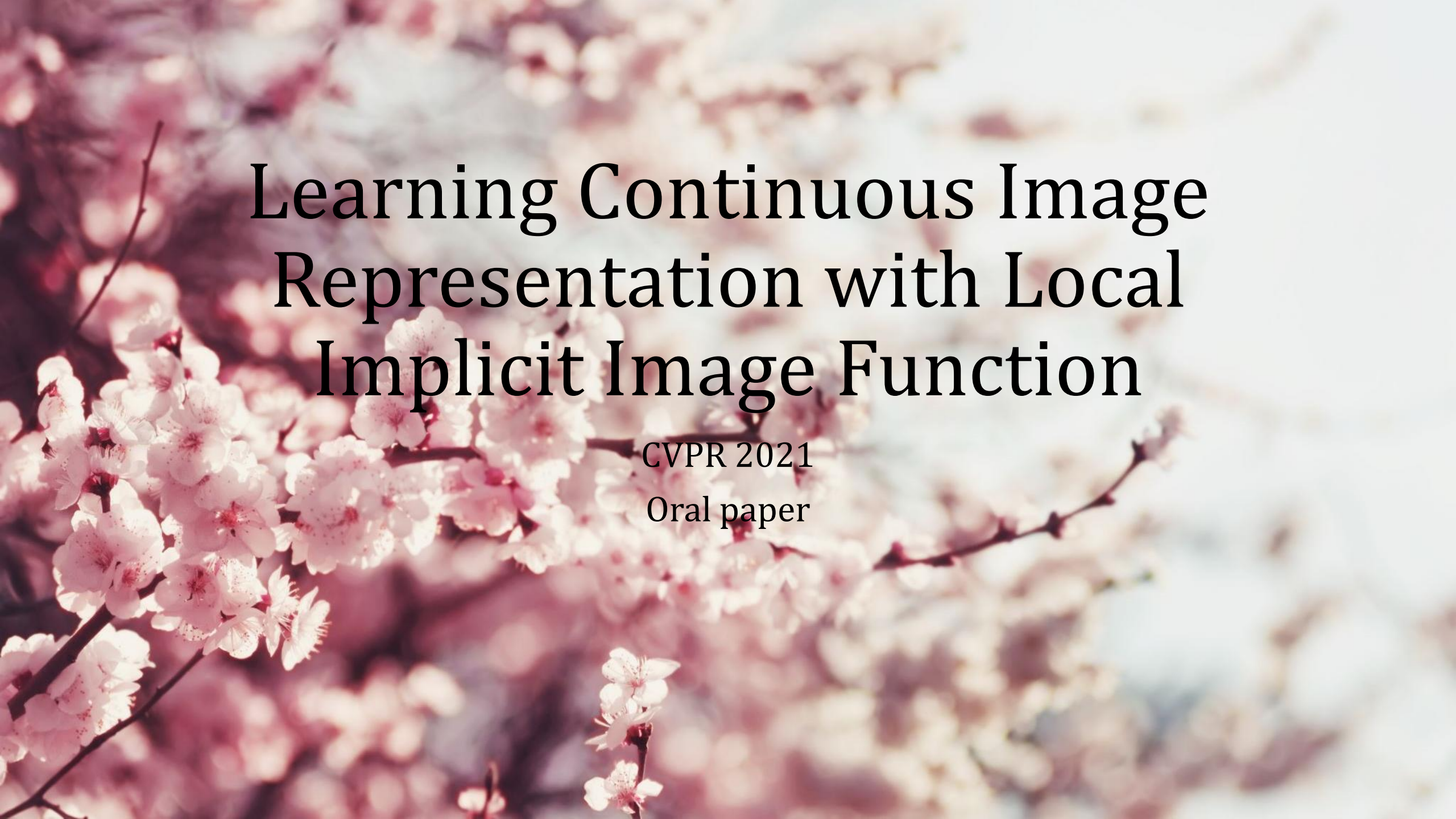


Learning Continuous Image Representation with Local Implicit Image Function

CVPR 2021

Oral paper

간략 소개

- 문제
  - How to represent an image?
    1. Visual world
       - 연속적임.
         - Our visual world is continuous.
    2. Machine
       - 이산적으로 표현함.
         - Represent the images as 2D arrays of pixels.
  - 이 둘 사이의 gap 차이 발생.
    - 예를 들어, 서로 다른 해상도를 가진 정답 영상이 주어졌을 때, 학습을 하기 위해서는 두 영상을 같은 사이즈로 줄여야 하는데 이 때 정보의 손실이 발생함.

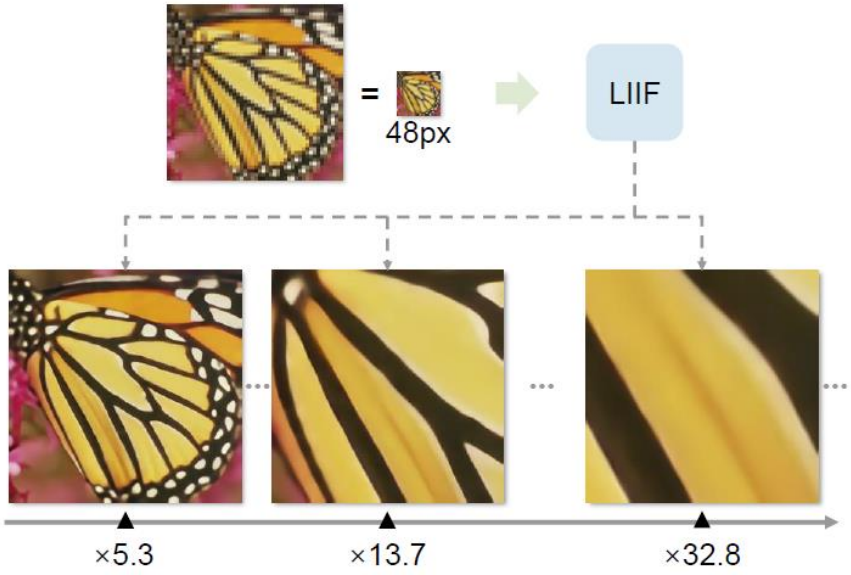


Figure 1: Local Implicit Image Function (LIIF) represents an image in continuous domain, which can be presented in arbitrary high resolution.

- 제안하는 방식
  - 영상을 고정된 해상도에서 표현하는 대신, 영상을 연속적인 domain 에 정의된 함수로 모델링을 통하여 연속적인 표현 방식을 연구함.
    - We propose to study a continuous representation for images. By modeling an image as a function defined in a continuous domain, we can restore and generate image in arbitrary resolution if needed.
- 효과
  - 이를 통해서, 원하는 무작위의 해상도로 영상을 생성 할 수 있음.
    - We can restore and generate the image in arbitrary resolution if needed.

핵심 아이디어

- How do we represent an image as a continuous function?
  - Inspired by Implicit neural representation.

The key idea of implicit neural representation is to represent an object as a function that maps coordinates to the corresponding signal, where the function is parameterized by a deep neural network.
- Implicit neural representation 은 deep neural network 에 의하여 학습된 함수로 이 함수를 통하여 좌표계를 이에 대응되는 신호로 변환 할 수 있음.

    - 하지만 주로, 3D shape reconstruction 에서 자주 사용되었으며 기존 방식들은 숫자와 같이 단순한 영상은 표현에 성공 하였지만 충실도가 높은(high fidelity) 자연계의 영상을 표현하는데 실패하였음.
- 논문의 저자들은 복잡한 자연계 영상을 연속적으로 표현하는 방식인 Local Implicit Image Function(LIIF) 를 제안함.

Implicit neural representation(Coordinate-based representation) 이란?

The key idea of implicit neural representation is to represent an object as a function that maps coordinates to the corresponding signal, where the function is parameterized by a deep neural network.

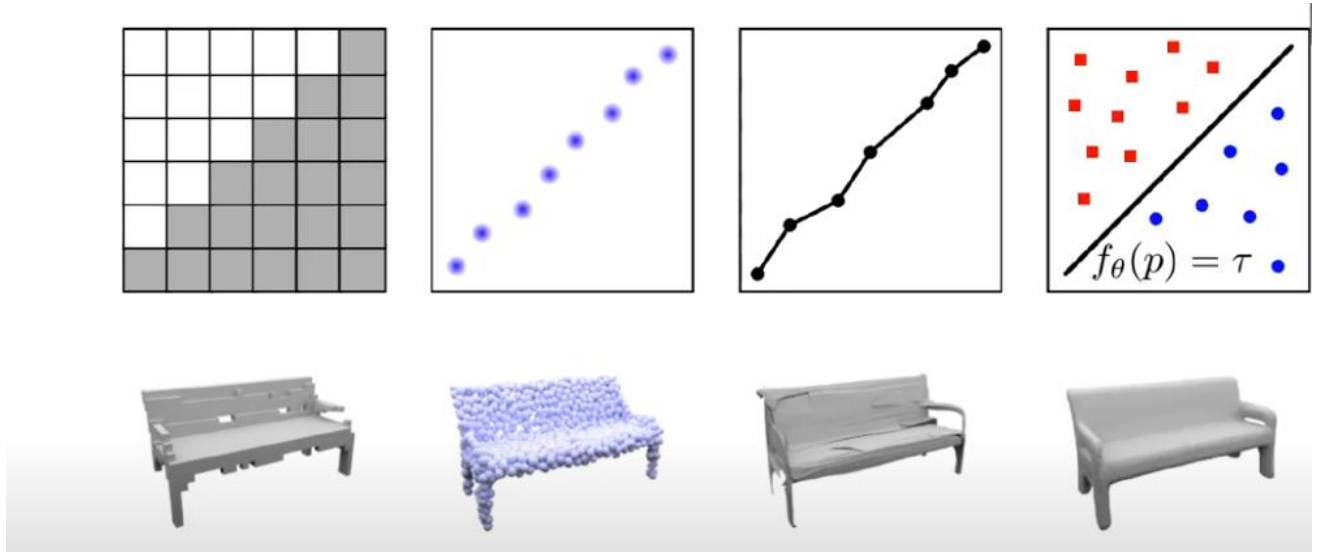
- Implicit neural representation 은 deep neural network 에 의하여 학습된 함수로 이 함수를 통하여 좌표계를 이에 대응되는 신호로 변환 할 수 있음.

- 모든 종류의 신호를 parameterize 하는 새로운 방식임.
  - Implicit Neural Representation(coordinate-based representations) are a novel way to parameterize signals of all kinds.
- 전통적인 signal representations 은 보통 **이산적임**.
  - 예를 들어, 영상 – 이산적인 픽셀 grid 에서 표현, 음성 신호 – 이산적인 amplitude 의 sample 들로 표현, 3D shape – voxels, point clouds, meshes 의 이산적인 grid 에서 표현 됨.
- 반면, Implicit Neural Representations 은 신호를 **연속적인 함수로 매개변수화(parameterize a signal as a continuous function)** 하며 **continuous function** 은 신호 domain 을 해당 좌표에 있는 모든 것에 mapping 함.
  - Implicit Neural Representations parameterize a signal as a continuous function that maps the domain of the signal (i.e., a coordinate, such as a pixel coordinate for an image) to whatever is at that coordinate (for an image, an R,G,B color).
- 실제로, 이러한 함수는 분석적으로 다루기 어려움.
  - 자연적인 영상 신호를 수학적인 공식으로 매개변수화 하는 함수를 기록하는 것은 불가능함.
    - It is impossible to "write down" the function that parameterizes a natural image as a mathematical formula.
- 따라서, Implicit Neural Representations 가 **neural network 을 통해서 함수를 근사화** 함.
  - Implicit Neural Representations thus approximate that function via a neural network.

https://github.com/vsitzmann/awesome-implicit-representations

전통적인 representation 방식 vs Implicit representation 방식

- E.g., 3D Representations



- Traditional Explicit Representations
  - Discrete
- Implicit Neural Representations
  - Continuous

https://www.youtube.com/watch?v=3uodIr6Ho24

Implicit neural representation 이 주목받는 이유

- 장점
  1. 연속 함수이기 때문에 영상이 픽셀 수에 결합되는 방식과 같이 **더 이상 공간 해상도와 결합되지 않음.**
     - 따라서, 신호를 parametrize 하기 위하여 필요한 메모리는 공간 해상도에 독립적이며 내제된 신호의 복잡도에 따라 scale 됨.
  2. **무한한 해상도를 가질 수 있음.**
     - 따라서, 무작위의 공간 해상도에서 sampling 될 수 있음.
- Application
  - Super-resolution, or in parameterizing signals in 3D and higher dimensions, where memory requirements grow intractably fast with spatial resolution.
- 쟁점
  - 핵심 쟁점은 이러한 표현 공간에서 직접적으로 작동하는 알고리즘에 있음.
    - The key promise of implicit neural representations lie in algorithms that directly operate in the space of these representations.
  - 다시 말하면, implicit representations 에 의하여 표현되는 영상 속에서 작동하는 neural network 과 동일한 convolution neural network 는 무엇인가?
    - What's the convolutional neural network equivalent of a neural network operating on images represented by implicit representations?
  - 이러한 질문은 공간 해상도와 무관한 알고리즘 class 로 가는 길을 제공함.

https://github.com/vsitzmann/awesome-implicit-representations

Implicit neural representation 이 주목받는 이유

- 좀 더 쉽게 풀어 말하면..
  - 영상을 비롯한 각종 데이터를 신경망으로 나타내는 것을 목표로 하며, 기존의 컴퓨터 비전 연구에서 영상을 다룰 때, 보통 픽셀의 위치마다 RGB 값을 가지는 하나의 행렬로 표현하는 것이 일반적이었음. 하지만, 결과적으로 영상의 해상도에 따라 저장에 필요한 용량과 영상을 다룰 수 있는 모델의 크기가 좌우되었음. 하지만, 요새는 영상을 **픽셀 위치인 (x, y) 좌표에서 (r, g, b) 값으로 변환해 주는 하나의 함수로 표현하는 아이디어**가 떠오르고 있음. 이 **함수를 신경망으로 표현해 학습**하는 것이 Implicit neural representations 의 핵심 아이디어임. 영상을 이렇게 함수로 표현하면 영상을 해상도와 관계 없이 저장하고 컴퓨터 비전 모델에 적용할 수 있게 됨. 뿐만 아니라, 영상을 고화질로 변환해주는 초해상도 기술도 매우 단순해 질 수 있음.

https://eiric.or.kr/manpower/rising_view.php?Seq=21

Local Implicit Image Function(LIIF)

- 핵심 관점
  - LIIF 에서 영상은 공간 차원에서 분포된 latent code 들의 집합으로 표현 할 수 있음.
    - In LIIF, an image is represented as a set of latent codes distributed in spatial dimensions.
- 입출력
  - 입력
    1. 좌표계 정보
       - The coordinate information.
    2. 좌표계 주변의 local latent codes.
       - Queries the local latent codes around the coordinate.
  - 출력
    - 주어진 좌표계의 RGB 값을 출력.
      - The predicts the RGB value at the given coordinate as inputs.
    - 이때, 좌표는 연속적인 값이 주어질 수 있기 때문에 무작위의 해상도를 가지는 영상이 출력됨.

Latent code 를 LIIF 의 입력으로 넣어주는 이유임.

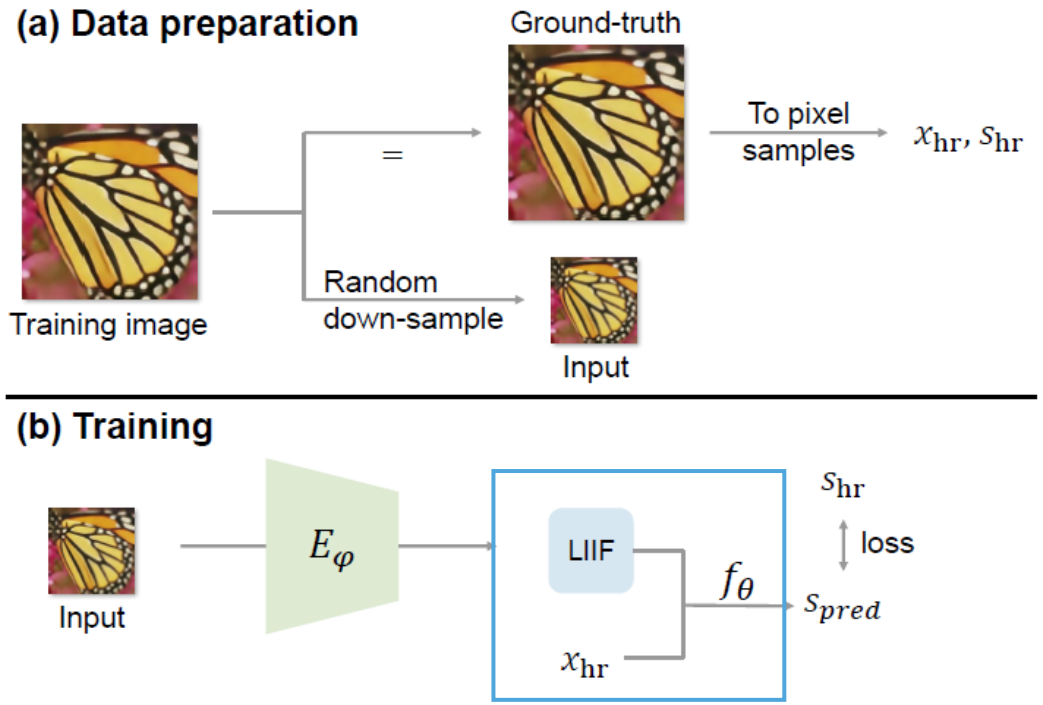


Figure 4: **Learning to generate continuous representation for pixel-based images.** An encoder is jointly trained with the LIIF representation in a self-supervised super-resolution task, in order to encourage the LIIF representation to maintain high fidelity in higher resolution.

Encoder

- 목표
  - 픽셀 값으로 표현된 영상에서 연속적인 표현(continuous representation) 을 생성하는 것을 목표로 함.
    - Why? 생성된 연속적인 표현이 입력 영상보다 더 높은 정확도로 일반화 할 수 있기 때문임.
      - Since we hope the generated continuous representation can generalize to higher precision than the input image.
    - LIIF 의 representation 으로서 영상에서 2D feature map 을 mapping 함.
      - The encoder maps the input image to a 2D feature map as its LIIF representation.
- 학습 방법
  - LIIF representation 과 self-supervised 방식으로 같이 학습.

- 전체적인 학습 과정 동안 제공되는 입력과 정답 영상은 up-sampling scale 을 연속적으로 변화시키면서 제공됨.
  - 따라서, LIIF 는 서로 다른 resolution 에 제공되는 정보를 exploit 할 수 있음.
  - 정답 영상들을 모두 같은 size 로 resize 할 필요 없음.

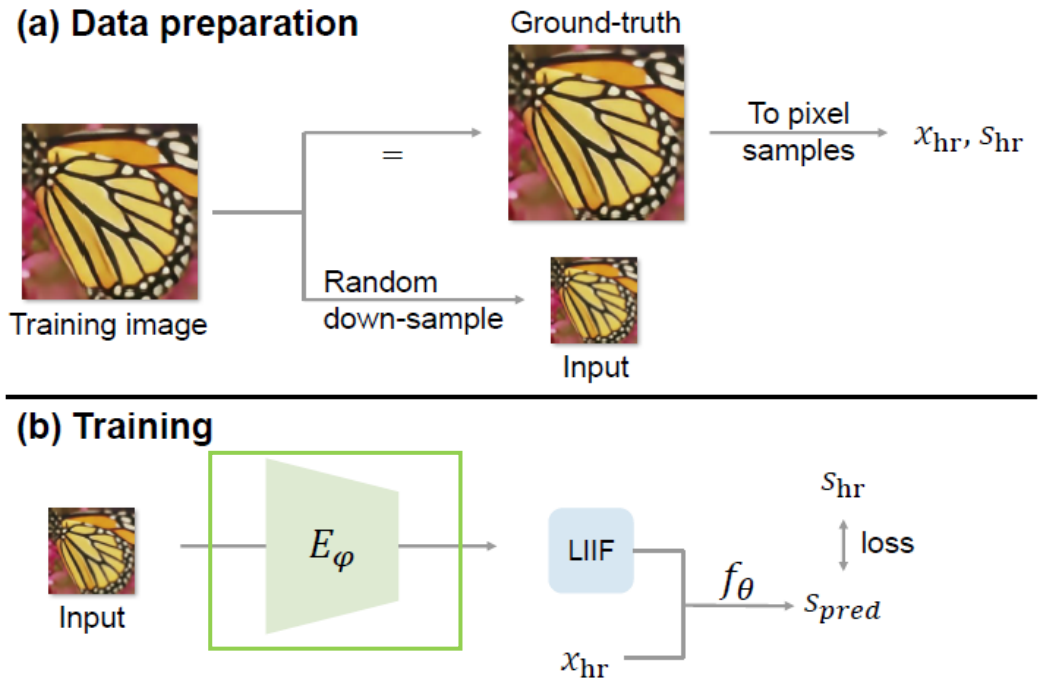


Figure 4: **Learning to generate continuous representation for pixel-based images.** An encoder is jointly trained with the LIIF representation in a self-supervised super-resolution task, in order to encourage the LIIF representation to maintain high fidelity in higher resolution.

Encoder

- Baseline
  - EDSR

```yaml
model:
  name: liif
  args:
    encoder_spec:
      name: edsr-baseline
      args:
        no_upsampling: true
```

```
    self.encoder = models.make(encoder_spec)

    if imnet_spec is not None:
        imnet_in_dim = self.encoder.out_dim
        if self.feat_unfold:
            imnet_in_dim *= 9
        imnet_in_dim += 2 # attach coord
        if self.cell_decode:
            imnet_in_dim += 2
        self.imnet = models.make(imnet_spec, args={'in_dim': imne
    else:
        self.imnet = None

def gen_feat(self, inp):
    self.feat = self.encoder(inp)
    return self.feat
```

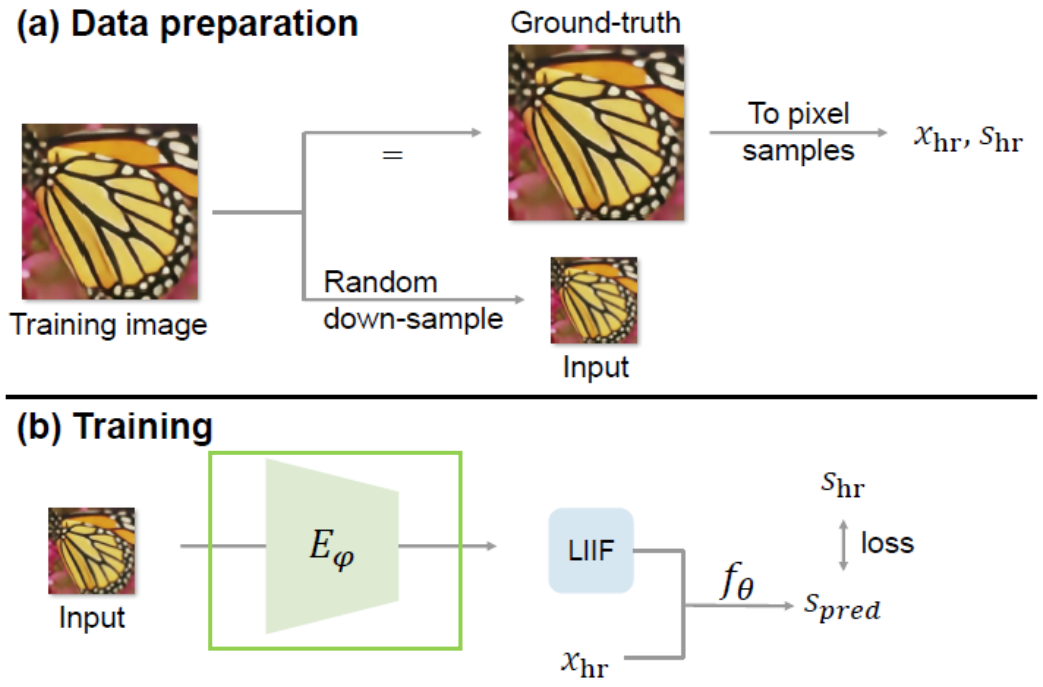


Figure 4: **Learning to generate continuous representation for pixel-based images.** An encoder is jointly trained with the LIIF representation in a self-supervised super-resolution task, in order to encourage the LIIF representation to maintain high fidelity in higher resolution.

Learning with size-varied GT

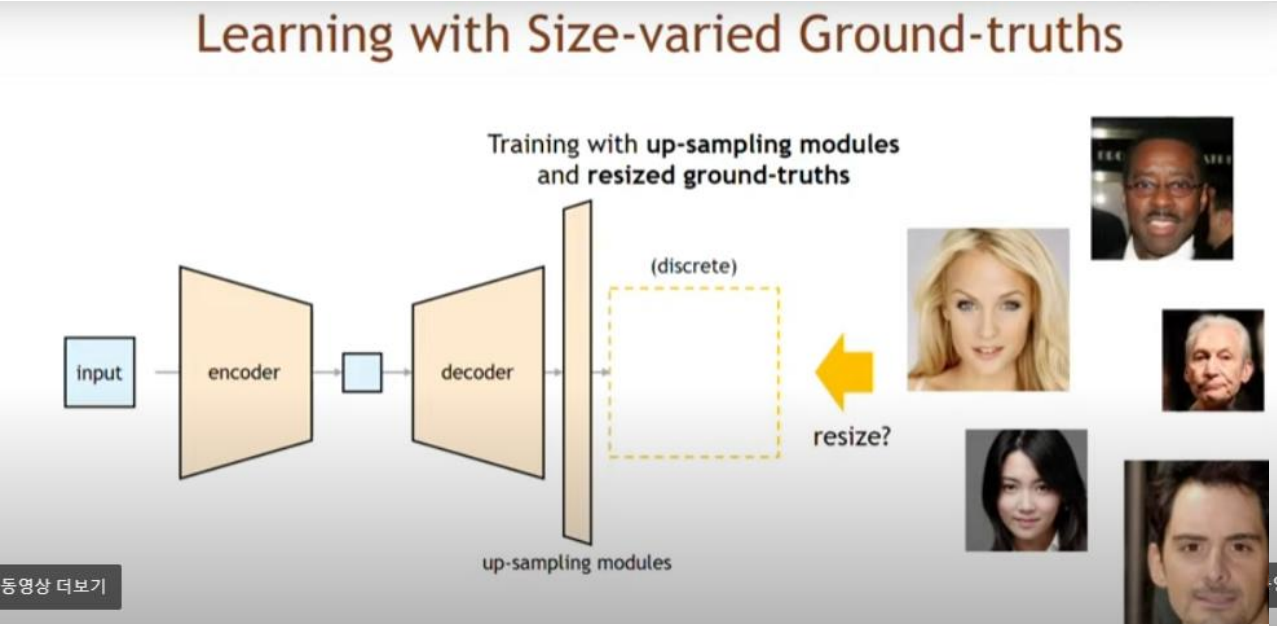


그림. 기존 방식

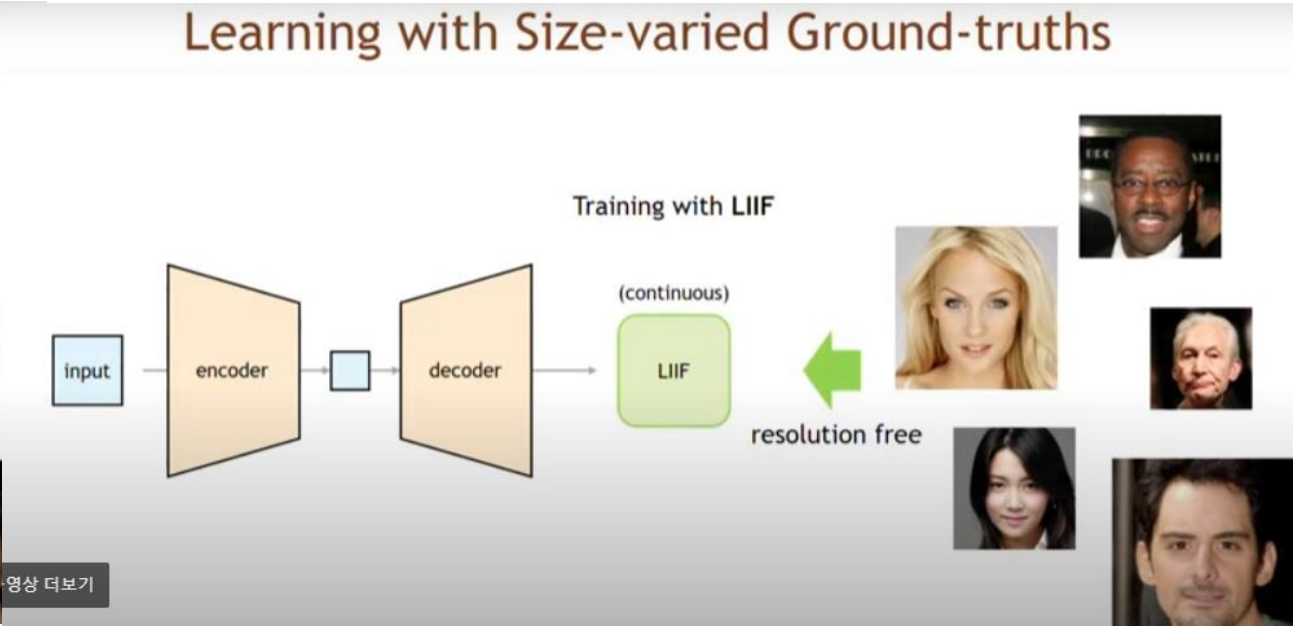


그림. LIIF 의 학습 방식

- 전체적인 학습 과정 동안 제공되는 입력과 정답 영상은 up-sampling scale 을 연속적으로 변화시키면서 제공됨.
  - 따라서, LIIF 는 서로 다른 resolution 에 제공되는 정보를 exploit 할 수 있음.
  - 정답 영상들을 모두 같은 size 로 resize 할 필요 없음.

제안하는 방식

- 주목해야 할 점은 f_θ 가 continuous representation 로서 encoder 의 출력 feature map 의 해당 좌표 혹은 주변 좌표에 해당하는 vector 를 입력으로 받는다는 점임.
- MLP 로 함수, f_θ 의 parameter 가 학습됨.

LIIF 에서 연속적인 영상 I 는 2D feature map M 으로 표현 될 수 있음.
In LIIF, each continuous image I is represented as a 2D feature map M.

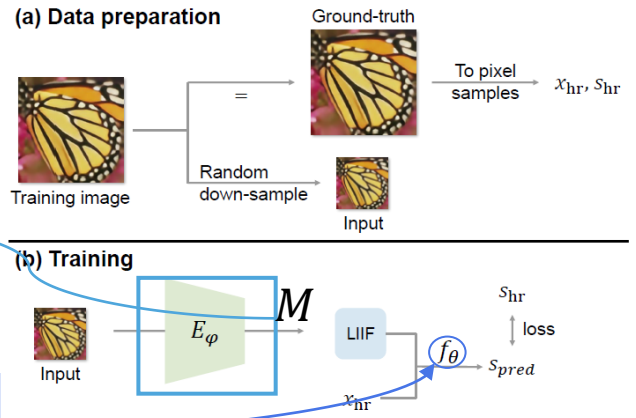


Figure 4: **Learning to generate continuous representation for pixel-based images.** An encoder is jointly trained with the LIIF representation in a self-supervised super-resolution task, in order to encourage the LIIF representation to maintain high fidelity in higher resolution.

Decoding function, f_θ

- Local Implicit Image Function $s = f_\theta(x, z)$
  - 입력
    1. $x(x_{hr})$
       - 영상(고해상도) domain 내에 있는 2D coordinate.
    2. z
       - encoder 에서 나온 2D feature map, $M \in \mathbb{R}^{H \times W \times D}$ 은 2차원의 연속적인 영상 공간 차원에서 균등하게 분포된 $H \times W$ latent code 로 해석 할 수 있음.
         - a continuous image is represented as a 2D feature map $M \in \mathbb{R}^{H \times W \times D}$

         Which is viewed as $H \times W$ latent code evenly spread in the 2D domain.
       - 따라서, M 에서 좌표계를 부여 가능하기 때문에 M 안에 있는 latent code 인 z 가 요구된 좌표계에 위치 혹은 인접한 latent code 벡터로 표현 가능함.
         - 이로 인하여, z 는 연속적인 영상의 지역적인 조각(local piece) 로 표현됨.
           - each latent code, z in M represented a local piece of the continuous image.
         - 결국, 각 local piece of the continuous image, z 는 자기 자신과 가장 가까운 좌표 집합의 신호를 예측하는 역할을 할 수 있음.
           - it is responsible for predicting the signal of the set of coordinates that are closest to itself.
  - 출력
    - RGB 신호, s 를 예측함.

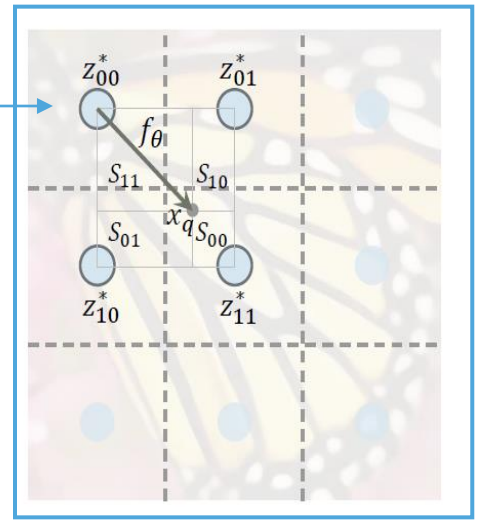


Figure 2: **LIIF representation with local ensemble.** A continuous image is represented as a 2D feature map with a decoding function f_θ shared by all the images. The signal is predicted by ensemble of the local predictions, which guarantees smooth transition between different areas.

RGB values. We assume the $H \times W$ feature vectors (we call them latent codes from now on) of $M^{(i)}$ are evenly distributed in the 2D space of the continuous image domain of $I^{(i)}$ (e.g. blue circles in Figure 2), then we assign a 2D co-

제안하는 방식

- Local Implicit Image Function $s = f_\theta(x, z)$
  - 모델 architecture

```yaml
imnet_spec:
  name: mlp
  args:
    out_dim: 3
    hidden_list: [256, 256, 256, 256]
```

```python
@register('mlp')
class MLP(nn.Module):

    def __init__(self, in_dim, out_dim, hidden_list):
        super().__init__()
        layers = []
        lastv = in_dim
        for hidden in hidden_list:
            layers.append(nn.Linear(lastv, hidden))
            layers.append(nn.ReLU())
            lastv = hidden
        layers.append(nn.Linear(lastv, out_dim))
        self.layers = nn.Sequential(*layers)

    def forward(self, x):
        shape = x.shape[:-1]
        x = self.layers(x.view(-1, x.shape[-1]))
        return x.view(*shape, -1)
```

제안하는 방식

- Local Implicit Image Function
  - $s = f_\theta(x, z)$ 의 의미
    - $f_\theta(\cdot, z)$ 는 neural implicit function 으로 좌표계 domain $x \in \mathcal{X}$ 을 RGB domain s $\in \mathcal{S}$으로 mapping 시키는 함수.
  - 표현
    - 연속적인 영상, I 에 대하여 x_q 좌표 에서 RGB 값은 다음과 같이 정의됨.

$$I(x_q) = f_\theta(z^*, x_q - v^*)$$

    - $x_q = query\ coordinate$
    - $z^* = M$ 에 존재하는 x_q 로 부터 가장 가까운 latent code. (is nearest latent code from x_q in M.)
      - 현재의 그림에서는 z^* 는 z^*_{11} 에 대응됨.
    - v^* =영상 domain 에서 latent code 의 좌표 값.
    - $I(x_q)$ = 연속적인 영상에서 임의의(real value 가능) 좌표계에 대한 RGB value.
  - 입력된 좌표 값, x 과 latent code, z 를 RGB value 로 출력시키는 함수인 f_θ는 영상 전반에 걸쳐서 공유되며, 출력된 연속적인 영상은 2D feature map, M 으로 표현되며 이는 2D domain 에 있는 latent code 들로 해석 할 수 있음.

As a summary, with a function f_θ shared by all the images, a continuous image is represented as a 2D feature map $M^{(i)} \in \mathbb{R}^{H \times W \times D}$ which is viewed as $H \times W$ latent codes evenly spread in the 2D domain. Each latent code z in $M^{(i)}$

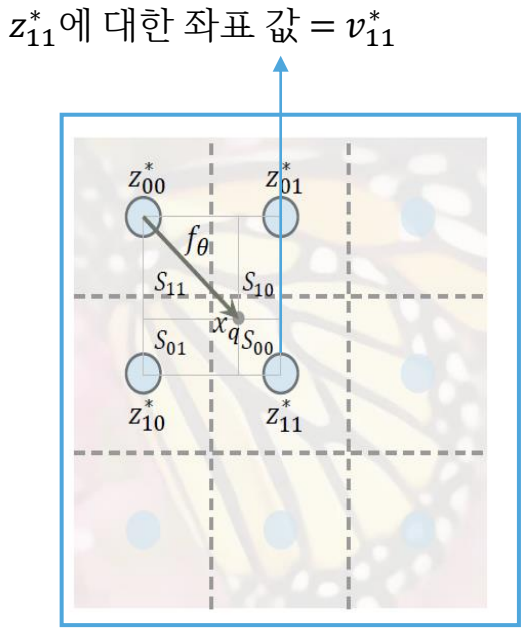


Figure 2: **LIIF representation with local ensemble.** A continuous image is represented as a 2D feature map with a decoding function f_θ shared by all the images. The signal is predicted by ensemble of the local predictions, which guarantees smooth transition between different areas.

제안하는 방식

- Local Implicit Image Function

```
# field radius (global: [-1, 1])
rx = 2 / feat.shape[-2] / 2
ry = 2 / feat.shape[-1] / 2

feat_coord = make_coord(feat.shape[-2:], flatten=False).cuda() \
    .permute(2, 0, 1) \
    .unsqueeze(0).expand(feat.shape[0], 2, *feat.shape[-2:])
```

```
preds = []
areas = []
for vx in vx_lst:
    for vy in vy_lst:
        coord_ = coord.clone()
        coord_[:, :, 0] += vx * rx + eps_shift
        coord_[:, :, 1] += vy * ry + eps_shift
        coord_.clamp_(-1 + 1e-6, 1 - 1e-6)
        q_feat = F.grid_sample(
            feat, coord_.flip(-1).unsqueeze(1),
            mode='nearest', align_corners=False)[:, :, 0, :] \
            .permute(0, 2, 1)
        q_coord = F.grid_sample(
            feat_coord, coord_.flip(-1).unsqueeze(1),
            mode='nearest', align_corners=False)[:, :, 0, :] \
            .permute(0, 2, 1)
        rel_coord = coord - q_coord
        rel_coord[:, :, 0] *= feat.shape[-2]
        rel_coord[:, :, 1] *= feat.shape[-1]
        inp = torch.cat([q_feat, rel_coord], dim=-1)

        if self.cell_decode:
            rel_cell = cell.clone()
            rel_cell[:, :, 0] *= feat.shape[-2]
            rel_cell[:, :, 1] *= feat.shape[-1]
            inp = torch.cat([inp, rel_cell], dim=-1)

        bs, q = coord.shape[:2]
        pred = self.imnet(inp.view(bs * q, -1)).view(bs, q, -1)
        preds.append(pred)

        area = torch.abs(rel_coord[:, :, 0] * rel_coord[:, :, 1])
        areas.append(area + 1e-9)
```

제안하는 방식

- Local Implicit Image Function

```python
def make_coord(shape, ranges=None, flatten=True):
    """ Make coordinates at grid centers.
    """
    coord_seqs = []
    for i, n in enumerate(shape):
        if ranges is None:
            v0, v1 = -1, 1
        else:
            v0, v1 = ranges[i]
        r = (v1 - v0) / (2 * n)
        seq = v0 + r + (2 * r) * torch.arange(n).float()
        coord_seqs.append(seq)
    ret = torch.stack(torch.meshgrid(*coord_seqs), dim=-1)
    if flatten:
        ret = ret.view(-1, ret.shape[-1])
    return ret
```

TORCH.MESHGRID

`torch.meshgrid(*tensors)` [SOURCE]

Take N tensors, each of which can be either scalar or 1-dimensional vector, and create N N-dimensional grids, where the i^{th} grid is defined by expanding the i^{th} input over dimensions defined by other inputs.

Parameters

tensors (*list of Tensor*) – list of scalars or 1 dimensional tensors. Scalars will be treated as tensors of size $(1,)$ automatically

Returns

If the input has k tensors of size $(N_1,), (N_2,), \ldots, (N_k,)$, then the output would also have k tensors, where all tensors are of size $(N_1, N_2, \ldots, N_k)$.

Return type

seq (sequence of Tensors)

Example:

```
>>> x = torch.tensor([1, 2, 3])
>>> y = torch.tensor([4, 5, 6])
>>> grid_x, grid_y = torch.meshgrid(x, y)
>>> grid_x
tensor([[1, 1, 1],
        [2, 2, 2],
        [3, 3, 3]])
>>> grid_y
tensor([[4, 5, 6],
        [4, 5, 6],
        [4, 5, 6]])
```

https://pytorch.org/docs/stable/generated/torch.meshgrid.html

제안하는 방식

- Local Implicit Image Function – feature unfolding
  - 목표
    - M 에 있는 각 latent code 에 포함된 정보를 풍부하게 하는 것이 목표임.
  - 방법
    - Feature unfolding 을 적용하여 M 에서 $\widehat{M}$ 을 얻어냄.
      - Feature unfolding
        - M 에서 3×3 으로 인접한 latent codes 을 concatenation 는 방식으로 다음과 같이 표현됨.

$$\widehat{M}_{j,k} = Concat(\{M_{j+l,k+m}\}_{l,m\in\{-1,0,1\}})$$

F.unfold

- 목표
  - Extract sliding local blocks from a batched input tensor.
- Torch.nn.Unfold(kernel_size, dilation=1, padding=0, stride=1)
  - Input: $(N, C, *)$
  - Output: $(N, C \times \prod(\text{kernel\_size}), L)$ as described above

Consider a batched `input` tensor of shape $(N, C, *)$, where N is the batch dimension, C is the channel dimension, and $*$ represent arbitrary spatial dimensions. This operation flattens each sliding `kernel_size`-sized block within the spatial dimensions of `input` into a column (i.e., last dimension) of a 3-D `output` tensor of shape $(N, C \times \prod(\text{kernel\_size}), L)$, where $C \times \prod(\text{kernel\_size})$ is the total number of values within each block (a block has $\prod(\text{kernel\_size})$ spatial locations each containing a C-channeled vector), and L is the total number of such blocks:

$$L = \prod_d \left\lfloor \frac{\text{spatial\_size}[d] + 2 \times \text{padding}[d] - \text{dilation}[d] \times (\text{kernel\_size}[d] - 1) - 1}{\text{stride}[d]} + 1 \right\rfloor,$$

https://bowenroom.github.io/myBlog/pytorch/fastai2/2020/06/01/torch-unfold.html

F.unfold

Examples:

```
>>> unfold = nn.Unfold(kernel_size=(2, 3))
>>> input = torch.randn(2, 5, 3, 4)
>>> output = unfold(input)
>>> # each patch contains 30 values (2x3=6 vectors, each of 5 channels)
>>> # 4 blocks (2x3 kernels) in total in the 3x4 input
>>> output.size()
torch.Size([2, 30, 4])
```

```
In [37]: input = torch.arange(8).float()# (2, 5, 3, 4)

In [38]: input = input.view(1,1,2,4)

In [43]: unfold = nn.Unfold(kernel_size=(2, 2))

In [44]: output = unfold(input)

In [45]: input

tensor([[[[0., 1., 2., 3.],
          [4., 5., 6., 7.]]]])

In [46]: output

tensor([[[0., 1., 2.],
         [1., 2., 3.],
         [4., 5., 6.],
         [5., 6., 7.]]])
```

https://bowenroom.github.io/myBlog/pytorch/fastai2/2020/06/01/torch-unfold.html

제안하는 방식

- Local Implicit Image Function – feature unfolding

```
gen_feat(self, inp):
self.feat = self.encoder(inp)
return self.feat

query_rgb(self, coord, cell=None):
feat = self.feat
```

```
if self.feat_unfold:
    feat = F.unfold(feat, 3, padding=1).view(
        feat.shape[0], feat.shape[1] * 9, feat.shape[2], feat.shape[3
```

제안하는 방식

- Local Implicit Image Function – local ensemble
  - $I(x_q) = f_\theta(z^*, x_q - v^*)$ 으로 연속적인 영상, continuous image 를 예측할 때 발생하는 문제점
    - Discontinuous prediction
      - x_q 의 움직임에 따른 z^* 에 대한 선택 변화에 의하여 연속적이지 않은 이산적인 값이 예측됨.
        - 2 차원 영상 좌표에 있는 x_q 가 움직일 때, x_q 에서 가장 인접한 latent code 인 z^* 에 대하여 query 하면서 신호의 예측이 이루어 지기 때문에 z^* 에 대한 선택도 갑자기 바뀔 수 있음. (the selection of nearest latent code changes)
        - 그림과 같이, x_q 가 회색 선을 따라 교차 하는 경우를 가정해 보자.
          - z^* 의 선택이 바뀌는 좌표계 근처에서, 매우 가까운 두개의 신호는 서로 다른 latent 에서$(z^*_{00} \rightarrow z^*_{11})$ 선택될 것임.
            - Around those coordinates where the selection of z^* switches, the signal of two infinitely close coordinates will be predicted from different latent codes.

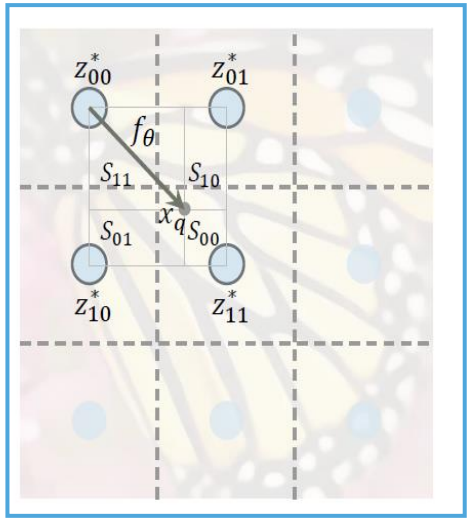


Figure 2: **LIIF representation with local ensemble.** A continuous image is represented as a 2D feature map with a decoding function f_θ shared by all the images. The signal is predicted by ensemble of the local predictions, which guarantees smooth transition between different areas.

제안하는 방식

- Local Implicit Image Function – local ensemble
  - 해결 방법
    - local ensemble
      - 식 $I(x_q) = f_\theta(z^*, x_q - v^*)$ 를 다음과 같이 좀 더 넓은 grid 에서 보도록 확장함.

$$I(x_q) = \sum_{t \in \{00,01,10,11\}} \frac{S_t}{S} f_\theta(z_t^*, x_q - v_t^*)$$

- 가장 인접한 하나의 latent code 를 보는 것이 아니라, grid 내에 있는(모든 방향의) 가장 인접한 Latent code 들을 가지고 continuous image 를 예측하도록 하는 것이 핵심임.
  - z_t^* ($t \in \{00,01,10,11\}$) – nearest latent code in top-left, top-right, bottom-left, bottom-right sub-spaces.
  - v_t^* - coordinate of z_t^*.
  - $\frac{S_t}{S}$ 의 의미
    - 각 방향에 대하여 얼마나 가중치를 둘 것인가를 의미함.
    - x_q 에 가까운 방향의 latent code 일수록 더 높은 영향을 주기 때문에 이를 반영하기 위하여 더 높은 가중치를 부여함.
    - How?
      - x_q 에 가까운 방향일 수록, x_q 와 가장 가까운 latent code z^* 의 rectangular area 는 작아지고 반대로 대각 방향으로 가장 먼 latent code z^* 의 rectangular area 는 증가함.
      - x_q 에 가까운 방향일 수록, 높은 가중치를 부여하고 싶기 때문에 각 latent code 에 대한 대각 성분의 rectangular area 에 가중치를 부여함.
      - S_t - area of rectangle between x_q and $v_{t'}^*$ (t' is diagonal to t)
        - E.g., 00 to 11, 10 to 01
      - 전체 사각형의 넓이 $S = \sum_t S_t$ 로 각 대각 성분의 rectangular area 를 정규화 하여 [0,1] 사이의 가중치 값을 생성.

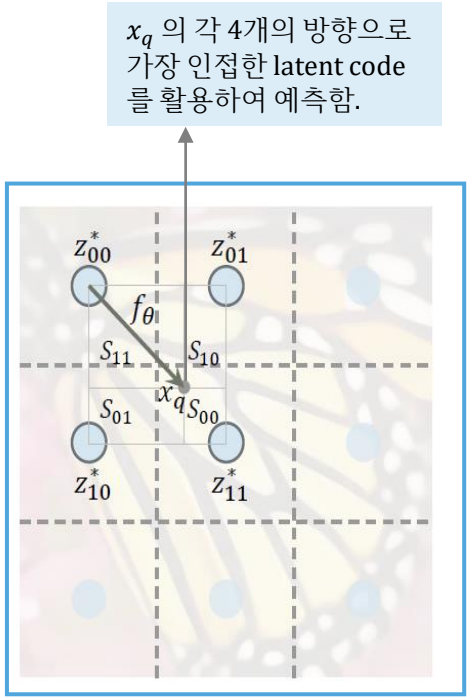


Figure 2: **LIIF representation with local ensemble.** A continuous image is represented as a 2D feature map with a decoding function f_θ shared by all the images. The signal is predicted by ensemble of the local predictions, which guarantees smooth transition between different areas.

제안하는 방식

- Local Implicit Image Function – local ensemble
  - 의미
    - z^* 의 좌표 변화에 대하여 연속적인 변형이 가능해짐.
      - It achieves continuous transition at coordinates where z^* switches.
    - Why?
      - local ensemble 를 통하여 local latent code 들에 의하여 나타나는 local pieces 가 인접한 pieces 와 overlap 이 되도록 하여 각 좌표에서 신호를 독립적으로 예측하기 위한 4가지 latent code 들이 있도록 설계됨.
        - Intuitively, this is to let the local pieces represented by local latent codes overlap with its neighboring pieces so that at each coordinate there are four latent codes for independently predicting the signal.
      - 뿐만 아니라 4가지 latent code 들이 합쳐 질 때, 거리가 가까운 latent code 일수록, 높은 가중치가 부여됨.
        - These four predictions are then merged by voting with normalized confidences, which are proportional to the area of the rectangle between the query point and its nearest latent code's diagonal counterpart, thus the confidence gets higher when the query coordinate is closer.

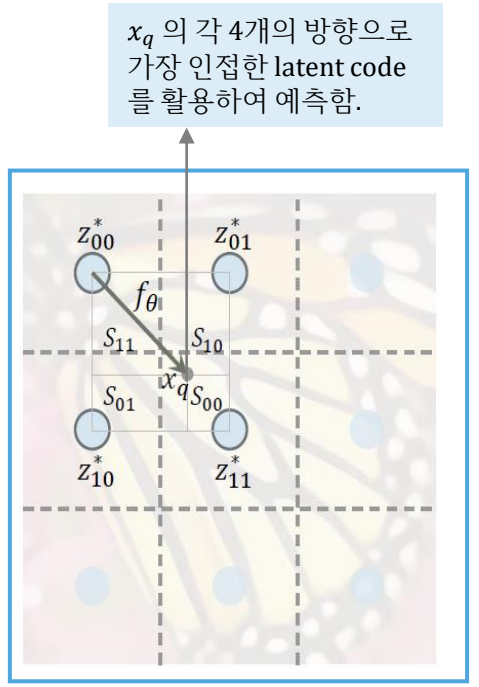


Figure 2: **LIIF representation with local ensemble.** A continuous image is represented as a 2D feature map with a decoding function f_θ shared by all the images. The signal is predicted by ensemble of the local predictions, which guarantees smooth transition between different areas.

제안하는 방식

- Local Implicit Image Function – local ensemble

```
if self.local_ensemble:
    vx_lst = [-1, 1]
    vy_lst = [-1, 1]
    eps_shift = 1e-6
else:
    vx_lst, vy_lst, eps_shift = [0], [0], 0

# field radius (global: [-1, 1])
rx = 2 / feat.shape[-2] / 2
ry = 2 / feat.shape[-1] / 2

feat_coord = make_coord(feat.shape[-2:], flatten=False).cuda() \
    .permute(2, 0, 1) \
    .unsqueeze(0).expand(feat.shape[0], 2, *feat.shape[-2:])
```

```
preds = []
areas = []
for vx in vx_lst:
    for vy in vy_lst:
        coord_ = coord.clone()
        coord_[:, :, 0] += vx * rx + eps_shift
        coord_[:, :, 1] += vy * ry + eps_shift
        coord_.clamp_(-1 + 1e-6, 1 - 1e-6)
        q_feat = F.grid_sample(
            feat, coord_.flip(-1).unsqueeze(1),
            mode='nearest', align_corners=False)[:, :, 0, :] \
            .permute(0, 2, 1)
        q_coord = F.grid_sample(
            feat_coord, coord_.flip(-1).unsqueeze(1),
            mode='nearest', align_corners=False)[:, :, 0, :] \
            .permute(0, 2, 1)
        rel_coord = coord - q_coord
        rel_coord[:, :, 0] *= feat.shape[-2]
        rel_coord[:, :, 1] *= feat.shape[-1]
        inp = torch.cat([q_feat, rel_coord], dim=-1)

        if self.cell_decode:
            rel_cell = cell.clone()
            rel_cell[:, :, 0] *= feat.shape[-2]
            rel_cell[:, :, 1] *= feat.shape[-1]
            inp = torch.cat([inp, rel_cell], dim=-1)

        bs, q = coord.shape[:2]
        pred = self.imnet(inp.view(bs * q, -1)).view(bs, q, -1)
        preds.append(pred)

        area = torch.abs(rel_coord[:, :, 0] * rel_coord[:, :, 1])
        areas.append(area + 1e-9)

tot_area = torch.stack(areas).sum(dim=0)
if self.local_ensemble:
    t = areas[0]; areas[0] = areas[3]; areas[3] = t
    t = areas[1]; areas[1] = areas[2]; areas[2] = t
ret = 0
for pred, area in zip(preds, areas):
    ret = ret + pred * (area / tot_area).unsqueeze(-1)
return ret

forward(self, inp, coord, cell):
self.gen_feat(inp)
return self.query_rgb(coord, cell)
```

제안하는 방식

- Local Implicit Image Function – cell decoding
  - 궁극적으로, LIF 표현을 임의의 해상도에 대하여 픽셀 기반 형식으로 표시 할 수 있는 것이 목표임.
    - In practice, we want that the LIIF representation can be presented as the pixel-based form in arbitrary resolution.
  - 의문점
    - Query pixel, x 에 대한 정보를 질문(query)하여야 할 것인가?
    - If) 단순한 방법으로는 연속적인 representation, I 에서 픽셀의 중심 좌표의 RGB 값을 요청(query) 하는 것임.
      - Suppose the desired resolution is given, a straight-forward way is to query the RGB values at the coordinates of pixel centers in the continuous representation.
      - 문제점
        - 하지만, query pixel 에 대하여 예측된 RGB 값은 크기에 독립적이기 때문에 중심 좌표를 제외한 pixel 영역에 대한 정보는 모두 무시되는 문제점이 있음.
        - since the predicted RGB value of a query pixel is independent of its size, the information in its pixel area is all discarded except the center value.
  - 해결 방법
    - Cell decoding
      - 핵심) 중심에 대한 좌표 정보 뿐만 아니라, cell 의 모양 정보까지 제공하여 pixel 에 대한 영역 정보가 추가적으로 제공되도록 설계함.

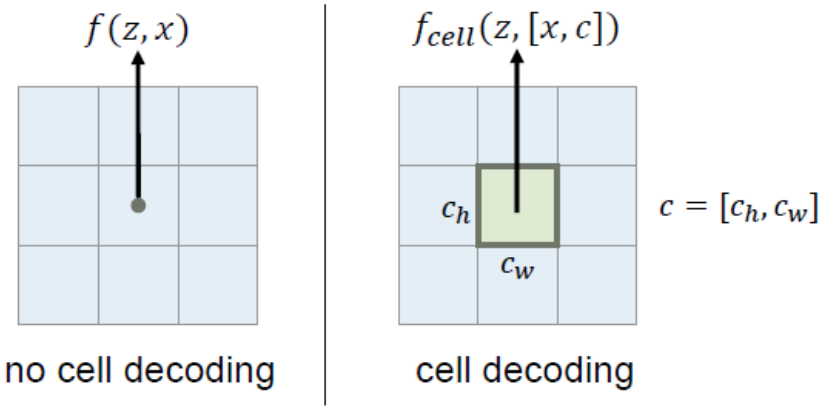


Figure 3: **Cell decoding.** With cell decoding, the decoding function takes the shape of the query pixel as an additional input and predicts the RGB value for the pixel.

제안하는 방식

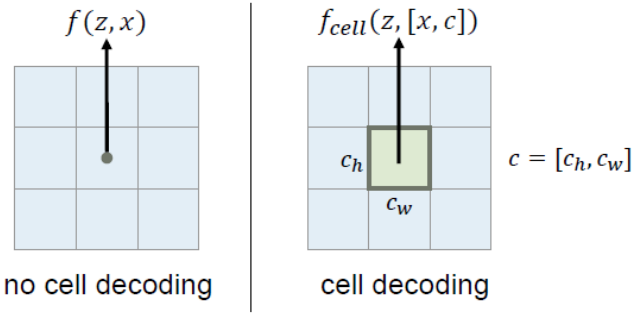


Figure 3: **Cell decoding.** With cell decoding, the decoding function takes the shape of the query pixel as an additional input and predicts the RGB value for the pixel.

- Local Implicit Image Function – cell decoding
  - 방법
    - Query pixel 의 모양 정보, c 를 neural implicit function, $f_\theta(\cdot, z)$ 에 다음과 같이 부여 할 수 있음.

$$s = f_{\theta,cell}(z,[x,c])$$

      - $c = [c_h, c_w]$
        - $c_h = height\ of\ the\ query\ pixel.$
        - $c_w = width\ of\ the\ query\ pixel.$
      - $[x,c] = concatenation\ operation.$
  - 의미
    - Pixel 에 대한 중심 정보, x 와 모양 정보, c 를 제공하였을 때 이에 해당하는 RGB 값이 어떠한 값이 되어야 하는가를 의미함.
    - Query pixel 에 대한 모양 정보를 제공하면 궁극적으로 주어진 해상도에서 연속적인 표현을 나타내고 싶을 때 효과적임,
      - As we will show in the experiments, having an extra input c can be beneficial when presenting the continuous representation in a given resolution.

제안하는 방식

- Local Implicit Image Function – cell decoding

```python
def __getitem__(self, idx):
    img_lr, img_hr = self.dataset[idx]
    p = idx / (len(self.dataset) - 1)
    w_hr = round(self.size_min + (self.size_max - self.size_min)
    img_hr = resize_fn(img_hr, w_hr)

    if self.augment:
        if random.random() < 0.5:
            img_lr = img_lr.flip(-1)
            img_hr = img_hr.flip(-1)

    if self.gt_resize is not None:
        img_hr = resize_fn(img_hr, self.gt_resize)

    hr_coord, hr_rgb = to_pixel_samples(img_hr)

    if self.sample_q is not None:
        sample_lst = np.random.choice(
            len(hr_coord), self.sample_q, replace=False)
        hr_coord = hr_coord[sample_lst]
        hr_rgb = hr_rgb[sample_lst]

    cell = torch.ones_like(hr_coord)
    cell[:, 0] *= 2 / img_hr.shape[-2]
    cell[:, 1] *= 2 / img_hr.shape[-1]

    return {
        'inp': img_lr,
        'coord': hr_coord,
        'cell': cell,
        'gt': hr_rgb
    }
```

```python
    if self.cell_decode:
        rel_cell = cell.clone()
        rel_cell[:, :, 0] *= feat.shape[-2]
        rel_cell[:, :, 1] *= feat.shape[-1]
        inp = torch.cat([inp, rel_cell], dim=-1)
```

제안하는 방식

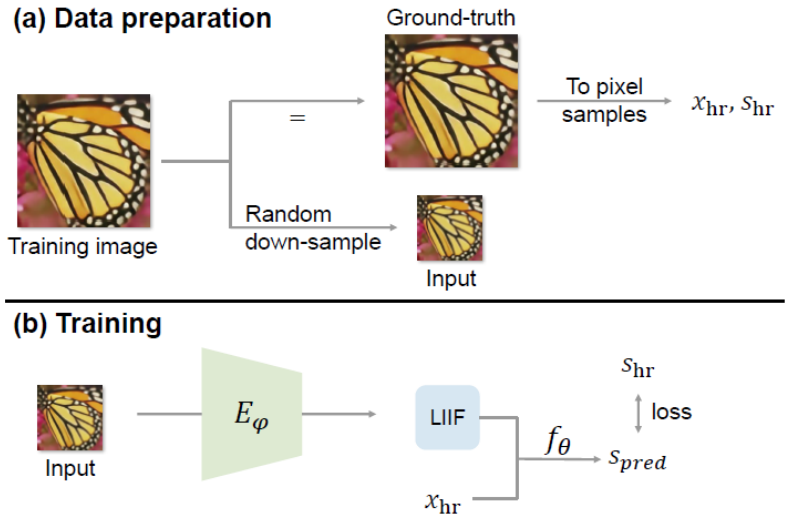


Figure 4: **Learning to generate continuous representation for pixel-based images.** An encoder is jointly trained with the LIIF representation in a self-supervised super-resolution task, in order to encourage the LIIF representation to maintain high fidelity in higher resolution.

- Learning Continuous Image Representation
  - 영상에 대한 연속적인 표현을 어떻게 학습할 것인가?
  - 궁극적인 목표
    - 학습 셋이 주어졌을 때, 관측하지 않은 영상에 대하여 연속적인 표현(continuous representation)을 생성해 내는 것임.
  - 기본 아이디어
    1. 영상을 LIIF representation 으로서 2D feature map 으로 mapping 하는 encoder 학습.
    2. 이를 입력으로 받아 모든 영상에 걸쳐 공유되는 continuous image 로 변환해 주는 neural function, $f_{\theta,cell}$ 을 함께 학습.
       - The general idea is to train an encoder that maps an image to a 2D feature map as its LIIF representation, the function $f_{\theta,cell}$ shared by all the images is jointly trained.
    - 뿐만 아니라, continuous representation 을 위하여 고해상도의 영상이 주어졌을 때 높은 충실도(high fidelity) 를 유지하기 위하여, self-supervised 방식으로 SR 알고리즘을 학습하는 것을 제안.
  - Self-supervised training
    - Ground-truth
      - 학습 데이터(정답 영상)이 주어졌을 때, 무작위의 scale 로 down-sampling 하여 입력 영상을 생성함.
        - 정답은 학습 영상을 pixel samples 로 나타내며(representing the training image as pixel samples) 다음과 같이 표현됨.
          - $x_{hr} = center\ coordinates\ of\ pixels\ in\ the\ image\ domain.$
            - 이는 LIIF representation 에 대한 query 로 사용 되어 $f_{\theta,cell}$ 가 LIIF representation 기반하여 해당 좌표에 대한 RGB 값을 예측함.
          - $s_{hr} = corresponding\ RGB\ values\ of\ the\ pixels.$
    - Loss
      - $f_{\theta,cell}$ 가 예측한 RGB 값과 정답 RGB 값 s_{hr} 과의 $L_1\ loss.$

제안하는 방식

- Ground-truth
  - 학습 데이터(정답 영상)이 주어졌을 때, 무작위의 scale 로 down-sampling 하여 입력 영상을 생성함.
    - 정답은 학습 영상을 pixel samples 로 나타내며(representing the training image as pixel samples) 다음과 같이 표현됨.
      - $x_{hr} = center\ coordinates\ of\ pixels\ in\ the\ image\ domain.$
        - 이는 LIIF representation 에 대한 query 로 사용 되어 $f_{\theta,cell}$ 가 LIIF representation 기반하여 해당 좌표에 대한 RGB 값을 예측함.
      - $s_{hr} = corresponding\ RGB\ values\ of\ the\ pixels.$

```python
def make_coord(shape, ranges=None, flatten=True):
    """ Make coordinates at grid centers.
    """
    coord_seqs = []
    for i, n in enumerate(shape):
        if ranges is None:
            v0, v1 = -1, 1
        else:
            v0, v1 = ranges[i]
        r = (v1 - v0) / (2 * n)
        seq = v0 + r + (2 * r) * torch.arange(n).float()
        coord_seqs.append(seq)
    ret = torch.stack(torch.meshgrid(*coord_seqs), dim=-1)
    if flatten:
        ret = ret.view(-1, ret.shape[-1])
    return ret


def to_pixel_samples(img):
    """ Convert the image to coord-RGB pairs.
        img: Tensor, (3, H, W)
    """
    coord = make_coord(img.shape[-2:])
    rgb = img.view(3, -1).permute(1, 0)
    return coord, rgb
```

제안하는 방식

- Local Implicit Image Function $s = f_\theta(x, z)$
  - 모델 호출

```
if imnet_spec is not None:
    imnet_in_dim = self.encoder.out_dim
    if self.feat_unfold:
        imnet_in_dim *= 9
    imnet_in_dim += 2 # attach coord
    if self.cell_decode:
        imnet_in_dim += 2
    self.imnet = models.make(imnet_spec, args={'in_dim': imnet_in_di
else:
    self.imnet = None
```

  - 모델 입력
    1. Encoder
       - Low resolution image
    2. LIIF
       - Coordinate
       - Cell
       - RGB value
       - Latent code in output of encoder

```
def __getitem__(self, idx):
    img_lr, img_hr = self.dataset[idx]
    p = idx / (len(self.dataset) - 1)
    w_hr = round(self.size_min + (self.size_max - self.size_min)
    img_hr = resize_fn(img_hr, w_hr)

    if self.augment:
        if random.random() < 0.5:
            img_lr = img_lr.flip(-1)
            img_hr = img_hr.flip(-1)

    if self.gt_resize is not None:
        img_hr = resize_fn(img_hr, self.gt_resize)

    hr_coord, hr_rgb = to_pixel_samples(img_hr)

    if self.sample_q is not None:
        sample_lst = np.random.choice(
            len(hr_coord), self.sample_q, replace=False)
        hr_coord = hr_coord[sample_lst]
        hr_rgb = hr_rgb[sample_lst]

    cell = torch.ones_like(hr_coord)
    cell[:, 0] *= 2 / img_hr.shape[-2]
    cell[:, 1] *= 2 / img_hr.shape[-1]

    return {
        'inp': img_lr,
        'coord': hr_coord,
        'cell': cell,
        'gt': hr_rgb
    }
```

제안하는 방식

- 모델 입력
  1. Encoder
     - Low resolution image
  2. LIIF
     - Coordinate
     - Cell
     - RGB value
     - Latent code in output of encoder
- 출력
  - Query coordinate 의 RGB values.

```
    for batch in tqdm(train_loader, leave=False, desc='train'):
        for k, v in batch.items():
            batch[k] = v.cuda()

        inp = (batch['inp'] - inp_sub) / inp_div
        pred = model(inp, batch['coord'], batch['cell'])

        gt = (batch['gt'] - gt_sub) / gt_div
        loss = loss_fn(pred, gt)

        train_loss.add(loss.item())

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        pred = None; loss = None

    return train_loss.item()
```

```
    def forward(self, inp, coord, cell):
        self.gen_feat(inp)
        return self.query_rgb(coord, cell)
```

```
def __getitem__(self, idx):
    img_lr, img_hr = self.dataset[idx]
    p = idx / (len(self.dataset) - 1)
    w_hr = round(self.size_min + (self.size_max - self.size_min)
    img_hr = resize_fn(img_hr, w_hr)

    if self.augment:
        if random.random() < 0.5:
            img_lr = img_lr.flip(-1)
            img_hr = img_hr.flip(-1)

    if self.gt_resize is not None:
        img_hr = resize_fn(img_hr, self.gt_resize)

    hr_coord, hr_rgb = to_pixel_samples(img_hr)

    if self.sample_q is not None:
        sample_lst = np.random.choice(
            len(hr_coord), self.sample_q, replace=False)
        hr_coord = hr_coord[sample_lst]
        hr_rgb = hr_rgb[sample_lst]

    cell = torch.ones_like(hr_coord)
    cell[:, 0] *= 2 / img_hr.shape[-2]
    cell[:, 1] *= 2 / img_hr.shape[-1]

    return {
        'inp': img_lr,
        'coord': hr_coord,
        'cell': cell,
        'gt': hr_rgb
    }
```