學號：b08901135　系級：電機一　　姓名：曾憲揚

1. 請從 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%)

   大model: Teacher net (ResNet18) ImageNet pretrained & fine-tune: 88.41%
   大小:43Mb
   Knowledge Distillation: Hw3 model當student，大model當teacher。
   大小:50Mb
   結果:

   ```
   epoch 197: train loss: 1.6011, acc 0.8741 val loss: 1.5455,
   acc 0.7998
   epoch 198: train loss: 1.5837, acc 0.8773 val loss: 2.0133,
   acc 0.7904
   epoch 199: train loss: 1.5594, acc 0.8746 val loss: 1.9212,
   acc 0.7900
   ```

   Quantization: 將大model從32bit轉成16bit。
   用validation set測正確率: 89.2%
   大小:22.4Mb
   Quantization只是完全單純的壓縮model，KD的主體還是小model，雖然有teacher，但大小有限制的情況下其實學不到太多的東西，因此Quantization比較好應該是正常的.

   以下三題只需要選擇兩者即可，分數取最高的兩個。

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:
   `hw7_Architecture_Design.ipynb`裡面的`model`，參數量大約為`20~30W`

   a. Teacher net (ResNet18) from scratch: 80.09%
   b. Teacher net (ResNet18) ImageNet pretrained & fine-tune: 88.41%
   c. Your student net from scratch: val acc 64.67%
   d. Your student net KD from (a.): 79.12%
   e. Your student net KD from (b.): 83.18%

Accuracy的結果完全合理，老師的強度的排序依序為e>d>c，有越強大的老師就可以學到越多的知識，因此KD的老師的能力直接反映在accuracy結果上面。c完全是靠自己，參數量少又沒有KD，因此performance會差很多。

3. [Network Pruning] 請使用兩種以上的 pruning rate 畫出 X 軸為參數量，Y 軸為 validation accuracy 的折線圖。你的圖上應該會有兩條以上的折線。(2%)
沒做0.0。

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

```
class Classifier(nn.Module):
    def __init__(self):
        super(Classifier, self).__init__()
        # torch.nn.Conv2d(in_channels, out_channels, kernel_size,
stride, padding)
        # torch.nn.MaxPool2d(kernel_size, stride, padding)
        # input 維度 [3, 128, 128]
        self.cnn = nn.Sequential(
            nn.Conv2d(3, 25, 3, 1, 1),  # [64, 128, 128]
            nn.BatchNorm2d(25),
            nn.ReLU(),
            nn.MaxPool2d(2, 2, 0),      # [64, 64, 64]

            nn.Conv2d(25, 50, 3, 1, 1),  # [64, 128, 128]
            nn.BatchNorm2d(50),
            nn.ReLU(),
            nn.MaxPool2d(2, 2, 0),      # [64, 64, 64]

            nn.Conv2d(50, 100, 3, 1, 1),  # [64, 128, 128]
            nn.BatchNorm2d(100),
            nn.ReLU(),
            nn.MaxPool2d(4, 4, 0),      # [64, 64, 64]

            nn.Conv2d(100, 192, 3, 1, 1),  # [64, 128, 128]
            nn.BatchNorm2d(192),
            nn.ReLU(),
            nn.MaxPool2d(4, 4, 0),      # [64, 64, 64]
        )
```

```
        self.fc = nn.Sequential(
            nn.Linear(192*2*2, 64),
            nn.ReLU(),
            nn.Linear(64, 11),
        )

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

results:

```
[026/030] Val Acc: 0.417493 loss: 0.013527
[027/030] Val Acc: 0.416910 loss: 0.013403
[028/030] Val Acc: 0.419825 loss: 0.013324
[029/030] Val Acc: 0.424490 loss: 0.013256
[030/030] Val Acc: 0.430612 loss: 0.013163
```

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

P.S. 這裡基本上是抄助教給的studentnet，只有稍微做一點點修改

```
self.cnn = nn.Sequential(

            nn.Sequential(
                nn.Conv2d(3, 16, 3, 1, 1),
                nn.BatchNorm2d(16),
                nn.ReLU6(),
                nn.Conv2d(16, 16, 3, 1, 1),
                nn.BatchNorm2d(16),
                nn.ReLU6(),
                nn.MaxPool2d(2, 2, 0),
            ),
            nn.Sequential(

                nn.Conv2d(16, 16, 3, 1, 1, groups=16),

                nn.BatchNorm2d(16),
                nn.ReLU6(),
```

```
        nn.Conv2d(16, 32, 1),
        nn.Conv2d(32, 32, 3, 1, 1, groups=32),

        nn.BatchNorm2d(32),
        nn.ReLU6(),

        nn.Conv2d(32, 32, 1),
        nn.MaxPool2d(2, 2, 0),
    ),
    nn.Sequential(
        nn.Conv2d(32, 32, 3, 1, 1, groups=32),
        nn.BatchNorm2d(32),
        nn.ReLU6(),
        nn.Conv2d(32, 64, 1),
        nn.Conv2d(64, 64, 3, 1, 1, groups=64),
        nn.BatchNorm2d(64),
        nn.ReLU6(),
        nn.Conv2d(64, 64, 1),
        nn.Conv2d(64, 64, 3, 1, 1, groups=64),
        nn.BatchNorm2d(64),
        nn.ReLU6(),
        nn.Conv2d(64, 64, 1),
        nn.MaxPool2d(2, 2, 0),
    ),
    nn.Sequential(
        nn.Conv2d(64, 64, 3, 1, 1, groups=64),
        nn.BatchNorm2d(64),
        nn.ReLU6(),
        nn.Conv2d(64, 128, 1),
        nn.Conv2d(128, 128, 3, 1, 1, groups=128),
        nn.BatchNorm2d(128),
        nn.ReLU6(),
        nn.Conv2d(128, 128, 1),
        nn.Conv2d(128, 128, 3, 1, 1, groups=128),
        nn.BatchNorm2d(128),
        nn.ReLU6(),
        nn.Conv2d(128, 128, 1),
        nn.MaxPool2d(2, 2, 0),
    ),
```

```
            nn.Sequential(
                nn.Conv2d(128, 128, 3, 1, 1, groups=128),
                nn.BatchNorm2d(128),
                nn.ReLU6(),
                nn.Conv2d(128, 256, 1),
                nn.Conv2d(256, 256, 3, 1, 1, groups=256),
                nn.BatchNorm2d(256),
                nn.ReLU6(),
                nn.Conv2d(256, 256, 1),
                nn.Conv2d(256, 256, 3, 1, 1, groups=256),
                nn.BatchNorm2d(256),
                nn.ReLU6(),
                nn.Conv2d(256, 256, 1),
            ),

            nn.AdaptiveAvgPool2d((1, 1)),
        )
        self.fc = nn.Sequential(
            nn.Linear(256, 50),
            nn.Linear(50 , 11),
        )
```

```
results:
[026/030] Val Acc: 0.653332 loss: 0.008219
[027/030] Val Acc: 0.642912 loss: 0.008812
[028/030] Val Acc: 0.611125 loss: 0.009991
[029/030] Val Acc: 0.658279 loss: 0.008249
[030/030] Val Acc: 0.651176 loss: 0.008972
```

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)

P.S.這裡也是跟上一題幾乎一樣，但是就在group的地方隨便改一改

results:

```
self.cnn = nn.Sequential(

            nn.Sequential(
                nn.Conv2d(3, 16, 3, 1, 1),
                nn.BatchNorm2d(16),
```

```
        nn.ReLU6(),
        nn.Conv2d(16, 16, 3, 1, 1),
        nn.BatchNorm2d(16),
        nn.ReLU6(),
        nn.MaxPool2d(2, 2, 0),
    ),
    nn.Sequential(

        nn.Conv2d(16, 32, 3, 1, 1, groups=16//2),

        nn.BatchNorm2d(16),
        nn.ReLU6(),
        nn.Conv2d(32, 32, 3, 1, 1, groups=32//4),

        nn.BatchNorm2d(32),
        nn.ReLU6(),
        nn.MaxPool2d(2, 2, 0),
    ),
    nn.Sequential(
        nn.Conv2d(32, 64, 3, 1, 1, groups=32//2),
        nn.BatchNorm2d(32),
        nn.ReLU6(),
        nn.Conv2d(64, 64, 3, 1, 1, groups=64//4),
        nn.BatchNorm2d(64),
        nn.ReLU6(),
        nn.Conv2d(64, 64, 3, 1, 1, groups=64//8),
        nn.BatchNorm2d(64),
        nn.ReLU6(),
        nn.MaxPool2d(2, 2, 0),
    ),
    nn.Sequential(
        nn.Conv2d(64, 128, 3, 1, 1, groups=64//2),
        nn.BatchNorm2d(64),
        nn.ReLU6(),
        nn.Conv2d(128, 128, 3, 1, 1, groups=128//4),
        nn.BatchNorm2d(128),
        nn.ReLU6(),
        nn.Conv2d(128, 128, 3, 1, 1, groups=128//8),
        nn.BatchNorm2d(128),
        nn.ReLU6(),
```

```
            nn.MaxPool2d(2, 2, 0),
        ),

        nn.Sequential(
            nn.Conv2d(128, 256, 3, 1, 1, groups=128//2),
            nn.BatchNorm2d(128),
            nn.ReLU6(),
            nn.Conv2d(256, 256, 3, 1, 1, groups=256//4),
            nn.BatchNorm2d(256),
            nn.ReLU6(),
            nn.Conv2d(256, 256, 3, 1, 1, groups=256//8),
            nn.BatchNorm2d(256),
            nn.ReLU6(),
        ),
        nn.AdaptiveAvgPool2d((1, 1)),
    )
    self.fc = nn.Sequential(
        nn.Linear(256, 50),
        nn.Linear(50 , 11),
    )
```

結果:

```
[026/030] Val Acc: 0.560927 loss: 0.009123
[027/030] Val Acc: 0.578812 loss: 0.009334
[028/030] Val Acc: 0.602098 loss: 0.009880
[029/030] Val Acc: 0.601934 loss: 0.009597
```