

```
In 1 1 from __future__ import print_function, division
2 import os
3 import torch
4 import pandas as pd
5 from skimage import io, transform
6 import numpy as np
7 import matplotlib.pyplot as plt
8 from torch.utils.data import Dataset, DataLoader
9 from torchvision import transforms, utils
10
11 # Ignore warnings
12 import warnings
13 warnings.filterwarnings("ignore")
14
15 plt.ion() # interactive mode
Executed at 2023.12.01 22:50:33 in 97ms
> /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 0x10657f20>

```
In 2 1 # from my utils notebook
2 def add_to_class(klass):
3     """Register them functions"""
4     def wrapper(obj):
5         # setattr(object, name, value) → sets the value of the attribute
6         setattr(klass, obj.__name__, obj)
7     return wrapper
Executed at 2023.12.01 22:50:33 in 4ms
```

```
In 3 1 class RedWebDataset(Dataset):
2     def __init__(self, root_dir, transform=None):
3         """
4             Root Dir is REDWEB_V1
5                 Imgs → Monocular Images
6                 RDs → Corresponding Response (Heatmap)
7         """
8         self.root_dir = root_dir
9         self.transform = transform
10        self.monocular_folder = os.path.join(root_dir, 'Imgs')
11        self.heatmap_folder = os.path.join(root_dir, 'RDs')
12
13        # Monocular images are in jpgs and the heatmap images are in pngs
14        self.monocular_images = sorted(os.listdir(self.monocular_folder))
15        self.heatmap_images = sorted(os.listdir(self.heatmap_folder))
16
Executed at 2023.12.01 22:50:33 in 2ms
```

now we can get images from our loaders:

```
In 4 1 loader = RedWebDataset("../RedWeb_V1")
2 loader.monocular_images[:2]
Executed at 2023.12.01 22:50:33 in 58ms
```

Out 4 ['1014775965_3367569158.jpg', '10147764085_481cd7d72f.jpg']

```
In 5 1 loader.heatmap_images[:2]
```

Executed at 2023.12.01 22:50:33 in 16ms

Out 5 ['1014775965_3367569158.png', '10147764085_481cd7d72f.png']

```
In 6 1 # add a simple __len__ method
2 @add_to_class(RedWebDataset)
3 def __len__(self):
4     assert len(self.monocular_images) == len(self.heatmap_images), "Hein?"
5     return len(self.monocular_images)
Executed at 2023.12.01 22:50:33 in 12ms
```

In 7 1 # now we can call len on our loader

2 len(loader)

Executed at 2023.12.01 22:50:33 in 10ms

Out 7 3600

```
In 8 1 # lets write for practice that makes a subplot which shows
2 # image and their corresponding heatmap on a single subplot
3
4 @add_to_class(RedWebDataset)
5 def _show_sample(self, name = None):
6
7     if name:
8         name = name if "." not in name else name.split(".")[0]
9
10    # show random index
11    rand_index = np.random.randint(low=0,high = len(self) , size=1).item()
12    name = self.monocular_images[rand_index].split(".")[0]
13
14    # read images
15    # mono_img = plt.imread(loader.monocular_folder +"/"+ name + ".jpg")
16    # heat_img = plt.imread(loader.heatmap_folder +"/"+ name + ".png")
17    mono_img = io.imread(self.monocular_folder +"/"+ name + ".jpg")
18    heat_img = io.imread(self.heatmap_folder +"/"+ name + ".png")
19
20    fig, ax = plt.subplot_mosaic("AB")
21    ax["A"].set_title(f"Monocular Image({name})", fontsize=8)
22    ax["A"].imshow(mono_img)
23    ax["B"].set_title(f"Heatmap Image({name})", fontsize=8)
24    ax["B"].imshow(heat_img, cmap="inferno")
25    ax["A"].grid(False)
26    ax["B"].grid(False)
27    ax["A"].axis("off")
28    ax["B"].axis("off")
29    plt.show()
Executed at 2023.12.01 22:50:33 in 7ms
```

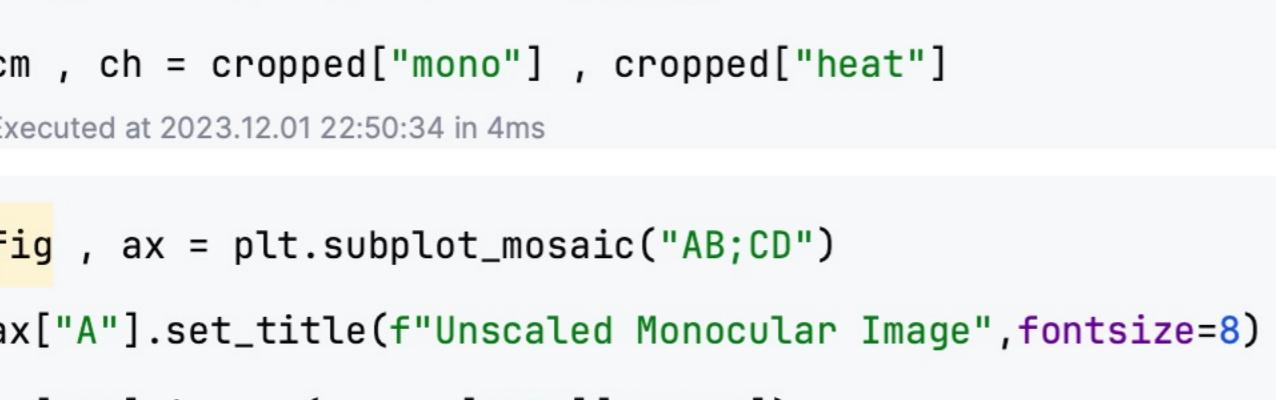
In 9 1 loader._show_sample()

Executed at 2023.12.01 22:50:33 in 96ms



In 10 1 loader._show_sample()

Executed at 2023.12.01 22:50:33 in 94ms



In 11 1 # now lastly just implement __getitem__

2 @add_to_class(RedWebDataset)

3 def __getitem__(self, idx):

4 if torch.is_tensor(idx):

5 idx = idx.tolist()

6
7 monocular_image = io.imread(self.monocular_folder +"/"+ self.monocular_images[idx])

8 heatmap_image = io.imread(self.heatmap_folder +"/"+ self.heatmap_images[idx])

9 sample = {'mono': monocular_image, 'heat': heatmap_image }

10
11 if self.transform:

12 sample = self.transform(sample)

13
14 return sample

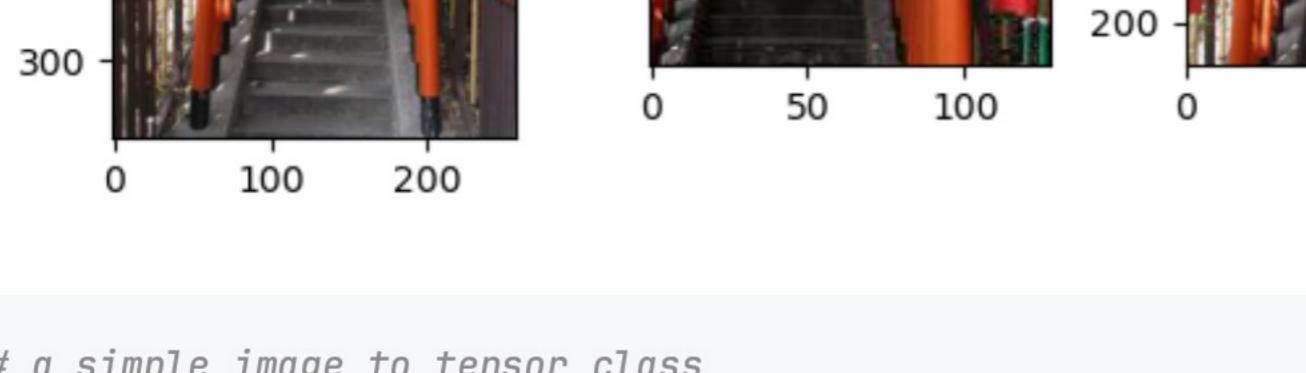
Executed at 2023.12.01 22:50:33 in 6ms

In 12 1 # now we can call index on our loader

2 plt.imshow(loader[1234]["mono"])

Executed at 2023.12.01 22:50:33 in 145ms

Out 12 <matplotlib.image.AxesImage at 0x10657f20>



Now lets Develop some useful data transformation

i) A rescaler

```
In 13 1 class Rescale(object):
2     """
3         simple Rescaler for my monocular and heatmap images
4     """
5     def __init__(self, output_size):
6         assert isinstance(output_size, (int, tuple))
7         self.output_size = output_size
8
9     def __call__(self, sample):
10        mono, heat = sample['mono'], sample['heat']
11        h, w = mono.shape[:2]
12
13        if isinstance(self.output_size, int):
14            if h > w:
15                new_h, new_w = self.output_size * h / w, self.output_size
16            else:
17                new_h, new_w = self.output_size, self.output_size * w / h
18            else:
19                new_h, new_w = self.output_size
20
21        new_h, new_w = int(new_h), int(new_w)
22
23        mono = transform.resize(mono, (new_h, new_w))
24        heat = transform.resize(heat, (new_h, new_w))
25
26
27        return {'mono': mono, 'heat': heat}
Executed at 2023.12.01 22:50:33 in 4ms
```

In 14 1 rescaler_64 = Rescale((64,64))

2 scaled = rescaler_64(loader[1234])

3 m64, h64 = scaled["mono"], scaled["heat"]

Executed at 2023.12.01 22:50:33 in 6ms

In 15 1 fig, ax = plt.subplot_mosaic("AB;CD")

2 ax["A"].set_title(f"Unscaled Monocular Image", fontsize=8)

3 ax["A"].imshow(loader[1234]["mono"])

4
5 ax["B"].set_title(f"Unscaled Heatmap Image", fontsize=8)

6 ax["B"].imshow(loader[1234]["heat"], cmap="inferno")

7
8 ax["C"].set_title(f"Mono 64x64", fontsize=8)

9 ax["C"].imshow(m64)

10
11 ax["D"].set_title(f"Heat 64x64", fontsize=8)

12 ax["D"].imshow(h64, cmap="inferno")

13
14 plt.show()

Executed at 2023.12.01 22:50:34 in 239ms

Out 15 <matplotlib.image.AxesImage at 0x10657f20>

ii) A random cropper

In 16 1 class RandomCrop(object):

2 """
3 Crop randomly the image in a sample.
4 """
5 def __init__(self, output_size):
6 assert isinstance(output_size, (int, tuple))
7 if isinstance(output_size, int):
8 self.output_size = (output_size, output_size)
9 else:
10 assert len(output_size) == 2
11 self.output_size = output_size
12
13 def __call__(self, sample):
14 mono, heat = sample['mono'], sample['heat']
15
16 h, w = mono.shape[:2]
17 new_h, new_w = self.output_size
18
19 top = np.random.randint(0, h - new_h)
20 left = np.random.randint(0, w - new_w)
21
22 mono = mono[top:top + new_h, left:left + new_w]
23 heat = heat[top:top + new_h, left:left + new_w]
24
25
26 return {'mono': mono, 'heat': heat}

Executed at 2023.12.01 22:50:34 in 9ms

In 17 1 # a random cropper which randomly crops and returns back an image of 64x64

2 r_cropper = RandomCrop((64,64))

3 cropped = r_cropper(loader[1234])

4 cm, ch = cropped["mono"], cropped["heat"]

Executed at 2023.12.01 22:50:34 in 4ms

In 18 1 fig, ax = plt.subplot_mosaic("AB;CD")

2 ax["A"].set_title(f"Unscaled Monocular Image", fontsize=8)

3 ax["A"].imshow(loader[1234]["mono"])

4
5 ax["B"].set_title(f"Unscaled Heatmap Image", fontsize=8)

6 ax["B"].imshow(loader[1234]["heat"], cmap="inferno")

7
8 ax["C"].set_title(f"rcropped Mono 64x64", fontsize=8)

9 ax["C"].imshow(cm)

10
11 ax["D"].set_title(f"rcropped Heat 64x64", fontsize=8)

12 ax["D"].imshow(ch, cmap="inferno")

13
14 plt.show()

Executed at 2023.12.01 22:50:34 in 249ms

Out 18 <matplotlib.image.AxesImage at 0x10657f20>

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

In 19 1 scale = Rescale(256)

2 crop = RandomCrop(128)

3 composed = transforms.Compose([Rescale(256),

4 RandomCrop(128)])

5
6 # Apply each of the above transforms on sample.
7 fig = plt.figure()
8 sample = loader[65] # lets start using a different example now
9 for tsfrm in enumerate([scale, crop, composed]):
10 transformed_sample = tsfrm(sample)
11
12 ax = plt.subplot(1, 3, i + 1)
13 plt.tight_layout()
14 ax.set_title(type(tsfrm).__name__)
15 plt.imshow(transformed_sample["mono"])
16
17 plt.show()

Executed at 2023.12.01 22:50:34 in 239ms

In 20 1 # a simple image to tensor class

2 class ToTensor(object):

3
4 def __init__(self):
5 pass
6
7 def __call__(self, sample):
8 mono, heat = sample['mono'], sample['heat']
9
10 mono = np.array(mono)
11 heat = np.array(heat)
12
13 mono = torch.from_numpy(mono).float()
14 heat = torch.from_numpy(heat).float()
15
16 sample = {'mono': mono, 'heat': heat}
17
18 return sample
19
20 def __repr__(self):
21 return self.__class__.__name__
22
23 def __getstate__(self):
24 return self.__dict__
25
26 def __setstate__(self, state):
27 self.__dict__.update(state)
28
29 def __hash__(self):
30 return id(self)
31
32 def __eq__(self, other):
33 if type(self) != type(other):
34 return False
35
36 if self.__dict__ != other.__dict__:
37 return False
38
39 return True
40
41 def __ne__(self, other):
42 if type(self) != type(other):
43 return True
44
45 if self.__dict__ != other.__dict__:
46 return True
47
48 return False
49
50 def __str__(self):
51 return self.__class__.__name__
52
53 def __repr__(self):
54 return self.__class__.__name__
55
56 def __getstate__(self):
57 return self.__dict__
58
59 def __setstate__(self, state):
60 self.__dict__.update(state)
61
62 def __hash__(self):
63 return id(self)
64
65 def __eq__(self, other):
66 if type(self) != type(other):
67 return False
68
69 if self.__dict__ != other.__dict__:
70 return False
71
72 return True
73
74 def __ne__(self, other):
75 if type(self) != type(other):
76 return True
77
78 if self.__dict__ != other.__dict__:
79 return True
80
81 return False
82
83 def __str__(self):
84 return self.__class__.__name__
85
86 def __repr__(self):
87 return self.__class__.__name__
88
89 def __getstate__(self):
90 return self.__dict__
91
92 def __setstate__(self, state):
93 self.__dict__.update(state)
94
95 def __hash__(self):
96 return id(self)
97
98 def __eq__(self, other):
99 if type(self) != type(other):
100 return False
101
102 if self.__dict__ != other.__dict__:
103 return False
104
105 return True
106
107 def __ne__(self, other):
108 if type(self) != type(other):
109 return True
110
111 if self.__dict__ != other.__dict__:
112 return True
113
114 return False
115
116 def __str__(self):
117 return self.__class__.__name__
118
119 def __repr__(self):
120 return self.__class__.__name__
121
122 def __getstate__(self):
123 return self.__dict__
124
125 def __setstate__(self, state):
126 self.__dict__.update(state)
127
128 def __hash__(self):
129 return id(self)
130
131 def __eq__(self, other):
132 if type(self) != type(other):
133 return False
134
135 if self.__dict__ != other.__dict__:
136 return False
137
138 return True
139
140 def __ne__(self, other):
141 if type(self) != type(other):
142 return True
143
144 if self.__dict__ != other.__dict__:
145 return True
146
147 return False
148
149 def __str__(self):
150 return self.__class__.__name__
151
152 def __repr__(self):
153 return self.__class__.__name__
154
155 def __getstate__(self):
156 return self.__dict__
157
158 def __setstate__(self, state):
159 self.__dict__.update(state)
160
161 def __hash__(self):
162 return id(self)
163
164 def __eq__(self, other):
165 if type(self) != type(other):
166 return False
167
168 if self.__dict__ != other.__dict__:
169 return False
170
171 return True
172
173 def __ne__(self, other):
174 if type(self) != type(other):
175 return True
176
177 if self.__dict__ != other.__dict__:
17

```
8 sample = loader[65] # lets start using a different example now
9 for i, tsfrm in enumerate([scale, crop, composed]):
10     transformed_sample = tsfrm(sample)
11
12     ax = plt.subplot(1, 3, i + 1)
13     plt.tight_layout()
14     ax.set_title(type(tsfrm).__name__)
15     plt.imshow(transformed_sample["mono"])
16
17 plt.show()
```

Executed at 2023.12.01 22:50:34 in 235ms



```
In 20 1 # a simple image to tensor class
2 class ToTensor(object):
3     """Convert ndarrays in sample to Tensors."""
4
5     def __call__(self, sample):
6         mono, heat = sample['mono'], sample['heat']
7
8         # for monocular image :
9         # swap color axis because
10        # numpy image: H x W x C
11        # torch image: C X H X W
12        mono = mono.transpose((2, 0, 1))
13
14        # dont have to do anything for single channel image
15
16        return {'mono': torch.from_numpy(mono),
17                'heat': torch.from_numpy(heat)}
```

Executed at 2023.12.01 22:50:34 in 4ms

```
In 21 1 to_tensor = ToTensor()
2 tensored= to_tensor(loader[65])
3 print(type(tensored["mono"]) , type(tensored["heat"]))
4
5 Executed at 2023.12.01 22:50:34 in 7ms
```

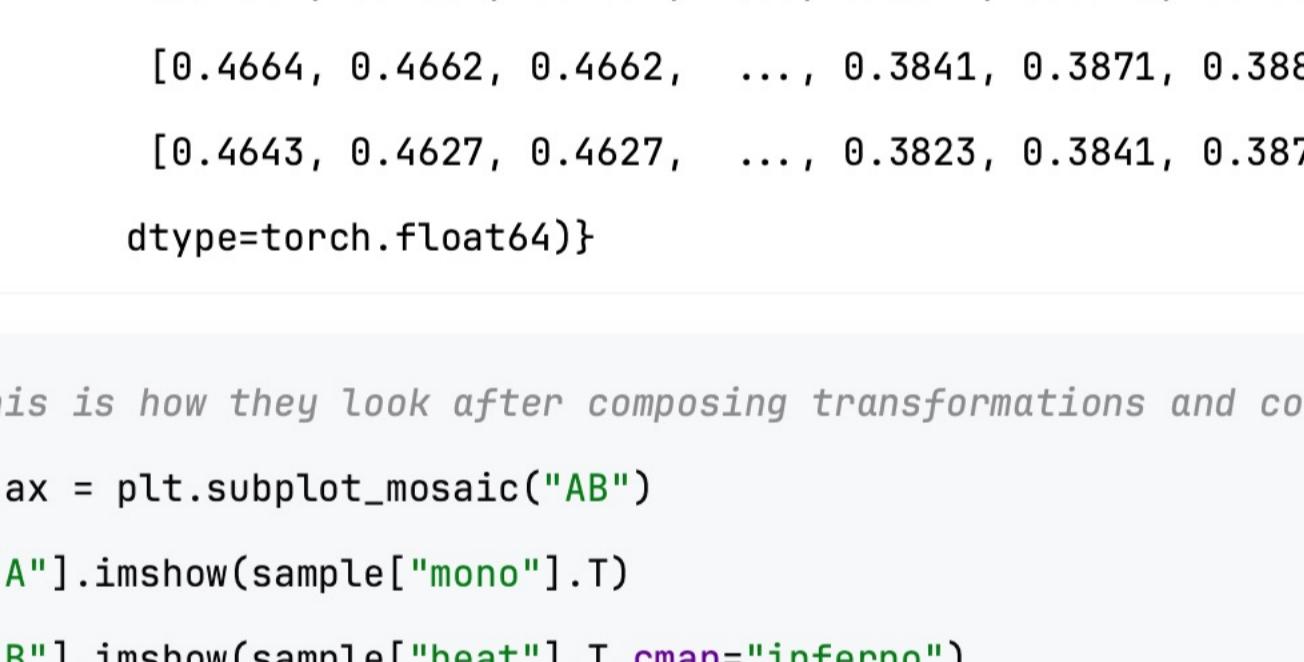
<class 'torch.Tensor'> <class 'torch.Tensor'>

```
In 22 1 # and their shape also must be the same in height and width .
2 # (channel , height , width) ← torch.img
3 print(tensored["mono"].shape , tensored["heat"].shape)
4
5 Executed at 2023.12.01 22:50:34 in 12ms
```

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

```
In 23 1 # but that shouldnt change anything .pairwise
2 # need to plot its transpose as plt.subplots does not take channel as its first dimension
3 fig,ax = plt.subplot_mosaic("AB")
4 ax["A"].imshow(tensored["mono"].T)
5 ax["B"].imshow(tensored["heat"].T,cmap="inferno")
6
7 Executed at 2023.12.01 22:50:34 in 109ms
```

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



```
In 24 1 # and now we can iterate through the dataset combining all the operations we have made thus far
2 transformed_dataset = RedWebDataset(root_dir="../ReDWeb_V1",transform=transforms.Compose([
3     Rescale(256),
4     RandomCrop(225),
5     ToTensor()
6 ]))
7
8 sample = transformed_dataset[99]
9 sample
10
11 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],

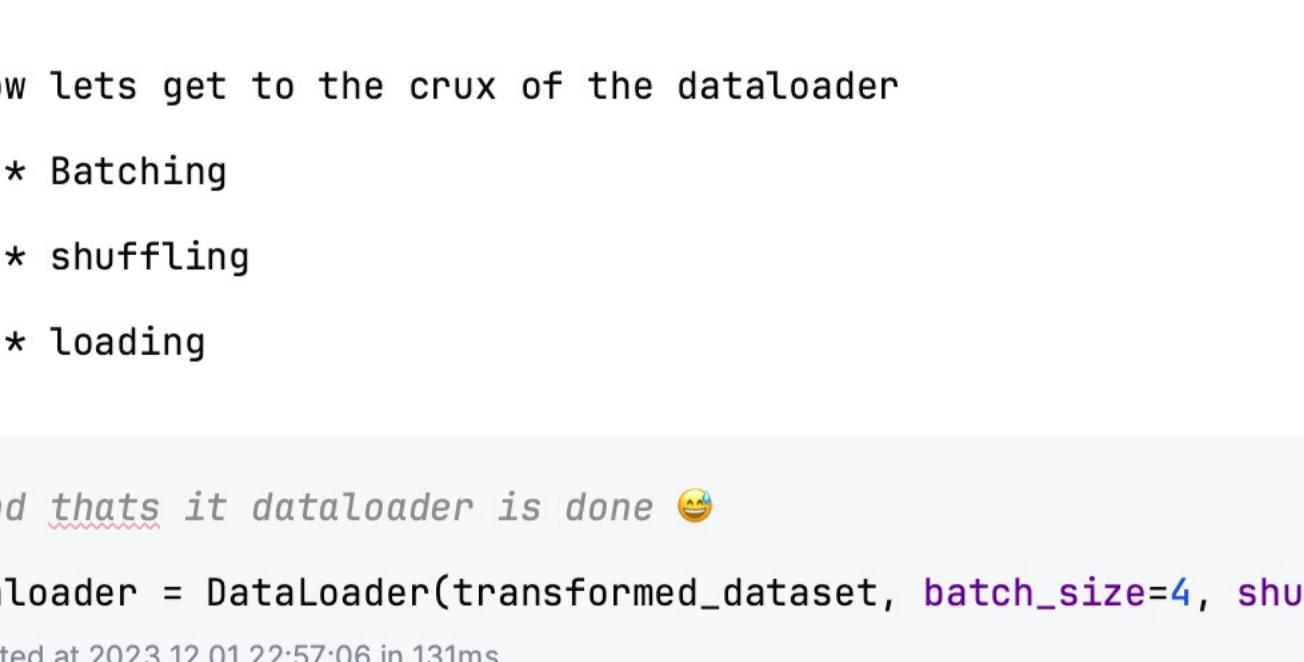
[0.4664, 0.4662, 0.4662, ..., 0.3841, 0.3871, 0.3882],

[0.4643, 0.4627, 0.4627, ..., 0.3823, 0.3841, 0.3870]],

dtype=torch.float64)}

In 25 1 # this is how they look after composing transformations and converting them into tensor objects
2 fig,ax = plt.subplot_mosaic("AB")
3 ax["A"].imshow(sample["mono"].T)
4 ax["B"].imshow(sample["heat"].T,cmap="inferno")
5
6 Executed at 2023.12.01 22:50:34 in 119ms

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



In 26 1 # this is how they (originally) looked from our simple class loader
2 fig,ax = plt.subplot_mosaic("AB")
3 ax["A"].imshow(loader[99]["mono"])
4 ax["B"].imshow(loader[99]["heat"],cmap="inferno")
5
6 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 1 # and thats it dataloader is done 😊
2 dataloader = DataLoader(transformed_dataset, batch_size=4, shuffle=True, num_workers=4) # but an iterable . cant call getitem on this obviously
3
4 Executed at 2023.12.01 22:57:06 in 131ms
```