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

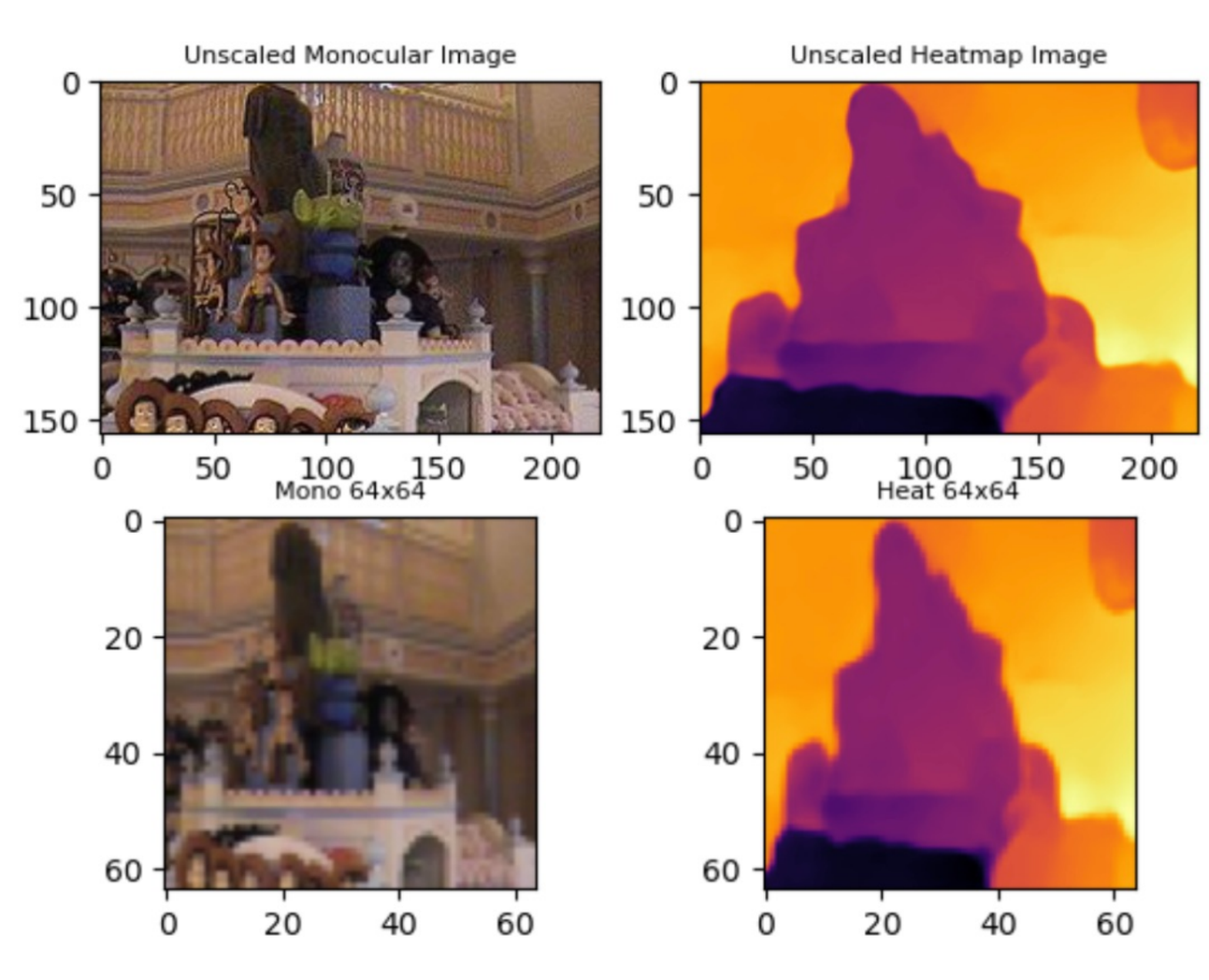
i) A rescaler

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>
```



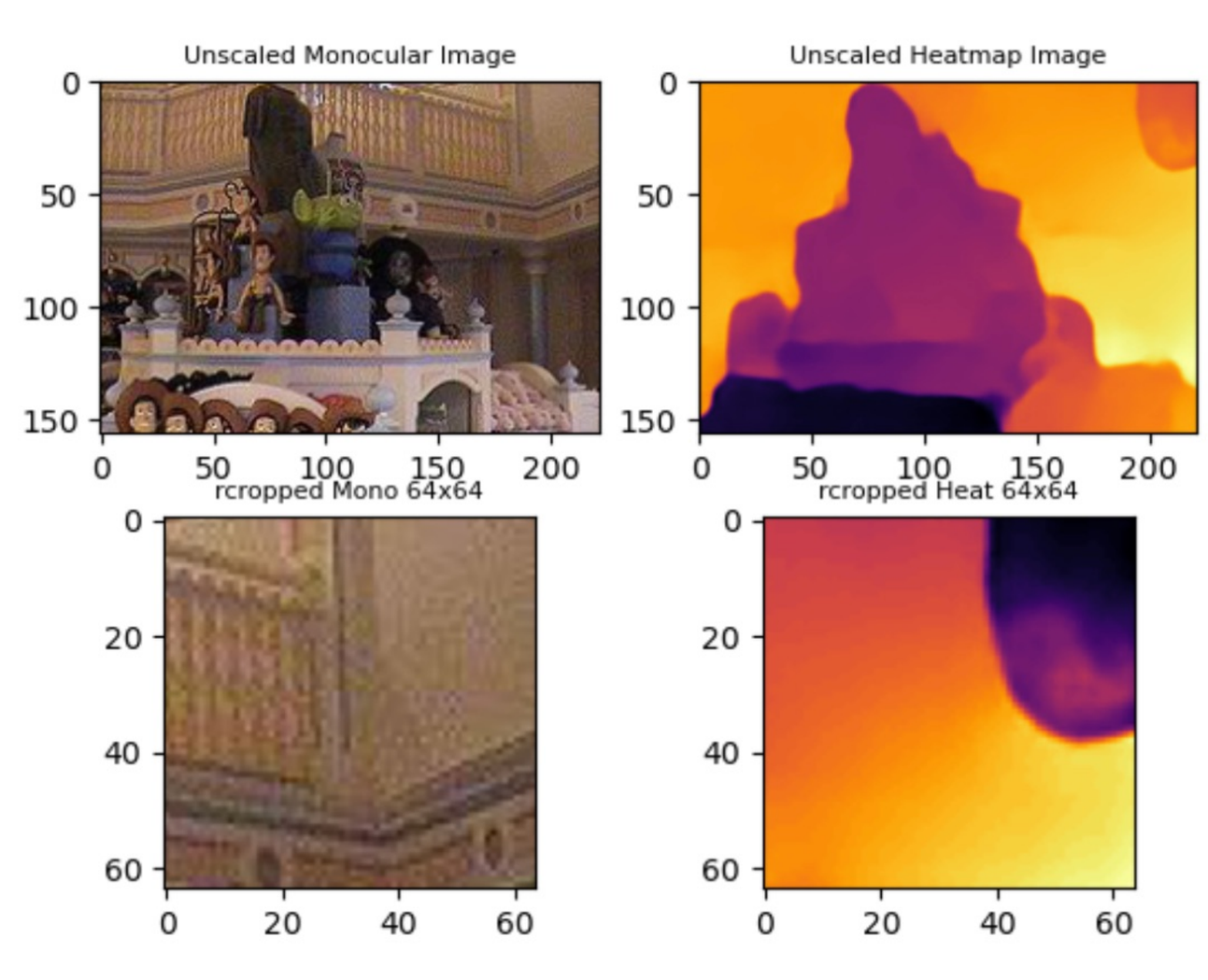
ii) A random cropper

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

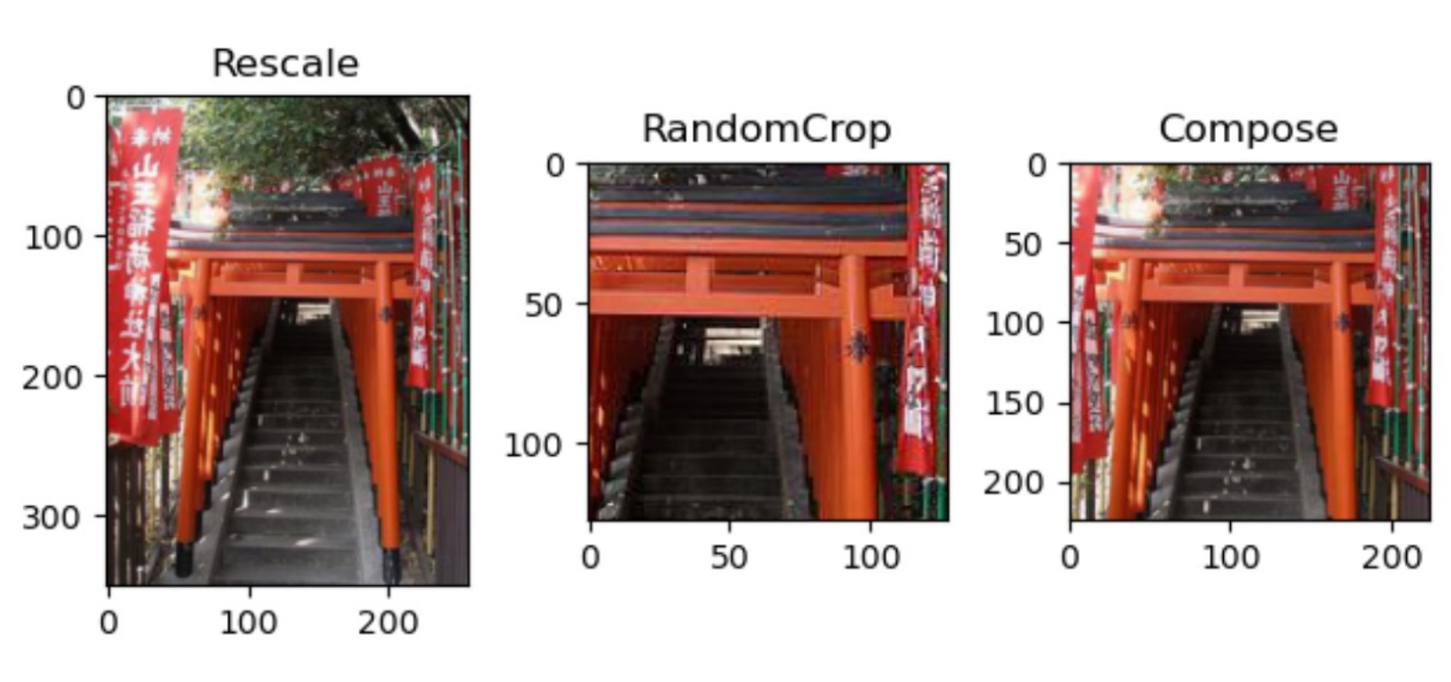
In 18
```python
fig , ax = plt.subplot_mosaic("AB;CD")
ax["A"].set_title(f"Unscaled Monocular Image",fontsize=8)
ax["A"].imshow(loader[1234]["mono"])

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>
```



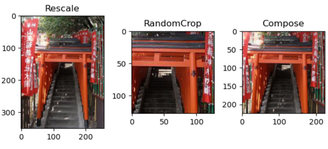
You can composite transforms composed = (rescale o cropped)(img)

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

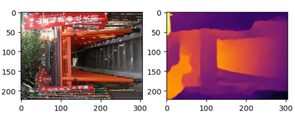
```
<class 'torch.Tensor'> <class 'torch.Tensor'>
```

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>
```

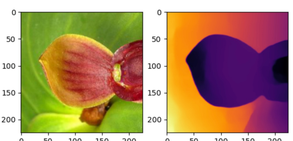


In 24
```
# and now we can iterate through the dataset combining all the operations we have made thus far
transformed_dataset = RedWebDataset(root_dir="../ReDWeb_V1",transform=transforms.Compose([
    Rescale(256),
    RandomCrop(225),
    ToTensor()
]))

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)}
```

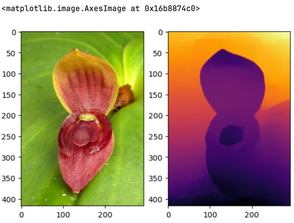
In 25
```
# this is how they look after composing transformations and converting them into tensor objects
fig,ax = plt.subplot_mosaic("AB")
ax["A"].imshow(sample["mono"].T)
ax["B"].imshow(sample["heat"].T,cmap="inferno")
```
Executed at 2023.12.01 22:50:34 in 119ms

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