ML_hw3 Report

電機四 B02504086 陳品融

執行 code 的方式與助教提供的範本有些微差異，詳細用法請見 README

1. Supervised learning

(1) **toolkit**: keras，backend: tensorflow

(2) **model architecture**:

```
Input(32,32,3)→ elu(Conv(32,3,3))*2→ Maxpooling→ Dropout(0.25)→ elu(Conv(64,3,3))*2→
Maxpooling→ Dropout(0.25)→ elu(Conv(128,3,3))*2→ Maxpooling→ Dropout(0.25)→ Flatten→
elu(Dense(256))→ Dropout(0.5)→ elu(Dense(32))→ Dropout(0.5)→ Softmax(Dense(10))
```

其中每層 cnn 以及 dense layer 都有加 W_regularizer=l2(1e-3)以避免 overfit，而 training 使用的 loss 爲 categorical_crossentropy，optmizer 則爲 Adam。

(3) **data preprocess**:

每筆 data 都除以 255 作 normalize，並將每個 class 隨機切 10%的 data 作爲 validation_set

(4) **method**:

batch_size=128，epoch=100，比較使用 ImageDataGenerator(weight shift=0.1, height shift=0.1,horizontal flip=True, sample_per_epoch=len(X))與否之差異。

(5) **performance**:

未使用 datagen: val_acc=66.2%， 使用 datagen: val_acc=74%。由此可知，在 training data 量相同的情況下，使用 ImageDataGenerator 可得到明顯較好的結果。

2. Semi-supervised learning(1)

(1) **model architecture**:

與 supervised learning 的架構大致相同，差別在於拿掉 W_regularizer，並在每層 cnn 及 dense 之後加 **batch normalization**(output layer 除外)。

(2) **method**:

使用 ImageDataGenerator + self-training。self-training 的步驟如下：

(a) 將 4500 筆 labeled training data 丟入 datagen 產生 2 倍量的 data，train 40 個 epoch

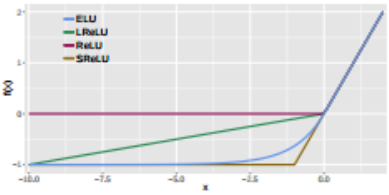
(b) 利用步驟(i)得到的 model 去 predict unlabeled data(unlabel + test)，將有信心的 data(機率>threshold)加入 X_add(X_add 初始爲 labeled training data)並丟入 datagen 產生等量的 data，train 10 個 epoch

(c) 重複步驟(a)，(b) 15 個 iteration 或者當 X_add 的 data 量大於 50000 筆即終止 training

※details: elu(如圖藍線所示)作爲 activation function，使 input 小於 0 仍有值。此外，加入 EarlyStopping 與 ModelCheckPoint 可縮短 training 時間並且避免 overfit。

(3) **performance**:

在 train 了 15 個 iteration 之後，X_add 通常會加到大約 45000～50000 筆 data，而最後的 val_acc 爲 86%，上傳到 kaggle 則爲 84%。關於影響此方法 performance 的因子會在第四點作更詳細的討論。



3. Semi-supervised learning(2)

(1) **model architecture**:

auto encoder

```
Input(32,32,3)→ relu(Conv(16,3,3))→ Maxpooling→ (relu(Conv(8,3,3))→ Maxpooling)*2→
encoded→ (relu(Conv(8,3,3)→ Upsampling)*2→ relu(Conv(16,3,3))→ Upsampling→
sigmoid(Conv(3,3,3))→ decoded
```

dnn

```
elu(Dense(256,input_dim=128)→ batch_normalize)→dropout(0.25)→ elu(Dense(512)→
batch_normalize)→dropout(0.25)→ elu(Dense(32)→batch_normalize)→dropout(0.5)→
softmax(Dense(10))
```

(2) **method**:

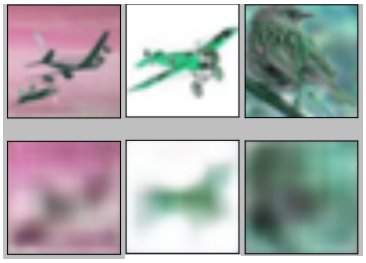
(a) train 一個 cnn 的 auto encoder，其中 batch_size=128, epoch=30, loss=binary_crossentropy，training data 爲 labeled+unlabeled+test。如此可將原本的 data 轉爲 128 維的 vector 作爲 feature。

(b) 將 feature 丟入 dnn model 作類似 2.所述的 self-training(沒有使用 datagen)，總共 5 個 iteration，每次 iteration train labeled data 100 個 epoch，X_add train 10 個 epoch。

(3) **performance**:

val_acc=51%，kaggle score=46%

※analysis: 由右邊的圖可發現，經過 auto encoder encode 再 decode 的圖會變得非常糢糊，所以可推論此 128 維的 code 不足以代表原圖，因此即便再怎麼 self-training 也無法達到好的 performance。



4. Compare and analyze your results

由於只有 ImageDataGenerator + self-training 的正確率較高，故此處我將只針對這一個方法作分析與比較。根據觀察實際 training 的過程與結果，我認爲以下幾個因素對於這次 task 的 performance 有比較顯著的影響。

(1) **threshold 的設定**:

我一開始的 threshold 是設爲 0.999，因爲我覺得必須要非常正確的 data 才能把他加進來 training，可是 val_acc 頂多只能到 80%。後來經過觀察與分析，我認爲可能的原因有以下兩點：首先，當 training data 到 40000 筆以上之後，後面要再加新的 unlabeled data 進來會變得很困難，甚至常常發生完全沒有加的情況，導致 model 一直重複 train 一樣的 data，因此便會 train 不太上去，甚至有可能 overfit；此外，把 threshold 設這麼高有另一個潛在的問題是，如果這個 model 對於那些 unlabeled data 已經有很大的把握時，那麼再把它們加進來似乎對於 model 的 update 比較沒有幫助。因此，最後我決定採用 adaptive threshold，意即 training data 加到某一定量時，便把 threshold 調小，如此一來就可解決上述的問題。至於爲何不把 threshold 就固定在一個較小的值，我認爲由於一開始的 model 不太可靠，因此 threshold 的初始值還是不要設得過低比較好。

(2) **batch normalization**:

所謂的 batch normalization，就是把某層 layer 的 mini-batch 變成平均爲 0，變異數爲 1，其作用之一是可以避免 overfit。我原本一開始用來避免 overfit 的方法是加 l2 norm 的 weight regularizer，可是發現 val_acc 會上不太去，後來改用 batch normalization，performance 便有所提升。

(3) **labeled 與 unlabeled data 的權重**:

我一開始的方法是只要 unlabeled data 的機率大於 threshold 就把他加進 training data 一起 train，可是後來覺得似乎不該如此相信 unlabeled data，應該讓 labeled data 相對的權重再大一點比較好。因此我後來改成 train labeled data 40 個 epoch，train labeled+unlabeled data 10 個 epoch。此比例應該可再作調整，不過這個方法確實讓我能突破原本 80%的瓶頸。

(4) **model 大小**:

由於此次 task 的圖不大，只有 32*32*3，所以 train_acc 很容易一下子就到 90%以上，若 model 太深或參數太多，則 train_acc 很有可能會跑到 97~98%。爲了避免 overfit，我的 convolution filter 數以及 dense 的 neuron 數都用得比較少一點。