

# 109550031\_HW5

## Environment details

	version
python	3.7.10
numpy	1.21.6
pandas	1.4.2
torch	1.21.1

## Dataset

利用助教們提供的 dataset，大約能達到 95 % 的準確率，為了提升準確率，我利用以下的程式碼來產生額外的驗證碼照片，並達到 99% 的準確率。

```
from captcha.image import ImageCaptcha
import matplotlib.pyplot as plt
import numpy as np
import random
import string
import csv

characters = string.digits + string.ascii_lowercase
width, height, n_len, n_class = 96, 72, 4, len(characters)
generator = ImageCaptcha(width=width, height=height)

with open('./dataset/train/annotations.csv', 'a', newline='') as csvfile:
    csv_writer = csv.writer(csvfile)
    for i in range(40000):
        random_str = ''.join([random.choice(characters) for j in range(4)])
        img = generator.generate_image(random_str)
        img.save(f"./dataset/train/task3/again{i}.png")
        csv_writer.writerow([f"task3/again{i}.png", random_str])
```

## Implementation Details

### One Hot Encoding

在此作業中，為了將 captcha 的資料轉換成方便 CNN model 進行訓練的樣子，我使用了 one hot encoding，替每個類別（數字+小寫英文字母）新增一個欄位，用 0/1 表示是否為此。在 task 1 因為只有一個字母，所以經過 one hot encoding 是長度為 36 的 list，task 2 經過 one hot encoding 的長度為 72，以此類推，task 3 經過 one hot encoding 的長度為 144。

以下為 one hot encoding 的函式：

```
def one_hot_encode(self, label):
    onehot = [0] * (ALL_CHAR_SET_LEN * self.captcha_len)
    for i, l in enumerate(label):
        idx = ALL_CHAR_SET.index(l) + i * ALL_CHAR_SET_LEN
        onehot[idx] = 1
    return np.array(onehot)
```

## Model Architecture

在這次的作業裡，我使用 pytorch 來建立我的 CNN 模型，此模型主要是由卷積層(Conv2d)、池化層(MaxPool2d)、激活函數(ReLU)組成，並且利用 BatchNorm 加速模型的收斂速度。

主要的結構如下：

1. 第一個卷積層 `nn.Conv2d(1, 4, 3, padding=(1, 1))` 代表輸入通道數為 1，輸出通道數為 4，kernel size 的長寬都為 3，padding 為 (1, 1)，可確保輸出的長寬不變。以 task 1 和 task 2 為例，輸入的 shape 為 `[batch, 1, 72, 72]`，通過這層卷積層後，輸出的 shape 為 `[batch, 4, 72, 72]`
2. 接著通過批規範層 `nn.BatchNorm2d(4)`，可加速模型的收斂速度。
3. 第二個卷積層 `nn.Conv2d(4, 16, 3, padding=(1, 1))` 代表輸入通道數為 4，輸出通道數為 16，kernel size 的長寬一樣為 3，padding 一樣為 (1, 1)。一樣以 task 1 和 task 2 為例，輸入的 shape 為 `[batch, 4, 72, 72]`，通過這層卷積層後，輸出的 shape 為 `[batch, 16, 72, 72]`。
4. 通過池化層 `nn.MaxPool2d(2, 2)`，池化窗口的長寬皆為 2，輸出的長寬為輸入的一半。繼續以 task 1 和 task 2 為例，輸入的 shape 為 `[batch, 16, 72, 72]`，通過這層池化層後，輸出的 shape 為 `[batch, 16, 36, 36]`。
5. 接著通過批規範層 `nn.BatchNorm2d(16)`，可加速模型的收斂速度。
6. 使用激活函數 `nn.ReLU()` 是為了引入非線性，讓模型可以擬合更複雜的函數，它會把小於 0 的值賦予 0，大於 0 的值不變。

繼續以 task 1 和 task 2 為例，經過四層如上結構後，得到一個 shape 為 `[batch, 1024, 4, 4]` 的張量，`x = x.view(-1, 1024*4*4)` 將改變此張量的 shape 為 `[batch, 1024*4*4]`，接著利用兩層的線性轉換，得到模型的輸出，也就是 one hot encoding 的型式。

以下為模型的 code：

```
class Model(nn.Module):
    def __init__(self, output_len):
        super(Model, self).__init__()
        self.output_len = output_len
        self.conv = nn.Sequential(
            # batch*1*72*72 / batch*1*72*96
            nn.Conv2d(1, 4, 3, padding=(1, 1)),
            nn.BatchNorm2d(4),
            nn.Conv2d(4, 16, 3, padding=(1, 1)),
            nn.MaxPool2d(2, 2),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            # batch*16*36*36 / batch*16*36*48
            nn.Conv2d(16, 32, 3, padding=(1, 1)),
            nn.BatchNorm2d(32),
            nn.Conv2d(32, 64, 3, padding=(1, 1)),
            nn.MaxPool2d(2, 2),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            # batch*64*18*18 / batch*64*18*24
            nn.Conv2d(64, 128, 3, padding=(1, 1)),
            nn.BatchNorm2d(128),
            nn.Conv2d(128, 256, 3, padding=(1, 1)),
            nn.MaxPool2d(2, 2),
            nn.BatchNorm2d(256),
            nn.ReLU(),
            # batch*256*9*9 / batch*256*9*12
            nn.Conv2d(256, 512, 3, padding=(1, 1)),
            nn.BatchNorm2d(512),
            nn.Conv2d(512, 1024, 3, padding=(1, 1)),
            nn.MaxPool2d(2, 2),
            nn.BatchNorm2d(1024),
            nn.ReLU(),
```

```

        # batch*1024*4*4 / batch*1024*4*6
    )
    self.fc = nn.Linear(1024*4*4, 1024)
    self.fc_task3 = nn.Linear(1024*4*6, 1024)
    self.fc2 = nn.Linear(1024, 256)
    self.out = nn.Linear(256, self.output_len)
    self.dropout = nn.Dropout(0.2)

    def forward(self, x):
        b, h, w = x.shape
        x = x.view(b,1,h,w)
        x = self.conv(x)
        if self.output_len > 100:
            x = x.view(-1, 1024*4*6)
            x = self.fc_task3(x)
        else:
            x = x.view(-1, 1024*4*4)
            x = self.fc(x)
        x = self.dropout(x)
        x = self.fc2(x)
        x = self.dropout(x)
        x = self.out(x)
        return x

```

## Training

舉 task 1 為例，訓練模型的流程如下：

1. 定義 CNN model。
2. 定義優化器 `torch.optim.Adam(modelfortask1.parameters())` 和 loss function `nn.CrossEntropyLoss()`，並且定義 learning rate = 1e-5。
3. 每次取一個 batch 作訓練，計算 loss，將優化梯度設為 0，並且將 loss 向後傳播，計算梯度，更新優化器的參數。
4. 每訓練完一個 epoch，計算其準確率，如果準確率比先前訓練的 epoch 都要來的高，保存此模型。

重複執行以上四個步驟，在 task 1，共執行 10 個 epoch。

以下為訓練模型的 code：

```

modelfortask1 = Model(ALL_CHAR_SET_LEN * 1).to(device)
optimizer = torch.optim.Adam(modelfortask1.parameters(), lr=1e-5)
loss_fn = nn.CrossEntropyLoss()
best_acc = 0

for epoch in range(10):
    print(f"Epoch [{epoch+1}]")
    modelfortask1.train()
    for image, label in train_dl_1:
        image = image.to(device)
        label = label.to(device, dtype=torch.float)

        pred = modelfortask1(image)
        loss = loss_fn(pred, label)

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

    sample_count = 0
    correct_count = 0
    modelfortask1.eval()
    for image, label in val_dl_1:
        image = image.to(device)
        label = label.to(device, dtype=torch.float)

```

```

pred = modelfortask1(image)
loss = loss_fn(pred, label)

pred = torch.argmax(pred.T[0:10], dim=0)
label = torch.argmax(label.T[0:10], dim=0)


sample_count += len(image)
correct_count += (label == pred).sum()


acc = correct_count / sample_count
print("accuracy (validation):", acc)
if acc > best_acc:
    best_acc = acc
    PATH_1 = "task_1_model.pt"
    torch.save({
        'model_state_dict': modelfortask1.state_dict(),
        'optimizer_state_dict': optimizer.state_dict(),
    }, PATH_1)

```

## Result

- task 1：準確率可達 99.94 %
- task 2：準確率可達 99.80 %
- task 3：準確率可達 98.08 %
- on kaggle public leaderboard：準確率可達 99.16 %

13	109550031		0.99160	19	1d
----	-----------	---	---------	----	----



**Your Best Entry!**  
Your most recent submission scored 0.99160, which is an improvement of your previous score of 0.99140. Great job!

Tweet this

## Model Weight

<https://drive.google.com/drive/folders/1R-nM5A3JeBstliBSgS2xK4YWJH-4mi0P?usp=sharing>