學號：b06902006 系級：資工三 姓名：王俊翔

1. 請從 Network Pruning/Quantization/Knowledge Distillation/Low Rank Approximation/Design Architecture 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%)

原 model	0.8076
Design Architecture	0.8009
Knowledge Distillation	0.7560

```
multiplier = [1, 2, 4, 8, 16, 16, 16, 16]
bandwidth = [ base * m for m in multiplier]
self.cnn = nn.Sequential(
    nn.Sequential(
        nn.Conv2d(3, bandwidth[0], 3, 1, 1), # 16 1
        nn.BatchNorm2d(bandwidth[0]),
        nn.ReLU6(),
        nn.MaxPool2d(2, 2, 0), # 16 /2
    ),
    nn.Sequential(
        nn.Conv2d(bandwidth[0], bandwidth[1], 3, 1, 1),
        nn.BatchNorm2d(bandwidth[1]),
        nn.ReLU6(),
        nn.MaxPool2d(2, 2, 0), # 32 /4
    ),
    nn.Sequential(
        nn.Conv2d(bandwidth[1], bandwidth[2], 3, 1, 1),
        nn.BatchNorm2d(bandwidth[2]),
        nn.ReLU6(),
        nn.MaxPool2d(2, 2, 0), # 64 /8
    ),
    nn.Sequential(
        nn.Conv2d(bandwidth[2], bandwidth[3], 3, 1, 1),
        nn.BatchNorm2d(bandwidth[3]),
        nn.ReLU6(),
        nn.MaxPool2d(2, 2, 0), # 128 /16
    ),
    nn.Sequential(
        nn.Conv2d(bandwidth[3], bandwidth[4], 3, 1, 1),
        nn.BatchNorm2d(bandwidth[4]),
        nn.ReLU6(),
    ),
    nn.Sequential(
        nn.Conv2d(bandwidth[4], bandwidth[5], 3, 1, 1),
        nn.BatchNorm2d(bandwidth[5]),
        nn.ReLU6(),
    ),
    nn.Sequential(
        nn.Conv2d(bandwidth[5], bandwidth[6], 3, 1, 1),
        nn.BatchNorm2d(bandwidth[6]),
        nn.ReLU6(),
    ),
    nn.AdaptiveAvgPool2d((1, 1)),
)
self.fc = nn.Sequential(
    nn.Linear(bandwidth[6], 11),
)
```

```
multiplier = [1, 2, 4, 8, 16, 16, 16, 16]
bandwidth = [ base * m for m in multiplier]
self.cnn = nn.Sequential(
    nn.Sequential(
        nn.Conv2d(3, bandwidth[0], 3, 1, 1), # 16 1
        nn.BatchNorm2d(bandwidth[0]),
        nn.ReLU6(),
        nn.MaxPool2d(2, 2, 0), # 16 /2
    ),
    nn.Sequential(
        nn.Conv2d(bandwidth[0], bandwidth[0], 3, 1, 1, groups=bandwidth[0]),
        nn.BatchNorm2d(bandwidth[0]),
        nn.ReLU6(),
        nn.Conv2d(bandwidth[0], bandwidth[1], 1),
        nn.MaxPool2d(2, 2, 0), # 32 /4
    ),
    nn.Sequential(
        nn.Conv2d(bandwidth[1], bandwidth[1], 3, 1, 1, groups=bandwidth[1]),
        nn.BatchNorm2d(bandwidth[1]),
        nn.ReLU6(),
        nn.Conv2d(bandwidth[1], bandwidth[2], 1),
        nn.MaxPool2d(2, 2, 0), # 64 /8
    ),
    nn.Sequential(
        nn.Conv2d(bandwidth[2], bandwidth[2], 3, 1, 1, groups=bandwidth[2]),
        nn.BatchNorm2d(bandwidth[2]),
        nn.ReLU6(),
        nn.Conv2d(bandwidth[2], bandwidth[3], 1),
        nn.MaxPool2d(2, 2, 0), # 128 /16
    ),
    nn.Sequential(
        nn.Conv2d(bandwidth[3], bandwidth[3], 3, 1, 1, groups=bandwidth[3]),
        nn.BatchNorm2d(bandwidth[3]),
        nn.ReLU6(),
        nn.Conv2d(bandwidth[3], bandwidth[4], 1), #256 /16
    ),
    nn.Sequential(
        nn.Conv2d(bandwidth[4], bandwidth[4], 3, 1, 1, groups=bandwidth[4]),
        nn.BatchNorm2d(bandwidth[4]),
        nn.ReLU6(),
        nn.Conv2d(bandwidth[4], bandwidth[5], 1), #256 /16
    ),
    nn.Sequential(
        nn.Conv2d(bandwidth[5], bandwidth[5], 3, 1, 1, groups=bandwidth[5]),
        nn.BatchNorm2d(bandwidth[5]),
        nn.ReLU6(),
        nn.Conv2d(bandwidth[5], bandwidth[6], 1), #256 /16
    ),
    nn.AdaptiveAvgPool2d((1, 1)),
)
self.fc = nn.Sequential(
    nn.Linear(bandwidth[6], 11),
)

def forward(self, x):
    out = self.cnn(x)
    out = out.view(out.size()[0], -1)
    return self.fc(out)
```

```
    multiplier = [1, 2, 4, 6, 8, 16, 16]
    bandwidth = [ base * m for m in multiplier]
    self.cnn = nn.Sequential(
        nn.Sequential(
            nn.Conv2d(3, bandwidth[0], 3, 1, 1), # 16 1
            nn.BatchNorm2d(bandwidth[0]),
            nn.ReLU6(),
            nn.MaxPool2d(2, 2, 0), # 16 /2
        ),
        nn.Sequential(
            nn.Conv2d(bandwidth[0], bandwidth[1], 3, 1, 1),
            nn.BatchNorm2d(bandwidth[1]),
            nn.ReLU6(),
            nn.MaxPool2d(2, 2, 0), # 32 /4
        ),
        nn.Sequential(
            nn.Conv2d(bandwidth[1], bandwidth[2], 3, 1, 1),
            nn.BatchNorm2d(bandwidth[2]),
            nn.ReLU6(),
            nn.MaxPool2d(2, 2, 0), # 64 /8
        ),
        nn.Sequential(
            nn.Conv2d(bandwidth[2], bandwidth[3], 3, 1, 1),
            nn.BatchNorm2d(bandwidth[3]),
            nn.ReLU6(),
            nn.MaxPool2d(2, 2, 0), # 128 /16
        ),
        nn.Sequential(
            nn.Conv2d(bandwidth[3], bandwidth[4], 3, 1, 1),
            nn.BatchNorm2d(bandwidth[4]),
            nn.ReLU6(),
        ),
        nn.AdaptiveAvgPool2d((1, 1)),
    )
    self.fc = nn.Sequential(
        nn.Linear(bandwidth[4], 11),
    )

def forward(self, x):
    out = self.cnn(x)
    out = out.view(out.size()[0], -1)
    return self.fc(out)
```

(base 皆為 16，每題都是)

Ans： 這裡我選擇 Design Architecture 和 Knowledge Distillation，原 model 如上圖 1 所示，是一個一般的 CNN structure，參數量為 1577611。Design Architecture 的部分，我把他所有的 layer 全部改成 DW+PW 去做處理(如上圖 2 這樣做改變)，其餘部分不變，這樣的參數量變成 187915；knowledge distillation 的部分設計出如上圖 3 的一般 CNN structure (層數不能太多參數會爆)，參數量為 191787，由上表可以知道，原 accuracy 與 design architecture 後準確率並沒有差距太多，可見這樣的 design 其實蠻適合這個 case 的；然而，knowledge distillation 的效果並沒有想像中好，最大的原因我認為是 architecture 太差了，如果使用的是上述 Design Architecture 的架構，相信結果會提升不少。

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，並嘗試解釋為甚麼有這樣的結果。你的 Student Net 的參數量必須要小於 Teacher Net 的參數量。(2%)

   x. Teacher net architecture and # of parameters: torchvision’s ResNet18, with 11,182,155 parameters.

y. Student net architecture and # of parameters: mobilenet v1, with 276267 parameters.

a. Teacher net (ResNet18) from scratch: 80.09%

b. Teacher net (ResNet18) ImageNet pretrained & fine-tune: 88.41%

c. Your student net from scratch: 78.89%

d. Your student net KD from (a.): 83.62%

e. Your student net KD from (b.): 84.23%

這題的話我的 knowledge distillation 是使用 teacher 跟 student 的 mutual learning，student net 的架構是 mobilenet v1 只有層數稍微不一樣，我的 net 如果單跑的話只有約 79%的 accuracy，不過由於加上了 mutual learning，不管是與 a 或 b 一起跑都有明顯的提升，其中 fine tuned 過的稍微好一點點(由於跟 teacher 一起 train，teacher 也會一起進步)，差異並不明顯。

3. [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%)

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

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

c. 將 CNN model 的 Convolution Layer 換成參數量接近的 Group Convolution Layer (Group 數量自訂，但不要設為 1 或 in_filters)

```
    multiplier = [1, 2, 4, 8, 8, 16, 16]
    bandwidth = [ base * m for m in multiplier]
    self.cnn = nn.Sequential(
        nn.Sequential(
            nn.Conv2d(3, bandwidth[0], 3, 1, 1), # 16 1
            nn.BatchNorm2d(bandwidth[0]),
            nn.ReLU6(),
            nn.MaxPool2d(2, 2, 0), # 16 /2
        ),
        nn.Sequential(
            nn.Conv2d(bandwidth[0], bandwidth[1], 3, 1, 1),
            nn.BatchNorm2d(bandwidth[1]),
            nn.ReLU6(),
            nn.MaxPool2d(2, 2, 0), # 32 /4
        ),
        nn.Sequential(
            nn.Conv2d(bandwidth[1], bandwidth[2], 3, 1, 1),
            nn.BatchNorm2d(bandwidth[2]),
            nn.ReLU6(),
            nn.MaxPool2d(2, 2, 0), # 64 /8
        ),
        nn.Sequential(
            nn.Conv2d(bandwidth[2], bandwidth[3], 3, 1, 1),
            nn.BatchNorm2d(bandwidth[3]),
            nn.ReLU6(),
            nn.MaxPool2d(2, 2, 0), # 128 /16
        ),
        nn.Sequential(
            nn.Conv2d(bandwidth[3], bandwidth[4], 3, 1, 1),
            nn.BatchNorm2d(bandwidth[4]),
            nn.ReLU6(),
        ),

        nn.AdaptiveAvgPool2d((1, 1)),
    )
    self.fc = nn.Sequential(
        nn.Linear(bandwidth[4], 11),
    )

def forward(self, x):
    out = self.cnn(x)
    out = out.view(out.size()[0], -1)
    return self.fc(out)
```

(a)

```
    multiplier = [2, 4, 8, 8, 16, 16, 16]
    bandwidth = [ base * m for m in multiplier]
    self.cnn = nn.Sequential(
        nn.Sequential(
            nn.Conv2d(3, bandwidth[0], 3, 1, 1), # 16 1
            nn.BatchNorm2d(bandwidth[0]),
            nn.ReLU6(),
            nn.MaxPool2d(2, 2, 0), # 16 /2
        ),
        nn.Sequential(
            nn.Conv2d(bandwidth[0], bandwidth[1], 3, 1, 1, groups = 4),
            nn.BatchNorm2d(bandwidth[1]),
            nn.ReLU6(),
            nn.MaxPool2d(2, 2, 0), # 32 /4
        ),
        nn.Sequential(
            nn.Conv2d(bandwidth[1], bandwidth[2], 3, 1, 1, groups = 4),
            nn.BatchNorm2d(bandwidth[2]),
            nn.ReLU6(),
            nn.MaxPool2d(2, 2, 0), # 64 /8
        ),
        nn.Sequential(
            nn.Conv2d(bandwidth[2], bandwidth[3], 3, 1, 1, groups = 4),
            nn.BatchNorm2d(bandwidth[3]),
            nn.ReLU6(),
            nn.MaxPool2d(2, 2, 0), # 128 /16
        ),
        nn.Sequential(
            nn.Conv2d(bandwidth[3], bandwidth[4], 3, 1, 1, groups = 4),
            nn.BatchNorm2d(bandwidth[4]),
            nn.ReLU6(),
        ),
        nn.Sequential(
            nn.Conv2d(bandwidth[4], bandwidth[5], 3, 1, 1, groups = 4),
            nn.BatchNorm2d(bandwidth[5]),
            nn.ReLU6(),
        ),
        nn.AdaptiveAvgPool2d((1, 1)),
    )
    self.fc = nn.Sequential(
        nn.Linear(bandwidth[5], 11),
    )

def forward(self, x):
    out = self.cnn(x)
    out = out.view(out.size()[0], -1)
    return self.fc(out)
```

(c)

```
        multiplier = [3, 4, 8, 8, 16, 16, 16, 16, 16]

        # bandwidth: 每一層Layer所使用的ch數量
        bandwidth = [ base * m for m in multiplier]

        # 我們只Pruning第三層以後的Layer
        for i in range(3, 7):
            bandwidth[i] = int(bandwidth[i] * width_mult)

        self.cnn = nn.Sequential(
            nn.Sequential(
                nn.Conv2d(3, bandwidth[0], 3, 1, 1), # 16 1
                nn.BatchNorm2d(bandwidth[0]),
                nn.ReLU6(),
                nn.MaxPool2d(2, 2, 0), # 16 /2
            ),
```

```
        nn.Sequential(
            nn.Conv2d(bandwidth[0], bandwidth[0], 3, 2, 1, groups=bandwidth[0]),
            nn.BatchNorm2d(bandwidth[0]),
            nn.ReLU6(),
            nn.Conv2d(bandwidth[0], bandwidth[1], 1),
        ),

        nn.Sequential(
            nn.Conv2d(bandwidth[1], bandwidth[1], 3, 2, 1, groups=bandwidth[1]),
            nn.BatchNorm2d(bandwidth[1]),
            nn.ReLU6(),
            nn.Conv2d(bandwidth[1], bandwidth[2], 1),
            #nn.MaxPool2d(2, 2, 0), # 64 /8
        ),
        nn.Sequential(
            nn.Conv2d(bandwidth[2], bandwidth[2], 3, 2, 1, groups=bandwidth[2]),
            nn.BatchNorm2d(bandwidth[2]),
            nn.ReLU6(),
            nn.Conv2d(bandwidth[2], bandwidth[3], 1),
            #nn.MaxPool2d(2, 2, 0), # 128 /16
        ),
        nn.Sequential(
            nn.Conv2d(bandwidth[3], bandwidth[3], 3, 1, 1, groups=bandwidth[3]),
            nn.BatchNorm2d(bandwidth[3]),
            nn.ReLU6(),
            nn.Conv2d(bandwidth[3], bandwidth[4], 1), #256 /16
        ),
        nn.Sequential(
            nn.Conv2d(bandwidth[4], bandwidth[4], 3, 1, 1, groups=bandwidth[4]),
            nn.BatchNorm2d(bandwidth[4]),
            nn.ReLU6(),
            nn.Conv2d(bandwidth[4], bandwidth[5], 1), #256 /16
        ),
        nn.Sequential(
            nn.Conv2d(bandwidth[5], bandwidth[5], 3, 1, 1, groups=bandwidth[5]),
            nn.BatchNorm2d(bandwidth[5]),
            nn.ReLU6(),
            nn.Conv2d(bandwidth[5], bandwidth[6], 1), #256 /16
        ),
        nn.Sequential(
            nn.Conv2d(bandwidth[6], bandwidth[6], 3, 1, 1, groups=bandwidth[6]),
            nn.BatchNorm2d(bandwidth[6]),
            nn.ReLU6(),
            nn.Conv2d(bandwidth[6], bandwidth[7], 1), #256 /16
        ),
        nn.AdaptiveAvgPool2d((1, 1)),
    )
    self.fc = nn.Sequential(
        nn.Linear(bandwidth[7], 11),
    )

def forward(self, x):
    out = self.cnn(x)
    out = out.view(out.size()[0], -1)
    return self.fc(out)
```

(b)

Model	Accuracy	Parameters	Size
a	0.7787	247179	976
b	0.7889	276267	1126
c	0.7752	287371	1140

我的結果在參數量接近時，由於 b 可以塞的層數變多了，所以最後的結果是比較好的，不過若層數相近，可能會由於 b 的參數量及 group 的影響導致 b 的 accuracy 變低，至於 c 的部分，照理來說會在 a, b 間，不過可能是架構設計的不是很好，導致他的 accuracy 稍微偏低，不過還在可以接受的範圍 (b 的部分由於是過 strong 的那組所以架構較好是正常的，例如 maxpool 改

成 stride = 2)。