Method description　　404410089 廖慧萍

what are your reference codes?

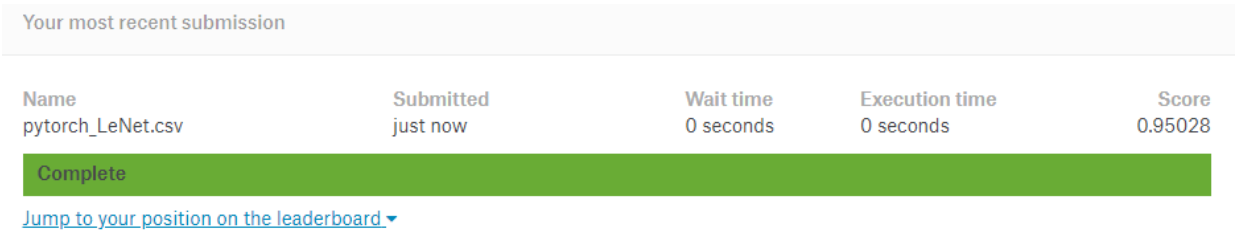
使用助教提供的程式碼
參考 AlexNet、VGG16 的架構
https://medium.com/%E9%9B%9E%E9%9B%9E%E8%88%87%E5%85%94%E5%85%94%E7%9A%84%E5%B7%A5%E7%A8%8B%E4%B8%96%E7%95%8C/%E6%A9%9F%E5%99%A8%E5%AD%B8%E7%BF%92-ml-note-cnn%E6%BC%94%E5%8C%96%E5%8F%B2-alexnet-vgg-inception-resnet-keras-coding-668f74879306

How to run your test?

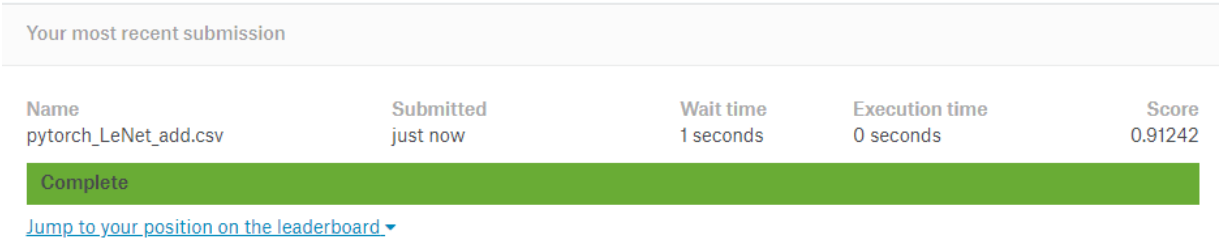
在 CONDA 環境中，開啟 jupyter notebook
執行 mnist.ipynb、mnist-add.ipynb、mnist-new0.96357.ipynb，各別得到 pytorch_LeNet.csv、pytorch_LeNet_add.csv、pytorch_new.csv 檔案。

Experimental results

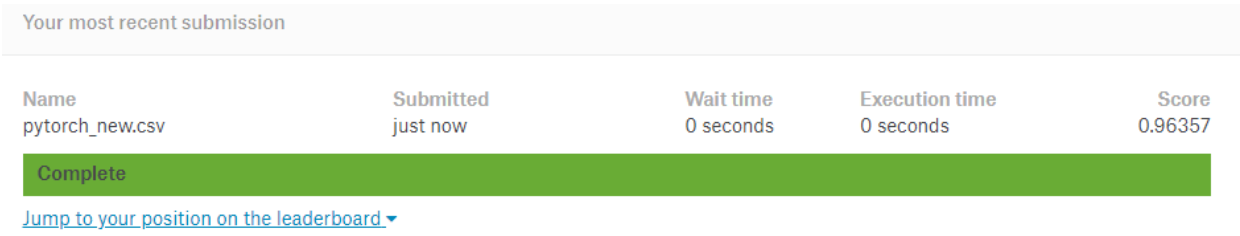
mnist.ipynb 執行後產生的 pytorch_LeNet.csv



mnist-add.ipynb 執行後產生的 pytorch_LeNet_add.csv



mnist-new.ipynb 執行後產生的 pytorch_new.csv



在 kaggle 上的排名依序為 2469、2731、2329

Disscussion

1. mnist.ipynb

在原本的程式碼中，self.conv1 = nn.Conv2d(1, 6, (5,5), padding=2)代表從原本的長寬為 28*28 的一張圖，轉變成 6 張 28*28 的圖，轉變計算方式如下

Shape:

- Input: $(N, C_{in}, H_{in}, W_{in})$
- Output: $(N, C_{out}, H_{out}, W_{out})$ where

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[1] - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

Variables:

- **weight** (*Tensor*) – the learnable weights of the module of shape (out_channels, in_channels, kernel_size[0], kernel_size[1])
- **bias** (*Tensor*) – the learnable bias of the module of shape (out_channels)

再經過 x = F.max_pool2d(F.relu(self.conv1(x)), (2,2))後，由 6*28*28 轉變成 6*14*14。

再來 self.conv2 = nn.Conv2d(6, 16, (5,5))使 6*14*14 轉變成 16*10*10。

再來 x = F.max_pool2d(F.relu(self.conv2(x)), (2,2))使 16*10*10 轉變成 16*5*5。

再經過 x = x.view(-1, self.num_flat_features(x))將 16*5*5flatten 成 400*1 的陣列。

再經過 self.fc1 = nn.Linear(16*5*5, 120)
self.fc2 = nn.Linear(120, 84)
self.fc3 = nn.Linear(84, 10) fully connected layer 從 400 個 node 依序轉變為 120 個、84 個、10 個，模型設計結束，因為資料集有 10 種數字。

2. mnist-add.ipynb

在原本的程式碼再加上一層 conv、fc 後，準確率變差了。因此我嘗試修改 fc1、fc2、fc3、fc4 的參數，但是發現嘗試過很多組數字，四層的 fc 依然不如三層的好。

同時，我也嘗試修改 conv1、conv2、conv3 的參數，這次的模型設計如下:

1*28*28 經過 self.conv1 = nn.Conv2d(1, 128, (5,5), padding=2)轉變成 128*28*28

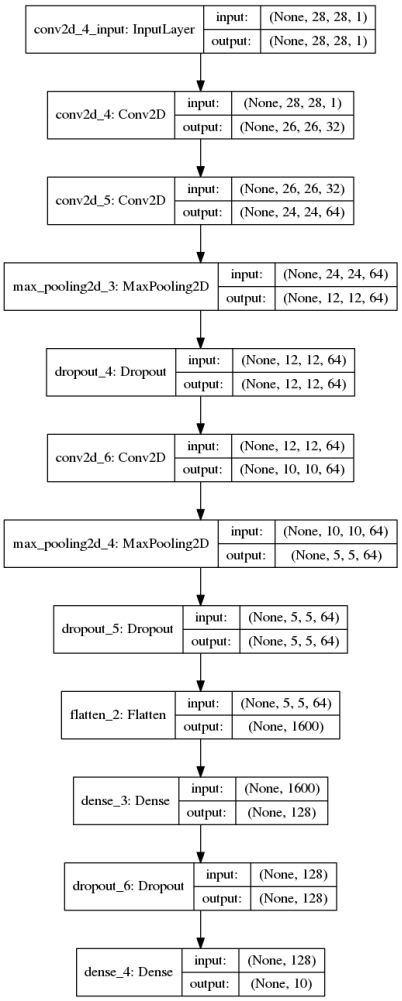
再經過 x = F.max_pool2d(F.relu(self.conv1(x)), (2,2))轉變成 128*14*14

再經過 self.conv2 = nn.Conv2d(128, 256, (3,3))轉變成 256*12*12

再經過 x = F.max_pool2d(F.relu(self.conv2(x)), (2,2))轉變成 256*6*6
再經過 self.conv3 = nn.Conv2d(256, 512, (3,3))轉變成 512*4*4
再經過 x = F.max_pool2d(F.relu(self.conv3(x)), (2,2))轉變成 512*2*2
再經過 x = x.view(-1, self.num_flat_features(x))將 512*2*2flatten 成 2048*1 的陣列。
再經過 self.fc1 = nn.Linear(512*2*2, 256)
self.fc2 = nn.Linear(256,128)
self.fc3 = nn.Linear(128,64)
self.fc4 = nn.Linear(64, 10) fully connected layer 從 2048 個 node 依序轉變為 256 個、128 個、64 個、10 個，模型設計結束，因為資料集有 10 種數字。

3. mnist-new.ipynb

因為修改後的準確率沒有提升，所以我參考了下圖的模型設計。



與原始的 LeNet-5 較大的不同在於有使用 Dropout，還有加了一層 conv、少了一層 fc。
1*28*28 經過 self.conv1 = nn.Conv2d(1, 32, (3,3))轉變成 32*26*26
再經過 self.conv2 = nn.Conv2d(32, 64, (3,3))轉變成 64*24*24
再經過 x = F.max_pool2d(x, 2, 2)轉變成 64*12*12
經過 x = F.dropout(x, p=0.25, training=self.training)後
再經過 self.conv3 = nn.Conv2d(64, 64, (3,3))轉變成 64*10*10
再經過 x = F.max_pool2d(x, 2, 2)轉變成 64*5*5
經過 x = F.dropout(x, p=0.35, training=self.training)後
再經過 x = x.view(-1, self.num_flat_features(x))將 64*5*5flatten 成 1600*1 的陣列。
再經過 self.fc1 = nn.Linear(1600, 128)
self.fc2 = nn.Linear(128,10)
fully connected layer 從 1600 個 node 依序轉變為 128 個、10 個，模型設計結束，因為資料集有 10 種數字。

在 mnist-new.ipynb 中，採用了 dropout，使模型較不容易發生 overfitting。準確率在這三個實驗中為最高的。

Problem and difficulties

在與助教給的程式碼相同的前提下，只增加一層 conv、一層 fc，其準確率竟然會下降。我推測可能是因為這次的圖片大小沒有很大(28*28)，想要增加 layer 的話，每層 layer 擷取的特徵數會更細部化，導致增加了太多特徵細節，無法更準確的辨識。