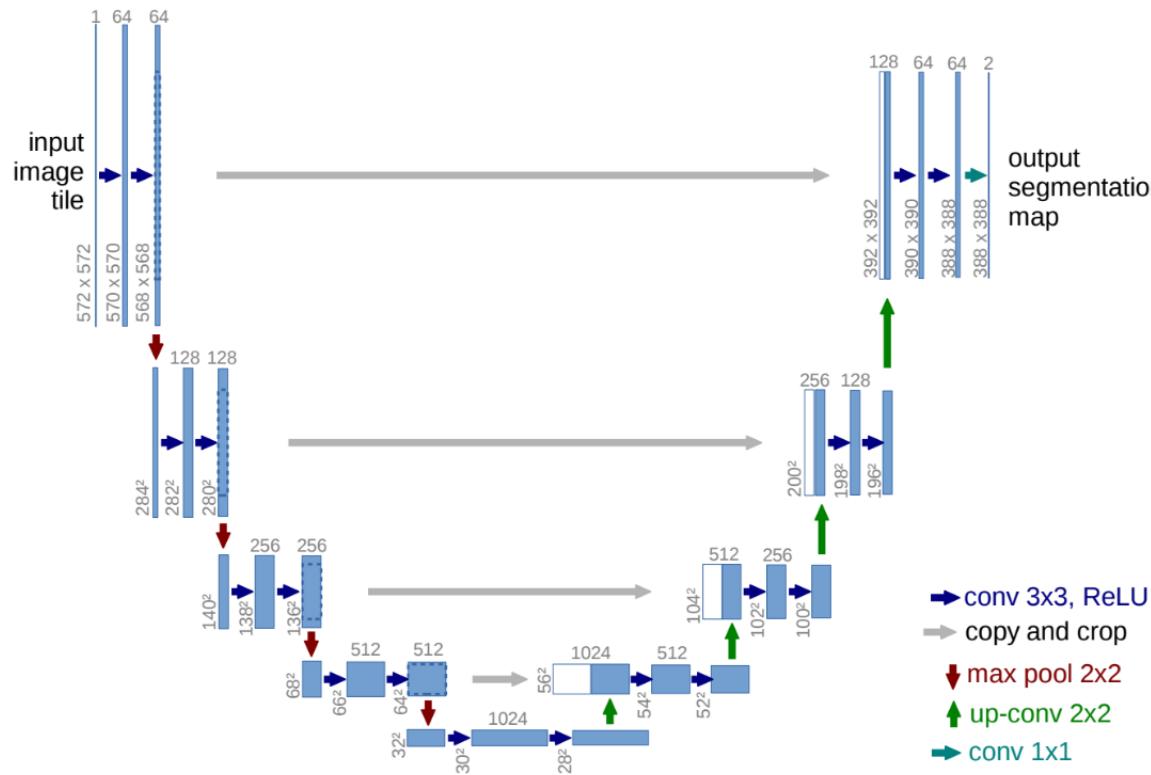


## Phần I: Lý Thuyết

### 1. Unet:

UNET là một mô hình mạng nơ-ron sâu được phát triển bởi Olaf Ronneberger, Philipp Fischer và Thomas Brox vào năm 2015, chủ yếu được sử dụng trong lĩnh vực xử lý ảnh và trí tuệ nhân tạo (AI). Mô hình này đã đạt được sự phổ biến rộng rãi trong các ứng dụng liên quan đến phân đoạn ảnh, như phân đoạn tế bào trong hình ảnh y học hoặc phân đoạn đối tượng trong hình ảnh chất lượng cao.

UNET sử dụng kiến trúc mạng Encoder-Decoder, với mục đích chính của lớp skip connection là tạo đường dẫn ngắn từ đầu vào đến đầu ra. Hàm kích hoạt thường được sử dụng trong UNET là Rectified Linear Unit (ReLU).



Hình 1: Image Super Resolution

## 2. Super Resolution:

Super-resolution (SISR) là một kỹ thuật quan trọng trong xử lý hình ảnh, nhằm tăng độ phân giải của hình ảnh. Phương pháp này có thể cải thiện chất lượng và chi tiết của hình ảnh, thường được áp dụng trong việc nâng cấp hình ảnh cũ, tăng cường hiệu suất giám sát video, và trong nhiều lĩnh vực ứng dụng khác.

Trong SISR, mô hình máy học được huấn luyện với bộ dữ liệu chứa các cặp hình ảnh, với mỗi cặp bao gồm một hình ảnh độ phân giải cao (SR/HR image) và một phiên bản thu nhỏ có độ phân giải thấp (LR image). Mô hình sử dụng thông tin từ các cặp này để tái tạo chi tiết bị mất mát và biến đổi hình ảnh thấp độ phân giải thành hình ảnh độ phân giải cao hơn.

Các phương pháp nội suy (truyền thống) như Nearest-neighbor Interpolation, Bilinear Interpolation và Bicubic Interpolation thường được sử dụng trong SISR để tái tạo các giá trị pixel bị thiếu trong quá trình tăng độ phân giải.

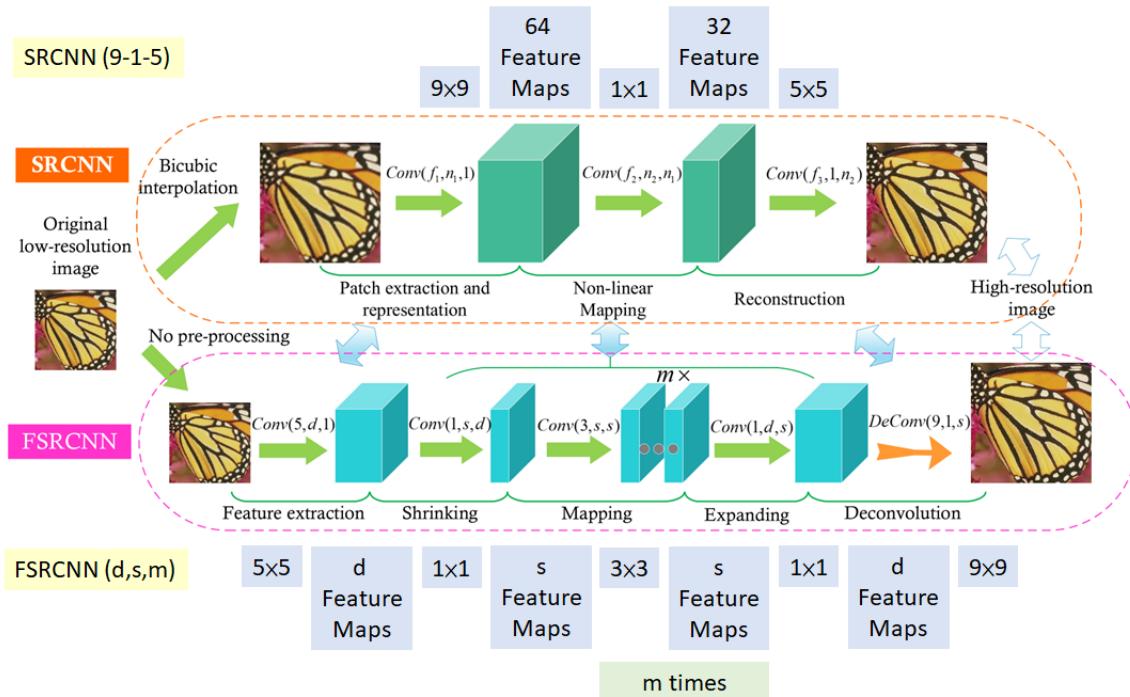
Một điểm đáng lưu ý là single-image super-resolution (SISR) thường khó khăn hơn so với multi-image super-resolution (MISR)/video super-resolution (VSR). SISR chỉ có một hình ảnh đầu vào để làm việc, trong khi MISR/VSR có thêm thông tin tham khảo từ nhiều hình ảnh hoặc video, giúp cải thiện khả năng tái tạo hình ảnh độ phân giải cao.

Trong SISR task, chúng ta cần khôi phục lại ảnh  $I_{SR}$  từ ảnh LR  $I_x$ .

$$I_{SR} = \mathcal{F}(I_x; \theta_{\mathcal{F}}) \quad (1)$$

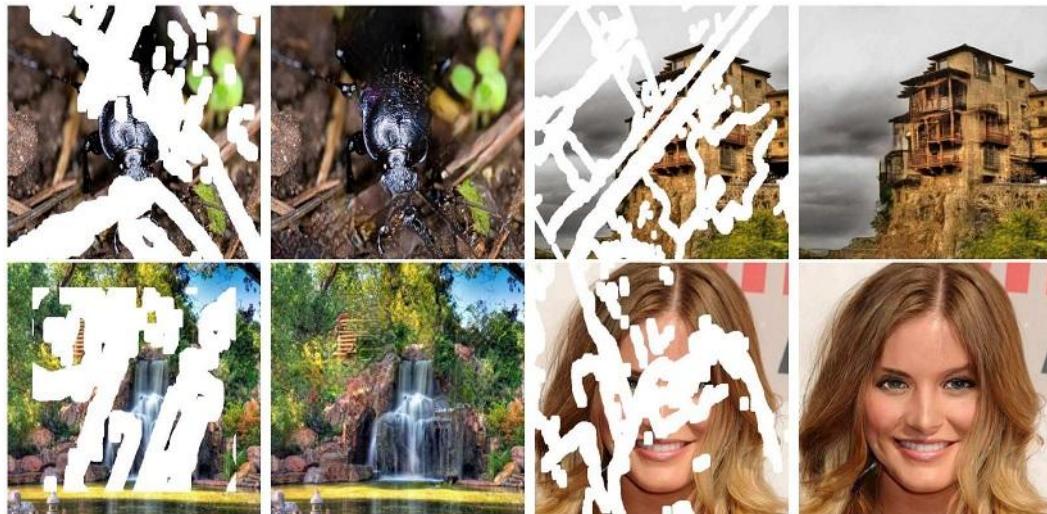
$\mathcal{F}$ : thuật toán hoặc super resolution model

$\theta_{\mathcal{F}}$ : là parameter của  $\mathcal{F}$



Hình 2: Image Super Resolution

### 3. Image Inpainting:



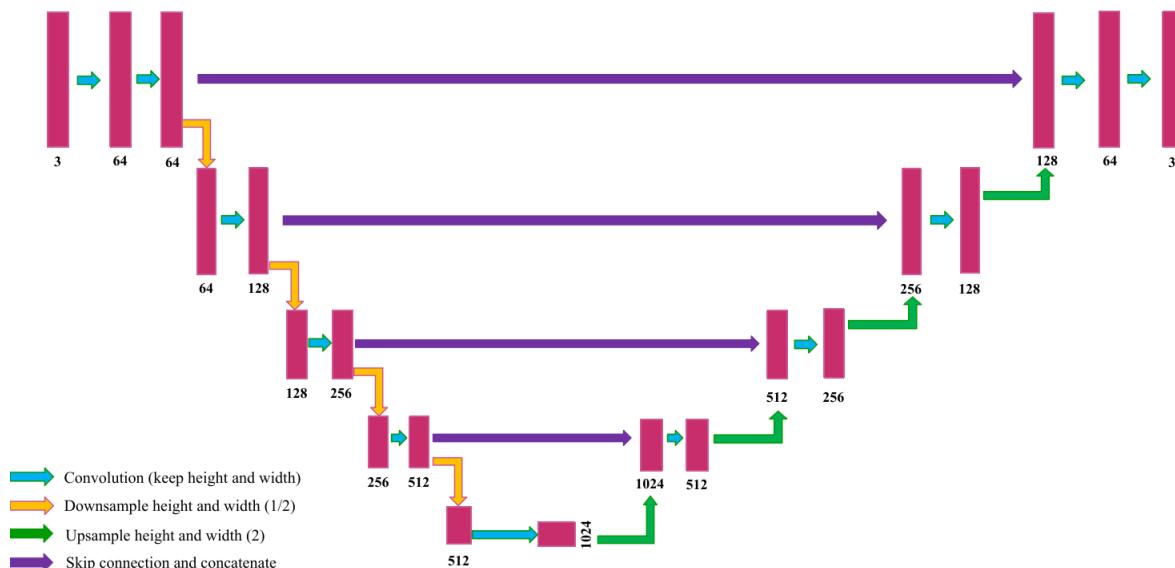
Hình 3: Image Inpainting

Inpainting là một phương pháp trong xử lý hình ảnh, được sử dụng để tái tạo hình ảnh bị thiếu một phần hoặc che khuất bằng cách dự đoán các pixel bị thiếu dựa trên thông tin có sẵn trong hình ảnh. Quá trình này có thể được thực hiện trong một hoặc hai giai đoạn. Trong inpainting, thông tin ngữ cảnh xung quanh các pixel bị thiếu là quan trọng để tái tạo hình ảnh một cách chính xác. Một số phương pháp inpainting sử dụng ví dụ hoặc các kỹ thuật khác để điền vào các pixel bị thiếu. Để biểu diễn một hình ảnh màu bị che khuất và mặt nạ (mask), ta thường tạo một hình ảnh kết hợp có bốn kênh, bao gồm hình ảnh gốc và mặt nạ (mask).

## Phần II: Bài Tập và Gợi Ý

Trong bài tập này các bạn sẽ xây dựng model cho riêng mình có kiến trúc giống Unet với mục đích là nhận Input là 1 ảnh màu (3 kênh) và generate ra 1 ảnh màu mới theo yêu cầu của từng task cụ thể (Bài tập 1). Sau đó các bạn chỉnh sửa model ở bài tập 1 để xây dựng model thực hiện task super resolution nhận ảnh input có kích thước  $64 \times 64$  và output ra ảnh có cùng nội dung nhưng kích thước tăng gấp 4 lần  $256 \times 256$  (Bài tập 2). Cuối cùng các bạn dùng model ở bài tập 1 để thực hiện task image inpainting cho ảnh có kích thước  $256 \times 256$ .

**1. Skip Connection Unet Architecture:** Xây dựng model có kiến trúc Unet có thông tin như hình 4 và code gợi ý bên dưới (các bạn có thể tham khảo thêm link file code ở trên) hoặc các bạn có thể xây dựng mode cho riêng mình theo yêu cầu là downsample sẽ thực hiện được 4 lần (giảm  $1/2, 1/4, 1/8, 1/16$  so với input image), sau đó sẽ upsample 4 lần để khôi phục lại size ảnh ban đầu ( $x2, x4, x8, x16$ ). Để kết hợp với skip connection các bạn nên dùng phép concatenate.



Hình 4: Kiến trúc Unet model

Bên dưới là đoạn code để các bạn tham khảo xây dựng network theo kiến trúc Unet:

```

1 class FirstFeature(nn.Module):
2     def __init__(self, in_channels, out_channels):
3         super(FirstFeature, self).__init__()
4         self.conv = nn.Sequential(
5             nn.Conv2d(in_channels, out_channels, 1, 1, 0, bias=False),
6             nn.LeakyReLU()
7         )
8
9     def forward(self, x):
10        return self.conv(x)
11
12
13 class ConvBlock(nn.Module):
14     def __init__(self, in_channels, out_channels):
15         super(ConvBlock, self).__init__()
16         self.conv = nn.Sequential(
17             nn.Conv2d(in_channels, out_channels, 3, 1, 1, bias=False),

```

```
18         nn.BatchNorm2d(out_channels),
19         nn.LeakyReLU(inplace=True),
20         nn.Conv2d(out_channels, out_channels, 3, 1, 1, bias=False),
21         nn.BatchNorm2d(out_channels),
22         nn.LeakyReLU(inplace=True),
23     )
24
25     def forward(self, x):
26         return self.conv(x)
27
28
29 class Encoder(nn.Module):
30     def __init__(self, in_channels, out_channels) -> None:
31         super().__init__()
32         self.encoder = nn.Sequential(
33             nn.MaxPool2d(2),
34             ConvBlock(in_channels, out_channels)
35         )
36
37     def forward(self, x):
38         x = self.encoder(x)
39         return x
40
41
42 class Decoder(nn.Module):
43     def __init__(self, in_channels, out_channels):
44         super(Decoder, self).__init__()
45         self.conv = nn.Sequential(
46             nn.UpsamplingBilinear2d(scale_factor=2),
47             nn.Conv2d(in_channels, out_channels, 1, 1, 0, bias=False),
48             nn.BatchNorm2d(out_channels),
49             nn.LeakyReLU(),
50         )
51         self.conv_block = ConvBlock(in_channels, out_channels)
52
53     def forward(self, x, skip):
54         x = self.conv(x)
55         x = torch.concat([x, skip], dim=1)
56         x = self.conv_block(x)
57         return x
58
59
60 class FinalOutput(nn.Module):
61     def __init__(self, in_channels, out_channels):
62         super(FinalOutput, self).__init__()
63         self.conv = nn.Sequential(
64             nn.Conv2d(in_channels, out_channels, 1, 1, 0, bias=False),
65             nn.Tanh()
66         )
67
68     def forward(self, x):
69         return self.conv(x)
70
71
72 class Unet(nn.Module):
73     def __init__(
74         self, n_channels=3, n_classes=3, features=[64, 128, 256, 512],
75     ):
76         super(Unet, self).__init__()
```

```

78         self.n_channels = n_channels
79         self.n_classes = n_classes
80
81         self.in_conv1 = FirstFeature(n_channels, 64)
82         self.in_conv2 = ConvBlock(64, 64)
83
84         self.enc_1 = Encoder(64, 128)
85         self.enc_2 = Encoder(128, 256)
86         self.enc_3 = Encoder(256, 512)
87         self.enc_4 = Encoder(512, 1024)
88
89         self.dec_1 = Decoder(1024, 512)
90         self.dec_2 = Decoder(512, 256)
91         self.dec_3 = Decoder(256, 128)
92         self.dec_4 = Decoder(128, 64)
93
94         self.out_conv = FinalOutput(64, n_classes)
95
96
97     def forward(self, x):
98         x = self.in_conv1(x)
99         x1 = self.in_conv2(x)
100
101        x2 = self.enc_1(x1)
102        x3 = self.enc_2(x2)
103        x4 = self.enc_3(x3)
104        x5 = self.enc_4(x4)
105
106        x = self.dec_1(x5, x4)
107        x = self.dec_2(x, x3)
108        x = self.dec_3(x, x2)
109        x = self.dec_4(x, x1)
110
111        x = self.out_conv(x)
112        return x

```

### FirstFeature Class:

- **Mục đích:** Tạo feature map ban đầu từ input
- **Thành phần:** Một lớp convolution duy nhất sau là hàm LeakyReLU. convolution này sử dụng kích thước kernel là 1, bước nhảy là 1 và không padding. Đây là một lớp đơn giản được thiết kế để mở rộng số lượng channel cho feature map

### ConvBlock Class:

- **Mục đích:** Khối convolution cơ bản để trích xuất đặc trưng
- **Thành phần:** Hai nhóm Conv-BatchNorm-LeakyReLU liên tục. Khối này là một khối cơ bản trong U-Net, được sử dụng cho cả down-sampling và up-sampling

### Encoder Class:

- **Mục đích:** Để giảm size của feature map và trích xuất các high-level feature.
- **Thành phần:** Một lớp Max Pooling sau là ConvBlock. Max Pooling giảm kích thước xuống một nửa, trong khi ConvBlock xử lý các đặc trưng.

### Decoder Class:

- **Mục đích:** Để tăng kích thước feature map và kết hợp với feature map tương ứng từ Encoder (skip connection).
- **Thành phần:** Upsampling (sử dụng nội suy bilinear) để tăng kích thước không gian. Một lớp convolution để giảm số lượng channel. Một ConvBlock để xử lý các feature được ghép (từ lớp upsampling và skip connection).

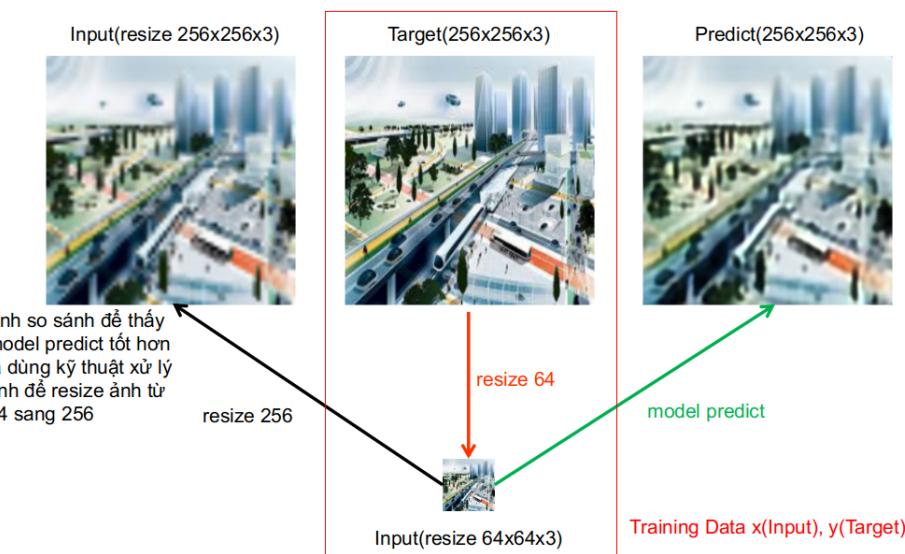
#### FinalOutput Class:

- **Mục đích:** Tạo ra đầu ra cuối cùng từ feature map cuối cùng.
- **Thành phần:** Một lớp convolution với hàm Tanh. Điều này giảm số lượng channel đầu ra xuống bằng số lượng channel của ảnh màu

#### Unet Class:

- **Mục đích:** Kết hợp tất cả các thành phần trên thành một kiến trúc U-Net đầy đủ.
- **Thành phần:** Xử lý ảnh đầu vào bằng FirstFeature và ConvBlock. Bốn lớp Encoder với số channel tăng dần, mỗi lớp tiếp tục downsample và xử lý feature map. Bốn lớp Decoder với số channel giảm dần, mỗi lớp tăng kích thước, kết hợp đặc trưng từ encoder (skip connection). Một lớp FinalOutput để tạo ra ảnh đã được xử lý.
- **Forward:** Đầu vào được xử lý qua các convolution ban đầu. Sau đó được downsample 4 lần, rồi được up sample lên 4 lần mỗi lần kết hợp với feature từ encoder. Cuối cùng đi qua lớp convolution cuối cùng để tạo ảnh đã xử lý

2. **Super Resolution with UNET:** Bài toán này giải quyết vấn đề khi một ảnh nhỏ resize lên một ảnh có kích thước lớn hơn sẽ khiến ảnh bị mờ, do đó các bạn sẽ xây dựng CNN model resize ảnh lớn hơn nhưng vẫn giữ được ảnh chất lượng tốt. Cho 1 tập data **Khoa\_LHR\_image.zip** đã được chia làm 2 phần train (685 ảnh), val (170 ảnh) và các ảnh có kích thước là 256x256x3. Các bạn **hãy tự tạo cho mình tập dataset với ảnh input chính là các ảnh trong tập data** nhưng kích thước ảnh đã giảm đi 4 lần (64x64x3). Sau đó dùng UNET model theo yêu cầu bên dưới để thực hiện bài toán, input là ảnh 64x64x3 và output là ảnh 256x256x3, target chính là ảnh gốc 256x256x3 của tập data



Yêu cầu model:

- Unet ở bài tập 1 (No skip connection: các bạn loại bỏ thành phần skip connection)
- Unet ở bài tập 1
- So sánh kết quả 2 models

Các bước thực hiện:

- Xây dựng dataset từ ảnh gốc (256x256x3). Khi load ảnh, mỗi sample các bạn tạo ra 2 ảnh (input và target). Input là ảnh gốc resize giảm đi 4 lần (64x64x3), target là ảnh gốc 256x256x3
- Chia data thành các tập train, validation (test tùy thuộc các bạn)
- Normalize data (dùng 1 trong các kỹ thuật chuẩn hóa đã được học). Kỹ thuật này phải phù hợp với activation của layer cuối cùng trong model để đảm bảo ảnh output từ model có giá trị trong range các giá trị của ảnh thông thường (có thể đã scale hoặc không).
- Điều chỉnh UNET model của bài tập 1 để nhận input là ảnh 64x64x3 output ra ảnh có kích thước 256x256x3
- Lựa chọn loss phù hợp cho việc output ra 1 ảnh từ model
- Config các hyperparameter
- Train và test kết quả

Đối với Super Resolution task chúng ta cần resize ảnh đầu vào từ 64 thành 256 (gấp 4 lần) trước khi đưa vào model. Do đó các bạn cần khai báo `resize_fnc` (dòng 1) và sử dụng trong forward method của model (dòng 3) theo đoạn code tham khảo bên dưới

```
1 self.resize_fnc = transforms.Resize((LOW_IMG_HEIGHT*4, LOW_IMG_HEIGHT*4),
2                                     antialias=True)
3 x = self.resize_fnc(x)
```

Đối với việc không sử dụng skip connection trong model chúng ta cần chỉnh sửa lại phần Decoder bằng cách không dùng concat và gấp đôi `out_channel` ở dòng 47, 48, 51

```
1 class FirstFeatureNoSkip(nn.Module):
2     def __init__(self, in_channels, out_channels):
3         super(FirstFeatureNoSkip, self).__init__()
4         self.conv = nn.Sequential(
5             nn.Conv2d(in_channels, out_channels, 1, 1, 0, bias=False),
6             nn.LeakyReLU()
7         )
8
9     def forward(self, x):
10        return self.conv(x)
11
12
13 class ConvBlockNoSkip(nn.Module):
14     def __init__(self, in_channels, out_channels):
15         super(ConvBlockNoSkip, self).__init__()
16         self.conv = nn.Sequential(
17             nn.Conv2d(in_channels, out_channels, 3, 1, 1, bias=False),
18             nn.BatchNorm2d(out_channels),
19             nn.LeakyReLU(inplace=True),
20             nn.Conv2d(out_channels, out_channels, 3, 1, 1, bias=False),
21             nn.BatchNorm2d(out_channels),
22             nn.LeakyReLU(inplace=True),
23         )
```

```
24
25     def forward(self, x):
26         return self.conv(x)
27
28
29 class EncoderNoSkip(nn.Module):
30     def __init__(self, in_channels, out_channels) -> None:
31         super(EncoderNoSkip, self).__init__()
32         self.encoder = nn.Sequential(
33             nn.MaxPool2d(2),
34             ConvBlockNoSkip(in_channels, out_channels)
35         )
36
37     def forward(self, x):
38         x = self.encoder(x)
39         return x
40
41
42 class DecoderNoSkip(nn.Module):
43     def __init__(self, in_channels, out_channels):
44         super(DecoderNoSkip, self).__init__()
45         self.conv = nn.Sequential(
46             nn.UpsamplingBilinear2d(scale_factor=2),
47             nn.Conv2d(in_channels, out_channels*2, 1, 1, 0, bias=False),
48             nn.BatchNorm2d(out_channels*2),
49             nn.LeakyReLU(),
50         )
51         self.conv_block = ConvBlockNoSkip(out_channels*2, out_channels)
52
53     def forward(self, x):
54         x = self.conv(x)
55         x = self.conv_block(x)
56         return x
57
58
59 class FinalOutputNoSkip(nn.Module):
60     def __init__(self, in_channels, out_channels):
61         super(FinalOutputNoSkip, self).__init__()
62         self.conv = nn.Sequential(
63             nn.Conv2d(in_channels, out_channels, 1, 1, 0, bias=False),
64             nn.Tanh()
65         )
66
67     def forward(self, x):
68         return self.conv(x)
69
70
71 class SR_Unet_NoSkip(nn.Module):
72     def __init__(self, n_channels=3, n_classes=3):
73         super(SR_Unet_NoSkip, self).__init__()
74
75         self.n_channels = n_channels
76         self.n_classes = n_classes
77         self.resize_fnc = transforms.Resize((LOW_IMG_HEIGHT*4, LOW_IMG_HEIGHT*4),
78                                           antialias=True)
79         self.in_conv1 = FirstFeatureNoSkip(n_channels, 64)
80         self.in_conv2 = ConvBlockNoSkip(64, 64)
81
82
83
```

```

84         self.enc_1 = EncoderNoSkip(64, 128)
85         self.enc_2 = EncoderNoSkip(128, 256)
86         self.enc_3 = EncoderNoSkip(256, 512)
87         self.enc_4 = EncoderNoSkip(512, 1024)
88
89         self.dec_1 = DecoderNoSkip(1024, 512)
90         self.dec_2 = DecoderNoSkip(512, 256)
91         self.dec_3 = DecoderNoSkip(256, 128)
92         self.dec_4 = DecoderNoSkip(128, 64)
93
94         self.out_conv = FinalOutputNoSkip(64, n_classes)
95
96
97     def forward(self, x):
98         x = self.resize_fnc(x)
99         x = self.in_conv1(x)
100        x = self.in_conv2(x)
101
102        x = self.enc_1(x)
103        x = self.enc_2(x)
104        x = self.enc_3(x)
105        x = self.enc_4(x)
106
107        x = self.dec_1(x)
108        x = self.dec_2(x)
109        x = self.dec_3(x)
110        x = self.dec_4(x)
111
112        x = self.out_conv(x)
113        return x

```

Dễ có thể tạo được super resolution data chúng ta có thể tham khảo đoạn code bên dưới

```

1 class ImageDataset(Dataset):
2     def __init__(self, img_dir, is_train=True):
3         self.resize = transforms.Resize((LOW_IMG_WIDTH, LOW_IMG_HEIGHT),
4                                         antialias=True)
5         self.is_train = is_train
6         self.img_dir = img_dir
7         self.images = os.listdir(img_dir)
8
9     def __len__(self):
10        return len(self.images)
11
12    def normalize(self, input_image, target_image):
13        input_image = input_image*2 - 1
14        target_image = target_image*2 - 1
15
16        return input_image, target_image
17
18    def random_jitter(self, input_image, target_image):
19        if torch.rand([]) < 0.5:
20            input_image = transforms.functional.hflip(input_image)
21            target_image = transforms.functional.hflip(target_image)
22
23        return input_image, target_image
24
25    def __getitem__(self, idx):
26        img_path = os.path.join(self.img_dir, self.images[idx])
27        image = np.array(Image.open(img_path).convert("RGB"))
28        image = transforms.functional.to_tensor(image)

```

```

28     input_image = self.resize(image).type(torch.float32)
29     target_image = image.type(torch.float32)
30
31     input_image, target_image = self.normalize(input_image, target_image)
32
33     if self.is_train:
34         input_image, target_image = self.random_jitter(input_image,
35                                         target_image)
36
37     return input_image, target_image

```

### **Khởi Tạo Class (`__init__` method):**

- Hàm khởi tạo nhận img\_dir và tham số tùy chọn is\_train. img\_dir là thư mục chứa ảnh, và is\_train chỉ ra liệu tập dữ liệu có được sử dụng cho việc huấn luyện hay không (mặc định là True).
- self.resize: Thay đổi kích thước ảnh theo chiều rộng và chiều cao.
- self.is\_train: Cho biết liệu tập dữ liệu có dùng cho huấn luyện không.
- self.images: Danh sách đường dẫn tệp cho tất cả ảnh trong img\_dir.

### **`__len__` method:**

- Trả về số lượng ảnh trong tập dữ liệu, cần thiết để PyTorch hiểu kích thước của tập dữ liệu.

### **Chuẩn Hóa (normalize method):**

- Normalize ảnh đầu vào và target. Các ảnh được chuyển từ [0, 255] sang [-1, 1]. Đây là normalize phổ biến để ổn định quá trình huấn luyện.

### **`random_jitter` method:**

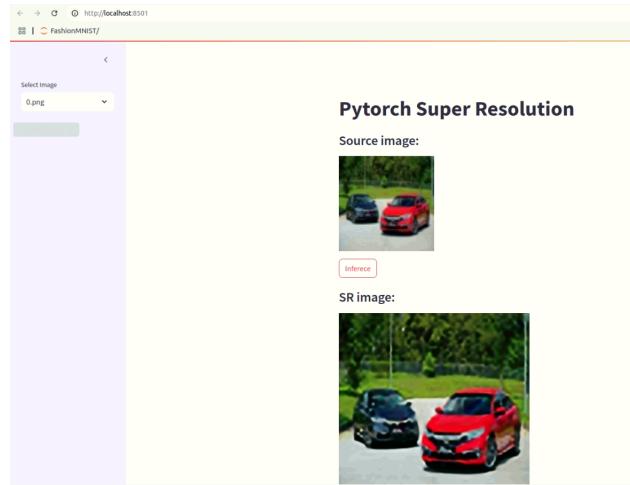
- Thực hiện tăng cường dữ liệu. Ngẫu nhiên áp dụng lật ngang cho cả ảnh đầu vào và target với xác suất 50%.

### **`__getitem__` method**

- `__getitem__` được sử dụng bởi PyTorch để lấy từng ảnh từ tập dữ liệu.
- img\_path: Lấy đường dẫn của ảnh tại idx nhất định.
- image: Đọc ảnh theo chế độ RGB.
- input\_image và target\_image: Ảnh gốc được thay đổi kích thước để tạo input\_image, và ảnh gốc cũng được giữ lại làm target\_image. Cả hai được chuyển đổi sang kiểu dữ liệu torch.float32.
- Các ảnh sau đó được chuẩn hóa bằng phương thức normalize.
- Nếu tập dữ liệu được sử dụng cho huấn luyện (self.is\_train là True), ‘random\_jitter’ được áp dụng cho cả input\_image và target\_image.

**Triển khai mô hình:** Dựa trên mô hình đã huấn luyện và thư viện streamlit để triển khai ứng dụng:

- [Github](#)
- Giao Diện



Hình 5: Giao diện ứng dụng

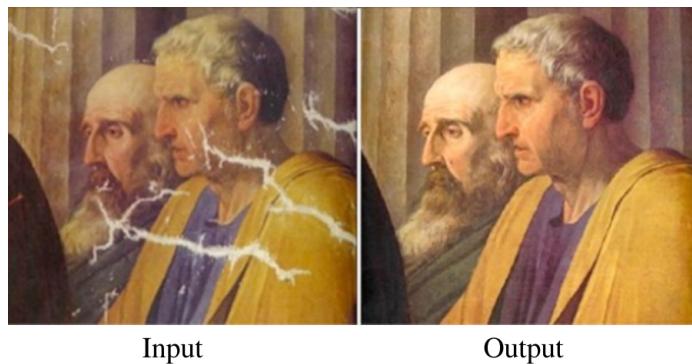
- 3. Image Inpainting with UNET:** Bài toán này giải quyết vấn đề khôi phục ảnh input bị hư tổn bằng cách loại bỏ lỗi và khôi phục lại ảnh. Các bạn chọn 1 tập data bất kỳ, ví dụ chon tập data **Khoa\_LHR\_image.zip** đã được chia làm 2 phần train (685 ảnh), val (170 ảnh) và các ảnh có kích thước là 256x256x3. Các bạn **hãy tự tạo cho mình tập dataset với ảnh input chính là các ảnh trong tập data nhưng đã được vẽ random các line vào trong ảnh.** Sau đó **tạo một UNET model input là ảnh bị hư tổn 256x256x3 và output là ảnh 256x256x3 đã khôi phục được ảnh bị hư, target chính là ảnh gốc 256x256x3 của tập data khi chưa được vẽ random các line**

Yêu cầu model:

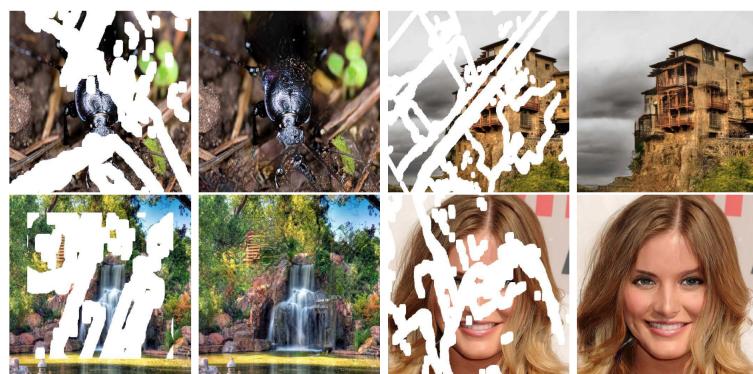
- Unet ở bài tập 1 (No skip connection: các bạn loại bỏ thành phần skip connection)
- Unet ở bài tập 1
- (Optional) Unet với pretrained weights
- So sánh kết quả 3 models

Các bước thực hiện:

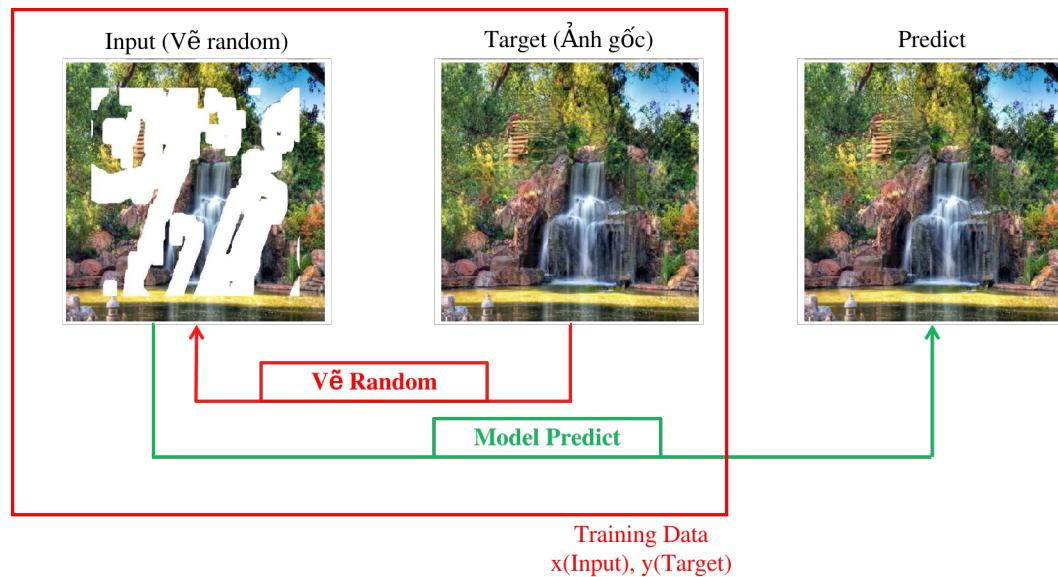
- Xây dựng datset từ ảnh gốc. Khi load ảnh mỗi sample các bạn tạo ra 2 ảnh (input và target). Input là ảnh gốc và được vẽ thêm các line random, target là ảnh gốc. (Cách vẽ line random các bạn có thể tùy chọn các cách mà các bạn tự học và tìm hiểu được)
- Chia data thành các tập train, validation (test tùy thuộc các bạn)
- Normalize data (dùng 1 trong các kỹ thuật chuẩn hóa đã được học). Kỹ thuật này phải phù hợp với activation của layer cuối cùng trong model để đảm bảo ảnh output từ model có giá trị trong range các giá trị của ảnh thông thường (có thể đã scale hoặc không).
- Xây dựng UNET model nhận input là ảnh bị hư tổn, ouput ra ảnh có cùng kích thước với input nhưng đã được khôi phục
- Lựa chọn loss phù hợp cho việc output ra 1 ảnh từ model
- Config các hyperparameter, Train và test kết quả



Hình 6: Ứng dụng khôi phục ảnh cũ



Hình 7: Một số ví dụ khi tạo inpainting dataset



Các bạn có thể tham khảo đoạn code dưới đây để vẽ random lên ảnh

```

1 class ImageDataset(Dataset):
2     def __init__(self, img_dir, is_train=True):
3         self.is_train = is_train
4         self.img_dir = img_dir
5         self.images = os.listdir(img_dir)

```

```

6
7     def __len__(self):
8         return len(self.images)
9
10    def normalize(self, input_image, target_image):
11        input_image = input_image*2 - 1
12        target_image = target_image*2 - 1
13
14        return input_image, target_image
15
16    def random_jitter(self, input_image, target_image):
17        if torch.rand([]) < 0.5:
18            input_image = transforms.functional.hflip(input_image)
19            target_image = transforms.functional.hflip(target_image)
20
21        return input_image, target_image
22
23    def create_mask(self, image):
24        masked_image = image.copy()
25        ## Prepare masking matrix
26        mask = np.full((IMG_WIDTH,IMG_HEIGHT,3), 0, np.uint8)
27        for _ in range(np.random.randint(1, 5)):
28            # Get random x locations to start line
29            x1, x2 = np.random.randint(1, IMG_WIDTH), np.random.randint(1, IMG_WIDTH)
30            # Get random y locations to start line
31            y1, y2 = np.random.randint(1, IMG_HEIGHT), np.random.randint(1, IMG_HEIGHT)
32
33            # Get random thickness of the line drawn
34            thickness = np.random.randint(1, 15)
35            # Draw line on the black mask
36            cv2.line(mask,(x1,y1),(x2,y2),(1,1,1),thickness)
37
38        masked_image = np.where(mask, 255*np.ones_like(mask), masked_image)
39        return masked_image
40
41    def __getitem__(self, idx):
42        img_path = os.path.join(self.img_dir, self.images[idx])
43        image = np.array(Image.open(img_path).convert("RGB"))
44
45        input_image = self.create_mask(image)
46        input_image = transforms.functional.to_tensor(input_image)
47        target_image = transforms.functional.to_tensor(image)
48
49        input_image = input_image.type(torch.float32)
50        target_image = target_image.type(torch.float32)
51
52        input_image, target_image = self.normalize(input_image, target_image)
53
54        if self.is_train:
55            input_image, target_image = self.random_jitter(input_image, target_image)
56
57        return input_image, target_image

```

### Khởi Tạo Class (`__init__` method):

- Hàm khởi tạo nhận `img_dir` và tham số tùy chọn `is_train`. `img_dir` là thư mục chứa ảnh, và `is_train` chỉ ra liệu tập dữ liệu có được sử dụng cho việc huấn luyện hay không (mặc định là True).
- `self.is_train`: Cho biết liệu tập dữ liệu có dùng cho huấn luyện không.

- self.img\_files: Danh sách đường dẫn tệp cho tất cả ảnh trong images.

#### **\_\_len\_\_ method:**

- Trả về số lượng ảnh trong tập dữ liệu, cần thiết để PyTorch hiểu kích thước của tập dữ liệu.

#### **Chuẩn Hóa (normalize method):**

- Normalize ảnh đầu vào và target. Các ảnh được chuyển từ [0, 255] sang [-1, 1]. Đây là normalize phổ biến để ổn định quá trình huấn luyện.

#### **random\_jitter method:**

- Thực hiện tăng cường dữ liệu. Ngẫu nhiên áp dụng lật ngang cho cả ảnh đầu vào và target với xác suất 50%.

#### **create\_mask method:**

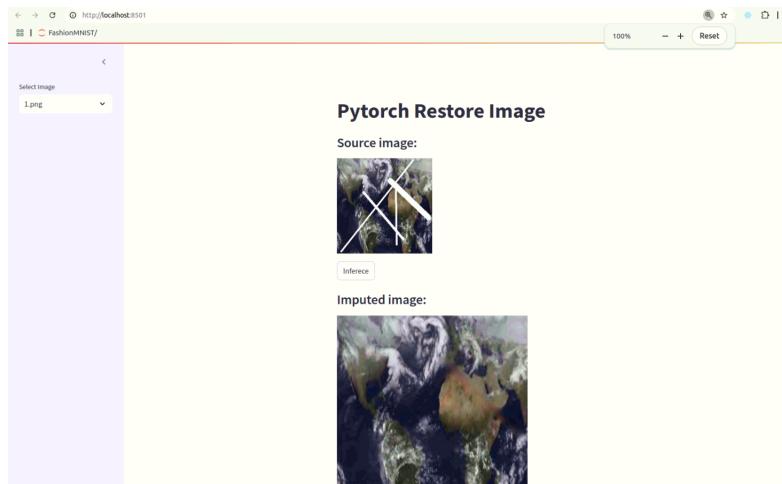
- Tạo và áp dụng một mask ngẫu nhiên lên hình ảnh.
- Vẽ các đường ngẫu nhiên trên hình ảnh, từ 1 đến 5 đường

#### **\_\_getitem\_\_ method**

- \_\_getitem\_\_ được sử dụng bởi PyTorch để lấy từng ảnh từ tập dữ liệu.
- img\_path: Lấy đường dẫn của ảnh tại idx nhất định.
- image: Đọc ảnh theo chế độ RGB.
- input\_image và target\_image: Ảnh gốc bị vẽ lên các đường ngẫu nhiên để tạo input\_image, và ảnh gốc cũng được giữ lại làm target\_image. Cả hai được chuyển đổi sang kiểu dữ liệu torch.float32.
- Các ảnh sau đó được chuẩn hóa bằng phương thức normalize.
- Nếu tập dữ liệu được sử dụng cho huấn luyện (self.is\_train là True), 'random\_jitter' được áp dụng cho cả input\_image và target\_image.

**Triển khai mô hình:** Dựa trên mô hình đã huấn luyện và thư viện streamlit để triển khai ứng dụng:

- [Github](#)
- Giao Diện



Hình 8: Giao diện ứng dụng

### Phần III: Trắc Nghiệm

1. Chức năng của các skip connection trong U-Net là gì?
  - (A). Tăng chi phí tính toán của mạng
  - (B)**. Kết hợp các feature map từ nhánh dowsampling với các feature map đã được upsample tương ứng trong nhánh mở rộng
  - (C). Dưa tính phi tuyến vào mạng
  - (D). Giảm số lượng tham số trong mạng
  
2. Ứng dụng chính của mô hình UNET là gì?
 

<b>(A)</b> . Phân loại văn bản	<b>(B)</b> . Phân đoạn hình ảnh
<b>(C)</b> . Nhận dạng giọng nói	<b>(D)</b> . Dự đoán giá cổ phiếu
  
3. Nếu không dùng skip connection trong U-Net có được không? Skip connection trong model U-Net có tác dụng gì?
  - (A) Được. Không có tác dụng gì
  - (B) Không. Giúp mô hình chạy được
  - (C)** Được. Nó giúp ảnh thu được nhiều thông tin hơn
  - (D) Không. Nó giúp ảnh thu được không thay đổi gì
  
4. Điều gì làm cho U-Net khác biệt với Mạng Convolutional Neural Network (CNN) thông thường được sử dụng để phân loại hình ảnh?
  - (A)**. U-Net có nhánh upsampling kết hợp với các skip connection
  - (B). U-Net không sử dụng các lớp convolutional
  - (C). U-Net chỉ được sử dụng cho ảnh grayscale
  - (D). U-Net chỉ được sử dụng cho ảnh màu
  
5. Phương pháp nào trong các phương pháp sau đây thực hiện nội suy tuyến tính tuần tự trên hai trục của hình ảnh và có khả năng tạo ra kết quả tốt hơn so với Nearest-neighbor Interpolation?

- (A). Bilinear Interpolation  
(C). Nearest-neighbor Interpolation  
(B). Bicubic Interpolation  
(D). Trilinear Interpolation
6. Super-resolution dùng để làm gì?  
(A). Phân loại hình ảnh  
(C). Làm mờ hình ảnh  
(B). Tăng độ phân giải hình ảnh  
(D). Tạo hình ảnh 3D
7. Trong quá trình huấn luyện super resolution model, data thường được tạo thành các cặp ảnh như?  
(A). Hình ảnh màu và hình ảnh đen trắng  
(B). Cặp ảnh thuộc 2 class khác nhau  
(C). Hình ảnh độ phân giải cao và hình ảnh độ phân giải thấp  
(D). Tất cả đều đúng
8. Vấn đề phổ biến nào có thể xảy ra trong các mạng deep learning mà kiến trúc của U-Net giúp giảm thiểu? ?  
(A). Overfitting  
(B). Underfitting  
(C). Mất thông tin không gian trong quá trình downsampling  
(D). Độ phức tạp tính toán
9. Mục tiêu chính của image inpainting là gì??  
(A). Nén ảnh mà không làm giảm đáng kể chất lượng  
(B) Tăng cường độ tương phản và màu sắc của ảnh  
(C) Phát hiện và xóa các đối tượng khỏi ảnh  
(D) Điền vào các vùng bị thiếu hoặc bị hỏng của ảnh bằng nội dung phù hợp
10. Ưu điểm của việc sử dụng phương pháp deep learning cho inpainting so với các phương pháp truyền thống là gì? ?  
(A). Các phương pháp deep learning luôn nhanh hơn.  
(B). Các phương pháp deep learning không yêu cầu bất kỳ dữ liệu huấn luyện nào  
(C). Các phương pháp deep learning xử lý tốt hơn các vùng khuyết thiếu lớn và texture phức tạp.  
(D). Các phương pháp deep learning đơn giản hơn để triển khai

## Phần IV: Phụ Lục

1. **Hint:** Các bạn được khuyến khích nên tham khảo code hướng dẫn (link bên dưới) trước. Sau đó tự xây dựng pipeline của riêng mình. Link data đã bao gồm trong file code. Các bạn lưu ý điều chỉnh batch size phù hợp với phần cứng mà mình có :

- [Link Bài tập 1](#)
- [Link Bài tập 2](#)
- [Link Bài tập 3](#)

2. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể tải tại [Link](#) ([Lưu ý](#) Sáng thứ 3 khi hết deadline phần bài tập ad mới copy các nội dung bài giải nêu trên vào đường dẫn)

### 3. Rubric:

Domain Conversion - Rubric		
Câu	Kiến Thức	Đánh Giá
1	<ul style="list-style-type: none"> <li>- Ôn tập về kiến trúc Unet</li> <li>- Ôn tập về các thành phần strong kiến trúc Unet, encoder, decoder, upsampling technique</li> <li>- Các ứng dụng của kiến trúc Unet</li> </ul>	<ul style="list-style-type: none"> <li>- Hiểu về khái niệm vanishing gradient</li> <li>- Có thể tự code và custom các thành phần trong kiến trúc Unet như encoder, decoder, upsampling</li> <li>- Có thể kết hợp và custom các thành phần để hình thành nên kiến trúc Unet-theo từng task với yêu cầu cụ thể.</li> </ul>
2	<ul style="list-style-type: none"> <li>- Khái niệm về Image super resolution</li> <li>- Tự tạo data dùng cho Image super resolution</li> <li>- Ôn tập và hiểu thêm tầm quan trọng của skip connection trong xây dựng model cho Image super resolution</li> <li>- Cách xây dựng và lựa chọn model trong Image super resolution task</li> </ul>	<ul style="list-style-type: none"> <li>- Xây dựng được model (dựa trên kiến trúc Unet) cơ bản cho task Image super resolution</li> <li>- Biết cách code và linh hoạt tinh chỉnh skip connection trong Unet</li> <li>- Biết cách can thiệp vào model để chỉnh sửa theo yêu cầu của Image super resolution task dựa vào kiến trúc Unet</li> </ul>
3	<ul style="list-style-type: none"> <li>- Khái niệm về Image inpainting</li> <li>- Tự tạo data dùng cho Image inpainting</li> <li>- Ôn tập và hiểu thêm tầm quan trọng của skip connection trong xây dựng model cho Image inpainting</li> <li>- Cách xây dựng và lựa chọn model trong Image inpainting task</li> </ul>	<ul style="list-style-type: none"> <li>- Xây dựng được model (dựa trên kiến trúc Unet) cơ bản cho task Image inpainting</li> <li>- Biết cách code và linh hoạt tinh chỉnh skip connection trong Unet</li> <li>- Biết cách can thiệp vào model để chỉnh sửa theo yêu cầu của Image inpainting task dựa vào kiến trúc Unet</li> <li>- Biết cách áp dụng chiến thuật pretrain cho Image inpainting</li> </ul>

- **Hết** -