

Environment details:

在最新版本的 Google Colab 上運行，且沒有額外下載的 lib (下圖是所有我 import 的東西)，因此不需要 !pip install。

唯一可能需要注意的是，因為我的線上 GPU 資源燒光了，所以我有額外花錢買 Colab Pro，環境應該是不影響，但我還是提供資訊讓助教知道。

```
[1] import csv
import cv2
import numpy as np
import random
import os

import torchvision
import torch
import torch.nn as nn
from torchvision.io import read_image
from torch.utils.data import Dataset, DataLoader

[6] from google.colab import drive
drive.mount('/content/drive')
TRAIN_PATH = "drive/MyDrive/captcha-hacker/train"
TEST_PATH = "drive/MyDrive/captcha-hacker/test"
device = "cuda"
# device = "cpu"
```

Implementation details

網路上已經有許多 MNIST 數字資料集的分類教學，但因為 [MNIST](#) 資料集의影像是灰階影像，而本次作業給的是有顏色的 RGB 圖案，所以我又額外多找找看有沒有更適合本次作業使用的模型，發現 ResNet18 採用的影像格式也為 RGB，而且操作起來相對簡單，因此選擇 ResNet18 作為本作業的模型。

使用 PyTorch 下的 torchvision.models.resnet18

<https://pytorch.org/vision/main/models/generated/torchvision.models.resnet18.html>

PyTorch 上的 ResNet18 是根據這篇論文實作的 <https://arxiv.org/pdf/1512.03385.pdf>

詳讀過 ResNet18 的模型結構後，我發現最後一層 fc layer 的輸出 shape 為 1000，而在本次作業中，我們分別只會用到 10、72、144 個輸出，因此呼叫 pretrained ResNet18 模型後，要額外加一行修改 fc layer 的輸出 (如下圖，task 1 2 3 的 out_features 分別改成 10、72、144)

```
16 model_task_3.fc = nn.Linear(in_features=512, out_features=144, bias=True).to(device)
```

在這篇文章當中我們可以看到 ResNet18 有許多可以改動的 Hyper parameter，

https://www.researchgate.net/figure/The-hyper-parameters-choices-for-ResNet18_tbl2_339580876

在本次作業當中，我只有在 DataLoader 那邊改 batch_size，其他都是用 pretrained 好的資料，沒有額外給參數。如下圖，我的 batch_size 在 task 1 2 3 都是 60。

```
train_ds = Task_Dataset(train_data, root=TRAIN_PATH, task = 1)
train_dl = DataLoader(train_ds, batch_size=60, num_workers=4, drop_last=True, shuffle=True)

val_ds = Task_Dataset(val_data, root=TRAIN_PATH, task = 1)
val_dl = DataLoader(val_ds, batch_size=60, num_workers=4, drop_last=False, shuffle=False)
```

Task 1 2 3，我分別使用 15、30、50 Epoch 進行訓練，訓練結束後輸出 weight(如下圖)

```
for epoch in range(50):
    print(f"Epoch [{epoch}]")
    model.train()
    for image, label in train_dl:
        image = image.to(device)
        label = label.to(device)

        pred = model(image)
        loss = loss_fn(pred, label)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    del image, label
torch.save(model.state_dict(), "/content/drive/MyDrive/model3")
```

其他部分，除了資料集處理有修改比較多(如下圖，我在 init 的地方加了一個參數 task，讓我的 class 可以分辨正在處理哪個 task)

```

class Task_Dataset(Dataset):
    def __init__(self, data, root, task, return_filename=False):
        if task == 1:
            self.data = [sample for sample in data if sample[0].startswith("task1")]
        if task == 2:
            self.data = [sample for sample in data if sample[0].startswith("task2")]
        if task == 3:
            self.data = [sample for sample in data if sample[0].startswith("task3")]
        self.return_filename = return_filename
        self.root = root
        self.task = task

    def __getitem__(self, index):

        filename, label = self.data[index]
        img_origin = read_image(f"{self.root}/{filename}")
        img_trans = transform(img_origin)
        if self.return_filename:
            return torch.FloatTensor(img_trans / 255), filename
        else:
            if self.task == 1:
                return torch.FloatTensor(img_trans / 255), int(label)
            elif self.task == 2:
                two_hot_table = [0 for i in range(72)]
                for i in range(2):
                    if label[i].isdigit():
                        two_hot_table[int(ord(label[i])) - 48 + i * 36] = 1
                    else:
                        two_hot_table[int(ord(label[i])) - 87 + i * 36] = 1
                return torch.FloatTensor(img_trans / 255), torch.tensor(two_hot_table)

            elif self.task == 3:
                four_hot_table = [0 for i in range(144)]
                for i in range(4):
                    if label[i].isdigit():
                        four_hot_table[int(ord(label[i])) - 48 + i * 36] = 1
                    else:
                        four_hot_table[int(ord(label[i])) - 87 + i * 36] = 1
                return torch.FloatTensor(img_trans / 255), torch.tensor(four_hot_table)

```

還有額外多增加一個 transform function 對圖片做 resize 跟 normalization 處理(下圖)

Normalize 的部分，採用 PyTorch 官方建議的參數([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

```







transform = torchvision.transforms.Compose([
    torchvision.transforms.ToPILImage(),
    torchvision.transforms.Resize(300),
    torchvision.transforms.ToTensor(),
    torchvision.transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])


```

另外在 loss function 的部分，因為 task 2 3 是 multi lable，因此我使用 MultiLabelSoftMarginLoss 作為 loss function(task1 的 loss 跟 sample code 一樣是 cross entropy)

除上面提到的之外，皆是按照助教給的 sample code 做修改，因此不多作介紹。

Screenshot

72	109550042		0.92760	22	5h
73	109705004		0.92740	24	9h
74	dalalalaa		0.92220	4	6h
75	109550053		0.91420	14	4h
76	0816181		0.91100	6	3h
77	0816170		0.90900	1	1s

 Your First Entry!
Welcome to the leaderboard!

本次作業的心得:

之前的作業，任務目標很明確，都已經清楚知道要做什麼、用什麼演算法，這次作業需要自己探索適合的學習模型，也因此有機會讀到平常較少看到的 Deep Learning documents，更瞭解深度學習的架構和原理，收穫頗豐。

downloadable model weight link

Task 1

https://drive.google.com/file/d/1-6gnvYxIT8OXxv-3KmDPBPMYe3PMIHmE/view?usp=share_link

Task 2

https://drive.google.com/file/d/1-3-3O5DHPzZPy93LPYELCr09C46adI-2/view?usp=share_link

Task 3

https://drive.google.com/file/d/1RAWcFCYQo_cbZCMz6zBx3gDy7aVQXI6x/view?usp=share_link