Environment details

Python 版本是3.8

其他的packages都在requirements.txt裡，沒有特別註明版本代表直接使用pip install -r requirements.txt即可。

Implementation details

model architecture:

data preprocessing and label vector:

三個task我都有針對每張圖片產生對應的灰階圖片，再用cv2.connectedComponentsWithStats根據我給定的參數去除掉照片中的小點點或是線條，再用cv2.connectedComponentsWithStats去算出連接在一起的方塊的片機大小，如果太小的就去除，接著就把這個產生出來的mask乘上r,g,b就會得到比較乾淨的圖片(產出mask的過程我寫在denoise.py裡)，如下圖

處理前：



處理後：



把處理完的圖片根據pytorch官方給resnet設定的preprocessing ([transforms.Resize(224), transforms.RandomHorizontalFlip(), transforms.ToTensor(), normalize]) 再經過一次處理，最終得到處理好的training data。

Label vector的部分，task1，ground truth如果是0那麼label vector 就是[1,0,0,0,0,0,0,0,0,0]，如果是1就是[0,1,0,0,0,0,0,0,0,0]依此類推；task2和task3都是採用multilabel的形式，每個字母或數字根據在abcdefghijklmnopqrstuvwxyz0123456789中的位置去決定label vector長怎樣，接著再把每個字母或數字的label vector concatenate在一起 如果是d8就是[0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]和

[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0] 連在一起變成

[0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0]，task3也是一樣的道理。

Network:

Network的部分就是resnet18去掉最後一層，再接上一層dropout layer和全連接層組成的，全連接層的輸出根據每個task的label vector大小有所改變，task1是10、task2是72，而task3是144。

Hyperparameters:

Learning rate是1e-3，和原本的sample code一樣，因為在loss 的折線圖看不出有lr太大的跡象，因此維持原本的數值。

Epochs在task1和task2都是100，因為在loss 的折線圖來看已經收斂的差多了，然後validation 的accuracy也蠻高的(有到1)，因此沒有做更動，而task3因為validation 的accuracyoa6u.32l4，因此有調高到200。

used deep learning framework: PyTorch