學號：B06901118　　系級：電機二　　姓名： 李高迪

1. **從 Network Pruning/Quantization/Knowledge Distillation/ Low Rank Approximation 選擇兩個方法(並詳述)，將同一個大 model 壓縮至同等數量級，並討論其 accuracy 的變化。 (2%)**

我將這一題分成兩部分，第一部分嘗試壓縮「單一模型」到不同的大小，並說明不同方法的結果和特點。第二部分則討論自己最後繳交模型的訓練方法以及心得。

第一部分： 壓縮「單一模型」

原本的大 model： 我在 HW3 訓練的「model16」 (準確率約 88%) (40.7MB)
(torchsummary 模型架構如下)
https://imgur.com/a/LUFqXrn?fbclid=IwAR0iteIQyep_dyS1GZZ1Pu1GjOSTNtfIZUn-XcH2Rbb1EK9oZeNpYBx8z7Y

註： 原本打算使用助教的模型壓縮，可是 data augmentation 的部分因此有了限制，最後決定採用自己的 HW3 模型，才能使用更多 augmentation

嘗試 1： 將模型壓縮到 10MB 左右

1. Network Pruning (更多詳情可參考報告第 3 題)

由於只使用基本的 Pruning + Fine-Tune，效果沒有很好，在壓縮約 10%後準確率下降許多。如果在每次 pruning 後 initialize weight 重新訓練，則非常耗費時間。最後決定將模型壓縮到所需大小後(10MB)，再重新訓練一次。

重新訓練後的準確率居於 80-85%左右 (因爲 initialize 的 weight 不一樣而有所浮動)
剪枝後準確率沒有大幅下降，可以猜想是原本的模型很龐大，因此許多連接其實沒有很大作用。

2. Quantization

目標是將模型壓縮到原本的 25%，因此立刻想到要用 8bit 表示原本的 fp32。可是，如單純壓縮到 uint8 準確率會有極大幅的下降，因此參考助教建議的作法，先做 min-max normalization、scaling 後以 uint8 存取參數。

壓縮後的模型大小約爲 11MB，準確率約 86%。由此可見，假如一開始是以 fp32 / fp16 (half precision)訓練模型，最後仍可以在不太影響準確率下，將模型壓縮到原來的一半 / 25%。

3. Knowledge Distillation

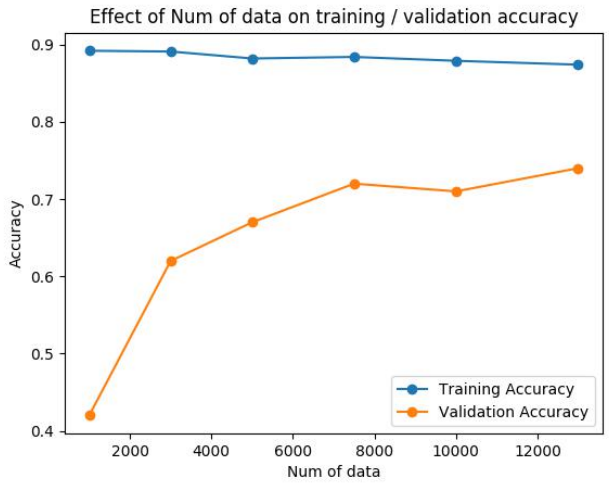
KD 的概念是在 train 小模型時，加入大模型預測的機率分布來幫助小模型學習。大模型在預測時會輸出「soft label」，代表圖片屬於不同 class 的機率。而訓練資料原本的 label 則是「hard label」。KD 實作時一般結合「soft label」及「hard label」，並以參數 alpha 控制 soft label 及 hard label 對 loss 影響的比例。

我實作時的主要方向，是測試使用不同的 alpha，並在訓練資料數量有變化時，看看得出的結果有甚麼趨勢。我訓練時使用原本的 1 萬筆 training data (同時有 soft / hard label)，3 千筆 testing data (只有 soft label)，然後保留 3 千筆 validation data 作測試。

A. 只使用 hard label

這情況跟 HW3 時一樣，所以應該不用多加以解釋 XD。使用跟原本的模型 (Acc~88%)類似的架構及 augmentation，並減少部分 conv filter 及 FC hidden neuron 後，得出的準確率約爲 84%。在模型大小大幅縮減的情況下，準確率下降了約 4%。如果不使用 dropout 及 augmentation，準確率則只有約 60%。

B. 只使用 soft label 且**不使用** dropout / augmentation

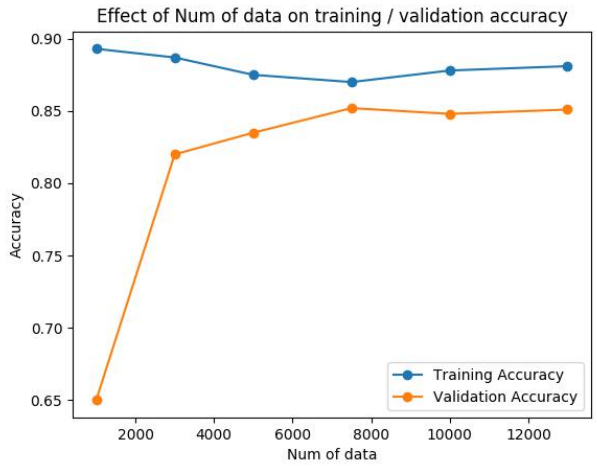


從上圖可見，在不使用 regularization 的情況下，只使用 soft label 作訓練可以在 training set 上達到幾乎完美的效果 (相當接近原本 soft label 的準確率)，然而在 validation set 的結果卻不太理想：模型在 overfit training set 的 soft label。隨著訓練資料量的上升，validation set 的準確率有逐漸上升的趨勢。

我認爲比較特別的地方，是同樣在不使用 dropout / augmentation 的情況下，使用 soft label 訓練比使用 hard label 訓練，val 準確率提升了超過 10%。這證明 soft label 的確代表了大模型的機率分佈，能協助小模型更有效率地學習。

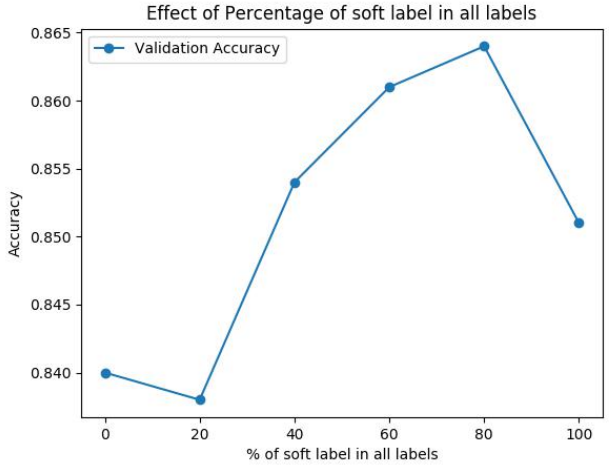
我在 HW4 實作 semi-supervised 時，曾嘗試只使用 unlabel data 的 soft label 訓練(且沒有使用 regularization)，結果在未看過的資料中準確率反而有所提升。
原因是 HW4 的資料非常充足(100 萬筆以上)，因此根本無法在 soft label 的「constraint」下強制記憶 data 與 output 的關係 (也就是不能 overfit)。這次的 task 資料沒有太多，所以只使用 testing data (3000 筆) 的 soft label 訓練就會出現強制記憶的情況。加上 1 萬筆 training data 後情況有所改善，但如果不使用減輕 overfitting 的措施，仍有 overfitting 的情況。

C. 只使用 soft label 且**使用** dropout / augmentation



使用 dropout + augmentation 後，val 準確率明顯提升了許多

D. 以 alpha 調整 soft label 佔全部 label 的比例，觀察模型最佳的表現



在有使用 dropout + augmentation 的情況下，使用 80% soft label + 20% hard label 得出的準確率最高。藉此可以推測訓練的時候，使用較多的 soft label 及較少的 hard label，能比只使用 soft label / hard label 取得更好的結果。但由於取樣數量不足，因此不敢確定推測是否合理。

4. Design Architecture

我覺得如果只是單純獨立使用 Design Architecture，訓練模型時已完全沒有用到原本模型的資訊，因此很難直接跟其他方法比較準確率。因此，我在報告第 4 題才用另外的架構實作 Design Architecture，詳情可參考報告第 4 題。

小結：

方法	最佳 Validation Accuracy
原本模型	88%
Network Pruning	85%
Quantization	86%
Knowledge Distillation	86.5%

相信在結合以上三種做法後，Validation Accuracy 可以非常接近原本的模型！

嘗試 2： 將模型壓縮到 1MB 左右

1. Network Pruning

要將模型壓縮到原本的 2.5%非常困難，實作中只成功壓到原本的 10%，之後就出現莫名奇妙的 bug。不過，在壓到約 4MB 的時候，模型的準確率只剩 40%左右，可以推算再壓下去結果只會更不樂觀。

2. Quantization

直接以 Quantization 將模型從 40MB 壓到 1MB 不太可行。因此，我先用嘗試 1 實作的 Knowledge Distillation 訓練出 10MB 的模型(準確率約 86.5%)，然後再用 uint8 表示，最後的 val acc 約 84%，模型大小 2.5MB。

3. Knowledge Distillation

參考嘗試 1 將模型壓縮到 10MB 的表現，我選擇使用 75% soft label 及 25% hard label 作訓練。最後在 1MB 的模型上，training acc 爲 87%，val acc 爲 85%。

小結：

只有 Knowledge Distillation 能直接把模型從 40MB 壓到 1MB。

我認爲助教提供大家參考的建議相當合理，也就是先用 design architecture 設計出較爲「輕巧」的架構，然後使用 soft label 作 knowledge distillation，最後使用 Quantization 將模型大小壓到原本的 25%。如果模型大小還差一點點，才再考慮 network pruning (但最多只能 prune 10%以下，不然準確率下降非常多)

第二部分： 我最後繳交的小模型

	Activation shape	參數數目
Input	(3, 64, 64)	
Conv 1a	(16, 64, 64)	448
Conv 1b	(16, 64, 64)	2320
MaxPool 1	(16, 32, 32)	
Conv 2a	(32, 32, 32)	4640
Conv 2b	(32, 32, 32)	9248
MaxPool 2	(32, 16, 16)	
Conv 3a	(64, 16, 16)	18496
Conv 3b	(64, 16, 16)	36928
MaxPool 3	(64, 8, 8)	
Conv 4	(64, 16, 16)	36928
MaxPool 4	(64, 8, 8)	
Full Connect	(64*4*4, 11)	11275

訓練參數量：120859 模型大小：276.8 KB

原本的大 model： HW3 8 個模型的 ensemble (準確率約 92.1%)

Knowledge Distillation

我是取我 HW3 的 ensemble，輸出在 testing data 上預測的 soft label。接著將 soft label normalize 後(如數值太大或太小很難訓練)，再丟給小模型作 soft label data。訓練 450Epoch 後的準確率約爲 90.5-91.5%。

不使用 Dropout 或 data augmentation

只在 testing set 上做 KD 能在 testing set 得到很好的 performance，但卻只是在 overfit testing set (在未看過的 data 只有 50% accuracy)。如果將 training 的 soft label 都放進去跟 testing set 一起 train (不含 validating set)，則能在 validating set 取得 80%以上的 performance。雖然使用 dropout 及 data augmentation 後能讓 validating set 的準確率大幅提升，但由於 kaggle 只看 testing set(額...)，因此我的目標反而是要讓模型盡力 overfit testing set。

Quantization

Pytorch 訓練時預設使用 fp32 浮點數，但也支援使用 fp16 訓練 (half precision)。只要將 x,y 改成 half tensor，並設「model = Classifier().cuda().half()」即可。這樣就能在幾乎完全不影響正確率的前提下，減少模型一半大小。

我曾嘗試在訓練結束後進一步壓縮模型，壓縮到 int8 後準確率極大幅度下降，壓縮到 fp8 (取 min-max、scale 及 normalize 後以 uint8 存取)後準確率則下降約 1%。由於原本的模型已符合 300KB 以下的要求，因此我最後沒有採用壓縮到 8bit 的模型。

Low Rank Approximation (Design Architecture)

雖然使用(NMkk / Nkk+NM)後，能讓模型在 100KB 的大小達到約 89%的準確率，但即使增加 filter / FC 直到模型大小到達 300KB，準確率還是無法超越 90%，因此最後還是只使用普通的 CNN。

註： 我亦曾嘗試將以上方法套用到助教提供的 Resnet 18，在 testing data 上作 training，最後在 testing data 得出的準確率約爲 88%。

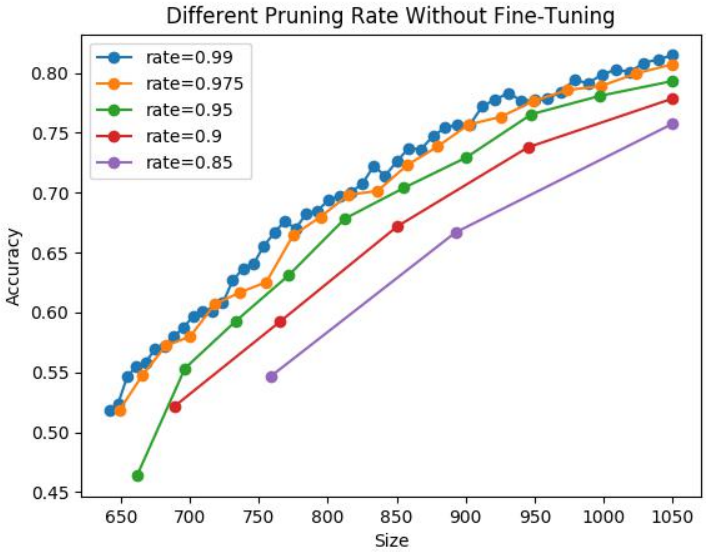
2. **[Knowledge Distillation] 請嘗試比較以下 validation accuracy (兩個 Teacher Net 由助教提供)以及 student 的總參數量以及架構，並嘗試解釋爲甚麼有這樣的結果。你的 Student Net 的參數量必須要小於 Teacher Net 的參數量。(2%)**

因爲時間不太足夠，而且已在第一題詳細說明實作 KD 後的結果和觀察，所以這題只好先略過 QQ

3. **[Network Pruning] 請使用兩種以上的 pruning rate 畫出 X 軸爲參數量，Y 軸爲 validation accuracy 的折線圖。(2%)**

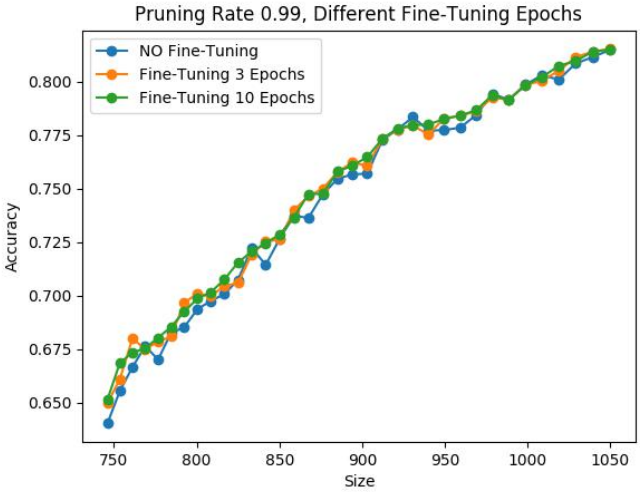
我在這一題使用的原始模型，爲 hw7_Architecture_Design.ipynb 中的 StudentNet，原始準確率約爲 81.5%，模型大小約爲 1.05MB。

1. 不使用任何 Fine-Tuning，調整 Pruning Rate



從上圖可見，每一次 Prune 的 neuron 數量愈小(Prune rate 愈高)，準確率下降的速度就會較慢。因此，在相同模型大小下，Prune rate 愈高準確率愈高。

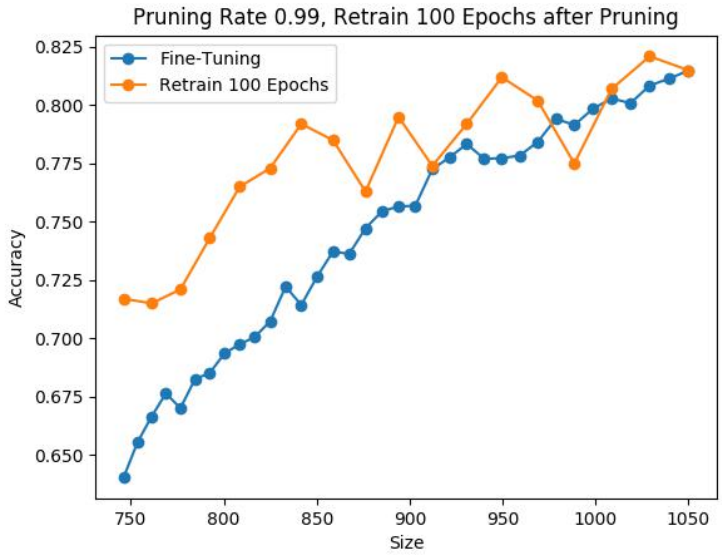
2. 固定 Pruning Rate 0.99，調整 Fine-Tuning Epochs 數量



(我計算 Fine-Tuning 後準確率的方式是取所有 Epochs 中的最高 val acc，因此實質上 Fine-Tuning 後的準確率應該還要再低一點點)

從上圖可見，不論在每次 Prune 後 Fine-Tuning 了多少 Epochs，對準確率的影響不大。我在最後的觀察中會嘗試解釋這現象。

3. 固定 **Pruning rate 0.99，Fine-Tuning or 重新訓練?**



由於 Fine-Tuning 看似沒有太大作用，而且在不少 paper 看到比起 Fine-Tuning，prune 後的模型「train from scratch」往往會有更好的 performance (以下圖爲例)

Dataset	Model	Unpruned	Pruned Model	Fine-tuned	Scratch-E	Scratch-B
CIFAR-10	VGG-16	93.63 (±0.16)	VGG-16-A	93.41 (±0.12)	93.62 (±0.11)	**93.78** (±0.15)
	ResNet-56	93.14 (±0.12)	ResNet-56-A	92.97 (±0.17)	92.96 (±0.26)	**93.09** (±0.14)
			ResNet-56-B	92.67 (±0.14)	92.54 (±0.19)	**93.05** (±0.18)
	ResNet-110	93.14 (±0.24)	ResNet-110-A	93.14 (±0.16)	**93.25** (±0.29)	93.22 (±0.22)
			ResNet-110-B	92.69 (±0.09)	92.89 (±0.43)	**93.60** (±0.25)
ImageNet	ResNet-34	73.31	ResNet-34-A	72.56	72.77	**73.03**
			ResNet-34-B	72.29	72.55	**72.91**

因此，我嘗試在每 prune 一次後，重新 initialize weight，訓練 100 Epochs，然後才繼續 pruning。這樣的步驟一直重複下去，得出的結果明顯進步了許多。只是，由於每次重新訓練時 initialize 的 weight 是隨機的，因此訓練時的隨機性較大，在 Pruning 的過程中準確率有較大的上下震盪。

觀察及感想

1. Prune Rate 的極限

當 Prune Rate 愈靠近 1，準確率下降的幅度就愈低。我曾嘗試使用 Rate=0.999、0.9999，模型也有輕微的進步。然而，如將下降幅度的曲線向上推(interpolate)，即使將 Prune Rate 設到趨於 1 的極限(每次移除 1 個 neuron)，準確率依舊下降很多。似乎 Network Pruning 還是只適合作爲 network compression 的最後微調。

2. Fine-Tuning 沒有用？

許多有關 Network Pruning 的 paper 均使用 Fine-Tuning，因此 Fine-Tuning 應該是有用的(?)。抱持著 Fine-Tuning 應該要有用的心態，我嘗試 Prune HW3 自己的模型(Acc~88%)：

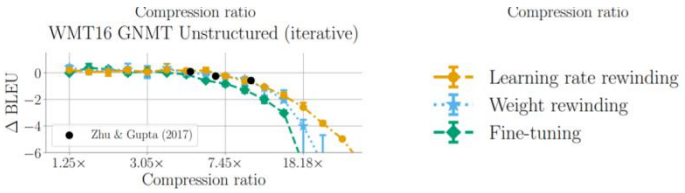
在 Prune 了 30%的 neuron 後，準確率估計爲 75%左右 。除了準確率下降的幅度比 StudentNet 低以外，還注意到使用 Fine-Tuning 後，準確率是有所提升的 (約 5%)

我認爲 Fine-Tuning 在這次 StudentNet 沒太大效果的可能原因，是(NMkk / Nkk+NM)的架構太「特別」，連接很 sparse，有可能令每一個 neuron 的重要性都比較大。此外，Prune 後的架構更爲怪異，可能在 forward/backpropogation 時參數 update 更困難 (因爲觀察到有時候 Fine-Tuning 後準確率反而有所降低)

3. 比重新訓練更有效率的方法?

由於每 prune 一次就重新訓練非常耗時，因此我嘗試將模型 prune 到原來的四分之一後，才重新訓練一次。得出的準確率約爲 68%，比沒有重新訓練的模型好(約 64%)，但比每 prune 一次都重新訓練的差(約 71%)。我認爲如果時間不足的話，可以考慮這種方法。

此外，我亦有在 paper 看到有可能比 Fine-tuning 效果更好的方法，如 weight rewinding 及其變異 learning rate rewinding。



參考資料

What is the State of Neural Network Pruning?
https://arxiv.org/pdf/2003.03033.pdf

RETHINKING THE VALUE OF NETWORK PRUNING
https://openreview.net/pdf?id=rJlnB3C5Ym

COMPARING REWINDING AND FINE-TUNING IN NEURAL NETWORK PRUNING
https://openreview.net/pdf?id=S1gSj0NKvB

我在 Prune 自己的 CNN 時用到的 code
https://github.com/jacobgil/pytorch-pruning

4. **[Low Rank Approx / Model Architecture] 請嘗試比較以下 validation accuracy，並且模型大小須接近 1 MB。 (2%)**

控制變數

1. Training data： 9866 筆 Validating data： 3430 筆
   (因爲要保留 validating data 來測試 accuracy，因此模型準確率並沒有很高)
2. 沒有使用其他 network compression 的方法
3. 由於 network 比較小，一律將圖片 resize 到 64x64
4. **Data augmentation**
   嘗試 data augmentation 時，發現使用太多 augmentation 在 1MB 的小模型會完全訓練不起來，甚至在 training data 上的準確率也無法超越 70% (HW3 中共使用 7 種 augmentation)。但如果完全不使用 augmentation，則會出現嚴重的 overfitting。因此最後只保留了較簡單的 augmentation (random horizontal flip, random rotation(15), colorjitter)，效果最好。
5. 訓練 200 Epoch： 取 validating accuracy 最高的 5 Epoch 作平均
6. **模型架構的相似度**
   爲了能更準確地比較不同 architecture 的準確率，我儘量讓架構在某些部分有同樣的設計，例如 2 層 FC Layer，Dropout(0.5)，相同的 optimizer 等等。由於使用 Depth/Pointwise/Group 後卷積層的參數數目明顯減少，因此我略爲增加 FC Layer neuron 的數目以及 filter 數目，以讓模型大小接近 1MB。

測試結果

架構	Training Acc	Validation Acc	模型大小	Groups
CNN model (用一般的 Convolution Layer)	83%	**72.05%**	1.008M	/
將 CNN model 的 Conv Layer 換成參數量接近的 Depthwise & Pointwise	77%	70.55%	0.688M	In_filter
	83%	**72.20%**	1.004M	In_filter
將 CNN model 的 Conv 換成參數量接近的 Group Conv	77%	70.5%	0.694M	In_filter/2
	79%	71.3%	0.711M	In_filter/4
	79%	**71.9%**	0.728M	In_filter/8
	84%	**72.5%**	0.992M	In_filter/8

實作詳情

a. 原始 CNN model (用一般的 Convolution Layer)

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 16, 64, 64]	448
BatchNorm2d-2	[-1, 16, 64, 64]	32
ReLU-3	[-1, 16, 64, 64]	0
Conv2d-4	[-1, 16, 64, 64]	2,320
BatchNorm2d-5	[-1, 16, 64, 64]	32
ReLU-6	[-1, 16, 64, 64]	0
MaxPool2d-7	[-1, 16, 32, 32]	0
Conv2d-8	[-1, 32, 32, 32]	4,640
BatchNorm2d-9	[-1, 32, 32, 32]	64
ReLU-10	[-1, 32, 32, 32]	0
Conv2d-11	[-1, 32, 32, 32]	9,248
BatchNorm2d-12	[-1, 32, 32, 32]	64
ReLU-13	[-1, 32, 32, 32]	0
MaxPool2d-14	[-1, 32, 16, 16]	0
Conv2d-15	[-1, 64, 16, 16]	18,496
BatchNorm2d-16	[-1, 64, 16, 16]	128
ReLU-17	[-1, 64, 16, 16]	0
Conv2d-18	[-1, 64, 16, 16]	36,928
BatchNorm2d-19	[-1, 64, 16, 16]	128
ReLU-20	[-1, 64, 16, 16]	0
MaxPool2d-21	[-1, 64, 8, 8]	0
Conv2d-22	[-1, 64, 8, 8]	36,928
BatchNorm2d-23	[-1, 64, 8, 8]	128
ReLU-24	[-1, 64, 8, 8]	0
MaxPool2d-25	[-1, 64, 4, 4]	0
Dropout2d-26	[-1, 64, 4, 4]	0
Linear-27	[-1, 128]	131,200
ReLU-28	[-1, 128]	0
Dropout2d-29	[-1, 128]	0
Linear-30	[-1, 11]	1,419

Total params: 242,203
Trainable params: 242,203
Non-trainable params: 0

我先模仿 HW3 的方式建構簡單的 CNN，並使用 Dropout(0.5)減輕 overfitting 的問題，得出的準確率約爲 72%。

觀察到約 6 成的參數量皆集中在 Conv Layer Flatten 到 Full-Connect Layer 的部分，而後續替換 Conv Layer 不能減少這部分的參數量。因此，我曾嘗試進一步減少 FC Layer 的 neuron 數量。測試後發現準確率下降了蠻多，只好維持 128 個 hidden neurons。

b. 將 CNN model 的 Conv Layer 換成參數量接近的 Depthwise & Pointwise

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 16, 64, 64]	448
BatchNorm2d-2	[-1, 16, 64, 64]	32
ReLU-3	[-1, 16, 64, 64]	0
Conv2d-4	[-1, 16, 64, 64]	2,320
BatchNorm2d-5	[-1, 16, 64, 64]	32
ReLU-6	[-1, 16, 64, 64]	0
MaxPool2d-7	[-1, 16, 32, 32]	0
Conv2d-8	[-1, 32, 32, 32]	4,640
BatchNorm2d-9	[-1, 32, 32, 32]	64
ReLU-10	[-1, 32, 32, 32]	0
Conv2d-11	[-1, 32, 32, 32]	9,248
BatchNorm2d-12	[-1, 32, 32, 32]	64
ReLU-13	[-1, 32, 32, 32]	0
MaxPool2d-14	[-1, 32, 16, 16]	0
Conv2d-15	[-1, 32, 16, 16]	320
BatchNorm2d-16	[-1, 32, 16, 16]	64
ReLU-17	[-1, 32, 16, 16]	0
Conv2d-18	[-1, 64, 16, 16]	2,112
Conv2d-19	[-1, 64, 16, 16]	640
BatchNorm2d-20	[-1, 64, 16, 16]	128
ReLU-21	[-1, 64, 16, 16]	0
Conv2d-22	[-1, 64, 16, 16]	4,160
MaxPool2d-23	[-1, 64, 8, 8]	0
Conv2d-24	[-1, 64, 8, 8]	640
BatchNorm2d-25	[-1, 64, 8, 8]	128
ReLU-26	[-1, 64, 8, 8]	0
Conv2d-27	[-1, 64, 8, 8]	4,160
MaxPool2d-28	[-1, 64, 4, 4]	0
Dropout2d-29	[-1, 64, 4, 4]	0
Linear-30	[-1, 128]	131,200
ReLU-31	[-1, 128]	0
Dropout2d-32	[-1, 128]	0
Linear-33	[-1, 11]	1,419

Total params: 161,819
Trainable params: 161,819
Non-trainable params: 0

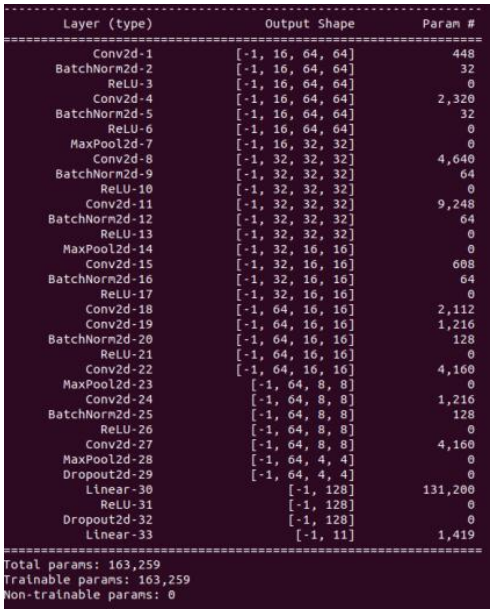
Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 16, 64, 64]	448
BatchNorm2d-2	[-1, 16, 64, 64]	32
ReLU-3	[-1, 16, 64, 64]	0
Conv2d-4	[-1, 16, 64, 64]	2,320
BatchNorm2d-5	[-1, 16, 64, 64]	32
ReLU-6	[-1, 16, 64, 64]	0
MaxPool2d-7	[-1, 16, 32, 32]	0
Conv2d-8	[-1, 32, 32, 32]	4,640
BatchNorm2d-9	[-1, 32, 32, 32]	64
ReLU-10	[-1, 32, 32, 32]	0
Conv2d-11	[-1, 32, 32, 32]	9,248
BatchNorm2d-12	[-1, 32, 32, 32]	64
ReLU-13	[-1, 32, 32, 32]	0
MaxPool2d-14	[-1, 32, 16, 16]	0
Conv2d-15	[-1, 32, 16, 16]	320
BatchNorm2d-16	[-1, 32, 16, 16]	64
ReLU-17	[-1, 32, 16, 16]	0
Conv2d-18	[-1, 64, 16, 16]	2,112
Conv2d-19	[-1, 64, 16, 16]	640
BatchNorm2d-20	[-1, 64, 16, 16]	128
ReLU-21	[-1, 64, 16, 16]	0
Conv2d-22	[-1, 64, 16, 16]	4,160
MaxPool2d-23	[-1, 64, 8, 8]	0
Conv2d-24	[-1, 64, 8, 8]	640
BatchNorm2d-25	[-1, 64, 8, 8]	128
ReLU-26	[-1, 64, 8, 8]	0
Conv2d-27	[-1, 96, 8, 8]	6,240
Conv2d-28	[-1, 96, 8, 8]	960
BatchNorm2d-29	[-1, 96, 8, 8]	192
ReLU-30	[-1, 96, 8, 8]	0
Conv2d-31	[-1, 96, 8, 8]	9,312
MaxPool2d-32	[-1, 96, 4, 4]	0
Dropout2d-33	[-1, 96, 4, 4]	0
Linear-34	[-1, 128]	196,736
ReLU-35	[-1, 128]	0
Dropout2d-36	[-1, 128]	0
Linear-37	[-1, 11]	1,419

Total params: 239,899
Trainable params: 239,899
Non-trainable params: 0

我先嘗試將原始 CNN model 全部 Conv Layer 替換成 Group = in_filters (即 Depthwise)，並加上 1x1 Convolution (Pointwise)，結果準確率下降很多，只剩 65%。其後將首三層 Conv Layer 改回普通的 Conv Layer，準確率提升到約 70.5%，模型大小爲原本的約 7 成。我測試了幾種 Conv / Depthwise 的搭配方式，準確率並沒比最後的架構(上頁左圖)高。

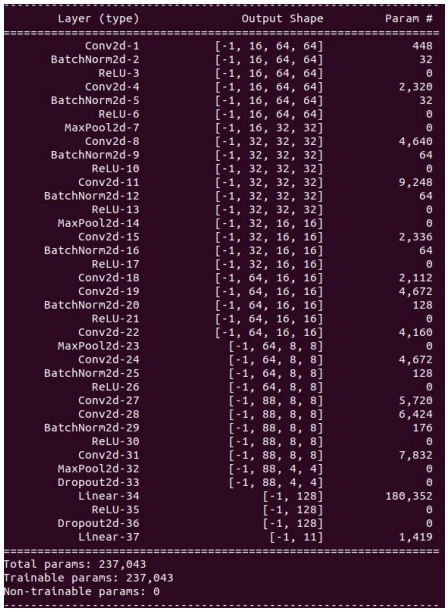
爲了讓模型大小接近 1MB，我嘗試調整 Filter 數量及 Conv Layer 數量。最後的架構額外增加了一層 Conv Layer 及增加 Filter 的數量(如上頁右圖)。準確率卻沒有預期的高，只有 72.2%，跟原始 CNN 模型差不多。

c. 將 CNN model 的 Conv 換成參數量接近的 Group Convolution Layer



Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 16, 64, 64]	448
BatchNorm2d-2	[-1, 16, 64, 64]	32
ReLU-3	[-1, 16, 64, 64]	0
Conv2d-4	[-1, 16, 64, 64]	2,320
BatchNorm2d-5	[-1, 16, 64, 64]	32
ReLU-6	[-1, 16, 64, 64]	0
MaxPool2d-7	[-1, 16, 32, 32]	0
Conv2d-8	[-1, 32, 32, 32]	4,640
BatchNorm2d-9	[-1, 32, 32, 32]	64
ReLU-10	[-1, 32, 32, 32]	0
Conv2d-11	[-1, 32, 32, 32]	9,248
BatchNorm2d-12	[-1, 32, 32, 32]	64
ReLU-13	[-1, 32, 32, 32]	0
MaxPool2d-14	[-1, 32, 16, 16]	0
Conv2d-15	[-1, 32, 16, 16]	608
BatchNorm2d-16	[-1, 32, 16, 16]	64
ReLU-17	[-1, 32, 16, 16]	0
Conv2d-18	[-1, 64, 16, 16]	2,112
Conv2d-19	[-1, 64, 16, 16]	1,216
BatchNorm2d-20	[-1, 64, 16, 16]	128
ReLU-21	[-1, 64, 16, 16]	0
Conv2d-22	[-1, 64, 16, 16]	4,160
MaxPool2d-23	[-1, 64, 8, 8]	0
Conv2d-24	[-1, 64, 8, 8]	1,216
BatchNorm2d-25	[-1, 64, 8, 8]	128
ReLU-26	[-1, 64, 8, 8]	0
Conv2d-27	[-1, 64, 8, 8]	4,160
MaxPool2d-28	[-1, 64, 4, 4]	0
Dropout2d-29	[-1, 64, 4, 4]	0
Linear-30	[-1, 128]	131,200
ReLU-31	[-1, 128]	0
Dropout2d-32	[-1, 128]	0
Linear-33	[-1, 11]	1,419

Total params: 163,259
Trainable params: 163,259
Non-trainable params: 0



Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 16, 64, 64]	448
BatchNorm2d-2	[-1, 16, 64, 64]	32
ReLU-3	[-1, 16, 64, 64]	0
Conv2d-4	[-1, 16, 64, 64]	2,320
BatchNorm2d-5	[-1, 16, 64, 64]	32
ReLU-6	[-1, 16, 64, 64]	0
MaxPool2d-7	[-1, 16, 32, 32]	0
Conv2d-8	[-1, 32, 32, 32]	4,640
BatchNorm2d-9	[-1, 32, 32, 32]	64
ReLU-10	[-1, 32, 32, 32]	0
Conv2d-11	[-1, 32, 32, 32]	9,248
BatchNorm2d-12	[-1, 32, 32, 32]	64
ReLU-13	[-1, 32, 32, 32]	0
MaxPool2d-14	[-1, 32, 16, 16]	0
Conv2d-15	[-1, 32, 16, 16]	2,336
BatchNorm2d-16	[-1, 32, 16, 16]	64
ReLU-17	[-1, 32, 16, 16]	0
Conv2d-18	[-1, 64, 16, 16]	2,112
Conv2d-19	[-1, 64, 16, 16]	4,672
BatchNorm2d-20	[-1, 64, 16, 16]	128
ReLU-21	[-1, 64, 16, 16]	0
Conv2d-22	[-1, 64, 16, 16]	4,160
MaxPool2d-23	[-1, 64, 8, 8]	0
Conv2d-24	[-1, 64, 8, 8]	4,672
BatchNorm2d-25	[-1, 64, 8, 8]	128
ReLU-26	[-1, 64, 8, 8]	0
Conv2d-27	[-1, 88, 8, 8]	5,720
Conv2d-28	[-1, 88, 8, 8]	6,424
BatchNorm2d-29	[-1, 88, 8, 8]	176
ReLU-30	[-1, 88, 8, 8]	0
Conv2d-31	[-1, 88, 8, 8]	7,832
MaxPool2d-32	[-1, 88, 4, 4]	0
Dropout2d-33	[-1, 88, 4, 4]	0
Linear-34	[-1, 128]	180,352
ReLU-35	[-1, 128]	0
Dropout2d-36	[-1, 128]	0
Linear-37	[-1, 11]	1,419

Total params: 237,043
Trainable params: 237,043
Non-trainable params: 0

Group Convolution Layer 中 Group 的數值介於 1 到 in_filters 之間，且要可以被 in_filter / out_filter 整除。我嘗試將原始 CNN model 全部 Conv Layer 替換成 Group Convolution 及 Pointwise 的架構，同樣發現首三層 Conv Layer 最好保持一般的 Conv Layer。接著，測試不同的 Group 值帶來的效果，以下舉幾個例子：

1. Group = in_filter / 2
準確率約 70.5%，模型大小爲 0.694MB
2. Group = in_filter / 4
準確率約 71.3%，模型大小爲 0.711MB
3. Group = in_filter / 8
準確率約 71.9%，模型大小爲 0.728MB
4. Group = 1 (即原始的 Conv 額外加上 1x1 Conv)
準確率約 72.4%，模型大小爲 1.04MB

觀察

1. Depthwise / Group + Pointwise Conv 有用嗎？

以相同的模型大小，使用 Depthwise / Group + Pointwise Conv Layer 後的準確率並沒有明顯的提升。然而，我仍認爲 NMkk / Nkk+NM 是 Network Compression 的大利器。

將原始 Conv 的準確率(72.05%)，與 Group Conv (group = in_filter/8) 的準確率(71.9%)比較，Group Conv 能在模型大小減少接近 30%後，仍然維持相若的準確率，算是相當厲害！

我有參考 MobileNet 的架構，但這次的模型大小還要比 MobileNet 小很多，因此尋找合適的架構還是有一定的困難。有時候多疊一層 Group Conv 後，準確率就大幅下降 (也不知道爲甚麼)。我覺得若不是要求很小的 network，使用普通的 Conv 倒好像還比較方便。

2. Group Convolution 的 Group 應該設多少？

測試不同的 Group 值後，發現 Group 愈小效果愈好。原因是將 Filter 分成不同的 Group，某程度上只是類近於「將 Fully Connect Layer 拔掉某些 connection」。因此，理論上原本的 Group=1 能達到更多 Group 的表現或是更好。

我覺得分成 4 個 Group (或 in_filter / 8 左右) 可能是比較好的做法。即使只是切成 4 個 Group，模型大小已經下降了接近 3 成，但準確率卻沒有大幅下降。如果切成更多 Group，模型大小下降的速度會愈來愈慢，且準確率會逐漸下降。Depthwise Conv 是 Group Conv 中，Group=in_filter 的特例，將每個 filter 設成一個 group，filter 之間在經過 1x1 conv 之前沒有互相影響。這次測試中 Depthwise Conv 的效果比較不好，某程度上印證了以上的想法。

謝謝助教！