This notebook was developed with torch tutorial @ https://pytorch.org/tutorials/recipes/recipes/custom_dataset_transforms_loader.html

In 1
```python
from __future__ import print_function, division
import os
import torch
import pandas as pd
from skimage import io, transform
import numpy as np
import matplotlib.pyplot as plt
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms, utils

# Ignore warnings
import warnings
warnings.filterwarnings("ignore")

plt.ion()   # interactive mode
```
Executed at 2023.12.01 22:50:33 in 1s 972ms

```
/Users/adi/opt/anaconda3/lib/python3.10/site-packages/torchvision/io/image.py:13: UserWarning: Failed to load image Pyth
```

Out 1
```
<contextlib.ExitStack at 0x10657f280>
```

In 2
```python
# from my utils notebook
def add_to_class(Class):
    """Register them functions"""
    def wrapper(obj):
        # setattr(object, name, value) → sets the value of the attribute
        setattr(Class ,obj.__name__ , obj)
    return wrapper
```
Executed at 2023.12.01 22:50:33 in 4ms

In 3
```python
class RedWebDataset(Dataset):
    def __init__(self, root_dir, transform=None):
        """
        Root Dir is REDWEB_V1
            Imgs → Monocular Images
            RDs → Corresponding Response (Heatmap)
        """
        self.root_dir = root_dir
        self.transform = transform
        self.monocular_folder = os.path.join(root_dir, 'Imgs')
        self.heatmap_folder = os.path.join(root_dir, 'RDs')
        # Monocular images are in jpgs and the heatmap images are in pngs
        self.monocular_images = sorted(os.listdir(self.monocular_folder))
        self.heatmap_images = sorted(os.listdir(self.heatmap_folder))

```
Executed at 2023.12.01 22:50:33 in 2ms

now we can get images from our loaders:

In 4
```python
loader = RedWebDataset("../ReDWeb_V1")
loader.monocular_images[:2]
```
Executed at 2023.12.01 22:50:33 in 58ms

Out 4
```
['10147759665_3367509158.jpg', '10147764085_481cd7d72f.jpg']
```

In 5
```python
loader.heatmap_images[:2]
```
Executed at 2023.12.01 22:50:33 in 15ms

Out 5
```
['10147759665_3367509158.png', '10147764085_481cd7d72f.png']
```

In 6
```python
# add a simple __len__ method
@add_to_class(RedWebDataset)
def __len__(self):
    assert len(self.monocular_images) == len(self.heatmap_images) , "Hein?"
    return len(self.monocular_images)
```
Executed at 2023.12.01 22:50:33 in 12ms

In 7
```python
# now we can call len on our loader
len(loader)
```
Executed at 2023.12.01 22:50:33 in 10ms

Out 7
```
3600
```

In 8
```python
# lets write for practice that makes a subplot which shows
# image and their coresponding heatmap on a single subplot

@add_to_class(RedWebDataset)
def _show_sample(self ,name = None):
    if name:
        name = name if "." not in name else name.spit(".")[0]
    else:
        # show random index
        rand_index = np.random.randint(low= 0,high =len(self) , size=1).item()
        name = self.monocular_images[rand_index].split(".")[0]

    # read images
    # mono_img =  plt.imread(loader.monocular_folder +"/"+ name + ".jpg")
    # heat_img =  plt.imread(loader.heatmap_folder +"/"+ name + ".png")
    mono_img =  io.imread(self.monocular_folder +"/"+ name + ".jpg")
    heat_img =  io.imread(self.heatmap_folder +"/"+ name + ".png")

    fig , ax = plt.subplot_mosaic("AB")
    ax["A"].set_title(f"Monocular Image{name}",fontsize=8)
    ax["A"].imshow(mono_img)
    ax["B"].set_title(f"Heatmap Image{name}",fontsize=8)
    ax["B"].imshow(heat_img,cmap="inferno")
    ax["A"].grid(False)
    ax["A"].axis('off')
    ax["B"].grid(False)
    ax["B"].axis('off')
    plt.show()
```
Executed at 2023.12.01 22:50:33 in 7ms

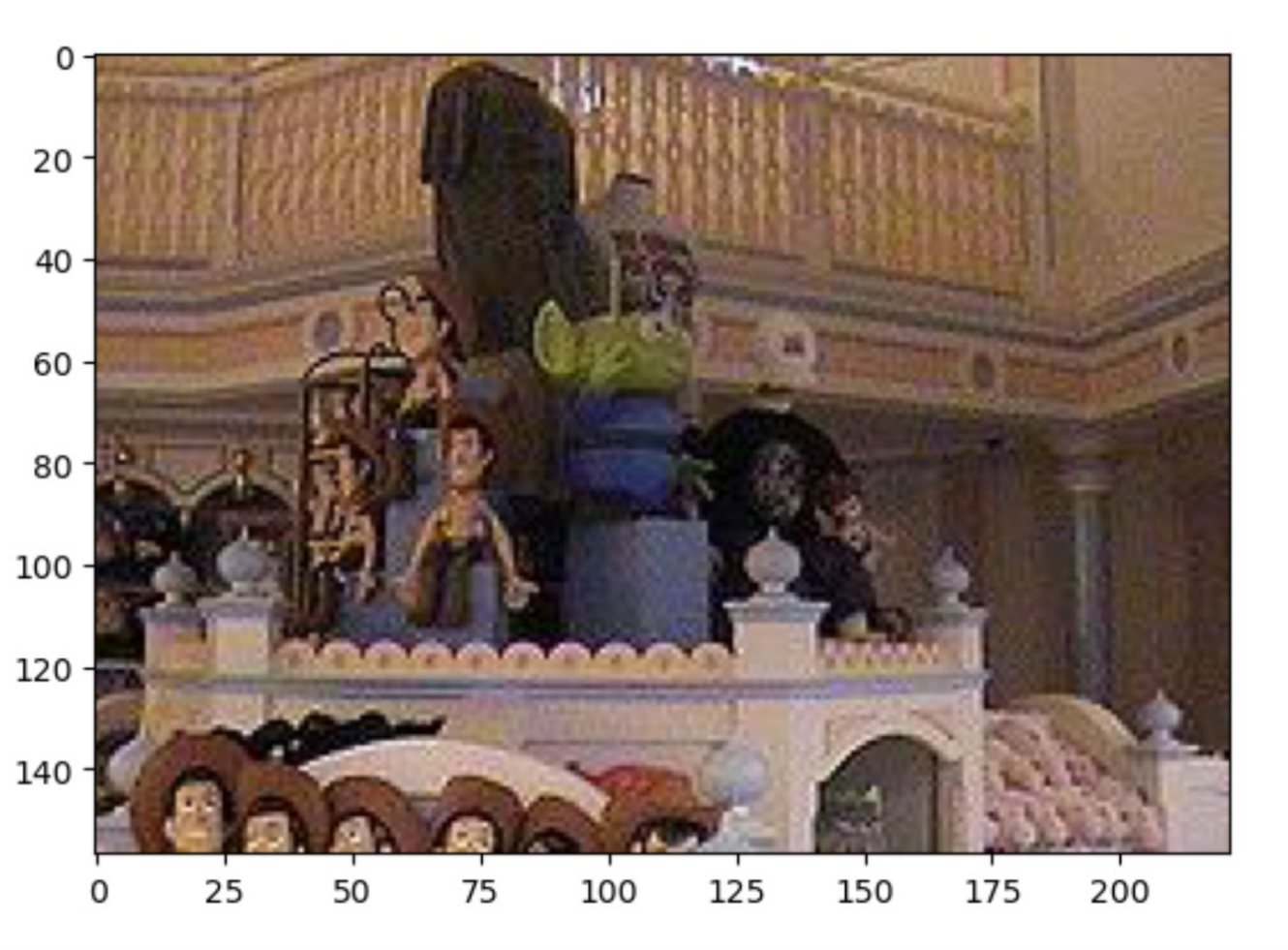
In 9
```python
loader._show_sample()
```
Executed at 2023.12.01 22:50:33 in 96ms

In 10
```python
loader._show_sample()
```
Executed at 2023.12.01 22:50:33 in 94ms

In 11
```python
# now lastly just implement __getitem__
@add_to_class(RedWebDataset)
def __getitem__(self, idx):
    if torch.is_tensor(idx):
        idx = idx.tolist()

    monocular_image = io.imread(self.monocular_folder+"/"+self.monocular_images[idx])
    heatmap_image = io.imread(self.heatmap_folder +"/"+ self.heatmap_images[idx])
    sample = {'mono': monocular_image, 'heat':heatmap_image }

    if self.transform:
        sample = self.transform(sample)

    return sample
```
Executed at 2023.12.01 22:50:33 in 6ms

In 12
```python
# now we can call index on our loader
plt.imshow(loader[1234]["mono"])
```
Executed at 2023.12.01 22:50:33 in 145ms

Out 12
```
<matplotlib.image.AxesImage at 0x16b03f3a0>
```



Now lets Develop some useful data transformation

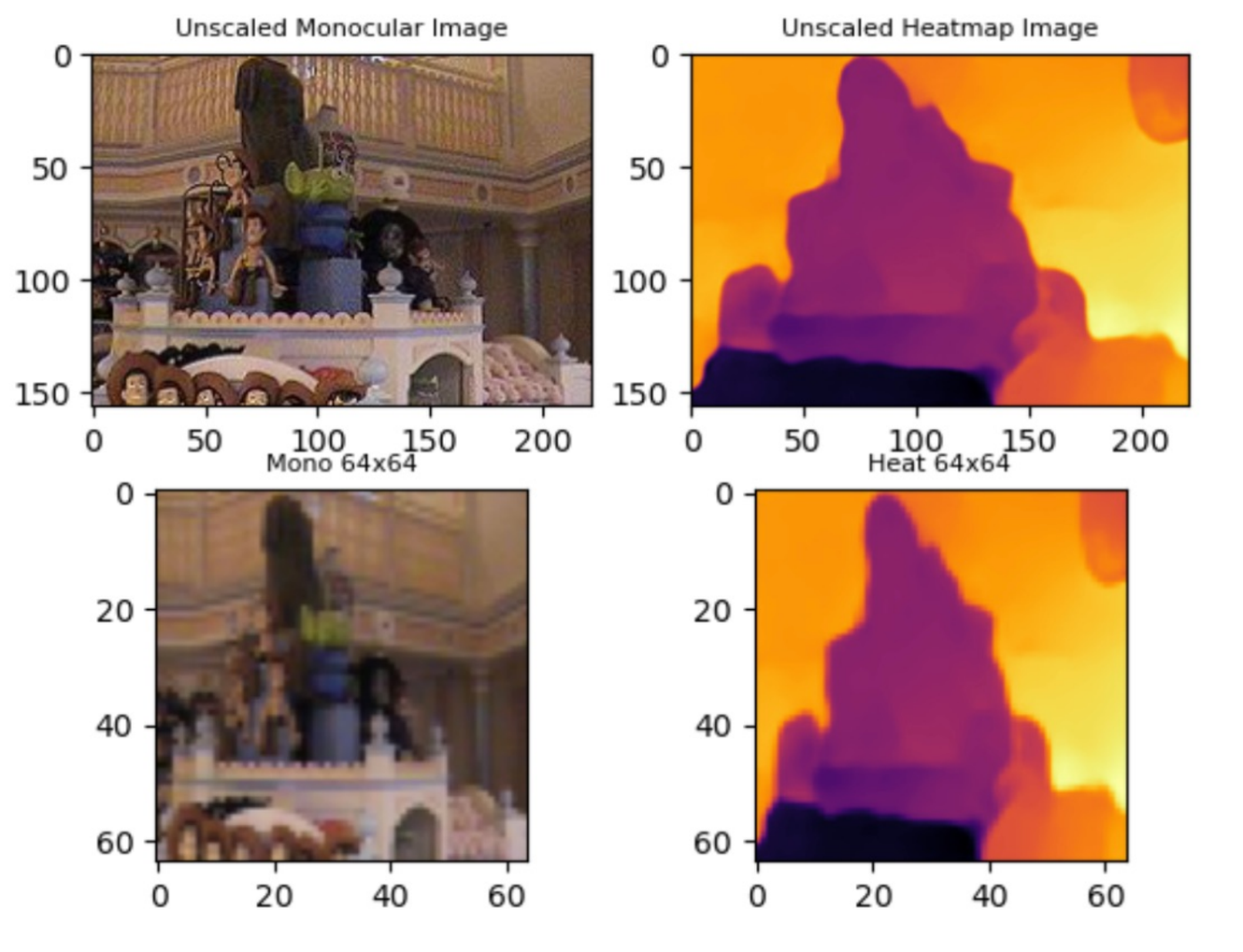
i) A rescaler

In 13
```python
class Rescale(object):
    """
    simple Rescaler for my monocular and heatmap images
    """
    def __init__(self, output_size):
        assert isinstance(output_size, (int, tuple))
        self.output_size = output_size

    def __call__(self, sample):
        mono, heat = sample['mono'], sample['heat']
        h, w = mono.shape[:2]
        if isinstance(self.output_size, int):
            if h > w:
                new_h, new_w = self.output_size * h / w, self.output_size
            else:
                new_h, new_w = self.output_size, self.output_size * w / h
        else:
            new_h, new_w = self.output_size

        new_h, new_w = int(new_h), int(new_w)

        mono = transform.resize(mono, (new_h, new_w))
        heat = transform.resize(heat, (new_h, new_w))

        return {'mono': mono, 'heat': heat}
```
Executed at 2023.12.01 22:50:33 in 4ms

In 14
```python
rescaler_64 = Rescale((64,64))
scaled = rescaler_64(loader[1234])
m64 ,h64 = scaled["mono"] , scaled["heat"]
```
Executed at 2023.12.01 22:50:33 in 6ms

In 15
```python
fig , ax = plt.subplot_mosaic("AB;CD")
ax["A"].set_title(f"Unscaled Monocular Image",fontsize=8)
ax["A"].imshow(loader[1234]["mono"])

ax["B"].set_title(f"Unscaled Heatmap Image",fontsize=8)
ax["B"].imshow(loader[1234]["heat"],cmap="inferno")

ax["C"].set_title(f"Mono 64x64",fontsize=8)
ax["C"].imshow(m64)

ax["D"].set_title(f"Heat 64x64",fontsize=8)
ax["D"].imshow(h64,cmap="inferno")

```
Executed at 2023.12.01 22:50:34 in 239ms

Out 15
```
<matplotlib.image.AxesImage at 0x16b1c8be0>
```



ii) A random cropper

In 16
```python
class RandomCrop(object):
    """
    Crop randomly the image in a sample.
    """

    def __init__(self, output_size):
        assert isinstance(output_size, (int, tuple))
        if isinstance(output_size, int):
            self.output_size = (output_size, output_size)
        else:
            assert len(output_size) == 2
            self.output_size = output_size

    def __call__(self, sample):
        mono, heat = sample['mono'], sample['heat']

        h, w = mono.shape[:2]
        new_h, new_w = self.output_size

        top = np.random.randint(0, h - new_h)
        left = np.random.randint(0, w - new_w)

        mono = mono[top: top + new_h, left: left + new_w]
        heat = heat[top: top + new_h, left: left + new_w]


        return {'mono': mono, 'heat': heat}

```
Executed at 2023.12.01 22:50:34 in 3ms

In 17
```python
# a random cropper which randomly crops and returns back an image of 64x64
r_cropper = RandomCrop((64,64))
cropped = r_cropper(loader[1234])
cm , ch = cropped["mono"] , cropped["heat"]
```
Executed at 2023.12.01 22:50:34 in 4ms

In 18
```python
fig , ax = plt.subplot_mosaic("AB;CD")
ax["A"].set_title(f"Unscaled Monocular Image",fontsize=8)
ax["A"].imshow(loader[1234]["mono"])

ax["B"].set_title(f"Unscaled Heatmap Image",fontsize=8)
ax["B"].imshow(loader[1234]["heat"],cmap="inferno")

ax["C"].set_title(f"rcropped Mono 64x64",fontsize=8)
ax["C"].imshow(cm)

ax["D"].set_title(f"rcropped Heat 64x64",fontsize=8)
ax["D"].imshow(ch,cmap="inferno")
```
Executed at 2023.12.01 22:50:34 in 241ms

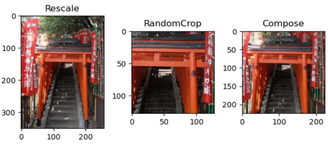
Out 18
```
<matplotlib.image.AxesImage at 0x16b378280>
```

You can composite transforms composed = (rescale o cropped)(img)

In 19
```python
scale = Rescale(256)
crop = RandomCrop(128)
composed = transforms.Compose([Rescale(256),
                               RandomCrop(224)])

# Apply each of the above transforms on sample.
fig = plt.figure()
sample = loader[65] # lets start using a different example now
for i, tsfrm in enumerate([scale, crop, composed]):
    transformed_sample = tsfrm(sample)

    ax = plt.subplot(1, 3, i + 1)
    plt.tight_layout()
    ax.set_title(type(tsfrm).__name__)
    plt.imshow(transformed_sample["mono"])

plt.show()
```
Executed at 2023.12.01 22:50:34 in 235ms

In 20
```python
# a simple image to tensor class
class ToTensor(object):
```

```
sample = loader[65] # lets start using a different example now
for i, tsfrm in enumerate([scale, crop, composed]):
    transformed_sample = tsfrm(sample)

    ax = plt.subplot(1, 3, i + 1)
    plt.tight_layout()
    ax.set_title(type(tsfrm).__name__)
    plt.imshow(transformed_sample["mono"])

plt.show()
```
Executed at 2023.12.01 22:50:34 in 235ms



In 20
```
# a simple image to tensor class
class ToTensor(object):
    """Convert ndarrays in sample to Tensors."""

    def __call__(self, sample):
        mono, heat = sample['mono'], sample['heat']

        # for monocular image :
        # swap color axis because
        # numpy image: H x W x C
        # torch image: C X H X W
        mono = mono.transpose((2, 0, 1))

        # dont have to do anything for single channel image

        return {'mono': torch.from_numpy(mono),
                'heat': torch.from_numpy(heat)}
```
Executed at 2023.12.01 22:50:34 in 4ms

In 21
```
to_tensor = ToTensor()
tensored= to_tensor(loader[65])
print(type(tensored["mono"]) , type(tensored["heat"]))
```
Executed at 2023.12.01 22:50:34 in 7ms

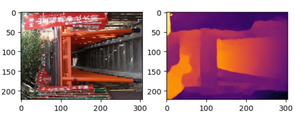
```
<class 'torch.Tensor'> <class 'torch.Tensor'>
```

In 22
```
# and their shape also must be the same in height and width .
# (channel , height , width) <- torch.img
print(tensored["mono"].shape , tensored["heat"].shape)
```
Executed at 2023.12.01 22:50:34 in 12ms

```
torch.Size([3, 305, 222]) torch.Size([305, 222])
```

In 23
```
# but that shouldnt change anything .pairwise
# need to plot its transpose as plt.subplots does not take channel as its first dimension
fig,ax = plt.subplot_mosaic("AB")
ax["A"].imshow(tensored["mono"].T)
ax["B"].imshow(tensored["heat"].T,cmap="inferno")
```
Executed at 2023.12.01 22:50:34 in 109ms

Out 23
```
<matplotlib.image.AxesImage at 0x16b570d30>
```

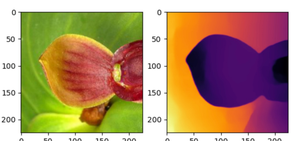


In 24
```
# and now we can iterate through the dataset combining all the operations we have made thus far
transformed_dataset = RedWebDataset(root_dir="../ReDWeb_V1",transform=transforms.Compose([
    Rescale(256),
    RandomCrop(225),
    ToTensor()
]))

sample = transformed_dataset[99]
sample
```
Executed at 2023.12.01 22:50:34 in 36ms

Out 24
```
 'heat': tensor([[0.8392, 0.8392, 0.8392,  ..., 0.9804, 0.9804, 0.9804],
         [0.8392, 0.8392, 0.8392,  ..., 0.9784, 0.9800, 0.9804],
         [0.8392, 0.8392, 0.8392,  ..., 0.9765, 0.9776, 0.9779],
         ...,
         [0.4667, 0.4667, 0.4667,  ..., 0.3844, 0.3875, 0.3883],
         [0.4664, 0.4662, 0.4662,  ..., 0.3841, 0.3871, 0.3882],
         [0.4643, 0.4627, 0.4627,  ..., 0.3823, 0.3841, 0.3870]],
        dtype=torch.float64)}
```

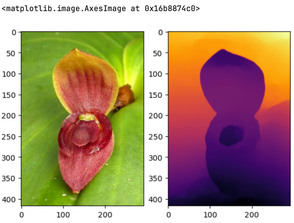
In 25
```
# this is how they look after composing transformations and converting them into tensor objects
fig,ax = plt.subplot_mosaic("AB")
ax["A"].imshow(sample["mono"].T)
ax["B"].imshow(sample["heat"].T,cmap="inferno")
```
Executed at 2023.12.01 22:50:34 in 119ms

Out 25
```
<matplotlib.image.AxesImage at 0x16b7b6e00>
```



In 26
```
# this is how they (originally) looked from our simple class loader
fig,ax = plt.subplot_mosaic("AB")
ax["A"].imshow(loader[99]["mono"])
ax["B"].imshow(loader[99]["heat"],cmap="inferno")

```
Executed at 2023.12.01 22:50:35 in 150ms

Out 26
```
<matplotlib.image.AxesImage at 0x16b8874c0>
```



```
# Now lets get to the crux of the dataloader
    * Batching
    * shuffling
    * loading
```

In 29
```
# and thats it dataloader is done 😄
dataloader = DataLoader(transformed_dataset, batch_size=4, shuffle=True, num_workers=4) # but an iterable . cant call getitem on this obviously
```
Executed at 2023.12.01 22:57:06 in 131ms

We would be overwriting the getitem method of our original Dataloader to encorporate Online Batch Sampling

We go on detail of its working in a seperate notebook in DepthSense/online_sampling

Check out that notebook/pdf before reading this

In 2

```python
from DataLoader import RedWebDataset , Rescale , RandomCrop , ToTensor
from __future__ import print_function, division
import os
import torch
from skimage import io, transform
import random
import matplotlib.pyplot as plt
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms, utils
```

Executed at 2023.12.02 17:20:51 in 8ms

In 4

```python
# from my utils notebook
def add_to_class(Class):
    """Register them functions"""
    def wrapper(obj):
        # setattr(object, name, value) --> sets the value of the attribute
        setattr(Class ,obj.__name__ , obj)
    return wrapper
```

Executed at 2023.12.02 17:22:46 in 10ms

In 80

```python
class OnlineRedWeb(RedWebDataset):
    def __init__(self, root_dir ,transform=None,N:int=10,sigma:int=0.02):
        super().__init__(root_dir,transform)
        self.N =N
        self.sigma = 0.02

    def __getitem__(self, index):
        # Call the __getitem__ method of the original dataset
        original_item = super(OnlineRedWeb, self).__getitem__(index)

        # Extract necessary information from the original_item
        mono = original_item['mono']
        heat = original_item['heat']

        # Implement online sampling logic here
        height, width = heat.shape
        point_a =[]
        point_b =[]
        labels = []

        for _ in range(self.N):
            i, j = random.randint(0, height-1), random.randint(0, width-1)
            k,l = random.randint(0, height-1), random.randint(0, width-1)

            ga, gb = heat[i, j], heat[k, l]  # Assuming heat is a 2D tensor

            if ga/ gb > 1 + self.sigma:
                label = 1
            elif ga/ gb < 1 - self.sigma:
                label = -1
            else:
                label = 0

            point_a.append((i,j))
            point_b.append((k,l))
            labels.append(label)

        # Update the original_item or create a new dictionary to return
        online_item = {'mono': mono,
                       'heat': heat,
                       'point_a':torch.tensor(point_a),"point_b": torch.tensor(point_b),
                       'labels': torch.tensor(labels)}

        return online_item
```

Executed at 2023.12.02 18:25:00 in 4ms

In 81

```python
@add_to_class(OnlineRedWeb)
def online_collater(batch):
    mono = torch.stack([item['mono'] for item in batch])
    heat = torch.stack([item['heat'] for item in batch])
    point_a = [item['point_a'] for item in batch]
    point_b = [item['point_b'] for item in batch]
    labels = [item['labels'] for item in batch]
    return{'mono': mono, 'heat': heat,
          'point_a':point_a ,"point_b":point_b,
          'labels': labels}
```

Executed at 2023.12.02 18:25:01 in 4ms

In 82

```python
# we can still call the old methods on this class
loader = OnlineRedWeb(root_dir="../ReDWeb_V1")
loader._show_sample()
```

Executed at 2023.12.02 18:25:01 in 163ms

In 83

```python
online_dataset = OnlineRedWeb(root_dir="../ReDWeb_V1",transform=transforms.Compose([
    Rescale(256),
    RandomCrop(225),
    ToTensor()
]))
```

Executed at 2023.12.02 18:25:03 in 15ms

In 84

```python
online_loader = DataLoader(online_dataset,batch_size=32,shuffle=True,collate_fn=custom_collate)
```

Executed at 2023.12.02 18:25:04 in 4ms

In 86

```python
for i, batch in enumerate(online_loader):
    if i > 0 :
        break
    print(batch["mono"].shape)
    print(batch["heat"].shape)
    print(batch["point_a"].shape)
    print(batch["point_b"].shape)
    print(batch["labels"].shape)
```

Executed at 2023.12.02 18:25:27 in 1s 761ms

```
torch.Size([32, 3, 225, 225])
torch.Size([32, 225, 225])
torch.Size([32, 10, 2])
torch.Size([32, 10, 2])
torch.Size([32, 10])
```

SSI_Loss.ipynb | DataLoader.py

Code | Managed: http://localhost:8850 | Python 3 (ipykernel) | Trusted

In 1
```
import numpy as np
import torch
import torch.nn.functional as F
from torchvision import transforms
from torch.utils.data import Dataset, DataLoader
# from skimage.metrics import structural_similarity as ssim
# from data.loaders.OnlineLoader import OnlineRedWeb
from data.loaders.DataLoader import RedWebDataset , Rescale,RandomCrop, ToTensor
```
Executed at 2023.12.03 13:34:17 in 1s 209ms

```
/Users/adi/opt/anaconda3/lib/python3.10/site-packages/torchvision/io/image.py:13: UserWarning: Failed to load image Pyth
```

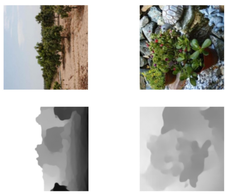
In 2
```
normal_dataset = RedWebDataset(root_dir="../data/ReDWeb_V1",transform=transforms.Compose([
    Rescale(256),
    RandomCrop(225),
    ToTensor()
]))
batcher = DataLoader(normal_dataset,batch_size=1,shuffle=True)
```
Executed at 2023.12.03 13:34:17 in 43ms

In 3
```
for sample1 in batcher:
    print(sample1["mono"].shape)
    print(sample1["heat"].shape)
    break

for sample2 in batcher:
    print(sample2["mono"].shape)
    print(sample2["heat"].shape)
    break
```
Executed at 2023.12.03 13:34:17 in 83ms

```
torch.Size([1, 3, 225, 225])
torch.Size([1, 3, 225, 225])
torch.Size([1, 3, 225, 225])
torch.Size([1, 3, 225, 225])
```

In 4
```
import matplotlib.pyplot as plt
fig ,ax = plt.subplot_mosaic("AB;CD")
ax["A"].imshow(sample1["mono"][0].T)
ax["C"].imshow(sample1["heat"][0].T)


ax["B"].imshow(sample2["mono"][0].T)
ax["D"].imshow(sample2["heat"][0].T)

for k in ax.keys():
    ax[k].axis("off")
    ax[k].grid("off")
```
Executed at 2023.12.03 13:34:17 in 91ms

```
/var/folders/f0/_f_gy0t91jqchv14f0hcl_gh0000gn/T/ipykernel_94717/2951813992.py:3: UserWarning: The use of `x.T` on tenso
```



```
# lets try to calculate the aggregate similarity score of these paiwise map
```

In 5
```
def gaussian(window_size, sigma):
    gauss = torch.exp(-(torch.arange(window_size) - window_size // 2)**2 / (2.0 * sigma**2))
    return gauss / gauss.sum()

def create_window(window_size, channel):
    _1D_window = gaussian(window_size, 1.5).unsqueeze(1)
    _2D_window = _1D_window.mm(_1D_window.t()).float().unsqueeze(0).unsqueeze(0)
    window = _2D_window.expand(channel, 1, window_size, window_size).contiguous()
    return window

def _ssim(img1, img2, window, window_size, channel, size_average=True):
    mu1 = F.conv2d(img1, window, padding=window_size // 2, groups=channel)
    mu2 = F.conv2d(img2, window, padding=window_size // 2, groups=channel)

    mu1_sq = mu1.pow(2)
    mu2_sq = mu2.pow(2)
    mu1_mu2 = mu1 * mu2

    sigma1_sq = F.conv2d(img1 * img1, window, padding=window_size // 2, groups=channel) - mu1_sq
    sigma2_sq = F.conv2d(img2 * img2, window, padding=window_size // 2, groups=channel) - mu2_sq
    sigma12 = F.conv2d(img1 * img2, window, padding=window_size // 2, groups=channel) - mu1_mu2

    C1 = (0.01 * 255)**2
    C2 = (0.03 * 255)**2

    ssim_map = ((2 * mu1_mu2 + C1) * (2 * sigma12 + C2)) / ((mu1_sq + mu2_sq + C1) * (sigma1_sq + sigma2_sq + C2))

    if size_average:
        return ssim_map.mean()
    else:
        return ssim_map.mean(1).mean(1).mean(1)

class SSIM(torch.nn.Module):
    def __init__(self, window_size=11, size_average=True):
        super(SSIM, self).__init__()
        self.window_size = window_size
        self.size_average = size_average
        self.channel = 1
        self.register_buffer('window', create_window(window_size, self.channel))

    def forward(self, img1, img2):
        (_, channel, _, _) = img1.size()

        if channel == self.channel and self.window.type() == img1.type():
            window = self.window
        else:
            window = create_window(self.window_size, channel)
            window = window.to(img1.device).type_as(img1)
            self.window = window
            self.channel = channel

        return _ssim(img1, img2, window, self.window_size, channel, self.size_average)

def ssim(img1, img2, window_size=11, size_average=True):
    (_, channel, _, _) = img1.size()
    window = create_window(window_size, channel)
    window = window.to(img1.device).type_as(img1)
    return _ssim(img1, img2, window, window_size, channel, size_average)
```
Executed at 2023.12.03 13:34:17 in 1ms

In 6
```
ssim_loss = SSIM()
```
Executed at 2023.12.03 13:34:17 in 39ms

In 7
```
ssim_loss(sample1["mono"],sample1["heat"])
```
Executed at 2023.12.03 13:34:18 in 274ms

Out 7
```
tensor(0.9881, dtype=torch.float64)
```

In 8
```
ssim_loss(sample2["mono"],sample2["heat"])
```
Executed at 2023.12.03 13:34:18 in 217ms

Out 8
```
tensor(0.9762, dtype=torch.float64)
```

In 1
```
import matplotlib.pyplot as plt
import torch
from torchvision.io import read_image
from torchvision import transforms
import torch.nn as nn
import torch.nn.functional as F



from torch.utils.data import DataLoader
```
Executed at 2023.12.04 20:06:38 in 1s 590ms

class **InverseDepthSmoothnessLoss** [source]

Criterion that computes image-aware inverse depth smoothness loss.

$$\text{loss} = |\partial_x d_{ij}| \, e^{-\|\partial_x I_{ij}\|} + |\partial_y d_{ij}| \, e^{-\|\partial_y I_{ij}\|}$$

Shape:

- Inverse Depth: $(N, 1, H, W)$
- Image: $(N, 3, H, W)$
- Output: scalar

In 2
```
from data.loaders.DataLoader import RedWebDataset , Rescale , RandomCrop
normal_dataset = RedWebDataset(root_dir="data/RedWeb_V1",transform=transforms.Compose([
    Rescale((256,256)),
]))
batcher = DataLoader(normal_dataset,batch_size=1,shuffle=True)
```
Executed at 2023.12.04 20:06:38 in 7ms

In 3
```
batched = iter(batcher)
sample0 = next(batched)
print(sample0["mono"].shape)
print(sample0["heat"].shape)

sample1 = next(batched)
print(sample1["mono"].shape)
print(sample1["heat"].shape)
```
Executed at 2023.12.04 20:06:38 in 12ms
```
torch.Size([1, 3, 256, 256])
torch.Size([1, 1, 256, 256])
torch.Size([1, 3, 256, 256])
torch.Size([1, 1, 256, 256])
```
```
/Users/adi/opt/anaconda3/lib/python3.10/site-packages/torchvision/transforms/functional.py:1603: UserWarning: The defaul
```

In 4
```
# lets see these images
import matplotlib.pyplot as plt
fig , ax = plt.subplot_mosaic("AB;CD")
ax["A"].imshow(sample0["mono"].squeeze(0).T)
ax["C"].imshow(sample1["mono"].squeeze(0).T)

ax["B"].imshow(sample0["heat"].squeeze(0).T , cmap="inferno")
ax["D"].imshow(sample1["heat"].squeeze(0).T, cmap="inferno")

for k in ax.keys():
    ax[k].grid("off")
    ax[k].axis("off")
```
Executed at 2023.12.04 20:06:38 in 93ms
```
/var/folders/f0/_fy0t9tj0chv14f0hcl_gh0000gn/T/ipykernel_12990/4250501436.py:4: UserWarning: The use of `x.T` on tensors of
dimension other than 2 to reverse their shape is deprecated and it will throw an error in a future release. Consider `x.mT` to
transpose batches of matrices or `x.permute(*torch.arange(x.ndim - 1, -1, -1))` to reverse the dimensions of a tensor. (Triggered
internally at /Users/runner/work/_temp/anaconda/conda-bld/pytorch_1699448803473/work/aten/src/ATen/native/TensorShape.cpp:3618.)
  ax["A"].imshow(sample0["mono"].squeeze(0).T)
```



In 5
```
# Based on
# https://github.com/tensorflow/models/blob/master/research/struct2depth/model.py#L625-L641


class InverseDepthSmoothnessLoss(nn.Module):
    r"""Criterion that computes image-aware inverse depth smoothness loss.

    .. math::

        \text{loss} = \left | \partial_x d_{ij} \right | e^{-\left \|
        \partial_x I_{ij} \right \| } + \left |
        \partial_y d_{ij} \right | e^{-\left \| \partial_y I_{ij} \right \|}


    Shape:
        - Inverse Depth: :math:`(N, 1, H, W)`
        - Image: :math:`(N, 3, H, W)`
        - Output: scalar

    Examples::

        >> idepth = torch.rand(1, 1, 4, 5)
        >> image = torch.rand(1, 3, 4, 5)
        >> smooth = tgm.losses.DepthSmoothnessLoss()
        >> loss = smooth(idepth, image)
    """

    def __init__(self) -> None:
        super(InverseDepthSmoothnessLoss, self).__init__()

    @staticmethod
    def gradient_x(img: torch.Tensor) -> torch.Tensor:
        assert len(img.shape) == 4, img.shape
        return img[:, :, :, :-1] - img[:, :, :, 1:]

    @staticmethod
    def gradient_y(img: torch.Tensor) -> torch.Tensor:
        assert len(img.shape) == 4, img.shape
        return img[:, :, :-1, :] - img[:, :, 1:, :]

    def forward(
            self,
            idepth: torch.Tensor,
            image: torch.Tensor) -> torch.Tensor:
        if not torch.is_tensor(idepth):
            raise TypeError("Input idepth type is not a torch.Tensor. Got {}"
                            .format(type(idepth)))
        if not torch.is_tensor(image):
            raise TypeError("Input image type is not a torch.Tensor. Got {}"
                            .format(type(image)))
        if not len(idepth.shape) == 4:
            raise ValueError("Invalid idepth shape, we expect BxCxHxW. Got: {}"
                             .format(idepth.shape))
        if not len(image.shape) == 4:
            raise ValueError("Invalid image shape, we expect BxCxHxW. Got: {}"
                             .format(image.shape))
        if not idepth.shape[-2:] == image.shape[-2:]:
            raise ValueError("idepth and image shapes must be the same. Got: {}"
                             .format(idepth.shape, image.shape))
        if not idepth.device == image.device:
            raise ValueError(
                "idepth and image must be in the same device. Got: {}" .format(
                    idepth.device, image.device))
        if not idepth.dtype == image.dtype:
            raise ValueError(
                "idepth and image must be in the same dtype. Got: {}" .format(
                    idepth.dtype, image.dtype))
        # compute the gradients
        idepth_dx: torch.Tensor = self.gradient_x(idepth)
        idepth_dy: torch.Tensor = self.gradient_y(idepth)
        image_dx: torch.Tensor = self.gradient_x(image)
        image_dy: torch.Tensor = self.gradient_y(image)

        # compute image weights
        weights_x: torch.Tensor = torch.exp(
            -torch.mean(torch.abs(image_dx), dim=1, keepdim=True))
        weights_y: torch.Tensor = torch.exp(
            -torch.mean(torch.abs(image_dy), dim=1, keepdim=True))

        # apply image weights to depth
        smoothness_x: torch.Tensor = torch.abs(idepth_dx * weights_x)
        smoothness_y: torch.Tensor = torch.abs(idepth_dy * weights_y)
        return torch.mean(smoothness_x) + torch.mean(smoothness_y)
```
Executed at 2023.12.04 20:06:38 in 1ms

In 6
```
sample0["mono"] =  sample0["mono"].to(torch.float)
sample0["heat"] =  sample0["heat"].to(torch.float)

sample1["mono"] = sample1["mono"].to(torch.float)
sample1["heat"] = sample1["heat"].to(torch.float)
```
Executed at 2023.12.04 20:06:38 in 5ms

In 7
```
inv_loss = InverseDepthSmoothnessLoss()
inv_loss(sample0["heat"],sample0["mono"])
```
Executed at 2023.12.04 20:06:38 in 20ms

Out 7
```
tensor(0.0443)
```

In 8
```
inv_loss = InverseDepthSmoothnessLoss()
inv_loss(sample0["heat"],sample1["mono"])
```
Executed at 2023.12.04 20:06:38 in 9ms

Out 8
```
tensor(0.4271)
```

In 1

```python
# So we will combine the ideas from the losses we have experimented with
# InverseDepth Loss works fine if there is no feathering in the subject . But if we penalize also on incorrectly detecting edges . we might improve upon the loss . So a edge-guide is needed

# And we will linearly combine with the loss that we experimented with Online sampling
```

Executed at 2023.12.10 18:16:31 in 4ms

In 4

```python
import torch
from torch import nn
import numpy as np
import torch.nn.functional as F
from torchvision import transforms

from data.loaders.DataLoader import RedWebDataset , Rescale , RandomCrop
from torch.utils.data import DataLoader

normal_dataset = RedWebDataset(root_dir="../data/ReDWeb_V1",transform=transforms.Compose([
    Rescale((256,256)),
]))
batcher = DataLoader(normal_dataset,batch_size=1,shuffle=True)
```

Executed at 2023.12.10 18:21:34 in 9ms

In 6

```python
# Lets Device our online sampling first

# Note this is derived completely from the Redweb paper
def onlineSampling(inputs, targets, masks, threshold, sample_num):

    # find A-B point pairs from predictions (mostly random)
    inputs_index = torch.masked_select(inputs, targets.gt(threshold))
    num_effect_pixels = len(inputs_index)
    shuffle_effect_pixels = torch.randperm(num_effect_pixels).cuda()
    rgb_a = inputs_index[shuffle_effect_pixels[0:sample_num*2:2]]
    rgb_b = inputs_index[shuffle_effect_pixels[1:sample_num*2:2]]

    # find corresponding pairs from ground truth
    depth_index = torch.masked_select(targets, targets.gt(threshold))
    depth_a = depth_index[shuffle_effect_pixels[0:sample_num*2:2]]
    depth_b = depth_index[shuffle_effect_pixels[1:sample_num*2:2]]

    # only compute the losses of point pairs with valid ground truth i.e consistent masked
    consistent_masks_index = torch.masked_select(masks, targets.gt(threshold))
    consistent_masks_A = consistent_masks_index[shuffle_effect_pixels[0:sample_num*2:2]]
    consistent_masks_B = consistent_masks_index[shuffle_effect_pixels[1:sample_num*2:2]]

    # The amount of A and B should be the same!!
    if len(depth_a) > len(depth_b):
        depth_a = depth_a[:-1]
        rgb_a = rgb_a[:-1]
        consistent_masks_A = consistent_masks_A[:-1]

    return rgb_a, rgb_b, depth_a, depth_b, consistent_masks_A, consistent_masks_B
```

Executed at 2023.12.10 18:32:20 in 2ms

In 7

```python
# now lets penalize wrong edges
# i.e if the edeges derived from the depth map does not map with the edges derived from the original image. there should be a corresponding penalty foir it.

# convenience wrapper function to get pixels
def ind2sub(idx, cols):
    r = idx / cols
    c = idx - r * cols
    return r, c
```

```python
def sub2ind(r, c, cols):
    idx = r * cols + c
    return idx


```

Executed at 2023.12.10 19:26:27 in 5ms

In 8

```python
def edgeGuidedSampling(inputs, targets, edges_img, thetas_img, masks, h, w):

    # find edges
    edges_max = edges_img.max()
    edges_mask = edges_img.ge(edges_max*0.1)
    edges_loc = edges_mask.nonzero()

    inputs_edge = torch.masked_select(inputs, edges_mask)
    targets_edge = torch.masked_select(targets, edges_mask)
    thetas_edge = torch.masked_select(thetas_img, edges_mask)
    minlen = inputs_edge.size()[0]

    # find anchor points (i.e, edge points)
    sample_num = minlen
    index_anchors = torch.randint(
        0, minlen, (sample_num,), dtype=torch.long).cuda()
    anchors = torch.gather(inputs_edge, 0, index_anchors)
    theta_anchors = torch.gather(thetas_edge, 0, index_anchors)
    row_anchors, col_anchors = ind2sub(edges_loc[index_anchors].squeeze(1), w)
    # compute the coordinates of 4-points,  distances are from [2, 30]
    distance_matrix = torch.randint(2, 31, (4, sample_num)).cuda()
    pos_or_neg = torch.ones(4, sample_num).cuda()
    pos_or_neg[:2, :] = -pos_or_neg[:2, :]
    distance_matrix = distance_matrix.float() * pos_or_neg
    col = col_anchors.unsqueeze(0).expand(4, sample_num).long(
    ) + torch.round(distance_matrix.double() * torch.cos(theta_anchors).unsqueeze(0)).long()
    row = row_anchors.unsqueeze(0).expand(4, sample_num).long(
    ) + torch.round(distance_matrix.double() * torch.sin(theta_anchors).unsqueeze(0)).long()

    # constrain 0=<c≤w, 0≤r≤h
    # Note: index should minus 1
    col[col < 0] = 0
    col[col > w-1] = w-1
    row[row < 0] = 0
    row[row > h-1] = h-1

    # a-b, b-c, c-d
    a = sub2ind(row[0, :], col[0, :], w)
    b = sub2ind(row[1, :], col[1, :], w)
    c = sub2ind(row[2, :], col[2, :], w)
    d = sub2ind(row[3, :], col[3, :], w)
    A = torch.cat((a, b, c), 0)
    B = torch.cat((b, c, d), 0)

    rgb_a = torch.gather(inputs, 0, A.long())
    rgb_b = torch.gather(inputs, 0, B.long())
    depth_a = torch.gather(targets, 0, A.long())
    depth_b = torch.gather(targets, 0, B.long())
    masks_A = torch.gather(masks, 0, A.long())
    masks_B = torch.gather(masks, 0, B.long())

    return rgb_a, rgb_b, depth_a, depth_b, masks_A, masks_B, sample_num
```

Executed at 2023.12.10 19:26:39 in 13ms

In _

```python
class EdgeguidedRankingLoss(nn.Module):
```

```python
    def __init__(self, point_pairs=10000, sigma=0.03, alpha=1.0, mask_value=-1e-8):
        super(EdgeguidedRankingLoss, self).__init__()
        self.point_pairs = point_pairs  # number of point pairs
        self.sigma = sigma  # used for determining the ordinal relationship between a selected pair
        self.alpha = alpha  # used for balancing the effect of = and (<,>)
        self.mask_value = mask_value
        # self.regularization_loss = GradientLoss(scales=4)

    def getEdge(self, images):
        n, c, h, w = images.size()
        a = torch.Tensor([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]]
                         ).cuda().view((1, 1, 3, 3)).repeat(1, 1, 1, 1)
        b = torch.Tensor([[1, 2, 1], [0, 0, 0], [-1, -2, -1]]
                         ).cuda().view((1, 1, 3, 3)).repeat(1, 1, 1, 1)
        if c == 3:
            gradient_x = F.conv2d(images[:, 0, :, :].unsqueeze(1), a)
            gradient_y = F.conv2d(images[:, 0, :, :].unsqueeze(1), b)
        else:
            gradient_x = F.conv2d(images, a)
            gradient_y = F.conv2d(images, b)
        edges = torch.sqrt(torch.pow(gradient_x, 2) + torch.pow(gradient_y, 2))
        edges = F.pad(edges, (1, 1, 1, 1), "constant", 0)
        thetas = torch.atan2(gradient_y, gradient_x)
        thetas = F.pad(thetas, (1, 1, 1, 1), "constant", 0)

        return edges, thetas

    def forward(self, inputs, targets, images, masks=None):
        if masks == None:
            masks = targets > self.mask_value
        # Comment this line if you don't want to use the multi-scale gradient matching term !!!
        # regularization_loss = self.regularization_loss(inputs.squeeze(1), targets.squeeze(1), masks.squeeze(1))
        # find edges from RGB
        edges_img, thetas_img = self.getEdge(images)

        # ============================
        n, c, h, w = targets.size()
        if n != 1:
            inputs = inputs.view(n, -1).double()
            targets = targets.view(n, -1).double()
            masks = masks.view(n, -1).double()
            edges_img = edges_img.view(n, -1).double()
            thetas_img = thetas_img.view(n, -1).double()

        else:
            inputs = inputs.contiguous().view(1, -1).double()
            targets = targets.contiguous().view(1, -1).double()
            masks = masks.contiguous().view(1, -1).double()
            edges_img = edges_img.contiguous().view(1, -1).double()
            thetas_img = thetas_img.contiguous().view(1, -1).double()

        # initialization
        loss = torch.DoubleTensor([0.0]).cuda()

        for i in range(n):
            # Edge-Guided sampling
            rgb_a, rgb_b, depth_a, depth_b, masks_A, masks_B, sample_num = edgeGuidedSampling(
                inputs[i, :], targets[i, :], edges_img[i], thetas_img[i], masks[i, :], h, w)
            # Random Sampling
            random_sample_num = sample_num
            random_rgb_a, random_rgb_b, random_depth_a, random_depth_b, random_masks_A, random_masks_B = onlineSampling(
                inputs[i, :], targets[i, :], masks[i, :], self.mask_value, random_sample_num)
```

```python

        # Combine EGS + RS
        rgb_a = torch.cat((rgb_a, random_rgb_a), 0)
        rgb_b = torch.cat((rgb_b, random_rgb_b), 0)
        depth_a = torch.cat((depth_a, random_depth_a), 0)
        depth_b = torch.cat((depth_b, random_depth_b), 0)
        masks_A = torch.cat((masks_A, random_masks_A), 0)
        masks_B = torch.cat((masks_B, random_masks_B), 0)

        # GT ordinal relationship
        target_ratio = torch.div(depth_a+1e-6, depth_b+1e-6)
        mask_eq = target_ratio.lt(
            1.0 + self.sigma) * target_ratio.gt(1.0/(1.0+self.sigma))
        labels = torch.zeros_like(target_ratio)
        labels[target_ratio.ge(1.0 + self.sigma)] = 1
        labels[target_ratio.le(1.0/(1.0+self.sigma))] = -1

        # consider forward-backward consistency checking, i.e, only compute losses of point pairs with valid GT
        consistency_mask = masks_A * masks_B

        equal_loss = (rgb_a - rgb_b).pow(2) * \
                     mask_eq.double() * consistency_mask
        unequal_loss = torch.log(
            1 + torch.exp((-rgb_a + rgb_b) * labels)) * (~mask_eq).double() * consistency_mask

        # Please comment the regularization term if you don't want to use the multi-scale gradient matching loss !!!
        # + 0.2 * regularization_loss.double()
        loss = loss + self.alpha * equal_loss.mean() + 1.0 * unequal_loss.mean()

    return loss[0].float()/n
```

Mini Batch Sampling

In 1
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from torchvision import transforms
from data.loaders.DataLoader import RedWebDataset , Rescale , RandomCrop , ToTensor
```
Executed at 2023.12.02 11:12:22 in 1s 485ms

```
/Users/adi/opt/anaconda3/lib/python3.10/site-packages/torchvision/io/image.py:13: UserWarning: Failed to load image Pyth
```

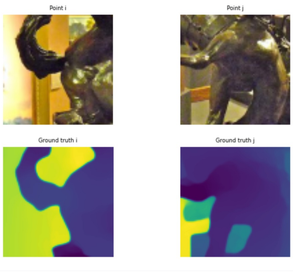
|# lets choose a sample image to show how mini batch sampling works to calculate ordinal relations

In 2
```
transformed_dataset = RedWebDataset(root_dir="../data/ReDWeb_V1",transform=transforms.Compose([
    Rescale(256),
    RandomCrop(225)
]))
```
Executed at 2023.12.02 11:12:22 in 41ms

In 3
```
sample_image = transformed_dataset[150]
fig,ax = plt.subplot_mosaic("AB")
ax["A"].imshow(sample_image["mono"])
ax["B"].imshow(sample_image["heat"])

for k in ax.keys():
    ax[k].axis("off")
    ax[k].grid("off")

plt.show()


```
Executed at 2023.12.02 11:12:22 in 106ms


In 4
```
# now we derive n ordinal relations from these images
# we will work out one such pair of example
# since we have random cropped and returned @ 225 . we can hardcode these values for any image

# Generate a random pixel
# low and high adjusting for window size. this would be adjusted and corrected while making the loss function
pi = np.random.randint(low=50, high=175, size= 2)
pj = np.random.randint(low=50, high=175, size=2)
```
Executed at 2023.12.02 11:12:22 in 3ms

In 5
```
# window padding to get idea about the region we are going to talk about
fig , ax = plt.subplot_mosaic("AB;CD")
ax["A"].set_title("Point i",fontsize=6)
ax["A"].imshow(sample_image["mono"][pi[0]-40 : pi[0]+40 , pi[1]-40:pi[1]+40 , :])
ax["B"].set_title("Point j",fontsize=6)
ax["B"].imshow(sample_image["mono"][pj[0]-40 : pj[0]+40 , pj[1]-40:pj[1]+40 , :])
ax["C"].set_title("Ground truth i",fontsize=6)
ax["C"].imshow(sample_image["heat"][pi[0]-40 : pi[0]+40 , pi[1]-40:pi[1]+40])
ax["D"].set_title("Ground truth j",fontsize=6)
ax["D"].imshow(sample_image["heat"][pj[0]-40 : pj[0]+40 , pj[1]-40:pj[1]+40])

for k in ax.keys():
    ax[k].axis("off")
    ax[k].grid("off")

```
Executed at 2023.12.02 11:12:22 in 117ms



In 6
```
# lets calculate the ordinal relation of the pair of ordinal relation we are going to calculate
gi =sample_image["heat"][pi[0],pi[1]]
gj =sample_image["heat"][pj[0],pj[1]]
print(gi,gj)
```
Executed at 2023.12.02 11:12:22 in 3ms

```
0.1798663350281251 0.08230662784977966
```

In 7
```
def ordinal_relation(gi,gj,sigma=0.02):
    if gi/gj > 1+sigma : return +1
    if gi/gj < 1+sigma : return -1
    return 0
```
Executed at 2023.12.02 11:12:23 in 2ms

In 8
```
# so the boundary of our ordinal relation pair is
[ordinal_relation(101,100) , ordinal_relation(102,100) , ordinal_relation(103,100)]
```
Executed at 2023.12.02 11:12:23 in 7ms

Out 8
```
[-1, 0, 1]
```

Out 8
[-1, 0, 1]

In 9
```
# here our ordinal relation will be
ordinal_relation(gi,gj)
```
Executed at 2023.12.02 11:12:23 in 22ms

Out 9
```
1
```

now we just repeat it n times for a given image to genreate such ordinal mappings

In 7

```python
import os
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
from torch.cuda.amp import GradScaler, autocast
from tqdm.auto import tqdm
import cv2 as cv
from PIL import Image
import segmentation_models_pytorch as smp
from sklearn.model_selection import train_test_split
import albumentations as A
from albumentations.pytorch import ToTensorV2
from torchmetrics.image import StructuralSimilarityIndexMeasure as SSIM
from torchmetrics.regression import MeanSquaredError as MSE
from torchmetrics.collections import MetricCollection
import gc
from torchvision.transforms import Normalize

import matplotlib.pyplot as plt
import torch
from torchvision import transforms
import torch.nn as nn


from torch.utils.data import DataLoader
from loss.StructureGuide import EdgeguidedRankingLoss

from data.loaders.DataLoader import RedWebDataset , Rescale , RandomCrop
normal_dataset = RedWebDataset(root_dir="../data/ReDWeb_V1",transform=transforms.Compose([
    Rescale((256,256)),
]))
batcher = DataLoader(normal_dataset,batch_size=1,shuffle=True)
```

In _

```python
import os

rgb_dir = "ReDWeb_V1/Imgs/"
depth_dir = "ReDWeb_V1/RDs/"
dataset_df = normal_dataset
```

In 9

```python
train_df = dataset_df[:3200]
test_df = dataset_df[3200:]
print(train_df.shape , test_df.shape)
```

```
(3200, 2) (400, 2)
```

In 10

```python
train_df, val_df = train_test_split(train_df,test_size=0.1,shuffle=True)
val_df, test_df = train_test_split(test_df, test_size=0.1,shuffle=True)
train_df.reset_index(drop=True,inplace=True)
val_df.reset_index(drop=True,inplace=True)
test_df.reset_index(drop=True,inplace=True)
len(train_df),len(val_df), len(test_df)
```

Out 10

```
(2880, 360, 40)
```

In 12

```python
sample_tfms = [
    A.HorizontalFlip(),
    A.GaussNoise(p=0.2),
    A.OneOf([
        A.MotionBlur(p=.3),
        A.MedianBlur(blur_limit=3, p=0.3),
        A.Blur(blur_limit=3, p=0.5),
    ], p=0.3),
    A.RGBShift(),
    A.RandomBrightnessContrast(),
    A.RandomResizedCrop(384,384),
    A.ColorJitter(),
    A.ShiftScaleRotate(shift_limit=0.1, scale_limit=0.3, rotate_limit=45, p=0.5),
    A.HueSaturationValue(p=0.3),
]
train_tfms = A.Compose([
    *sample_tfms,
    A.Resize(224,224),
    A.Normalize(always_apply=True),
    ToTensorV2()
],is_check_shapes=False)
valid_tfms = A.Compose([
    A.Resize(224,224),
    A.Normalize(always_apply=True),
    ToTensorV2()
],is_check_shapes=False)
```

In 14

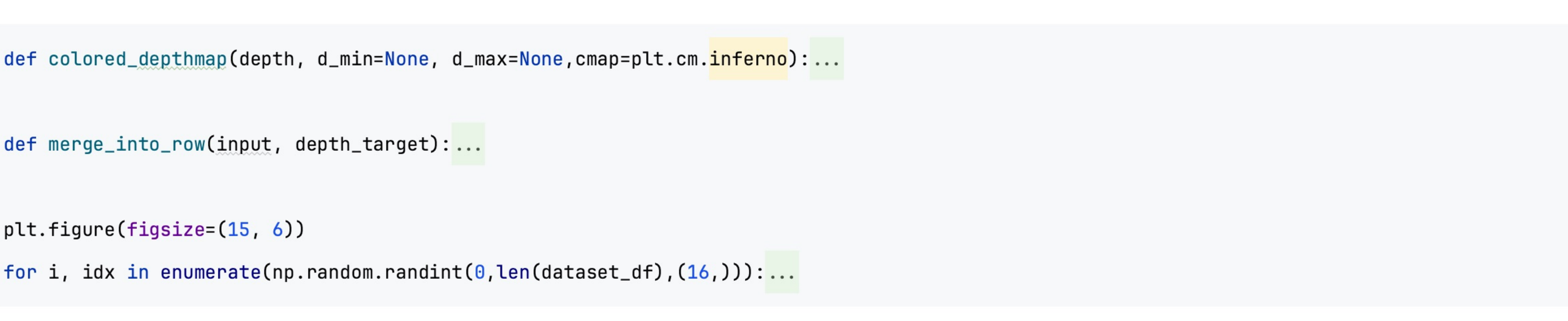
```python
train_ds = Dataset(train_df,train_tfms)
val_ds = Dataset(val_df,valid_tfms)
test_ds = Dataset(test_df, valid_tfms)
len(train_ds), len(val_ds), len(test_ds)
```

Out 14

```
(2880, 360, 40)
```

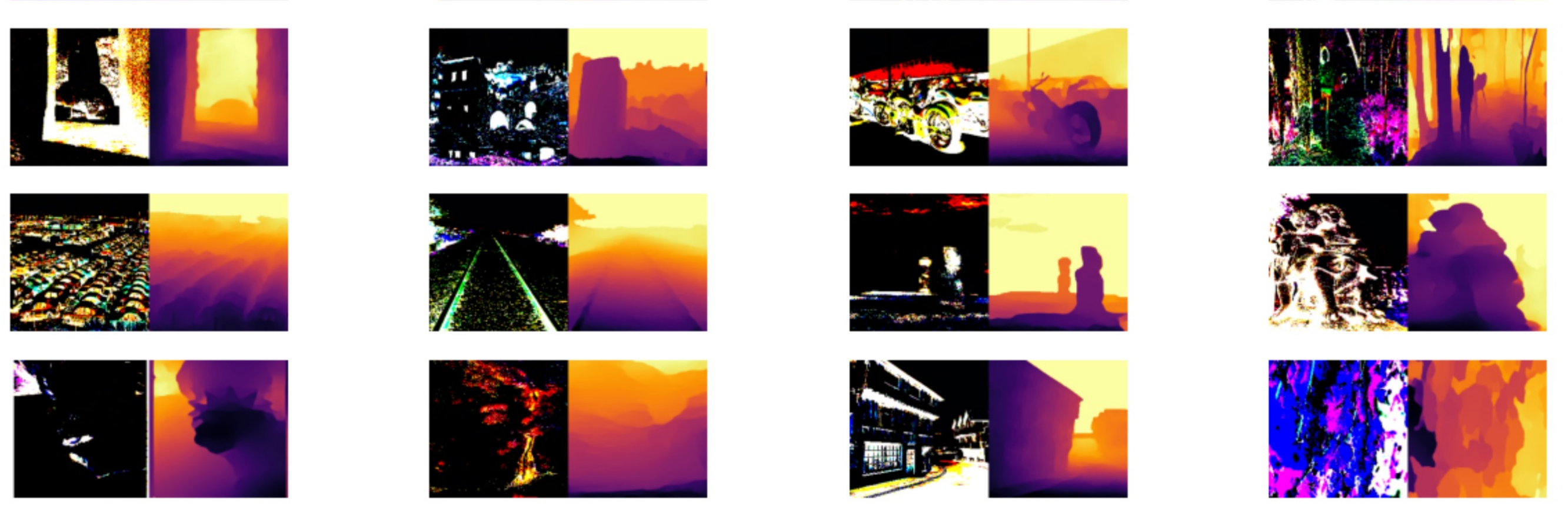
In 11

```python
def colored_depthmap(depth, d_min=None, d_max=None,cmap=plt.cm.inferno):...

def merge_into_row(input, depth_target):...

plt.figure(figsize=(15, 6))
for i, idx in enumerate(np.random.randint(0,len(dataset_df),(16,))):...
```



In 15

```python
randoms = [test_ds[idx] for idx in range(16)]
plt.figure(figsize=(15, 6))
for i, (img,mask) in enumerate(randoms):...
```



In 16

```python
train_dl = torch.utils.data.DataLoader(train_ds,shuffle=True,batch_size=64)
val_dl = torch.utils.data.DataLoader(val_ds,shuffle=False,batch_size=64)
test_dl = torch.utils.data.DataLoader(test_ds, shuffle=True, batch_size=4)
len(train_dl), len(val_dl), len(test_dl)
```

Out 16

```
(45, 6, 10)
```

In 17

```python

class ResnetFF(nn.Module):
    def __init__(self, encoder_name='resnext50_32x4d', in_channels=3, classes=1):
        super().__init__()
        self.encoder = smp.encoders.get_encoder(encoder_name, in_channels=in_channels)
        self.decoder = smp.unet.decoder.UnetDecoder(
            encoder_channels=self.encoder.out_channels,
            decoder_channels=(256, 128, 64, 32),
            n_blocks=(3, 2, 1, 0),
            use_batchnorm=True,
            center=False,
            attention_type=None,  # You can change this to 'scse' or 'cbam' for attention mechanisms
        )
        self.segmentation_head = smp.unet.model.UnetHead(
            in_channels=self.decoder.out_channels[-1], out_channels=classes, activation=None
        )

    def trainable_encoder(self, trainable=True):
        for p in self.encoder.parameters():
            p.requires_grad = trainable

    def forward(self, x):
        features = self.encoder(x)
        decoder_output = self.decoder(*features)
        logits = self.segmentation_head(decoder_output[-1])
        return logits

    def _num_params(self):
        return sum([p.numel() for p in self.parameters() if p.requires_grad])

model = ResnetFF(encoder_name='resnext50_32x4d', in_channels=3, classes=1)
```

In 18

```python
class UnNormalize(Normalize):...

@torch.no_grad()
def plot_vals(imgs, preds, targets,n=4,figsize=(6,2),title=''):...
```

In 19

```python
epochs = 5
freeze_epochs = 2
lr = 1e-3

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(device)

metrics = MetricCollection([
    SSIM(data_range=(0,1)),
    MSE()
]).to(device)
train_metrics = metrics.clone()
val_metrics = metrics.clone()

logs = pd.DataFrame()
logs[['loss_train','loss_val','ssim_train','ssim_val','mse_train','mse_val']]=None

model = model.to(device)
model.trainable_encoder(trainable=False)

loss_fn = EdgeguidedRankingLoss()
optim = torch.optim.AdamW(model.parameters(),lr=lr / 25.,weight_decay=0.02)
sched = torch.optim.lr_scheduler.OneCycleLR(optim,max_lr=lr,epochs=epochs,steps_per_epoch=(len(train_dl)))

scaler = GradScaler()
```

```
cuda
Downloading: "https://download.pytorch.org/models/resnext50_32x4d-7cdf4587.pth" to /root/
.cache/torch/hub/checkpoints/resnext50_32x4d-7cdf4587.pth
100%|██████████| 95.8M/95.8M [00:00<00:00, 303MB/s]
```

In 20

```python
best_ssim = -1e9
best_epoch = -1

print('training decoder only')

for epoch in tqdm(range(epochs)):

    model.train()

    if epoch == freeze_epochs:
        model.trainable_encoder(trainable=True)
        print('training encoder and decoder both')

    running_loss = 0.
    train_prog = tqdm(train_dl,total=len(train_dl))

    for img, mask in train_prog:

        with autocast():
            img, mask = img.to(device), mask.to(device)
            preds = model(img)

            loss = loss_fn(preds,mask)
            scaler.scale(loss).backward()
            scaler.unscale_(optim)
            nn.utils.clip_grad_norm_(model.parameters(), max_norm=2.0, norm_type=2)
            scaler.step(optim)
            scaler.update()
            sched.step()
            optim.zero_grad()

            running_loss += loss.item()

            train_prog.set_description(f'loss: {loss.item():.3f}')
            train_metrics(preds,mask)

        del img, mask, preds, loss

    m = train_metrics.compute()
    _ssim,_mse = m['StructuralSimilarityIndexMeasure'].cpu().item(), m['MeanSquaredError'].cpu().item()
    logs.loc[epoch,['loss_train','ssim_train','mse_train']] = (running_loss/len(train_dl),_ssim,_mse)
    train_metrics.reset()
    model.eval()

    with torch.no_grad():

        running_loss = 0.

        val_prog = tqdm(val_dl,total=len(val_dl))
        for img, mask in val_prog:

            with autocast():
                img, mask = img.to(device), mask.to(device)
                preds = model(img)
                loss = loss_fn(preds,mask)
                running_loss += loss.item()
                val_prog.set_description(f'loss: {loss.item():.3f}')

                val_metrics(preds, mask)

            del img, mask, preds, loss

        m = val_metrics.compute()
        _ssim,_mse = m['StructuralSimilarityIndexMeasure'].cpu().item(), m['MeanSquaredError'].cpu().item()
        logs.loc[epoch,['loss_val','ssim_val','mse_val']] = (running_loss/len(val_dl),_ssim,_mse)
        val_metrics.reset()

    if _ssim > best_ssim:
        best_ssim = _ssim
        best_epoch = epoch
        sd = model.state_dict()
        torch.save(sd,'ResnetFF.pt')

    print(f"\n\n{logs.tail(1)}\n\n")

    with torch.no_grad():
        with autocast():
            img, mask = next(iter(test_dl))
            img, mask = img.to(device), mask.to(device)
            preds = model(img)
            plot_vals(
                img.cpu(),
                preds.cpu(),
                mask.cpu()
            )

    gc.collect()
    torch.cuda.empty_cache()
```

```
training decoder only
0%| | 0/5 [00:00<?, ?it/s]

0%| | 0/45 [00:00<?, ?it/s]

/usr/local/lib/python3.10/dist-packages/torch/optim/lr_scheduler.py:136: UserWarning: Detected call of `lr_scheduler.step()` before
`optimizer.step()`. In PyTorch 1.1.0 and later, you should call them in the opposite order: `optimizer.step()` before `lr_scheduler.step
()`.  Failure to do this will result in PyTorch skipping the first value of the learning rate schedule. See more details at
https://pytorch.org/docs/stable/optim.html#how-to-adjust-learning-rate
  warnings.warn("Detected call of `lr_scheduler.step()` before `optimizer.step()`. "
0%| | 0/6 [00:00<?, ?it/s]


  loss_train loss_val ssim_train  ssim_val mse_train  mse_val
```

```
   loss_train loss_val ssim_train  ssim_val mse_train  mse_val
0   0.327318   0.2071   0.133579  0.425158  0.327318  0.20583
```

0%| | 0/45 [00:00<?, ?it/s]

0%| | 0/6 [00:00<?, ?it/s]

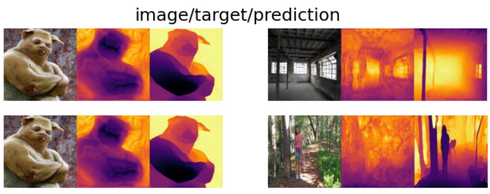
```
   loss_train  loss_val ssim_train  ssim_val mse_train   mse_val
1   0.050247  0.042086    0.58262  0.608989  0.050247  0.041778
```

training encoder and decoder both

0%| | 0/45 [00:00<?, ?it/s]

0%| | 0/6 [00:00<?, ?it/s]

```
   loss_train  loss_val ssim_train  ssim_val mse_train   mse_val
2   0.054153  0.047912   0.647623  0.622574  0.054153  0.047796
```



0%| | 0/45 [00:00<?, ?it/s]

0%| | 0/6 [00:00<?, ?it/s]

```
   loss_train  loss_val ssim_train  ssim_val mse_train   mse_val
3   0.047921  0.039307   0.680513  0.654263  0.047921  0.039069
```

0%| | 0/45 [00:00<?, ?it/s]

0%| | 0/6 [00:00<?, ?it/s]

```
   loss_train  loss_val ssim_train  ssim_val mse_train   mse_val
4   0.043474  0.038338   0.688893  0.655723  0.043474  0.038063
```

In 31

```
pip install dill
```

```
Collecting dill
  Downloading dill-0.3.7-py3-none-any.whl (115 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 115.3/115.3 kB 1.6 MB/s eta 0:00:00
Installing collected packages: dill
Successfully installed dill-0.3.7
```

In 33

```
import dill
filename = 'globalsave.pkl'
dill.dump_session(filename)
```

```
Traceback…
TypeError: no default __reduce__ due to non-trivial __cinit__
```

In 1
```
from google.colab import drive
import torch
drive.mount('/content/drive')
%cd /content/drive/MyDrive/DepthSense
```

```
Mounted at /content/drive
/content/drive/MyDrive/DepthSense
```

In 21
```
best_epoch
```

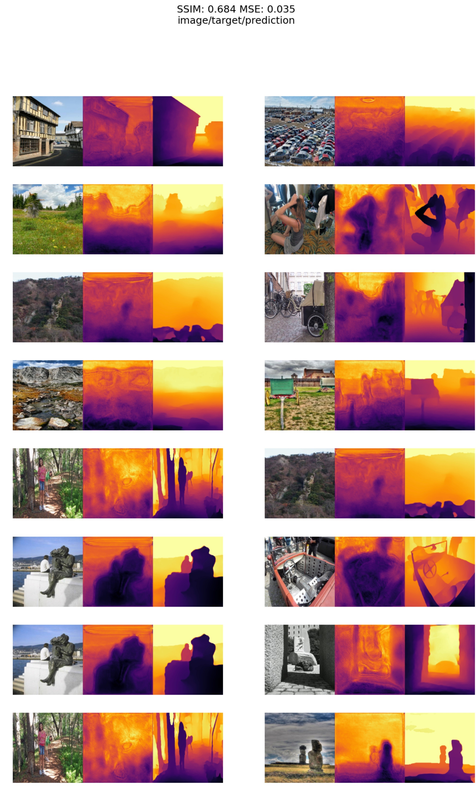
Out 21
```
4
```

In 23
```
from ResnetFF import ResnetFF
model = ResnetFF()
best_sd = torch.load('ResnetFF.pt')
model.load_state_dict(best_sd)
```

Out 23
```
<All keys matched successfully>
```

In 24
```
all_imgs, all_preds, all_targets = [], [], []
with torch.no_grad():
    with autocast():
        for img, mask in tqdm(test_dl,total=len(test_dl)):
            img, mask = img.to(device), mask.to(device)
            preds = model(img)
            all_imgs.append(img)
            all_preds.append(preds)
            all_targets.append(mask)

test_metrics = metrics.clone()
test_metrics(
    torch.vstack(all_preds),
    torch.vstack(all_targets)
)
m = test_metrics.compute()
title = f"SSIM: {m['StructuralSimilarityIndexMeasure'].cpu().item():.3f} MSE: {m['MeanSquaredError'].cpu().item():.3f}"
plot_vals(
    torch.vstack(all_imgs).cpu(),
    torch.vstack(all_preds).cpu(),
    torch.vstack(all_targets).cpu(),
    n=16,
    figsize=(10,15),
    title=title
)
```

```
0%| | 0/10 [00:00<?, ?it/s]
```

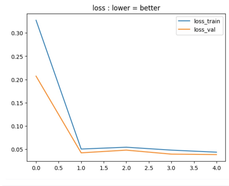


In 25
```
logs
```

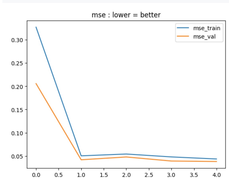
Out 25

	loss_train	loss_val	ssim_train	ssim_val	mse_train	mse_val
0	0.327318	0.20571	0.133579	0.425158	0.327318	0.20583
1	0.050247	0.042086	0.58262	0.608989	0.050247	0.041778
2	0.054153	0.047912	0.647623	0.622574	0.054153	0.047796
3	0.047921	0.039307	0.680513	0.654263	0.047921	0.039069
4	0.043474	0.038338	0.688893	0.655723	0.043474	0.038063

In 26
```
logs['loss_train'].plot()
logs['loss_val'].plot()
plt.title('loss : lower = better')
plt.legend()
plt.show()
```



In 27
```
logs['mse_train'].plot()
logs['mse_val'].plot()
plt.title('mse : lower = better')
plt.legend()
plt.show()
```



In 28
```
logs['ssim_train'].plot()
logs['ssim_val'].plot()
plt.title('ssim : higher = better')
plt.legend()
plt.show()
```