Lab 6 - Image, Datasets, Dataloaders, Augmentation

Part I - Images

Images in Python are usually represented as ndarrays. PyTorch, of course, supports also the image as a Tensor.

Depending upon the library, images may be encoded differently:

- PIL uses, by default, the RGB encoding. The image can be represented as a $h \times w \times 3$ ndarray by calling the `.asarray()` method of numpy
- opencv uses, by default, the BGR encoding (reverse than RGB). The image, in Python, is directly stored as a $h \times w \times 3$ ndarray.
- PyTorch prefers images to be stored as $3 \times h \times w$ tensors.

Reading an image

- With PIL, images can be read with the `PIL.Image.open(path)` method
- torchvision, a subpackage of PyTorch for Computer Vision, has a method `io.read_image(path)` for reading images as JPEG or PNG. For other formats, we must resort to PIL.

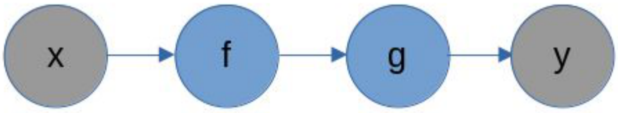
In [1]:

```python
import torchvision
import torch
from PIL import Image
import numpy as np
import os

img = Image.open("imgs/02/compgra1.jpg")
print(type(img)) # the image is a type on its own
img # I can visualize the image directly like this. No need for matplotlib or other auxiliary libs
```

```
<class 'PIL.JpegImagePlugin.JpegImageFile'>
```

Out[1]:



to print the raw content of an image, we must call `np.asarray` on it.

In [2]:

```python
img_array = np.asarray(img)
print(img_array.shape)
img_array
```

```
(115, 472, 3)
```

Out[2]:

```
array([[[255, 255, 255],
        [255, 255, 255],
        [255, 255, 255],
        ...,
        [255, 255, 255],
        [255, 255, 255],
        [255, 255, 255]],

       [[255, 255, 255],
        [255, 255, 255],
        [255, 255, 255],
        ...,
        [255, 255, 255],
        [255, 255, 255],
        [255, 255, 255]],

       [[255, 255, 255],
        [255, 255, 255],
        [255, 255, 255],
        ...,
        [255, 255, 255],
        [255, 255, 255],
        [255, 255, 255]],

       ...,

       [[255, 255, 255],
        [255, 255, 255],
        [255, 255, 255],
        ...,
        [255, 255, 255],
        [255, 255, 255],
        [255, 255, 255]],

       [[255, 255, 255],
        [255, 255, 255],
        [255, 255, 255],
        ...,
        [255, 255, 255],
        [255, 255, 255],
        [255, 255, 255]],

       [[255, 255, 255],
        [255, 255, 255],
        [255, 255, 255],
        ...,
        [255, 255, 255],
        [255, 255, 255],
        [255, 255, 255]]], dtype=uint8)
```

In [3]:

```python
img_torch = torchvision.io.read_image("imgs/03/dataloader01.jpg")
print(type(img_torch), img_torch.shape, "\n", img_torch)
# this time the image IS a tensor
# take a look at the shape
```

```
<class 'torch.Tensor'> torch.Size([3, 319, 600])
 tensor([[[255, 255, 255,  ..., 255, 255, 255],
         [255, 255, 255,  ..., 255, 255, 255],
         [255, 255, 255,  ..., 255, 255, 255],
         ...,
         [255, 255, 255,  ..., 255, 255, 255],
         [255, 255, 255,  ..., 255, 255, 255],
         [255, 255, 255,  ..., 255, 255, 255]],

        [[255, 255, 255,  ..., 255, 255, 255],
         [255, 255, 255,  ..., 255, 255, 255],
         [255, 255, 255,  ..., 255, 255, 255],
         ...,
         [255, 255, 255,  ..., 255, 255, 255],
         [255, 255, 255,  ..., 255, 255, 255],
         [255, 255, 255,  ..., 255, 255, 255]],

        [[255, 255, 255,  ..., 255, 255, 255],
         [255, 255, 255,  ..., 255, 255, 255],
         [255, 255, 255,  ..., 255, 255, 255],
         ...,
         [255, 255, 255,  ..., 255, 255, 255],
         [255, 255, 255,  ..., 255, 255, 255],
         [255, 255, 255,  ..., 255, 255, 255]]], dtype=torch.uint8)
```

Visualizing the image from a Tensor

In order to visualize the image, we can convert it to a PIL Image. But how?

In [4]:

```python
def tensor2PIL(tensor:torch.Tensor):

    return Image.fromarray(tensor.permute(1,2,0).numpy()) #change the order of dimensions and convert it into a tenso
    # your code here
```

Datasets

For this lab, we will use a custom dataset for classifying cats and dogs. It is a subset of the famous dataset from the cats vs. dogs Kaggle challenge. You can find it in `data/catsdogs`.

In [5]:

```
folder1 = "imgs/"
folder2 = "2"
folder1 + "/" + folder2 # I will habve ba double /, so to merge use os.path.join
```

Out[5]:

```
'imgs//2'
```

In [6]:

```
catsdogs_viz = [Image.open(os.path.join("data/catsdogs", im)) for im in os.listdir("data/catsdogs") if im.endswith(".
print("Tot images", len(catsdogs_viz))
```

```
Tot images 10
```

Let's have a look at the data

In [7]:

```
catsdogs_viz[1]
```

Out[7]:

In [8]:

```
catsdogs_viz[2]
```

Out[8]:

Building a custom dataset

Let us use our knowledge to build a custom dataset out of these images

In [9]:

```
class CatsVsDogsDataset(torch.utils.data.Dataset):
    def __init__(self, root):
        # I don't load it as PIL, but I use torchvision.io.read_image
        self.data = [torchvision.io.read_image(os.path.join(root, im)) for im in sorted(os.listdir(root)) if im.endsw
        #The alphabetical order of the images is not the same order in which imgs are stored in the file system!!!!
        self.labels = self._get_labels(os.path.join(root, "labels.txt"))

    def _get_labels(self, txt_path):
        with open(txt_path, "r") as f: #load the text file
            labels = [int(line.strip()) for line in f]
            #strip() is a sefatey method which removes whitespaces before and after something
        return labels

    def __len__(self):
        return len(self.data)

    def __getitem__(self, index):
        return self.data[index], self.labels[index]
```

now, the lazy version...

In [10]:

```
class CatsVsDogsDatasetLazy(torch.utils.data.Dataset):
    def __init__(self, root):
        self.data = [os.path.join(root, im) for im in sorted(os.listdir(root)) if im.endswith(".jpg")]
        self.labels = self._get_labels(os.path.join(root, "labels.txt"))

    def _get_labels(self, txt_path):
        with open(txt_path, "r") as f:
            labels = [int(line.strip()) for line in f]
        return labels

    def __len__(self):
        return len(self.data)

    def __getitem__(self, index):
        return torchvision.io.read_image(self.data[index]), self.labels[index]
```

let us try the new dataset:

In [12]:

```
dataset = CatsVsDogsDataset("data/catsdogs")

print(dataset.data)
print(dataset.labels)
print(len(dataset))
first_data = dataset[0]
print(type(first_data))
# Images are expressed by intgers from 0 to 256.
# So datatupe: dtype=torch.uint8 it's an unsigned int: must conbvert it into float!
```

```
[tensor([[[161, 184, 196,  ...,  59,  32,  21],
         [139, 158, 177,  ...,  44,  31,  34],
         [125, 114, 119,  ...,  28,  32,  46],
         ...,
         [154,  90,  64,  ...,  68, 137, 147],
         [115, 167, 104,  ...,  89, 162, 197],
         [108, 167, 134,  ..., 115, 127, 139]],

        [[199, 222, 234,  ..., 105,  78,  67],
         [177, 196, 215,  ...,  90,  77,  80],
         [166, 155, 160,  ...,  73,  77,  91],
         ...,
         [177, 115,  91,  ..., 111, 180, 192],
         [138, 192, 131,  ..., 132, 205, 242],
         [131, 192, 161,  ..., 156, 170, 182]],

        [[160, 183, 195,  ...,  68,  41,  30],
         [138, 157, 176,  ...,  51,  38,  41],
         [126, 115, 120,  ...,  34,  38,  52],
         ...,
         [135,  75,  50,  ...,  57, 126, 137],
         [ 96, 152,  90,  ...,  76, 149, 185],
         [ 89, 152, 120,  ..., 100, 114, 126]]], dtype=torch.uint8), tensor([[[172, 173, 174,  ..., 126, 134, 142],
         [173, 173, 175,  ..., 133, 139, 144],
         [181, 182, 183,  ..., 153, 155, 155],
         ...,
         [227, 226, 223,  ..., 177, 176, 176],
         [228, 227, 225,  ..., 177, 176, 176],
         [229, 228, 225,  ..., 177, 176, 176]],

        [[118, 119, 119,  ...,  97, 105, 114],
         [119, 119, 120,  ..., 102, 108, 114],
         [127, 128, 128,  ..., 119, 122, 122],
         ...,
         [196, 195, 194,  ..., 143, 142, 142],
         [197, 196, 196,  ..., 143, 142, 142],
         [198, 197, 196,  ..., 143, 142, 142]],

        [[ 56,  57,  55,  ...,  81,  87,  93],
         [ 57,  57,  56,  ...,  82,  87,  90],
         [ 65,  66,  64,  ...,  91,  91,  89],
         ...,
         [152, 151, 152,  ...,  97,  96,  96],
         [153, 152, 154,  ...,  97,  96,  96],
         [154, 153, 154,  ...,  97,  96,  96]]], dtype=torch.uint8), tensor([[[143, 143, 143,  ..., 149, 149, 149],
         [143, 143, 143,  ..., 148, 148, 149],
         [143, 143, 143,  ..., 147, 147, 147],
         ...,
         [148, 146, 143,  ...,  46,  52,  55],
         [144, 142, 140,  ...,  46,  52,  56],
         [144, 143, 142,  ...,  37,  42,  47]],

        [[164, 164, 164,  ..., 170, 170, 170],
         [164, 164, 164,  ..., 169, 169, 170],
         [164, 164, 164,  ..., 168, 168, 168],
         ...,
         [116, 114, 111,  ...,  44,  50,  53],
         [112, 110, 108,  ...,  44,  50,  54],
         [112, 111, 110,  ...,  35,  40,  45]],

        [[193, 193, 193,  ..., 199, 199, 199],
         [193, 193, 193,  ..., 198, 198, 199],
         [193, 193, 193,  ..., 197, 197, 197],
         ...,
         [ 93,  91,  88,  ...,  45,  51,  54],
         [ 89,  87,  85,  ...,  45,  51,  55],
         [ 89,  88,  87,  ...,  36,  41,  46]]], dtype=torch.uint8), tensor([[[119, 110, 106,  ..., 149, 152, 158],
         [131, 126, 124,  ..., 142, 145, 150],
         [157, 156, 158,  ..., 145, 147, 151],
         ...,
         [130, 131, 136,  ..., 126, 125, 125],
         [128, 130, 132,  ..., 125, 124, 123],
         [129, 131, 132,  ..., 123, 122, 121]],

        [[128, 119, 115,  ..., 152, 155, 161],
         [140, 135, 133,  ..., 145, 148, 153],
         [166, 165, 167,  ..., 148, 150, 154],
         ...,
         [130, 131, 136,  ..., 125, 124, 124],
         [128, 130, 132,  ..., 124, 123, 122],
         [129, 131, 132,  ..., 122, 121, 120]],

        [[137, 128, 124,  ..., 157, 160, 166],
         [149, 144, 142,  ..., 150, 153, 158],
         [175, 174, 176,  ..., 153, 155, 159],
         ...,
         [122, 123, 128,  ..., 121, 120, 120],
         [120, 122, 124,  ..., 120, 119, 118],
         [121, 123, 124,  ..., 118, 117, 116]]], dtype=torch.uint8), tensor([[[191, 194, 193,  ...,  21,  23,  24],
         [187, 187, 185,  ...,  19,  20,  21],
         [191, 185, 182,  ...,  16,  16,  17],
         ...,
         [245, 249, 250,  ...,  64,  68,  70],
         [245, 248, 251,  ...,  71,  72,  68],
         [236, 240, 243,  ...,  80,  79,  73]],

        [[194, 199, 201,  ...,  19,  21,  22],
         [190, 192, 193,  ...,  17,  18,  19],
         [194, 190, 190,  ...,  14,  14,  15],
         ...,
         [251, 255, 253,  ...,  64,  68,  70],
         [251, 254, 254,  ...,  71,  72,  68],
         [242, 246, 246,  ...,  80,  79,  73]],

        [[201, 203, 203,  ...,  20,  22,  23],
         [197, 196, 195,  ...,  18,  19,  20],
         [201, 194, 192,  ...,  15,  15,  16],
         ...,
         [241, 245, 244,  ...,  64,  68,  70],
         [241, 244, 245,  ...,  71,  72,  68],
         [232, 236, 237,  ...,  80,  79,  73]]], dtype=torch.uint8), tensor([[[120, 120, 120,  ..., 122, 121, 121],
         [120, 120, 120,  ..., 122, 121, 121],
         [120, 120, 120,  ..., 122, 121, 121],
         ...,
         [113, 113, 113,  ..., 102,  96,  91],
         [113, 113, 113,  ..., 105,  99,  91],
         [113, 113, 113,  ..., 107, 101,  91]],

        [[114, 114, 114,  ..., 113, 112, 112],
         [114, 114, 114,  ..., 113, 112, 112],
         [114, 114, 114,  ..., 113, 112, 112],
         ...,
         [109, 109, 109,  ..., 102,  96,  92],
         [109, 109, 109,  ..., 105,  99,  92],
         [109, 109, 109,  ..., 107, 101,  92]],

        [[ 56,  54,  54,  ...,  74,  73,  73],
         [ 56,  54,  54,  ...,  74,  73,  73],
         [ 56,  54,  52,  ...,  74,  73,  73],
         ...,
         [ 72,  72,  72,  ...,  90,  84,  78],
         [ 72,  72,  72,  ...,  93,  87,  78],
         [ 72,  72,  72,  ...,  95,  89,  78]]], dtype=torch.uint8), tensor([[[209, 137, 107,  ..., 190, 111,  21],
         [205, 146, 108,  ..., 148, 137,  41],
         [210, 192, 163,  ...,  64, 108,  74],
         ...,
         [160, 169, 187,  ..., 165, 159, 142],
         [156, 169, 191,  ..., 174, 147, 131],
         [168, 184, 203,  ..., 182, 138, 124]],

        [[211, 139, 109,  ..., 188, 111,  21],
         [206, 148, 110,  ..., 146, 137,  41],
         [211, 193, 164,  ...,  62, 108,  74],
         ...,
         [159, 168, 186,  ..., 166, 160, 143],
         [155, 168, 187,  ..., 175, 148, 132],
         [164, 180, 199,  ..., 183, 139, 125]],

        [[200, 128,  98,  ..., 176,  99,   9],
         [198, 137,  99,  ..., 134, 125,  29],
         [203, 185, 156,  ...,  50,  96,  62],
         ...,
         [155, 164, 182,  ..., 161, 155, 138],
         [151, 164, 184,  ..., 170, 143, 127],
         [161, 177, 196,  ..., 178, 134, 120]]], dtype=torch.uint8), tensor([[[182, 181, 180,  ..., 196, 198, 198],
         [182, 181, 180,  ..., 196, 197, 197],
         [182, 182, 181,  ..., 196, 196, 196],
         ...,
         [230, 231, 234,  ..., 182, 194, 198],
         [231, 232, 234,  ..., 173, 188, 195],
         [230, 231, 233,  ..., 148, 164, 173]],

        [[183, 182, 181,  ..., 193, 195, 195],
         [183, 182, 181,  ..., 193, 194, 194],
         [183, 183, 182,  ..., 193, 193, 193],
         ...,
         [231, 232, 235,  ..., 183, 195, 199],
         [232, 233, 235,  ..., 174, 189, 196],
         [231, 232, 234,  ..., 149, 165, 174]],

        [[175, 174, 173,  ..., 174, 178, 178],
         [175, 174, 173,  ..., 176, 177, 179],
         [175, 175, 174,  ..., 176, 176, 178],
         ...,
         [223, 224, 227,  ..., 175, 187, 191],
         [224, 225, 227,  ..., 166, 181, 188],
         [223, 224, 226,  ..., 141, 157, 166]]], dtype=torch.uint8), tensor([[[217, 217, 217,  ...,  52,  54,  56],
         [217, 217, 217,  ...,  55,  54,  53],
         [217, 217, 217,  ...,  56,  52,  48],
         ...,
         [216, 215, 215,  ...,   9,   9,   9],
         [214, 214, 215,  ...,   9,   9,   9],
         [214, 214, 215,  ...,   9,   9,   9]],

        [[204, 204, 204,  ...,  47,  49,  51],
         [204, 204, 204,  ...,  50,  49,  48],
         [204, 204, 204,  ...,  51,  47,  43],
         ...,
         [207, 206, 206,  ...,   9,   9,   9],
         [205, 205, 206,  ...,   9,   9,   9],
         [205, 205, 206,  ...,   9,   9,   9]],

        [[185, 185, 185,  ...,  41,  43,  45],
         [185, 185, 185,  ...,  44,  43,  42],
         [185, 185, 185,  ...,  45,  41,  37],
         ...,
         [198, 197, 197,  ...,   9,   9,   9],
         [196, 196, 197,  ...,   9,   9,   9],
         [196, 196, 197,  ...,   9,   9,   9]]], dtype=torch.uint8), tensor([[[ 74,  71,  67,  ...,  73,  75,  78],
         [ 71,  68,  67,  ...,  70,  72,  75],
         [ 71,  71,  72,  ...,  73,  74,  76],
         ...,
         [254, 249, 249,  ...,  51,  61,  45],
         [252, 247, 251,  ...,  52,  68,  57],
         [252, 252, 255,  ...,  51,  77,  71]],

        [[ 96,  90,  87,  ...,  85,  85,  87],
         [ 90,  87,  87,  ...,  84,  83,  84],
         [ 90,  90,  92,  ...,  88,  87,  86],
         ...,
         [247, 241, 241,  ...,  65,  75,  58],
         [245, 240, 243,  ...,  66,  82,  70],
         [245, 245, 248,  ...,  65,  91,  84]],

        [[109, 104,  98,  ...,  85,  87,  92],
         [104, 101,  98,  ...,  85,  87,  91],
         [104, 104, 103,  ...,  91,  93,  95],
         ...,
         [228, 222, 222,  ...,  68,  78,  64],
         [227, 221, 224,  ...,  69,  85,  76],
         [227, 227, 231,  ...,  68,  94,  90]]], dtype=torch.uint8)]
[0, 1, 0, 0, 0, 1, 1, 1, 0]
10
<class 'tuple'>
```

Part II - DataLoaders

DataLoaders can be quickly constructed from a Dataset...

In [13]:

```
dataloader = torch.utils.data.DataLoader(dataset, batch_size=2, shuffle=True) #non lazy version
```

let us loop through the dataloader:

In [14]:

```
list_of_images = []
for imgs, labels in dataloader:
    # Dataloader is an iterator: yield operator!
    # Each query returns a different item in the dataloader, and then it stiches them together. (collate function)
    print(imgs.shape, labels)
    for img in imgs:
        list_of_images.append(tensor2PIL(img))
```

```
torch.Size([2, 3, 216, 237]) tensor([0, 0])
torch.Size([2, 3, 216, 237]) tensor([1, 1])
torch.Size([2, 3, 216, 237]) tensor([1, 0])
torch.Size([2, 3, 216, 237]) tensor([0, 0])
torch.Size([2, 3, 216, 237]) tensor([1, 1])
```

Q: Do you notice something different w.r.t. what we saw before?

In [15]:

```
list_of_images[0] # I have shuffled them so I have a different order
```

Out[15]:

In [16]:

```
list_of_images[4]
```

Out[16]:

Part III - Data Augmentation

From the lecture, we know that we can construct custom augmentation pipelines using torchvision.

Let us quickly implement augmentability in our (non-lazy) dataset

In [ ]:
```
class CatsVsDogsDatasetAugmentable(CatsVsDogsDataset): # the leng method is inherited by CastVsDogsDataset
    def __init__(self, root, transform=None):
        super().__init__(root)
        self.transform = transform

    def __getitem__(self, index):
        img, label = super().__getitem__(index)
        if self.transform:
            img = self.transform(img)
        return img, label
```

Q: What about the `__len__` method?

In [ ]:
```
transform_pipeline = torchvision.transforms.Compose([
    torchvision.transforms.RandomHorizontalFlip(),
    torchvision.transforms.RandomRotation(degrees=15),
])

aug_dataset = CatsVsDogsDatasetAugmentable("data/catsdogs", transform=transform_pipeline)
```

In [ ]:
```
tensor2PIL(aug_dataset.data[0])
```

In [ ]:
```
tensor2PIL(aug_dataset[0][0])
```

Q: What is the difference between the two code snippets here above?

Problem to solve:

- Suppose we have a dataset composed of n images
- As opposed to the cats vs dogs example we saw before, the images don't have a common size $h \times w$
- **What are my possibilities for training an ANN on this dataset?**
  - crop imgs
  - batch size of 1 but and then effective batch size bigger, so each time I accumulate the gradient and after reaching the desired batch size.

End of "compulsory" lab. Next we have some optional suggestions for loading datasets doing and data augmentation in PyTorch

The "compulsory" augmentations: how to do them on our dataset

```
pipeline = T.Compose([
    …,
    T.ToTensor(),
    T.Normalize(mean=mean, std=std)
])
```

Essentially, we need to calculate `mean` and `std`.

We have our data in `dataset.data`. What can we do to get mean and std?

In [ ]:
```
# your code here
```

Train/Test splitting in PyTorch

We can apply a train/test split by using the `torch.utils.data.random_split` method

In [ ]:
```
pct_train = .7
len_train = int(len(aug_dataset) * pct_train)
```

```
len_test = len(aug_dataset) - len_train
# don't do:
# len_test = int(len(aug_dataset * (1-pct_train))
# for casting reasons: casting to integers = casting to floor!
trainset, testset = torch.utils.data.random_split(aug_dataset, [len_train, len_test])
```

Notice that now, the `trainset` and `aug_dataset` are of two different types!

In [ ]:
```
print(type(aug_dataset), type(trainset), type(testset))
```

we can recover the original dataset by accessing the `dataset` attribute of `torch.utils.data.dataset.Subset`

In [ ]:
```
trainset.dataset.labels
```

Despite the difference, both `trainset` and `aug_dataset` can be equally used to create DataLoaders...

Miscellaneous dataset helps

ImageFolder

It often happens that datasets are distributed with the following folder structure:

```
root_folder
  |
  - class 0
    |
    - images belonging to class 0
  |
  ...
  |
  - class i
    |
    - images belonging to class i
  |
  ...
```

without a corresponding `labels.txt` file (or similar file.)

When the situation is this one, without building exotic custom classes, we can use the `torchvision.datasets.ImageFolder(...)` class that automatically builds a (lazy) dataset for us.

Downloading widespread benchmark datasets

To download benchmark datasets like

- MNIST
- Cifar10 and Cifar100
- Fashion-MNIST
- Microsoft COCO
- ...

we can use the corresponding `torchvision.datasets` classes. Just a couple of notes:

- remember to set, preferrably, the flag `download` to True in the constructor (otherwise the dataset won't donwload)
- ImageNet won't download because of recent controversies on fairness and privacy. If you need it, download it (at your home) from here

References

- Pillow docs: https://pillow.readthedocs.io/en/stable/
- torch tutorial on datasets and dataloaders: https://pytorch.org/tutorials/beginner/basics/data_tutorial.html#datasets-dataloaders
- torchvision tutorials and docs
  - IO: https://pytorch.org/vision/0.8/io.html
  - Datasets: https://pytorch.org/vision/stable/datasets.html
  - Transformations: https://pytorch.org/vision/stable/transforms.html

Additional material

- Albumentations, library for more advanced data augmentation: https://albumentations.ai/