**Deep-Learning-for-Autonomous-Driving**

**Lab4 : Pruning & Quantization**

ID : 310605007

Member : 鄭晴立

Department : 機器人學程

目錄

[表目錄 2](#_Toc121430312)

[找不到圖表目錄。 **錯誤! 尚未定義書籤。**](#_Toc121430313)

[圖目錄 2](#_Toc121430314)

[1. Task 1: Training 3](#_Toc121430315)

[1.1. Result 3](#_Toc121430316)

[1.2. Data augmentation 4](#_Toc121430317)

[1.4. Dirty data 5](#_Toc121430318)

[2. Task 2 : Fine-grained Pruning 7](#_Toc121430319)

[2.1. Result 7](#_Toc121430320)

[2.2. Method 8](#_Toc121430321)

[2.3. Conclusion 8](#_Toc121430322)

[3. Task 3 : Coarse-grained Pruning 9](#_Toc121430323)

[3.1. Result 9](#_Toc121430324)

[4. Task 4 : Post Training Static Quantization 10](#_Toc121430325)

[5. Report 11](#_Toc121430326)

[5.1. What are the pros and cons of fine-grained pruning and coarse-grained pruning? Which one is more hardware-friendly and why? (5%) 11](#_Toc121430327)

[5.2. Give your opinions on the benchmark table. (about speed and accuracy) (5%) 11](#_Toc121430328)

[5.3. Why do we fuse modules in quantization flow? (5%) 11](#_Toc121430329)

[5.4. In step 3 of post-training static quantization (PTSQ), we insert observers. What’s the use of these observers? (5%) 12](#_Toc121430330)

[5.5. We use PTSQ in task 5. What’s the difference between PTSQ and post-training dynamic quantization? Which one is more hardware-friendly and why? (5%) 12](#_Toc121430331)

[5.6. Other (10%) 13](#_Toc121430332)

# 表目錄

[表 一 clean data及dirty data的Accuracy交叉比對 6](#_Toc121430333)

[表 二 clean data及dirty data的Confusion Matrix交叉比對 6](#_Toc121430334)

# 圖目錄

[圖 一 最佳model的Confusion Matrix 3](#_Toc121430335)

[圖 二 訓練過程曲線 3](#_Toc121430336)

[圖 三Data Loader匯入兩種資料 4](#_Toc121430337)

[圖 四 利用librosa函式做Data augmentation，(左圖)有提高及降低音頻，有增加雜訊。(右圖)有8種augmentation的方式，在load data時會隨機選擇，而每一種augmentation前面都會加上隨機不同程度的noise 4](#_Toc121430338)

[圖 五 40個檔案的波型，其中包含部分錯誤檔案，但完全無法從中去判斷 5](#_Toc121430339)

[圖 六 直接在 speech\_command\_dataset.py檔裡面匯入新的txt檔 6](#_Toc121430340)

[圖 七 model每一層layer 的 Sparsity table 7](#_Toc121430341)

[圖 八 Testing Accuracy和 Training Accuracy 7](#_Toc121430342)

[圖 九Testing Accuracy和 Training Accuracy 9](#_Toc121430343)

[圖 十 第一次移除20%連接，得到[43,88,176,512]、91.57%的Accuracy及337.4K的parameter 9](#_Toc121430344)

[圖 十一 第二次移除26%連接，得到[20,81,144,512]、92.1%的Accuracy及269.4K的parameter 9](#_Toc121430345)

[圖 十二Quantization 公式 12](#_Toc121430346)

[圖 十三 pytorch官方文件中的add\_observer函式 12](#_Toc121430347)

[圖 十四 計算register中的變量 12](#_Toc121430348)

# Task 1: Training

在本次的實驗中，我在Training過程做了兩個調整，一個是Data augmentation，一個是刪除壞掉的音訊資料，最好的model可以到**91.47%，**以下先展示本次的training結果。。

## Result

圖 一 是model的Confusion Matrix，可以發現二類、第六類及第七類容易混淆，其分別是up、on和off，其發音非常類似；而第零類及第八類，分別為yes和stop，因為發與其他有巨大差別，所以分辨率特別高。

最高的Testing Accuracy達到**91.47%**，下圖 二 看到的93.37%是因為在訓練時，我將有問題的data刪掉後，所得到的數值。

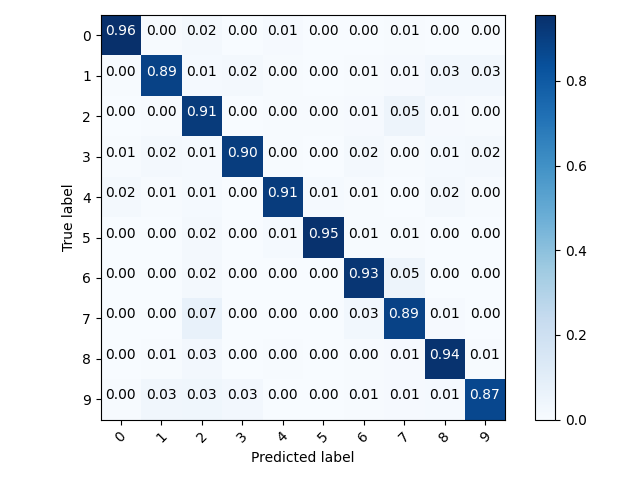


圖 一 最佳model的Confusion Matrix

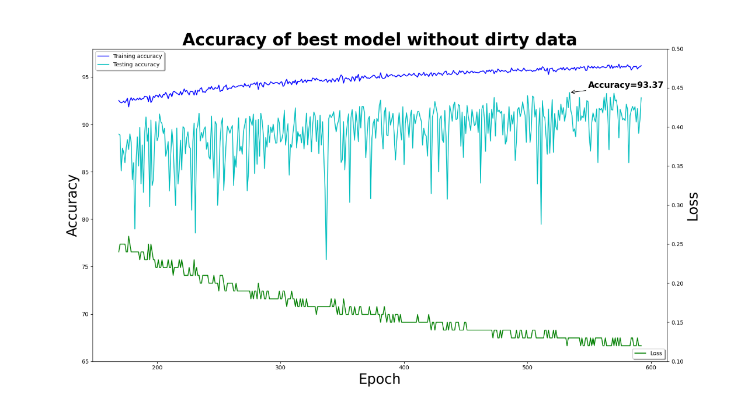


圖 二 訓練過程曲線

## Data augmentation

經過觀察，大多數的聲音檔噪音皆很大，有語調高低的不同、速度

的不同。原本我想利用torchaudio函式中sox\_effects.apply\_effects\_tensor去做處理

，但其僅支援linux系統，於是我只能改用librosa函式做處理，其處理方式是將聲音檔轉為一維的numpy，再將其作處理，最大的缺點就是他不在GPU運算，所以

速度會慢很多。

如圖 三，我是直接在做dataloader的時候就匯入沒有augmentation及有augmentation的data，等於直接將training data 變成兩倍。

一張含有 文字 的圖片

自動產生的描述

圖 三Data Loader匯入兩種資料

|  |  |
| --- | --- |
| 一張含有 文字 的圖片  自動產生的描述 |  |

圖 四 利用librosa函式做Data augmentation，(左圖)有提高及降低音頻，有增加雜訊。(右圖)有8種augmentation的方式，在load data時會隨機選擇，而每一種augmentation前面都會加上隨機不同程度的noise

## Dirty data

經過觀察，在Data裡面，有非常多音訊資料是有問題的，是人耳無法分辨的，包括幾種情況：

* 噪音太大，以至於完全聽不到正確發音
* 聽得懂在念什麼，但完全分類錯誤
* 聽不懂在念什麼，其發音完全無法分辨為該類別

我本來想利用波型去濾除錯誤檔案

，但是我發現根本無法判斷，如圖 五，因為每一種雜訊都太過隨機了，於是我利用第一次train出來的91%的model去找Confusion Matrix

，將每一次判斷錯的training data和testing data列出來，減少檔案數量後，再一個一個去聽，找出錯誤檔案，並列出正確的test\_list\_new及train\_list\_new，然後直接在 speech\_command\_dataset.py檔裡面匯入新的txt檔，如圖 六 所示，以下將有錯誤的資料稱為dirty data，沒有錯誤的稱為clean data。

從表 一 可以看到我的訓練結果，Accuracy並沒有預期中的提高，僅從91.07%提高到91.47%，但如果我用clean data去test的話就會提高到93.41%。

一張含有 桌 的圖片

自動產生的描述

圖 五 40個檔案的波型，其中包含部分錯誤檔案，但完全無法從中去判斷

表 一 clean data及dirty data的Accuracy交叉比對

|  |  |  |
| --- | --- | --- |
|  | 用dirty data去訓練 | 用clean data去訓練 |
| 在dirty data evaluation |  |  |
| 在clean data evaluation | - |  |

一張含有 文字 的圖片

自動產生的描述

圖 六 直接在 speech\_command\_dataset.py檔裡面匯入新的txt檔

表 二 clean data及dirty data的Confusion Matrix交叉比對

|  |  |  |
| --- | --- | --- |
|  | 用dirty data去訓練 | 用clean data去訓練 |
| 在dirty data evaluation |  |  |
| 在clean data evaluation | - |  |

# Task 2 : Fine-grained Pruning

## Result

從下圖 七 可以看到，Fine-grained之前Accuracy是前面所說的91.47%，sparsity為0；而Fine-grained之後Accuracy僅下降到90.98%，而sparsity可以降到68.96%。

從圖 八所示，我做了3次的fine grained第一次降到85.62%、第二次降到88.89%，後來發現都可以很順利train到89%以上，於是最後一次直接看到70.48%。

|  |  |
| --- | --- |
|  |  |

圖 七 model每一層layer 的 Sparsity table

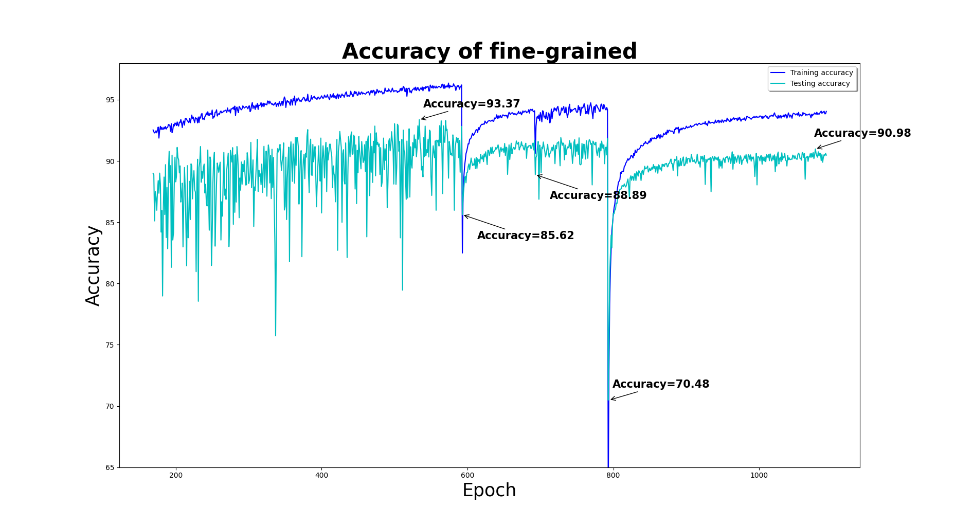


圖 八 Testing Accuracy和 Training Accuracy

## Method

首先，將原本的model做deepcopy，避免更改到原本的model，並將需要Pruning的layer取出來之後，將weight轉成numpy形式，再利用numpy的percentile函式取出絕對值最大的前幾百分位，將其作成一個mask，最後套上原本的weight就完成了。

一張含有 文字 的圖片

自動產生的描述

## Conclusion

從圖 八可以發現，當我Pruning 50%~70% 讓Training Accuracy下降到70%、80%之後，Train第一個epoch就可以快速上升90%以上，從這裡可以看出，model其實是被少數的weight所支配。

# Task 3 : Coarse-grained Pruning

## Result

我做了兩次的Coarse-grained Pruning，其方法就依照助教所提供的code去完成。第一次我弄掉20%的連接，得到[43,88,176,512]，而有91.57%的Accuracy和337.46K的parameter：第二次我弄掉26%的連接，得到[20,81,144,512]，而有92.1%的Accuracy和269.24K的parameter

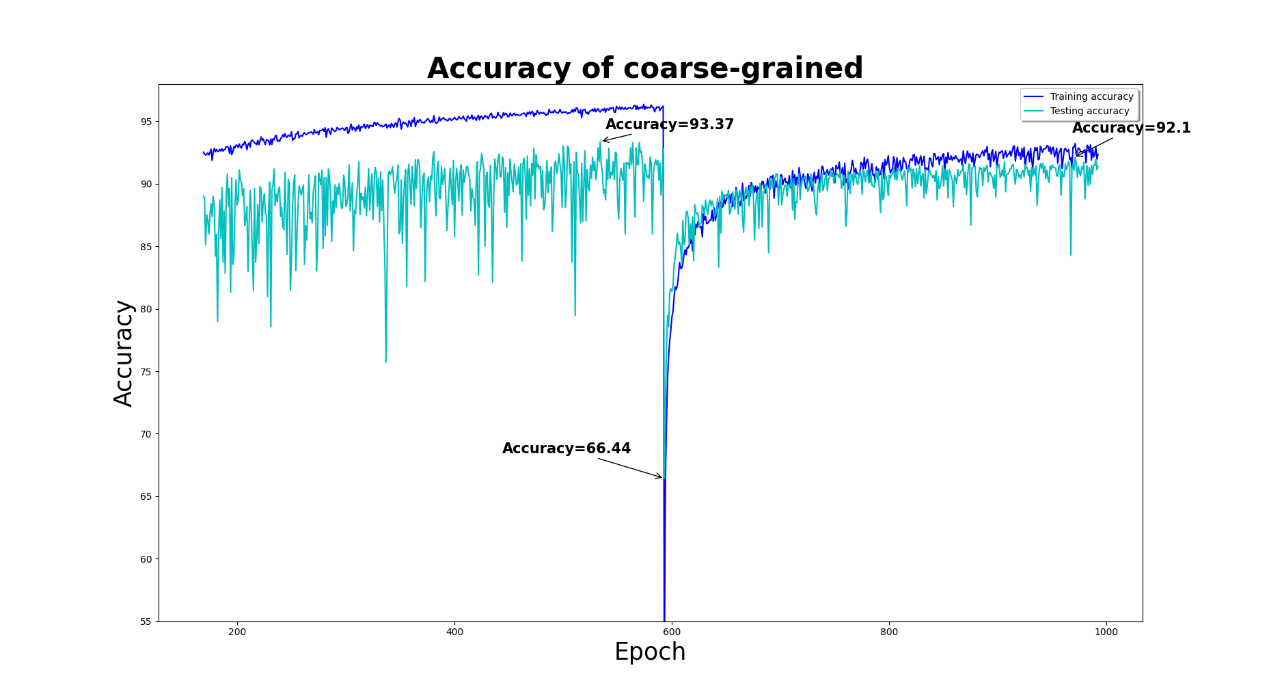


圖 九Testing Accuracy和 Training Accuracy

一張含有 文字 的圖片

自動產生的描述

圖 十 第一次移除20%連接，得到[43,88,176,512]、91.57%的Accuracy及337.4K的parameter

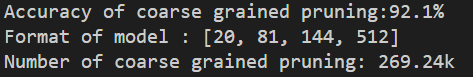


圖 十一 第二次移除26%連接，得到[20,81,144,512]、92.1%的Accuracy及269.4K的parameter

# Task 4 : Post Training Static Quantization

一張含有 文字, 螢幕擷取畫面, 監視器, 黑色 的圖片

自動產生的描述

# Report

## What are the pros and cons of fine-grained pruning and coarse-grained pruning? Which one is more hardware-friendly and why? (5%)

在本次的實驗中fine-grained僅將部分權重調成0，並沒有後續的壓縮或其他調整，在運算速度的提升是非常有限的，幾乎沒有提升。

而在coarse-grained，直接設置threshold的方式，也會讓模型架構有點不平均，應該有更多的調整方式，但整體來說效果比fine-grained好很多。

## Give your opinions on the benchmark table. (about speed and accuracy) (5%)

如上題所述，fine-grained在沒有其他調整的情況下，運算速度很難有所提升，而Quantized是個非常有效率且簡單的調整方式，只是受限於pytorch，在其他實作應用上，可能有些限制。

## Why do we fuse modules in quantization flow? (5%)

如同助教在影片所說，pytorch 所提供的quantization並不支援cuda，因此龐大的torch在cpu裡面會花很多時間去計算，因此fuse\_modules能將多層layer的model融合成一個模塊，以提升它的運算速度，但官方文件有說他也可以提升精確度，但我就不太確定是為什麼。

## In step 3 of post-training static quantization (PTSQ), we insert observers. What’s the use of these observers? (5%)

Quantization 轉化的方式為圖 十二，需要找出scale及zero-point。從官方文件可以看到，會先利用add\_parameter去加入observer，加入之後會再去計算register中的變量histogram、min\_val、max\_val，透過這幾個參數去找出開頭提的cale及zero-point，並計算最終quantization的值

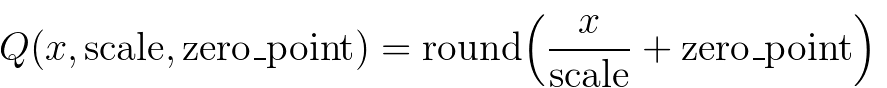


圖 十二Quantization 公式

一張含有 文字 的圖片

自動產生的描述

圖 十三 pytorch官方文件中的add\_observer函式

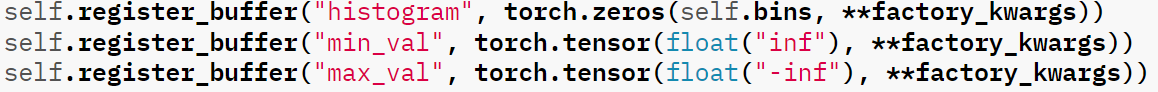


圖 十四 計算register中的變量

## We use PTSQ in task 5. What’s the difference between PTSQ and post-training dynamic quantization? Which one is more hardware-friendly and why? (5%)

兩者都是將float32轉換為int8，而post-training dynamic quantization只支援 nn.Linear and nn.LSTM，實際作用的layer比起PTSQ更少一些，效果也較差。

## Other (10%)

* Training:

若時間夠的話，多做幾次Confusion matrix 應該能把所有的錯誤資料挑出來，並且使用torchaudio函式做data augmentation，Accuracy和效率應該會好很多。

* Pruning:

fine-grained和coarse-grained，應該有更多的調整方式，提升Pruning

的效率，但遺憾沒時間做。而我不太懂fine-grained中的sparsity的意義，實際上並沒有任何的實際效率的提升，或是model的輕量化。

* Quantization

看起來是非常有效率的輕量化選擇，但從pytoech的說明看來，目前限制蠻多的，不知道在實際應用上會不會有困難。