HW7 Report

學號：b07901039　系級：電機二　　　姓名：劉知穎

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：使用助教 trained 的 torchvisions 的 resnet18。

   size: 43 MB。(44788712 bytes)、參數量：11.1 M。

   Valididation Loss: 0.55742 、Validation Acc: 0.885

   resnet18 模型架構：

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 64, 64]	9,408
BatchNorm2d-2	[-1, 64, 64, 64]	128
ReLU-3	[-1, 64, 64, 64]	0
MaxPool2d-4	[-1, 64, 32, 32]	0
Conv2d-5	[-1, 64, 32, 32]	36,864
BatchNorm2d-6	[-1, 64, 32, 32]	128
ReLU-7	[-1, 64, 32, 32]	0
Conv2d-8	[-1, 64, 32, 32]	36,864
BatchNorm2d-9	[-1, 64, 32, 32]	128
ReLU-10	[-1, 64, 32, 32]	0
BasicBlock-11	[-1, 64, 32, 32]	0
Conv2d-12	[-1, 64, 32, 32]	36,864
BatchNorm2d-13	[-1, 64, 32, 32]	128
ReLU-14	[-1, 64, 32, 32]	0
Conv2d-15	[-1, 64, 32, 32]	36,864
BatchNorm2d-16	[-1, 64, 32, 32]	128
ReLU-17	[-1, 64, 32, 32]	0
BasicBlock-18	[-1, 64, 32, 32]	0
Conv2d-19	[-1, 128, 16, 16]	73,728
BatchNorm2d-20	[-1, 128, 16, 16]	256
ReLU-21	[-1, 128, 16, 16]	0
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

(1) Low Rank Approximation

將 resnet18 架構中除第一層 convolution layer 外的 convolution layer，拆解成 depthwise 和 pointwise 的 convolution layer。

size: 5.7 MB。(5922784 bytes)、參數量：1.4 M

training 時將 data 隨機作 RandomCrop、RandomHorizontalFlip、RandomRotation、ColorJitter、RandomPerspective 這些 transform。

使用 AdamW optimizer、learning rate = 1e-3、batch size = 128、跑 250 個 epoch。

smallResnet 模型架構：

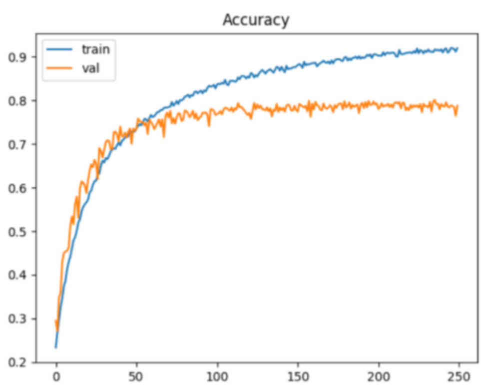
Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 64, 64]	9,408
BatchNorm2d-2	[-1, 64, 64, 64]	128
ReLU-3	[-1, 64, 64, 64]	0
MaxPool2d-4	[-1, 64, 32, 32]	0
Conv2d-5	[-1, 64, 32, 32]	576
BatchNorm2d-6	[-1, 64, 32, 32]	128
ReLU6-7	[-1, 64, 32, 32]	0
Conv2d-8	[-1, 64, 32, 32]	4,096
BatchNorm2d-9	[-1, 64, 32, 32]	128
ReLU6-10	[-1, 64, 32, 32]	0
Conv2d-11	[-1, 64, 32, 32]	576
BatchNorm2d-12	[-1, 64, 32, 32]	128
[illegible]	[-1, 64, 32, 32]	0
[illegible]	[-1, 64, 32, 32]	4,096
[illegible]	[-1, 64, 32, 32]	128
ReLU6-16	[-1, 64, 32, 32]	0
ReLU-17	[-1, 64, 32, 32]	0
BasicBlock-18	[-1, 64, 32, 32]	0
Conv2d-19	[-1, 64, 32, 32]	576
BatchNorm2d-20	[-1, 64, 32, 32]	128
ReLU6-21	[-1, 64, 32, 32]	0
Conv2d-22	[-1, 64, 32, 32]	4,096
BatchNorm2d-23	[-1, 64, 32, 32]	128
ReLU6-24	[-1, 64, 32, 32]	0
Conv2d-25	[-1, 64, 32, 32]	576
BatchNorm2d-26	[-1, 64, 32, 32]	128
ReLU6-27	[-1, 64, 32, 32]	0
Conv2d-28	[-1, 64, 32, 32]	4,096
BatchNorm2d-29	[-1, 64, 32, 32]	128
ReLU6-30	[-1, 64, 32, 32]	0
ReLU-31	[-1, 64, 32, 32]	0
BasicBlock-32	[-1, 64, 32, 32]	0
Conv2d-33	[-1, 64, 16, 16]	576
BatchNorm2d-34	[-1, 64, 16, 16]	128
ReLU6-35	[-1, 64, 16, 16]	0
Conv2d-36	[-1, 128, 16, 16]	8,192
BatchNorm2d-37	[-1, 128, 16, 16]	256
ReLU6-38	[-1, 128, 16, 16]	0
Conv2d-39	[-1, 128, 16, 16]	1,152
BatchNorm2d-40	[-1, 128, 16, 16]	256
ReLU6-41	[-1, 128, 16, 16]	0
Conv2d-42	[-1, 128, 16, 16]	16,384
BatchNorm2d-43	[-1, 128, 16, 16]	256
ReLU6-44	[-1, 128, 16, 16]	0
Conv2d-45	[-1, 128, 16, 16]	8,192
BatchNorm2d-46	[-1, 128, 16, 16]	256
ReLU-47	[-1, 128, 16, 16]	0
BasicBlock-48	[-1, 128, 16, 16]	0
Conv2d-49	[-1, 128, 16, 16]	1,152
BatchNorm2d-50	[-1, 128, 16, 16]	256
ReLU6-51	[-1, 128, 16, 16]	0
Conv2d-52	[-1, 128, 16, 16]	16,384
BatchNorm2d-53	[-1, 128, 16, 16]	256
ReLU6-54	[-1, 128, 16, 16]	0
Conv2d-55	[-1, 128, 16, 16]	1,152
BatchNorm2d-56	[-1, 128, 16, 16]	256
ReLU6-57	[-1, 128, 16, 16]	0
Conv2d-58	[-1, 128, 16, 16]	16,384
BatchNorm2d-59	[-1, 128, 16, 16]	256
ReLU6-60	[-1, 128, 16, 16]	0
ReLU-61	[-1, 128, 16, 16]	0
BasicBlock-62	[-1, 128, 16, 16]	0
Conv2d-63	[-1, 128, 8, 8]	1,152
BatchNorm2d-64	[-1, 128, 8, 8]	256
ReLU6-65	[-1, 128, 8, 8]	0
Conv2d-66	[-1, 256, 8, 8]	32,768
BatchNorm2d-67	[-1, 256, 8, 8]	512
ReLU6-68	[-1, 256, 8, 8]	0
Conv2d-69	[-1, 256, 8, 8]	2,304
BatchNorm2d-70	[-1, 256, 8, 8]	512
ReLU6-71	[-1, 256, 8, 8]	0
Conv2d-72	[-1, 256, 8, 8]	65,536
BatchNorm2d-73	[-1, 256, 8, 8]	512
ReLU6-74	[-1, 256, 8, 8]	0
Conv2d-75	[-1, 256, 8, 8]	32,768
BatchNorm2d-76	[-1, 256, 8, 8]	512
ReLU-77	[-1, 256, 8, 8]	0
BasicBlock-78	[-1, 256, 8, 8]	0
Conv2d-79	[-1, 256, 8, 8]	2,304
BatchNorm2d-80	[-1, 256, 8, 8]	512
ReLU6-81	[-1, 256, 8, 8]	0
Conv2d-82	[-1, 256, 8, 8]	65,536
BatchNorm2d-83	[-1, 256, 8, 8]	512
ReLU6-84	[-1, 256, 8, 8]	0
Conv2d-85	[-1, 256, 8, 8]	2,304
BatchNorm2d-86	[-1, 256, 8, 8]	512
ReLU6-87	[-1, 256, 8, 8]	0
Conv2d-88	[-1, 256, 8, 8]	65,536
BatchNorm2d-89	[-1, 256, 8, 8]	512
ReLU6-90	[-1, 256, 8, 8]	0
ReLU-91	[-1, 256, 8, 8]	0
BasicBlock-92	[-1, 256, 8, 8]	0
Conv2d-93	[-1, 256, 4, 4]	2,304
BatchNorm2d-94	[-1, 256, 4, 4]	512
ReLU6-95	[-1, 256, 4, 4]	0
Conv2d-96	[-1, 512, 4, 4]	131,072
BatchNorm2d-97	[-1, 512, 4, 4]	1,024
ReLU6-98	[-1, 512, 4, 4]	0
Conv2d-99	[-1, 512, 4, 4]	4,608
BatchNorm2d-100	[-1, 512, 4, 4]	1,024
ReLU6-101	[-1, 512, 4, 4]	0
Conv2d-102	[-1, 512, 4, 4]	262,144
BatchNorm2d-103	[-1, 512, 4, 4]	1,024
ReLU6-104	[-1, 512, 4, 4]	0
Conv2d-105	[-1, 512, 4, 4]	131,072
BatchNorm2d-106	[-1, 512, 4, 4]	1,024
ReLU-107	[-1, 512, 4, 4]	0
BasicBlock-108	[-1, 512, 4, 4]	0
Conv2d-109	[-1, 512, 4, 4]	4,608
BatchNorm2d-110	[-1, 512, 4, 4]	1,024
ReLU6-111	[-1, 512, 4, 4]	0
Conv2d-112	[-1, 512, 4, 4]	262,144
BatchNorm2d-113	[-1, 512, 4, 4]	1,024
ReLU6-114	[-1, 512, 4, 4]	0
Conv2d-115	[-1, 512, 4, 4]	4,608
BatchNorm2d-116	[-1, 512, 4, 4]	1,024
ReLU6-117	[-1, 512, 4, 4]	0
Conv2d-118	[-1, 512, 4, 4]	262,144
BatchNorm2d-119	[-1, 512, 4, 4]	1,024
ReLU6-120	[-1, 512, 4, 4]	0
ReLU-121	[-1, 512, 4, 4]	0
BasicBlock-122	[-1, 512, 4, 4]	0
AdaptiveAvgPool2d-123	[-1, 512, 1, 1]	0
Linear-124	[-1, 11]	5,643

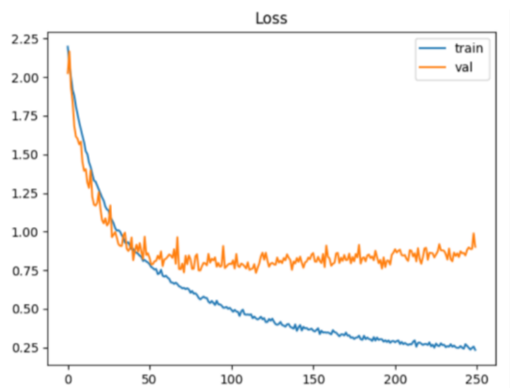
Total params: 1,454,603
Trainable params: 1,454,603
Non-trainable params: 0

對應

resnet18

Conv2d-5	[-1, 64, 32, 32]	36,864
BatchNorm2d-6	[-1, 64, 32, 32]	128
ReLU-7	[-1, 64, 32, 32]	0





Best Validation Loss: 0.807 、Validation Acc: 0.801

(2) Knowledge Distillation

以助教 trained 好的 resnet18 當 teacher net，用一般 CNN 架構(沒有用 depthwise 和 pointwise convolution)的小 model 當 student net。

size: 5.0 MB (5223289 bytes) 、參數量：1.7 M

training 時將 data 隨機作 RandomCrop、RandomHorizontalFlip、RandomRotation、ColorJitter、RandomPerspective 這些 transform。

使用 AdamW optimizer、learning rate = 1e-3、batch size = 128、跑 250 個 epoch。

loss function 定義為(alpha = 0.5, T= 20)

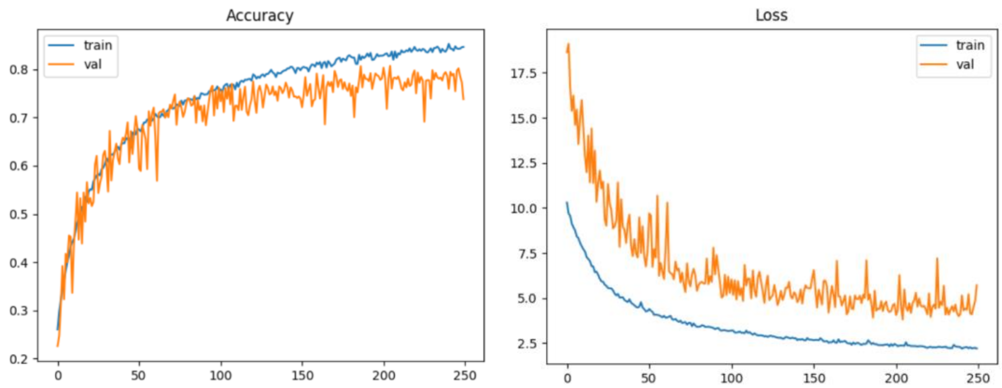
$$Loss = \alpha T^2 \times KL\left(\frac{\text{Teacher's Logits}}{T} \middle|\middle| \frac{\text{Student's Logits}}{T}\right) + (1-\alpha)(\text{原本的Loss})$$

studentnet 架構：

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 32, 64, 64]	864
BatchNorm2d-2	[-1, 32, 64, 64]	64
ReLU6-3	[-1, 32, 64, 64]	0
MaxPool2d-4	[-1, 32, 32, 32]	0
Conv2d-5	[-1, 64, 32, 32]	18,432
BatchNorm2d-6	[-1, 64, 32, 32]	128
ReLU6-7	[-1, 64, 32, 32]	0
MaxPool2d-8	[-1, 64, 16, 16]	0
Conv2d-9	[-1, 128, 8, 8]	73,728
BatchNorm2d-10	[-1, 128, 8, 8]	256
ReLU6-11	[-1, 128, 8, 8]	0
MaxPool2d-12	[-1, 128, 4, 4]	0
Conv2d-13	[-1, 128, 4, 4]	147,456
BatchNorm2d-14	[-1, 128, 4, 4]	256
ReLU6-15	[-1, 128, 4, 4]	0
Conv2d-16	[-1, 256, 2, 2]	294,912
BatchNorm2d-17	[-1, 256, 2, 2]	512
ReLU6-18	[-1, 256, 2, 2]	0
Conv2d-19	[-1, 256, 2, 2]	589,824
BatchNorm2d-20	[-1, 256, 2, 2]	512
ReLU6-21	[-1, 256, 2, 2]	0
Conv2d-22	[-1, 256, 2, 2]	589,824
BatchNorm2d-23	[-1, 256, 2, 2]	512
ReLU6-24	[-1, 256, 2, 2]	0
AdaptiveAvgPool2d-25	[-1, 256, 1, 1]	0
Linear-26	[-1, 11]	2,827

Total params: 1,720,107
Trainable params: 1,720,107
Non-trainable params: 0

Conv2d(3, 32, 2) / Conv2d(32, 64, 1) / Conv2d(64, 128, 2) / Conv2d(128, 128, 1) / Conv2d(128, 256, 2) / Conv2d(256, 256, 1) / Conv2d(256, 256, 1)



每個 Conv2d 後面都接 BatchNorm2d 和 ReLU6、前三個 ReLU6 之後接 MaxPool2d，最後接一層 Linear(256, 1) Best validation loss: 4.0890 、acc: 0.8050

	state_dict size	# of parameters	validation accuracy
resnet18	43 MB	11.1M	0.885
DW+PW conv	5.7 MB	1.4 M	0.801
Knowledge Distillation	5 MB	1.7 M	0.805

用上述兩個方法壓縮模型後，模型大小約為原本的八分之一，參數量約為原本的十分之一，validation accuracy 從 0.885 降至 0.801 和 0.805。

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. size: 43738 KB。
y. Student net architecture and # of parameters: 274763。size: 1107 KB。

student net 架構：

```
StudentNet(
  (cnn): Sequential(
    (0): Sequential(
      (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU6(inplace=True)
    )
    (1): Sequential(
      (0): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=32, bias=False)
      (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU6(inplace=True)
      (3): Conv2d(32, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (5): ReLU6(inplace=True)
    )
    (2): Sequential(
      (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), groups=64, bias=False)
      (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU6(inplace=True)
      (3): Conv2d(64, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (4): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (5): ReLU6(inplace=True)
    )
    (3): Sequential(
      (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=128, bias=False)
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU6(inplace=True)
      (3): Conv2d(128, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (4): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (5): ReLU6(inplace=True)
    )
    (4): Sequential(
      (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), groups=128, bias=False)
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU6(inplace=True)
      (3): Conv2d(128, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (5): ReLU6(inplace=True)
    )
    (5): Sequential(
      (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=256, bias=False)
      (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU6(inplace=True)
      (3): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (5): ReLU6(inplace=True)
    )
    (6): Sequential(
      (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=256, bias=False)
      (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU6(inplace=True)
      (3): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (5): ReLU6(inplace=True)
    )
    (7): Sequential(
      (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=256, bias=False)
      (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU6(inplace=True)
      (3): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (5): ReLU6(inplace=True)
    )
    (8): AdaptiveAvgPool2d(output_size=(1, 1))
  )
  (fc): Sequential(
    (0): Linear(in_features=256, out_features=11, bias=True)
  )
)
```

除了第一層 convonlution layer 之外，其他層都使用 deepwise+pointwise convolution。第一層 conv2d 之後接 batchnorm2d 和 ReLU6。其他層 layer 的架構是 deepwise conv2d / batchnorm2d / ReLU6 / pointwise conv2d / batchnorm2d / ReLU6。輸入輸出 channel 維度分別是 (3, 32) / (32, 64) / (64, 128) / (128, 128) / (128, 256) / (256, 256) / (256, 256) / (256, 256)。最後接 AdaptiveAvgPool 和一層 linear(256, 11)。
訓練方式：
(d)(e)小題以助教提供的 model 當 teacher net，loss 定義為 (alpha = 0.5, T = 20)

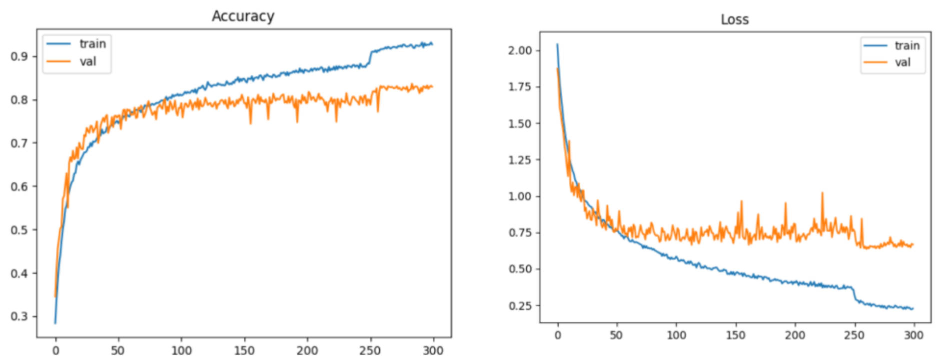
$$Loss = \alpha T^2 \times KL\left(\frac{\text{Teacher's Logits}}{T} \,||\, \frac{\text{Student's Logits}}{T}\right) + (1-\alpha)(\text{原本的Loss})$$

(c)小題只用 student net 訓練，loss 定義為 CrossEntropyLoss。
training 時將 data 隨機作 RandomCrop、RandomHorizontalFlip、RandomRotation、ColorJitter、RandomPerspective 這些 transform。
使用 AdamW optimizer 、learning rate = 1e-3、batch size = 32、跑 250 個 epoch。最後再用 SGD optimizer、learning rate = 1e-3、batch size = 32、跑 50 個 epoch。

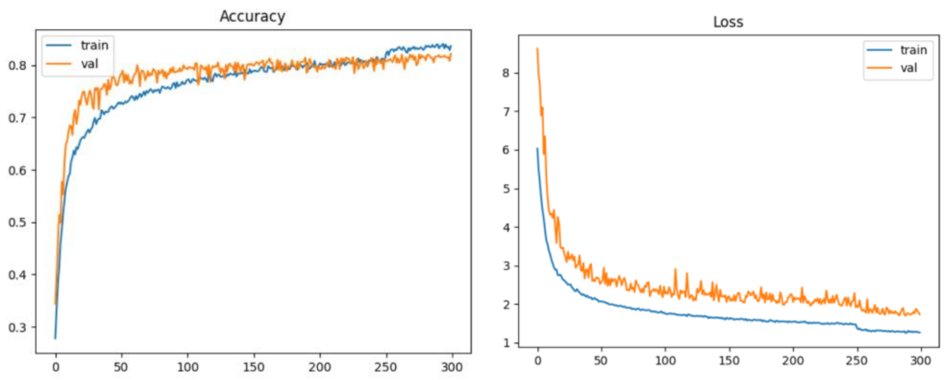
a. Teacher net (ResNet18) from scratch: 80.09%

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

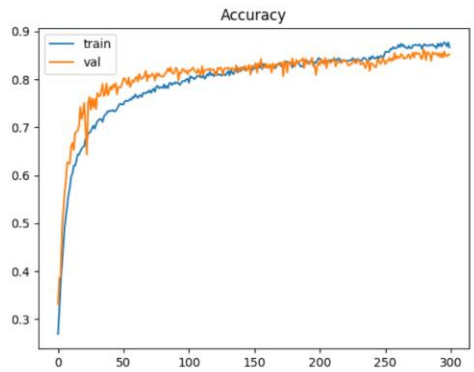
c. Your student net from scratch: 83.10%

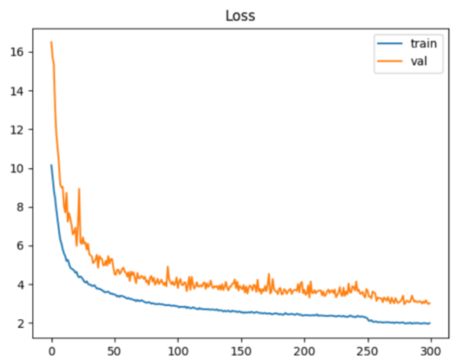


d. Your student net KD from (a.): 82.22 %



e. Your student net KD from (b.): 85.86 %





	validation accuracy
student net	83.10%
KD from resnet18 from scratch (80.09%)	82.22%
KD from resnet18 imagenet & fine-tuned(88.41%)	85.56%

以 resnet18 from scratch (80.09%)當 teacher net 使用 knowledge distillation 得出的結果較單純以 student net 訓練的結果稍差，而使用 resnet18 imagenet & fine-tuned(88.41%)當 teacher net 得出的結果較單純用 student net 訓練好。推測原因為 resnet18 from scratch 本身的 accuracy 太低，造成 student net 學習時，學到不正確的訊息。反之，resnet18 imagenet& fine-tuned 的 accuracy 夠高，student net 學習時不僅能學到較大比例正確的 label，也能學到額外的訊息，例如：某兩類圖形長的相像的訊息。

註：在 epoch = 250 時，loss 和 accuracy 曲線有明顯的變化是因為我在 epoch = 250 時將 optimizer 改成 SGD，以收斂到 loss 更小的點。

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%)

使用 knowledge distillation，以助教提供的 resnet18 當作 teacher net(Validation accuracy: 0.884) 訓練小模型 student net。student net 使用 depthwise 和 pointwise convolution。

training 時將 data 隨機作 RandomCrop、RandomHorizontalFlip、RandomRotation、ColorJitter、RandomPerspective 這些 transform。使用 AdamW optimizer、learning rate = 1e-3、batch size = 128、跑 250 個 epoch。

size：1022 KB。參數量：256K 。

Validation loss: 0.94757、 Validation Acc: 0.814。

student net 模型架構：

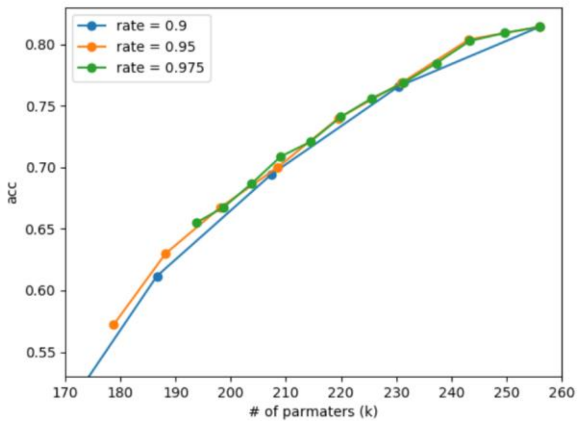
```
StudentNet(
  (cnn): Sequential(
    (0): Sequential(
      (0): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU6()
      (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
    (1): Sequential(
      (0): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=16)
      (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU6()
      (3): Conv2d(16, 32, kernel_size=(1, 1), stride=(1, 1))
      (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
    (2): Sequential(
      (0): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=32)
      (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU6()
      (3): Conv2d(32, 64, kernel_size=(1, 1), stride=(1, 1))
      (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
    (3): Sequential(
      (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=64)
      (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU6()
      (3): Conv2d(64, 128, kernel_size=(1, 1), stride=(1, 1))
      (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
```

```
    )
    (4): Sequential(
      (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=128)
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU6()
      (3): Conv2d(128, 256, kernel_size=(1, 1), stride=(1, 1))
    )
    (5): Sequential(
      (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=256)
      (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU6()
      (3): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
    )
    (6): Sequential(
      (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=256)
      (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU6()
      (3): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
    )
    (7): Sequential(
      (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=256)
      (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU6()
      (3): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
    )
    (8): AdaptiveAvgPool2d(output_size=(1, 1))
  )
  (fc): Sequential(
    (0): Linear(in_features=256, out_features=11, bias=True)
  )
)
```

network pruning 實作方法：

用 batchnorm layer 的 γ 因子來決定 neuron 的重要性。從第三層以後的 layer 做 network pruning，每次 prune 掉一定比例的 neurons。prune 完之後再 train 5 個 epoch，存下 validation accuracy 較高的模型。

使用 rate=0.975, 0.95, 0.9 做 network pruning。



結論：

一次 prune 較少 neuron(較大的 rate)，在參數量相同的情況下，準確率較好。