02_exploration

October 26, 2025

[1]:
```
import sys
import os

# Add root directory of rhe project into sys.path
sys.path.append(os.path.abspath(os.path.join("..")))
```

0.1 0. Libraries

[2]:
```
from src.visualization import display_images_and_targets
from src.dataset import LicensePlateDataset
from src.utils import print_tree
from src.config import (
    PROCESSED_DATA_DIR,
    IMAGE_SIZE,
    MEAN_NORMALIZATION,
    STD_NORMALIZATION,
    NUM_WORKERS,
    BATCH_SIZE
)

import torch
import os

from torchvision import transforms
from torch.utils.data import DataLoader
```

0.2 1. Physical devices

[3]:
```
# Check GPU
print(f'Does the device have a GPU? {torch.cuda.is_available()}')
print(f'The device has {torch.cuda.device_count()} GPU(s)')
for i in range(torch.cuda.device_count()):
    print(f'The GPU(s) name of device is: {torch.cuda.get_device_name(i)}')
```

```
Does the device have a GPU? True
The device has 1 GPU(s)
The GPU(s) name of device is: NVIDIA GeForce MX230
```

[4]:
```
num_workers = os.cpu_count()
print(f'The number of cores of your CPU: {num_workers}')
```

```
The number of cores of your CPU: 8
```

0.3 2. Dataset

[5]:
```
# Get the list of file paths in each folder
file_pathes = {}
for dir_name in os.listdir(PROCESSED_DATA_DIR):
    dir_path = os.path.join(PROCESSED_DATA_DIR, dir_name)
    file_pathes[dir_name] = []
    for filename in os.listdir(dir_path):
        file_pathes[dir_name].append(os.path.join(dir_path, filename))
```

[6]:
```
# Directory tree of processed data
print_tree(PROCESSED_DATA_DIR)
```

```
[DIR] processed
    [DIR] test
        [FILE] xemay1817_jpg.rf.119f2c447b36c4a29c10f0ef8e90e019.xml
        [FILE] CarLongPlateGen2231_jpg.rf.09a6ae0129bd4a3ccac80d66ba4e4b95.xml
        [FILE] xemay246_jpg.rf.530bba55def20c3ce976703557b39b7b.xml
        [FILE] xemay66_jpg.rf.a9939bfd9034efff251c176215d259cc.jpg
        [FILE] CarLongPlateGen2960_jpg.rf.bda1b9f4642a8866ed162c2edfbe3b94.xml
        […]
    [DIR] valid
        [FILE] CarLongPlateGen1530_jpg.rf.3289b63b8aff3d0c30701736cc0d7712.jpg
        [FILE] CarLongPlateGen1708_jpg.rf.2562fda9da07faf37e8f1cfad4393f9f.xml
        [FILE] xemay397_jpg.rf.5b5289344b4ebe46f7c772ac2fb435e3.jpg
        [FILE] CarLongPlateGen1809_jpg.rf.c451d9563fb97938eaf545a70ff4c457.jpg
        [FILE] CarLongPlateGen86_jpg.rf.6aa8cb57bb3b078e4f1a23b0efe49706.jpg
        […]
    [DIR] train
        [FILE] CarLongPlateGen2707_jpg.rf.92e2bb3a0499e548f3bc2e54cc4ce299.xml
        [FILE] 003df8cf2effae50_jpg.rf.85b92c041e14d9bcf4ed1fd70de9661f.jpg
        [FILE] CarLongPlateGen2372_jpg.rf.0d12d2a9b4f0f957096227bf11d30626.jpg
        [FILE] Cars168_png_jpg.rf.33aee6637f3a5a92f0bb1bd1e3005c3e.xml
        [FILE] CarLongPlate584_jpg.rf.f0130e378d8afc47359cbaca7e8a8969.xml
        […]
```

0.4 3. Data preprocessing

After downloading the dataset and storing it locally, now we will move on to the step of reading the images and getting the labels of each image based on the filename of each image and XML file, and for each folder, we will also be able to split the dataset into 3 separate datasets, namely `training set`, `validation set` and `test set`.

We will initialize the dataset by using the **Dataset** class of **Pytorch**, which is a special class that allows us to store the entire image in a variable without worrying about memory impact. **Dataset** helps load data in a **Lazy Loading** style (loading data when needed), saving memory and processing large data sets without having to load the entire data set into memory. When combined with the **DataLoader** class of **PyTorch**, Dataset allows us to automatically split the data into small **batches** and can **shuffle** the data to ensure that the samples are trained randomly. This is important to minimize overfitting and improve training efficiency.

```
<img
    src="../assets/lazy_loading.png"
    alt=""
    width="700"
/>
```

Based on the image above, we will mark from (1) to (4) for each image for easy description. Suppose we have `N` images locally (1), and the storage method of **Dataset** (2) is to store the entire path of `N` images of (1) and divide them into batches, suppose each batch will be 2 images, and every time the model or user calls the 2nd batch in `N` images (1), **Dataset** will use the **Lazy Loading** mechanism (3) and only load 2 images in the exact 2nd batch from (1) into memory and send to (4) - this can be the model calling or the user calling. Therefore, we can preserve the original data and keep functions such as batching, mixing, …

[7]:
```
# Transform images
transform = transforms.Compose([
    transforms.Resize(IMAGE_SIZE),
    transforms.ToTensor(),
    transforms.Normalize(mean=MEAN_NORMALIZATION, std=STD_NORMALIZATION)
])
```

[8]:
```
# Declare dataset and dataloader
train_dataset = LicensePlateDataset(file_paths=file_pathes["train"],␣
 ↪transform=transform)
valid_dataset = LicensePlateDataset(file_paths=file_pathes["valid"],␣
 ↪transform=transform)
test_dataset = LicensePlateDataset(file_paths=file_pathes["test"],␣
 ↪transform=transform)
```

For image problems, in addition to the pre-trained models provided by **Pytorch** through the **torchvision** module, the library also provides special classes combined with the **Dataset** class to help us preprocess image data while calling `__getitem__`, specifically the **Transform** class, this is a class that helps us make image features better through transformations such as image rotation, image normalization, image flipping, resizing,… To normalize, we just need to use the transformation `ToTensor()` of **Transform**, a library that will help us normalize and convert from **numpy** to **tensor**.

```
<img
    src="https://www.researchgate.net/publication/356632069/figure/fig3/AS:1095686288347139@163
    alt=""
    width="500"
/>
```

We can check the data folder and see that the images have been pre-processed by a third party, **Roboflow**, they have done some transformations like resizing all the images to `640x640x3`, rotating the images, removing white points, etc. Therefore, we don't need to do too much processing on this data anymore, what we need to do is resize the images to reduce the image size to speed up the training process and normalize the images to the range `[0, 1]`, this will avoid the problem of unit difference between pixels because each pixel of the image will have a value from `[0, 255]`, and normalization will also help the model converge faster and learn better. Based on the data labels are bounding boxes with range in image size, so we can also normalize these labels by taking the x_{min}, y_{min}, x_{max}, y_{max} points of the bounding box divided by the appropriate image size to bring this label to the range `[0, 1]`, for example

$$\frac{x_{min}}{width\ image}$$

and

$$\frac{y_{min}}{height\ image}$$

To convert back to the original value, we simply multiply back by the image size.

[9]:
```
# A sample in training dataset
train_dataset[0]
```

[9]:
```
(tensor([[[-2.1179, -2.1179, -2.1179,  …, -2.1179, -2.1179, -2.1179],
          [-2.1179, -2.1179, -2.1179,  …, -2.1179, -2.1179, -2.1179],
          [-2.1179, -2.1008, -2.1179,  …, -2.1008, -2.1179, -2.1008],
          …,
          [-2.0837, -1.8268, -1.1075,  …, -2.1179, -2.1179, -2.1179],
          [-1.8782, -1.3302, -0.7822,  …, -2.1179, -2.1179, -2.1179],
          [-2.0665, -2.0665, -2.1179,  …, -2.1179, -2.1179, -2.1179]],

         [[-2.0182, -2.0007, -2.0182,  …, -2.0357, -2.0357, -2.0182],
          [-2.0182, -2.0007, -2.0182,  …, -2.0357, -2.0357, -2.0182],
          [-2.0357, -2.0182, -2.0357,  …, -2.0182, -2.0182, -2.0007],
          …,
          [-1.9657, -1.7031, -0.9678,  …, -2.0357, -2.0357, -2.0357],
          [-1.7556, -1.1954, -0.6352,  …, -2.0357, -2.0357, -2.0357],
          [-1.9482, -1.9482, -2.0182,  …, -2.0357, -2.0357, -2.0357]],

         [[-1.8044, -1.7870, -1.8044,  …, -1.7347, -1.8044, -1.8044],
          [-1.8044, -1.7870, -1.8044,  …, -1.7696, -1.8044, -1.8044],
          [-1.7696, -1.7522, -1.7696,  …, -1.8044, -1.8044, -1.8044],
          …,
          [-1.7522, -1.4907, -0.7587,  …, -1.8044, -1.8044, -1.8044],
          [-1.5779, -1.0201, -0.4624,  …, -1.8044, -1.8044, -1.8044],
          [-1.7696, -1.7696, -1.8044,  …, -1.8044, -1.8044, -1.8044]]]),
 {'boxes': tensor([[ 56., 108.,  96., 190.]]), 'labels': tensor([1])})
```

[10]:
```
# Split dataset to batches
train_dataloader = DataLoader(
    dataset=train_dataset,
    batch_size=BATCH_SIZE,
    num_workers=NUM_WORKERS,
    persistent_workers=True,
    pin_memory=True,
    shuffle=True,
    prefetch_factor=10,
    collate_fn=lambda batch: tuple(zip(*batch))
)

valid_dataloader = DataLoader(
    dataset=valid_dataset,
    batch_size=BATCH_SIZE,
    num_workers=NUM_WORKERS,
    persistent_workers=True,
    pin_memory=True,
    shuffle=True,
    prefetch_factor=10,
    collate_fn=lambda batch: tuple(zip(*batch))
)

test_dataloader = DataLoader(
    dataset=test_dataset,
    batch_size=len(test_dataset),
    num_workers=NUM_WORKERS,
    persistent_workers=True,
    pin_memory=True,
    shuffle=True,
    prefetch_factor=10,
    collate_fn=lambda batch: tuple(zip(*batch))
)
```

0.5 4. Visualization

[11]:
```
print(f'The number of images in train set: {len(train_dataset)}')
print(f'The number of batch in train set: {len(train_dataloader)}\n')

print(f'The number of images in validation set: {len(valid_dataset)}')
print(f'The number of batch in validation set: {len(valid_dataloader)}\n')

print(f'The number of images in test set: {len(test_dataset)}')
print(f'The number of batch in test set: {len(test_dataloader)}')
```

```
The number of images in train set: 20576
The number of batch in train set: 1286
The number of images in validation set: 1973
The number of batch in validation set: 124

The number of images in test set: 978
The number of batch in test set: 1
```

[ ]:
```
def has_invalid_box(data):
    """
        Check the dataset whether contains invalid bounding or not
    """
    for i, sample in enumerate(data):
        for box in sample[1]["boxes"]:
            if box[2] - box[0] <= 0.0 or box[3] - box[1] <= 0.0:
                print(sample)
                print(data.data[i])
```

[13]:
```
has_invalid_box(train_dataset)
```

[14]:
```
has_invalid_box(valid_dataset)
```

[15]:
```
has_invalid_box(test_dataset)
```

[16]:
```
# Display images and labels
sample_idxes = torch.randperm(len(train_dataloader.dataset))[:10]
example_images = [train_dataset[idx][0] for idx in sample_idxes]
example_labels = [train_dataset[idx][1] for idx in sample_idxes]

display_images_and_targets(example_images, example_labels)
```