Machine Learning HW7 Report

學號：B06901045 系級：電機三 姓名：曹林熹

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%)**

ANS:

method 1)

可以先看看我採用 Architecture_Design 建構出一個 Student_Net，此部份的結構為 TA 給的 sample code，觀察架構如下圖。

num of parameters: 256,779

```
[ ] model = StudentNet()

    from torchsummary import summary
    summary(model, input_size=(3, 128, 128))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 16, 128, 128]             448
       BatchNorm2d-2         [-1, 16, 128, 128]              32
             ReLU6-3         [-1, 16, 128, 128]               0
         MaxPool2d-4           [-1, 16, 64, 64]               0
            Conv2d-5           [-1, 16, 64, 64]             160
       BatchNorm2d-6           [-1, 16, 64, 64]              32
             ReLU6-7           [-1, 16, 64, 64]               0
            Conv2d-8           [-1, 32, 64, 64]             544
         MaxPool2d-9           [-1, 32, 32, 32]               0
           Conv2d-10           [-1, 32, 32, 32]             320
      BatchNorm2d-11           [-1, 32, 32, 32]              64
            ReLU6-12           [-1, 32, 32, 32]               0
           Conv2d-13           [-1, 64, 32, 32]           2,112
        MaxPool2d-14           [-1, 64, 16, 16]               0
           Conv2d-15           [-1, 64, 16, 16]             640
      BatchNorm2d-16           [-1, 64, 16, 16]             128
            ReLU6-17           [-1, 64, 16, 16]               0
           Conv2d-18          [-1, 128, 16, 16]           8,320
        MaxPool2d-19            [-1, 128, 8, 8]               0
           Conv2d-20            [-1, 128, 8, 8]           1,280
      BatchNorm2d-21            [-1, 128, 8, 8]             256
            ReLU6-22            [-1, 128, 8, 8]               0
           Conv2d-23            [-1, 256, 8, 8]          33,024
           Conv2d-24            [-1, 256, 8, 8]           2,560
      BatchNorm2d-25            [-1, 256, 8, 8]             512
            ReLU6-26            [-1, 256, 8, 8]               0
           Conv2d-27            [-1, 256, 8, 8]          65,792
           Conv2d-28            [-1, 256, 8, 8]           2,560
      BatchNorm2d-29            [-1, 256, 8, 8]             512
            ReLU6-30            [-1, 256, 8, 8]               0
           Conv2d-31            [-1, 256, 8, 8]          65,792
           Conv2d-32            [-1, 256, 8, 8]           2,560
      BatchNorm2d-33            [-1, 256, 8, 8]             512
            ReLU6-34            [-1, 256, 8, 8]               0
           Conv2d-35            [-1, 256, 8, 8]          65,792
AdaptiveAvgPool2d-36            [-1, 256, 1, 1]               0
           Linear-37                   [-1, 11]           2,827
================================================================
Total params: 256,779
Trainable params: 256,779
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.19
Forward/backward pass size (MB): 13.13
Params size (MB): 0.98
Estimated Total Size (MB): 14.29
```

接下來，我把此結構去做 Knowledge_Distillation，使用的 Teacher_Net 為助教提 pretrained_resnet18，提供的架構如下。

```
from torchsummary import summary
summary(teacher_net, input_size=(3, 128, 128))
```

```
          Conv2d-22          [-1, 128, 16, 16]         147,456
     BatchNorm2d-23          [-1, 128, 16, 16]             256
          Conv2d-24          [-1, 128, 16, 16]           8,192
     BatchNorm2d-25          [-1, 128, 16, 16]             256
            ReLU-26          [-1, 128, 16, 16]               0
      BasicBlock-27          [-1, 128, 16, 16]               0
          Conv2d-28          [-1, 128, 16, 16]         147,456
     BatchNorm2d-29          [-1, 128, 16, 16]             256
            ReLU-30          [-1, 128, 16, 16]               0
          Conv2d-31          [-1, 128, 16, 16]         147,456
     BatchNorm2d-32          [-1, 128, 16, 16]             256
            ReLU-33          [-1, 128, 16, 16]               0
      BasicBlock-34          [-1, 128, 16, 16]               0
          Conv2d-35            [-1, 256, 8, 8]         294,912
     BatchNorm2d-36            [-1, 256, 8, 8]             512
            ReLU-37            [-1, 256, 8, 8]               0
          Conv2d-38            [-1, 256, 8, 8]         589,824
     BatchNorm2d-39            [-1, 256, 8, 8]             512
          Conv2d-40            [-1, 256, 8, 8]          32,768
     BatchNorm2d-41            [-1, 256, 8, 8]             512
            ReLU-42            [-1, 256, 8, 8]               0
      BasicBlock-43            [-1, 256, 8, 8]               0
          Conv2d-44            [-1, 256, 8, 8]         589,824
     BatchNorm2d-45            [-1, 256, 8, 8]             512
            ReLU-46            [-1, 256, 8, 8]               0
          Conv2d-47            [-1, 256, 8, 8]         589,824
     BatchNorm2d-48            [-1, 256, 8, 8]             512
            ReLU-49            [-1, 256, 8, 8]               0
      BasicBlock-50            [-1, 256, 8, 8]               0
          Conv2d-51            [-1, 512, 4, 4]       1,179,648
     BatchNorm2d-52            [-1, 512, 4, 4]           1,024
            ReLU-53            [-1, 512, 4, 4]               0
          Conv2d-54            [-1, 512, 4, 4]       2,359,296
     BatchNorm2d-55            [-1, 512, 4, 4]           1,024
          Conv2d-56            [-1, 512, 4, 4]         131,072
     BatchNorm2d-57            [-1, 512, 4, 4]           1,024
            ReLU-58            [-1, 512, 4, 4]               0
      BasicBlock-59            [-1, 512, 4, 4]               0
          Conv2d-60            [-1, 512, 4, 4]       2,359,296
     BatchNorm2d-61            [-1, 512, 4, 4]           1,024
            ReLU-62            [-1, 512, 4, 4]               0
          Conv2d-63            [-1, 512, 4, 4]       2,359,296
     BatchNorm2d-64            [-1, 512, 4, 4]           1,024
            ReLU-65            [-1, 512, 4, 4]               0
      BasicBlock-66            [-1, 512, 4, 4]               0
AdaptiveAvgPool2d-67           [-1, 512, 1, 1]               0
          Linear-68                   [-1, 11]           5,643
================================================================
Total params: 11,182,155
Trainable params: 11,182,155
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.19
Forward/backward pass size (MB): 20.50
Params size (MB): 42.66
Estimated Total Size (MB): 63.35
----------------------------------------------------------------
```

學習過程中，我一開始都使用 AdamW optimizer (lr = 10^(-3))，讓 epoch 跑到 170 次，此時在 val_acc 最高可以到 0.8105 ，不過在後期可以看到我們 train_acc 持續升高，但是 val_acc 卻停滯，我猜測因為在 adam 後期使得學習律非常低，因此必須重新設定 lr。

```
epoch cost time = 39.841712474823
epoch 150: train loss: 3.3255, acc 0.8712 valid loss: 4.7168, acc 0.7965
epoch cost time = 39.91655373573303
epoch 151: train loss: 3.3307, acc 0.8755 valid loss: 4.4318, acc 0.7866
epoch cost time = 39.9276807308197
epoch 152: train loss: 3.2856, acc 0.8757 valid loss: 4.4575, acc 0.8061
epoch cost time = 39.897618532180786
epoch 153: train loss: 3.2715, acc 0.8776 valid loss: 4.3784, acc 0.7921
epoch cost time = 39.87037920951843
epoch 154: train loss: 3.3043, acc 0.8774 valid loss: 4.2038, acc 0.8020
epoch cost time = 39.90630626678467
epoch 155: train loss: 3.2775, acc 0.8767 valid loss: 3.9685, acc 0.8017
epoch cost time = 39.92513561248779
epoch 156: train loss: 3.3288, acc 0.8743 valid loss: 4.1391, acc 0.8073
epoch cost time = 39.98978877067566
epoch 157: train loss: 3.3045, acc 0.8765 valid loss: 4.2554, acc 0.8006
epoch cost time = 40.01657962799072
epoch 158: train loss: 3.2445, acc 0.8804 valid loss: 4.3599, acc 0.8096
epoch cost time = 39.96568059921265
epoch 159: train loss: 3.2732, acc 0.8781 valid loss: 4.3227, acc 0.7997
epoch cost time = 39.877344369888306
epoch 160: train loss: 3.2768, acc 0.8750 valid loss: 4.1303, acc 0.8061
epoch cost time = 39.911781549453735
epoch 161: train loss: 3.2852, acc 0.8727 valid loss: 4.0196, acc 0.8050
epoch cost time = 40.02925157546997
epoch 162: train loss: 3.2441, acc 0.8822 valid loss: 4.1225, acc 0.8006
epoch cost time = 40.16031813621521
epoch 163: train loss: 3.2666, acc 0.8804 valid loss: 4.5577, acc 0.8009
epoch cost time = 40.085801124572754
epoch 164: train loss: 3.2179, acc 0.8832 valid loss: 4.3590, acc 0.7892
epoch cost time = 40.1453378200531
epoch 165: train loss: 3.2263, acc 0.8757 valid loss: 4.5547, acc 0.7933
epoch cost time = 40.13863682746887
epoch 166: train loss: 3.2114, acc 0.8798 valid loss: 4.5375, acc 0.7851
epoch cost time = 40.02513575553894
epoch 167: train loss: 3.1822, acc 0.8799 valid loss: 4.3064, acc 0.7997
epoch cost time = 40.02539896965027
epoch 168: train loss: 3.3152, acc 0.8728 valid loss: 4.1173, acc 0.7901
epoch cost time = 40.012587785720825
epoch 169: train loss: 3.1354, acc 0.8831 valid loss: 4.3338, acc 0.7959
epoch cost time = 40.031681537628174
```

因此我重新設定 lr = 10^(-4) 再 train epoch = 50，可以看到我們的 val_acc 有稍微提升，最後拿出我們的 model 丟到 kaggle（使用 hw3 kaggle 避免污染 Leaderboard），可以得到 0.86312 acc。

```
save model
epoch   0: train loss: 3.2402, acc 0.8750 valid loss: 3.6190, acc 0.8222
epoch cost time = 111.84169054031372

epoch   1: train loss: 3.0752, acc 0.8837 valid loss: 3.5267, acc 0.8204
epoch cost time = 109.01970791816711

save model
epoch   2: train loss: 3.0178, acc 0.8844 valid loss: 3.5369, acc 0.8259
epoch cost time = 109.01639747619629

epoch   3: train loss: 2.9738, acc 0.8845 valid loss: 3.6550, acc 0.8222
epoch cost time = 108.86228537559509

epoch   4: train loss: 3.0097, acc 0.8872 valid loss: 3.5062, acc 0.8233
epoch cost time = 108.84835362434387

epoch   5: train loss: 2.9753, acc 0.8927 valid loss: 3.5532, acc 0.8242
epoch cost time = 109.03948140144348

epoch   6: train loss: 2.9385, acc 0.8930 valid loss: 3.5716, acc 0.8242
epoch cost time = 108.64016675949097

epoch   7: train loss: 2.9882, acc 0.8905 valid loss: 3.5424, acc 0.8213
epoch cost time = 108.73926615715027

epoch   8: train loss: 2.9389, acc 0.8922 valid loss: 3.5306, acc 0.8224
epoch cost time = 109.12442922592163

epoch   9: train loss: 2.9130, acc 0.8967 valid loss: 3.5986, acc 0.8190
epoch cost time = 109.22598385810852

epoch  10: train loss: 2.9625, acc 0.8927 valid loss: 3.5028, acc 0.8227
epoch cost time = 109.62371230125427

epoch  11: train loss: 2.9128, acc 0.8976 valid loss: 3.4929, acc 0.8236
epoch cost time = 109.88148093223572

epoch  12: train loss: 2.9215, acc 0.8929 valid loss: 3.5143, acc 0.8254
epoch cost time = 109.93494391441345
```

method 2）

方法二我採用 Weight_Quantization ，此部份的初始結構為我們剛剛 train 好的 student_net，因此原本參數數量與上述作法的 student_net 一樣，主要是要來觀察若使用了 Weight_Quantization 的話，那準確率將會有多少變化。

首先，我們導入我們的模型，可以看到 original cost: 1047706 bytes，接著經由32 bit => 8 bit 轉換，可以看到 8-bit cost: 268471 bytes，變成原本的 1/4 倍。那我們就是要來觀察此壓縮後的模型最後預測的 prediction 丟入 kaggle 會得到多少的準確率，結果發現確實模型預測的結果比 32 bit 不準確，是因為用較少的存儲空間存模型，自然有這樣的結果，但是仍然丟到 kaggle hw3 可以得到 0.84817 acc。

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%)**

ANS：

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

y. Student net architecture and # of parameters: 256,779

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

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

c. Your student net from scratch: 64.19%

d. Your student net KD from (a.): 79.18%

e. Your student net KD from (b.): 82.59%

我使用的 student 為助教的 sample code student_net，epoch times 由自行觀察是否模型已經飽和，則不再 train。

c)　可以看到我的 model 的 val_acc 並不高，在參數量小的時候，儘管 train_acc 快到了飽和，但是 val_acc 頂多也才只有 0.64 左右，畢竟在我 hw3 的 CNN model 中，不使用 resnet 但是參數量 11,267,083 時，val_acc 也才 0.79~0.82，可見 from scratch 的 student_net 並不是個很理想的選擇。

```
[160/200] 15.74 sec(s) Train Acc: 0.966146 Loss: 0.000905 | Val Acc: 0.632362 loss: 0.018143
[161/200] 15.71 sec(s) Train Acc: 0.979931 Loss: 0.000476 | Val Acc: 0.627988 loss: 0.017978
[162/200] 15.70 sec(s) Train Acc: 0.985100 Loss: 0.000406 | Val Acc: 0.624198 loss: 0.019189
[163/200] 15.69 sec(s) Train Acc: 0.966450 Loss: 0.000746 | Val Acc: 0.629446 loss: 0.018166
[164/200] 15.71 sec(s) Train Acc: 0.980438 Loss: 0.000499 | Val Acc: 0.619534 loss: 0.019747
[165/200] 15.75 sec(s) Train Acc: 0.966957 Loss: 0.000793 | Val Acc: 0.641108 loss: 0.017929
[166/200] 15.72 sec(s) Train Acc: 0.983783 Loss: 0.000404 | Val Acc: 0.642566 loss: 0.018159
[167/200] 15.76 sec(s) Train Acc: 0.976992 Loss: 0.000579 | Val Acc: 0.641691 loss: 0.017466
[168/200] 15.70 sec(s) Train Acc: 0.987330 Loss: 0.000410 | Val Acc: 0.626531 loss: 0.019244
[169/200] 15.75 sec(s) Train Acc: 0.968174 Loss: 0.000873 | Val Acc: 0.615160 loss: 0.020194
[170/200] 15.70 sec(s) Train Acc: 0.951348 Loss: 0.001458 | Val Acc: 0.636443 loss: 0.017629
```

d)　可以看到我的 model 的 val_acc 與上面的 student_net from scratch

相比有比較高，但 train 了約 90 次（有重新設 lr)，我們可以看到 train_acc 與 val_acc 皆沒有繼續往上升，我猜測是進入了 local minimum（左圖）。我之後又再重新設定 lr，可以看到 acc 有提升一點（到 0.7910)，使用此模型不像最後的 hw7_best model 一樣好，畢竟 teacher_net 缺乏 pretrain，不過得到此準確率已經跟我們自己在 hw3 CNN 差不多了（右圖）。此模型的缺點就是，容易卡在 local point，因此要手動調 lr...。

```
epoch  40: train loss: 2.8242, acc 0.7569 valid loss: 2.8776, acc 0.7464
epoch cost time = 39.52186608314514

epoch  41: train loss: 2.8547, acc 0.7576 valid loss: 2.8385, acc 0.7458
epoch cost time = 39.52450227737427

epoch  42: train loss: 2.8660, acc 0.7516 valid loss: 2.9153, acc 0.7414
epoch cost time = 39.618900299072266

epoch  43: train loss: 2.8719, acc 0.7509 valid loss: 2.8564, acc 0.7440
epoch cost time = 39.712618350982666

epoch  44: train loss: 2.8680, acc 0.7562 valid loss: 2.8667, acc 0.7464
epoch cost time = 39.693859338760376

epoch  45: train loss: 2.8666, acc 0.7463 valid loss: 2.8854, acc 0.7461
epoch cost time = 39.589476346969604

epoch  46: train loss: 2.8681, acc 0.7511 valid loss: 2.8677, acc 0.7455
epoch cost time = 39.70133852958679

epoch  47: train loss: 2.8644, acc 0.7536 valid loss: 2.8759, acc 0.7501
epoch cost time = 39.55176758766174

epoch  48: train loss: 2.8407, acc 0.7556 valid loss: 2.8603, acc 0.7513
epoch cost time = 39.63163924217224

epoch  49: train loss: 2.8601, acc 0.7580 valid loss: 2.9039, acc 0.7466
epoch cost time = 39.57999777793884

epoch  50: train loss: 2.8627, acc 0.7545 valid loss: 2.8277, acc 0.7548
epoch cost time = 39.573840379714966
```

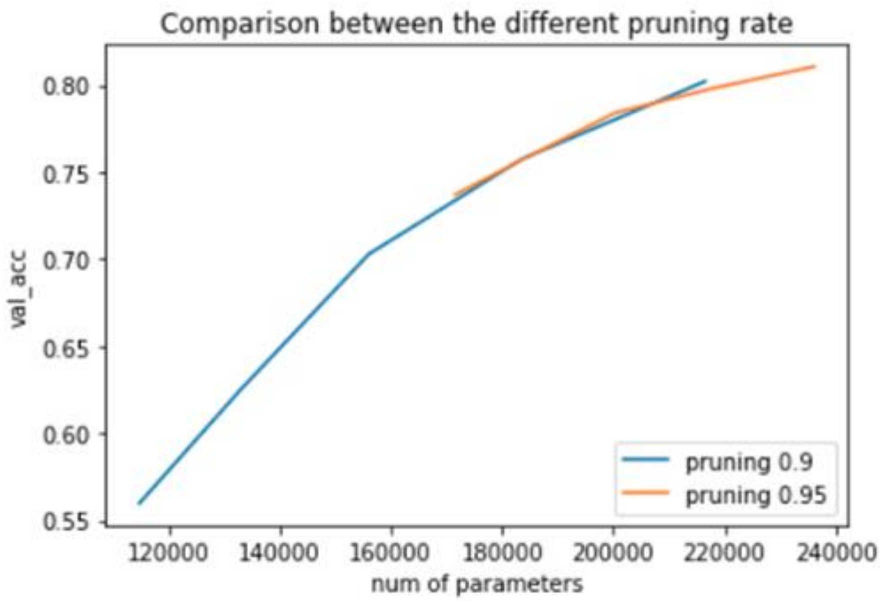
```
epoch  70: train loss: 1.9297, acc 0.8447 valid loss: 2.5633, acc 0.7732
epoch cost time = 40.3752543926239

epoch  71: train loss: 1.9026, acc 0.8462 valid loss: 2.7338, acc 0.7627
epoch cost time = 40.01913046836853

save model
epoch  72: train loss: 1.9452, acc 0.8445 valid loss: 2.2063, acc 0.7918
epoch cost time = 40.12100696563721

epoch  73: train loss: 1.9065, acc 0.8487 valid loss: 2.4376, acc 0.7691
epoch cost time = 40.06245708465576

epoch  74: train loss: 1.8911, acc 0.8492 valid loss: 2.5755, acc 0.7647
epoch cost time = 40.12340950965881

epoch  75: train loss: 1.9011, acc 0.8491 valid loss: 2.3654, acc 0.7589
epoch cost time = 40.0911545753479

epoch  76: train loss: 1.8793, acc 0.8477 valid loss: 2.3109, acc 0.7706
epoch cost time = 40.452666997909546

epoch  77: train loss: 1.8789, acc 0.8512 valid loss: 2.0856, acc 0.7778
epoch cost time = 40.72527623176575

epoch  78: train loss: 1.8804, acc 0.8526 valid loss: 2.6094, acc 0.7577
epoch cost time = 40.35807204246521

epoch  79: train loss: 1.8770, acc 0.8533 valid loss: 2.3743, acc 0.7781
epoch cost time = 40.23571753501892

epoch  80: train loss: 1.8725, acc 0.8518 valid loss: 2.3281, acc 0.7910
epoch cost time = 40.22321653366089
```

e）　此部份在第一小題有詳述過了，這部份的 acc 最高是因為他的 teacher 使用裡面最好的 resnet18_pretrain，因此學習的比較好。

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%)**

ANS:

我使用的 model 為在第一題 train 好的 student_net，分別使用了 pruning rate = 0.9 vs 0.95，我總共各 prune 五次，所以剩下的參數量約為 0.9^5*256,779 = 198690 與 0.95^5*256,779 = 151625，可以看到我們經過比較低的 pruning rate 產生出的 val_acc 較低，這個結果很正常因為參數量較少的關係。



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%）
   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)

ANS:

暫時無。