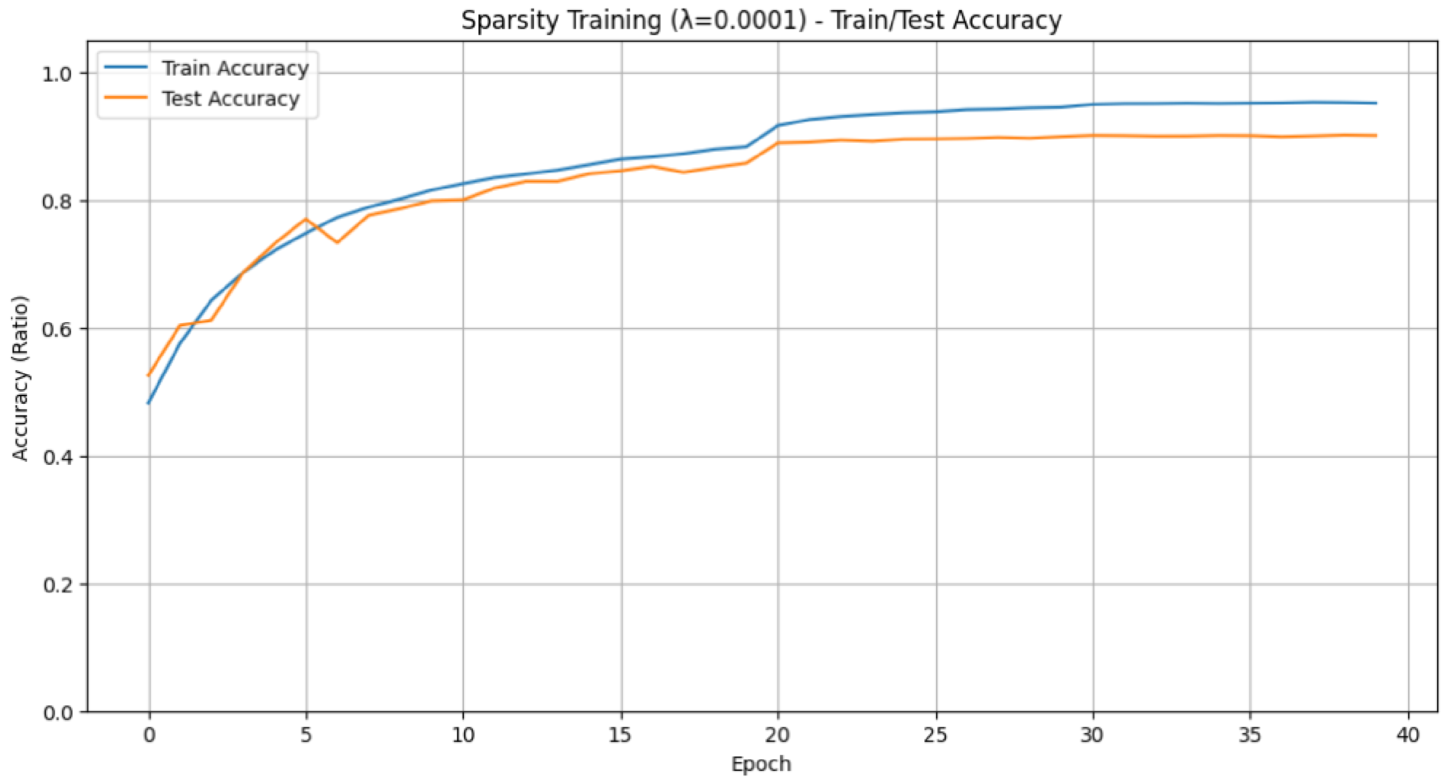
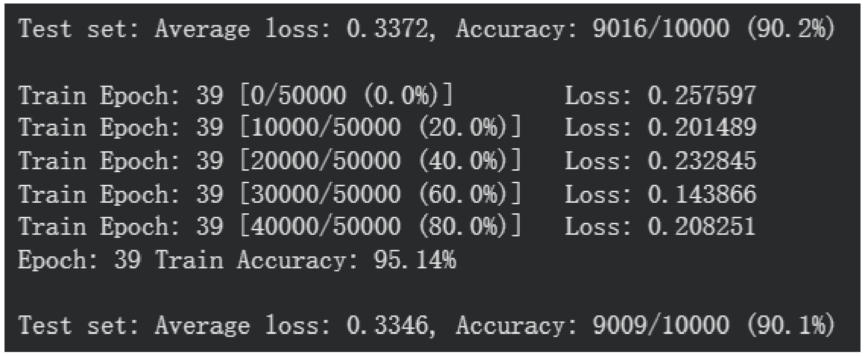
LAB2-Pruning Model

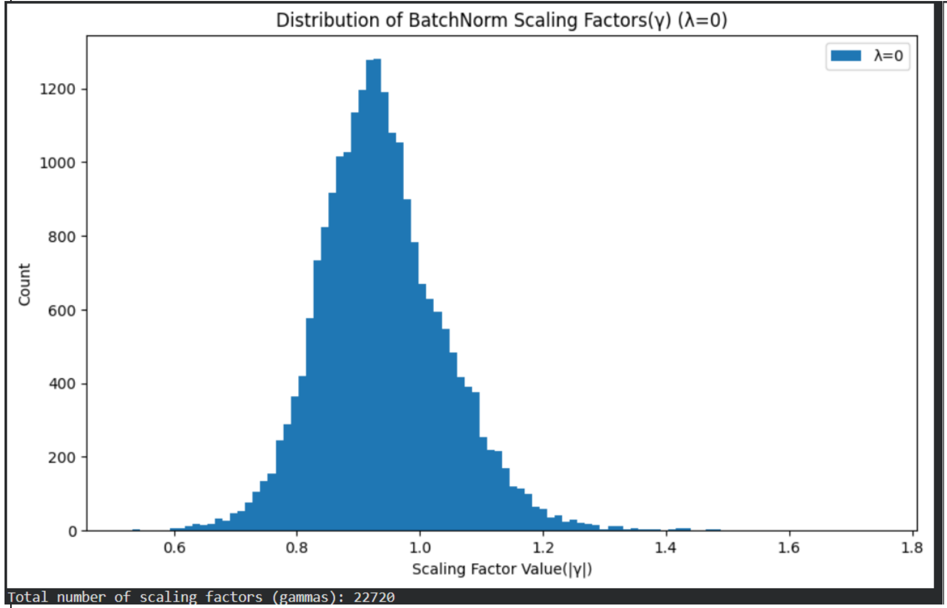
- **Plot** train accuracy and test accuracy of the original model in sparsity-training over epochs (Only λ = 1e-4 are needed.)





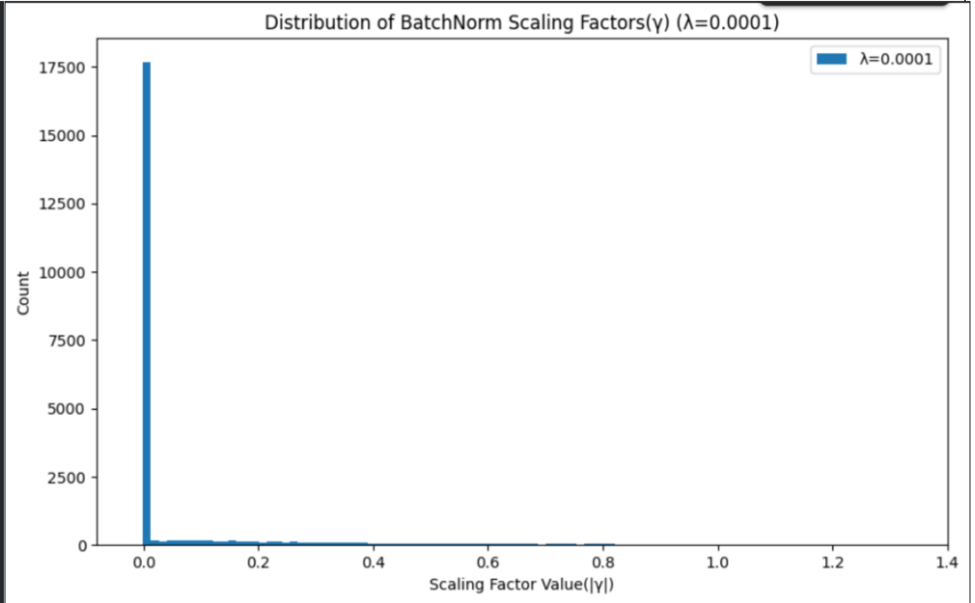
```
Test set: Average loss: 0.3372, Accuracy: 9016/10000 (90.2%)

Train Epoch: 39 [0/50000 (0.0%)]        Loss: 0.257597
Train Epoch: 39 [10000/50000 (20.0%)]   Loss: 0.201489
Train Epoch: 39 [20000/50000 (40.0%)]   Loss: 0.232845
Train Epoch: 39 [30000/50000 (60.0%)]   Loss: 0.143866
Train Epoch: 39 [40000/50000 (80.0%)]   Loss: 0.208251
Epoch: 39 Train Accuracy: 95.14%

Test set: Average loss: 0.3346, Accuracy: 9009/10000 (90.1%)
```

- **Plot** scaling factor distribution with 3 different λ value

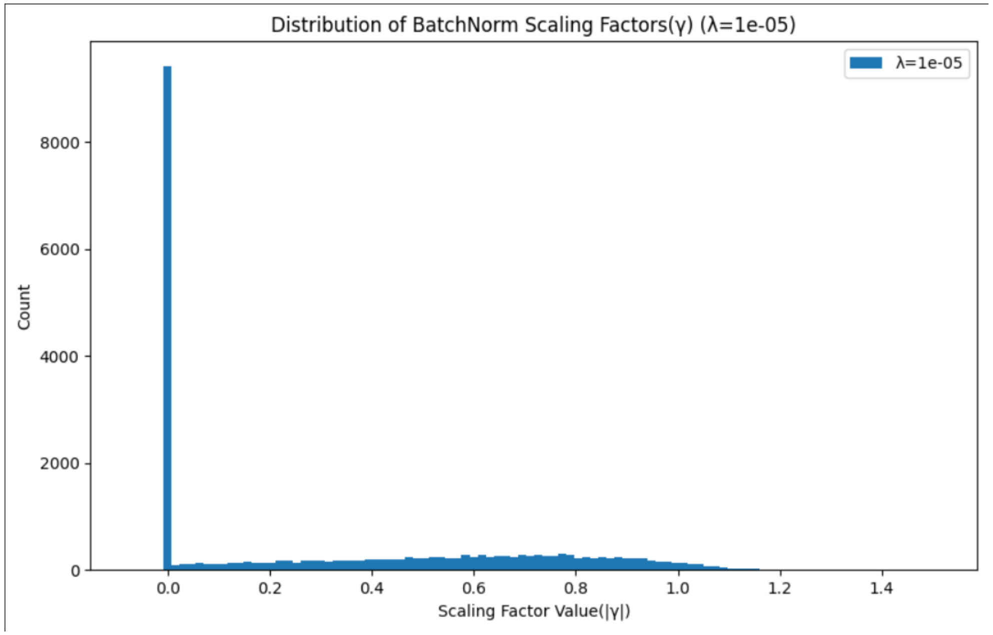


```
Total number of scaling factors (gammas): 22720
```

↑ Lambda=0



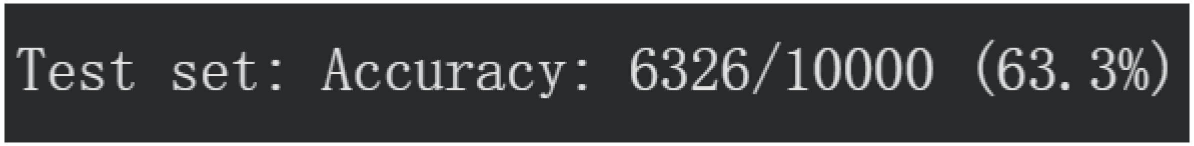
↑ Lambda=1e-4



↙ Lambda=1e-5

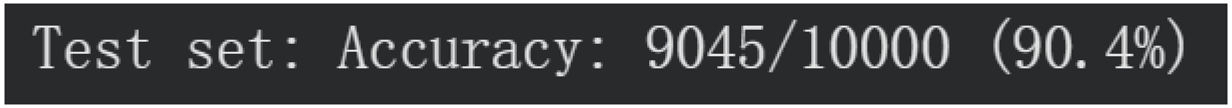
- **Show** the model test accuracy after pruning 50% and 90% channels (5%)

After pruning 90% channel Test Accuracy



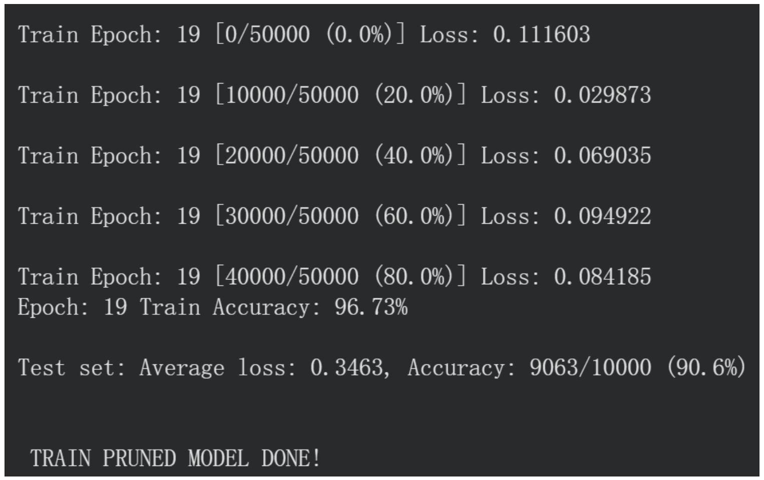
```
Test set: Accuracy: 6326/10000 (63.3%)
```

After pruning 50% channel Test Accuracy



```
Test set: Accuracy: 9045/10000 (90.4%)
```

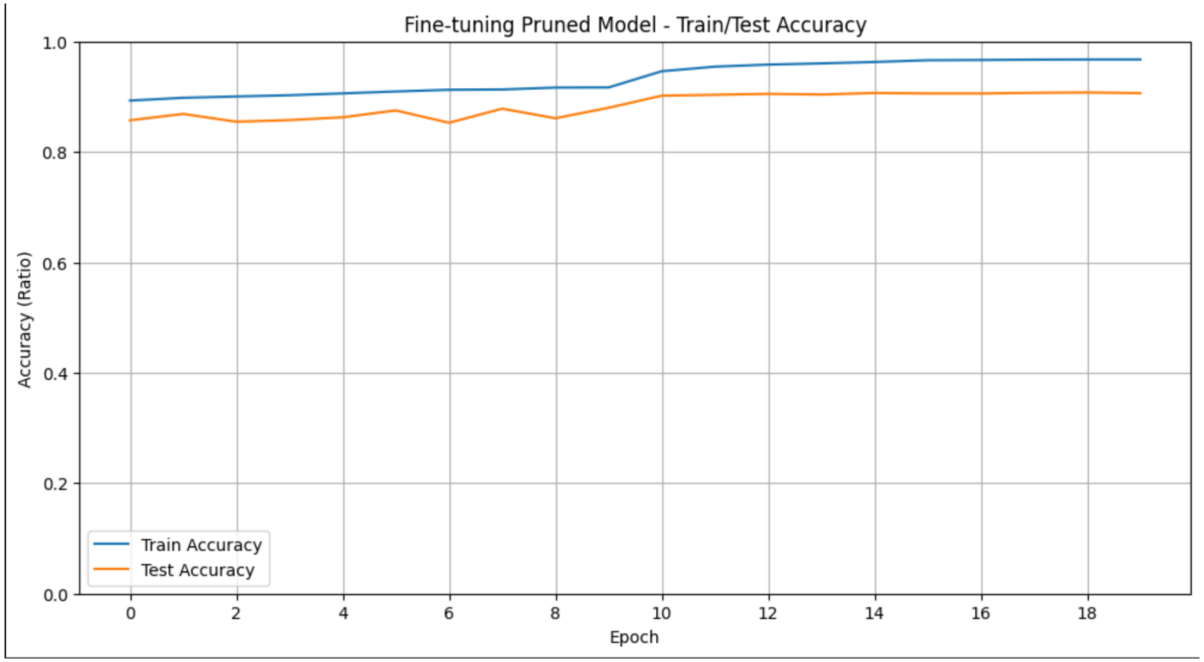
- **Plot** train accuracy and test accuracy after fine-tuning the pruned 90% model over epochs (5%)



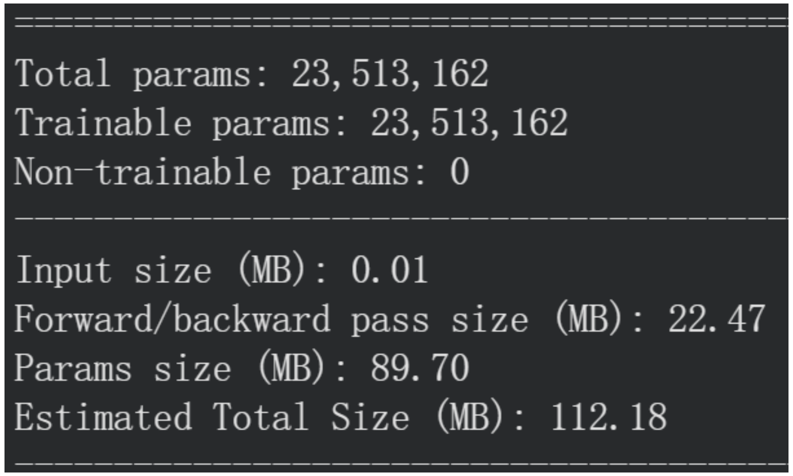
```
Train Epoch: 19 [0/50000 (0.0%)] Loss: 0.111603

Train Epoch: 19 [10000/50000 (20.0%)] Loss: 0.029873

Train Epoch: 19 [20000/50000 (40.0%)] Loss: 0.069035

Train Epoch: 19 [30000/50000 (60.0%)] Loss: 0.094922

Train Epoch: 19 [40000/50000 (80.0%)] Loss: 0.084185
Epoch: 19 Train Accuracy: 96.73%

Test set: Average loss: 0.3463, Accuracy: 9063/10000 (90.6%)


 TRAIN PRUNED MODEL DONE!
```

The last epoch – test acc=90.6% 、train acc=96.73%

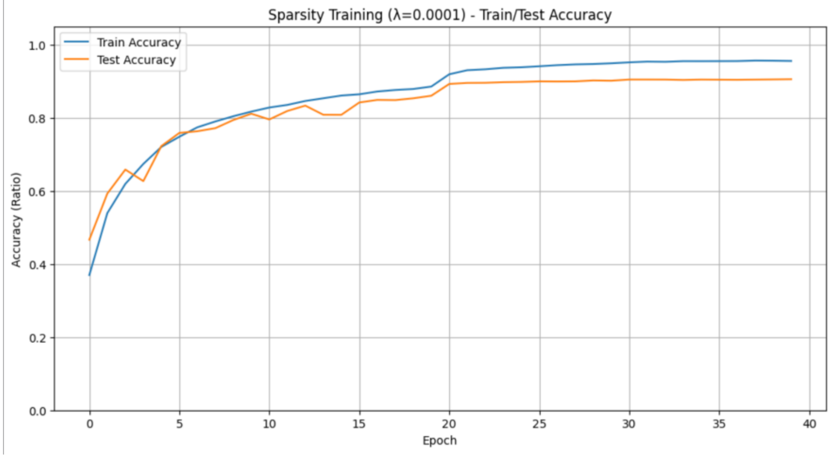
Note: The first epoch start from number 0



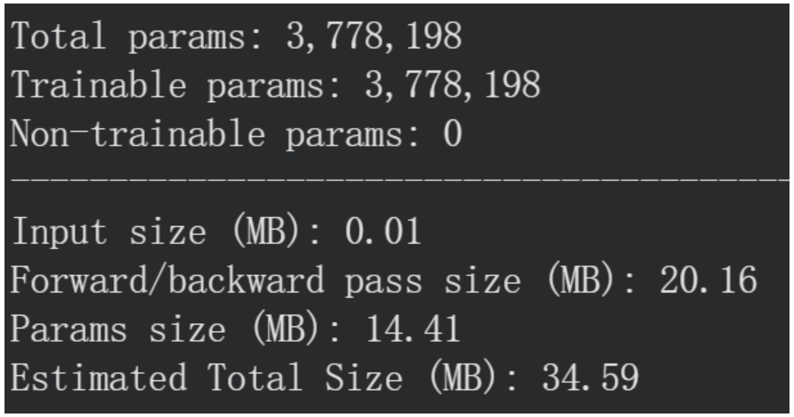
- **Show** and compare the test accuracy and model parameters among the original and fine-tuned models (5%)



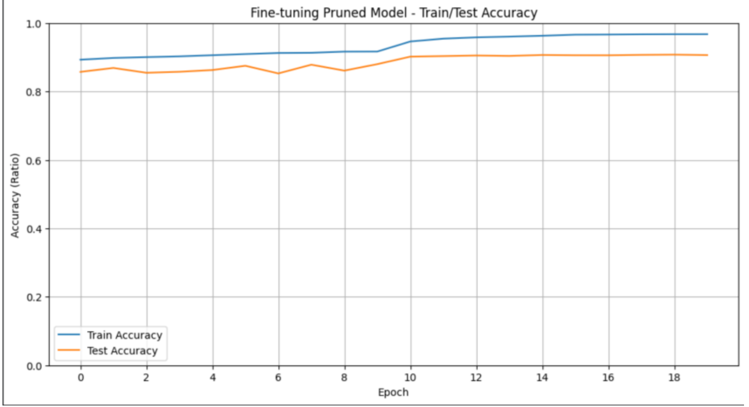
```
================================================================
Total params: 23,513,162
Trainable params: 23,513,162
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 22.47
Params size (MB): 89.70
Estimated Total Size (MB): 112.18
----------------------------------------------------------------
```



Before pruning , need 23.51M parameters, Test Accuracy 90.5% ,不過我最高有train過91%
那一次的結果並沒有截圖，是在某一次修model重train看到的
以目前這次實驗的結果最高是90.5%

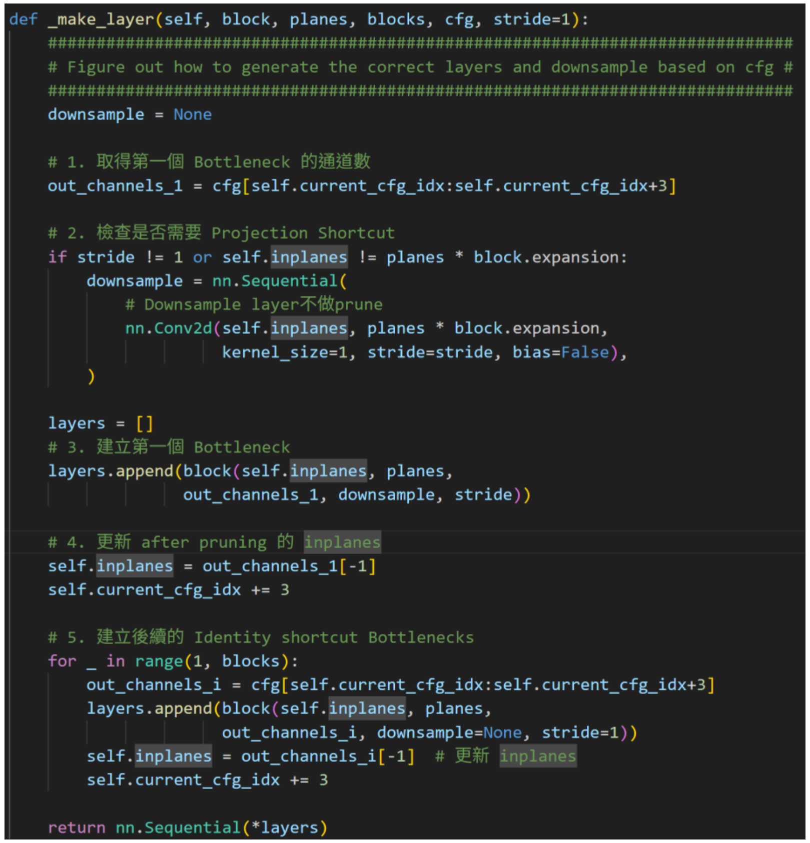


```
Total params: 3,778,198
Trainable params: 3,778,198
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 20.16
Params size (MB): 14.41
Estimated Total Size (MB): 34.59
```



After pruning , need 3.778M parameters, Test Accuracy 90.6%
是拿同一份model_best.pth,在fine-tune之前test acc只剩63.3% , 經過fine-tune之後能達到和原本prune之前的Accurate一樣高

• **Explain** how you modified resnet.py (5%)



```python
def _make_layer(self, block, planes, blocks, cfg, stride=1):
    ################################################################################
    # Figure out how to generate the correct layers and downsample based on cfg #
    ################################################################################
    downsample = None

    # 1. 取得第一個 Bottleneck 的通道數
    out_channels_1 = cfg[self.current_cfg_idx:self.current_cfg_idx+3]

    # 2. 檢查是否需要 Projection Shortcut
    if stride != 1 or self.inplanes != planes * block.expansion:
        downsample = nn.Sequential(
            # Downsample layer不做prune
            nn.Conv2d(self.inplanes, planes * block.expansion,
                      kernel_size=1, stride=stride, bias=False),
        )

    layers = []
    # 3. 建立第一個 Bottleneck
    layers.append(block(self.inplanes, planes,
                        out_channels_1, downsample, stride))

    # 4. 更新 after pruning 的 inplanes
    self.inplanes = out_channels_1[-1]
    self.current_cfg_idx += 3

    # 5. 建立後續的 Identity shortcut Bottlenecks
    for _ in range(1, blocks):
        out_channels_i = cfg[self.current_cfg_idx:self.current_cfg_idx+3]
        layers.append(block(self.inplanes, planes,
                            out_channels_i, downsample=None, stride=1))
        self.inplanes = out_channels_i[-1]  # 更新 inplanes
        self.current_cfg_idx += 3

    return nn.Sequential(*layers)
```

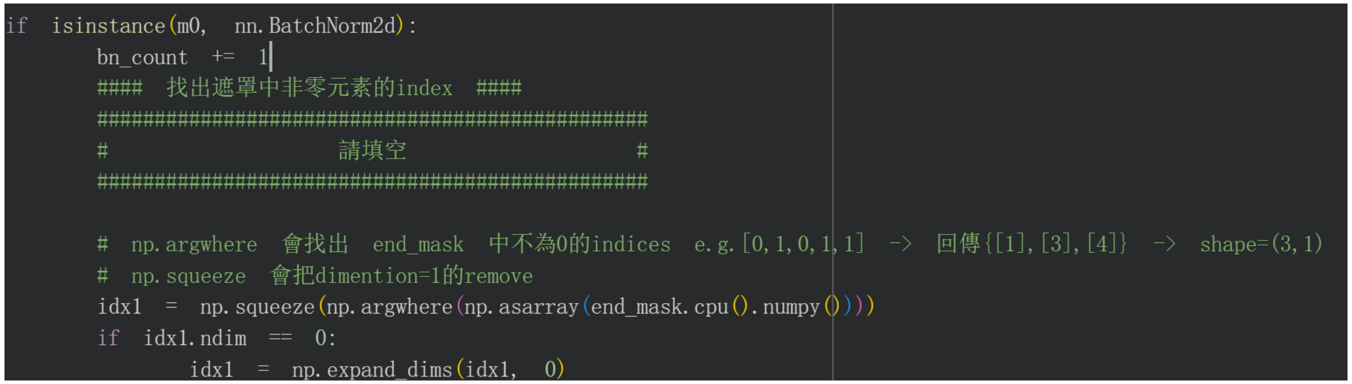
*1.*在 ResNet 類別中加入了一個 self.current_cfg_idx 變數，用來追蹤目前在 cfg 列表中的位置。

*2.***動態建立 Bottleneck,**
用 out_channels = cfg[self.current_cfg_idx : self.current_cfg_idx + 3] 來從 cfg 列表中一次讀取該 Bottleneck 所需的三個輸出通道數（分別對應 bn1, bn2, bn3）。

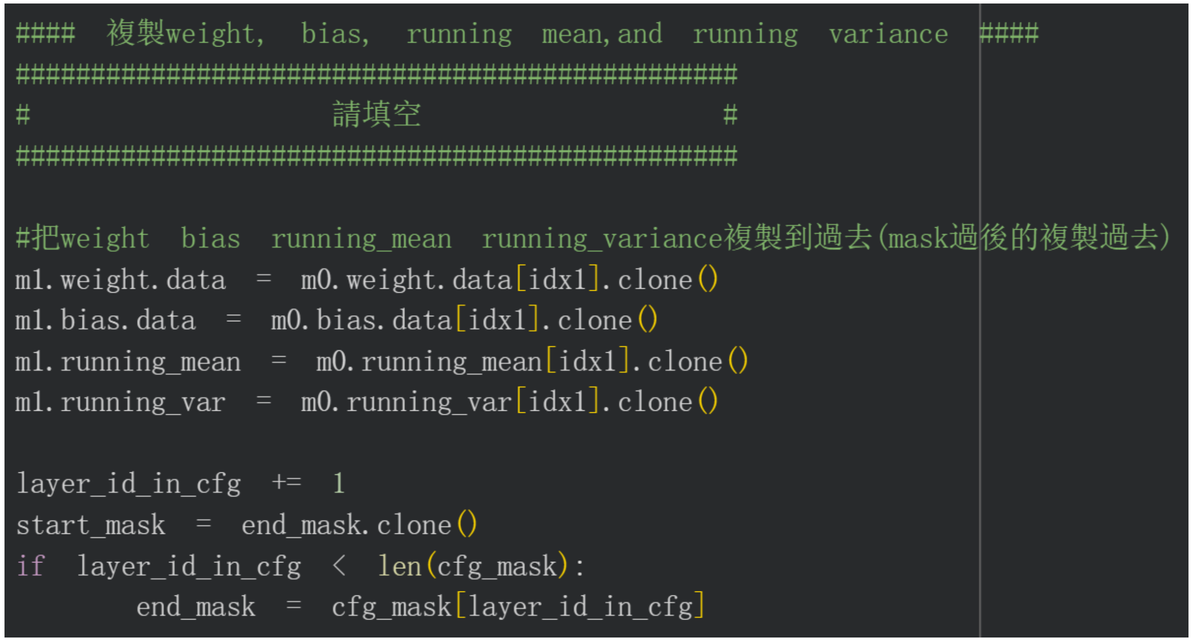
*3.***處理 Downsample**：對於每一層 的第一個block,確保Shortcut 連接的channel數能對齊避免crash

*4.***更新 inplanes**：在每個 Bottleneck 建立完成後，將 self.inplanes 更新為該block的最終輸出out_channels[]，這樣下一個區塊或最後的 fc 層才能接收到正確的輸入通道數。

- **Explain** how you copied the original model weights to the pruned model. (5%)

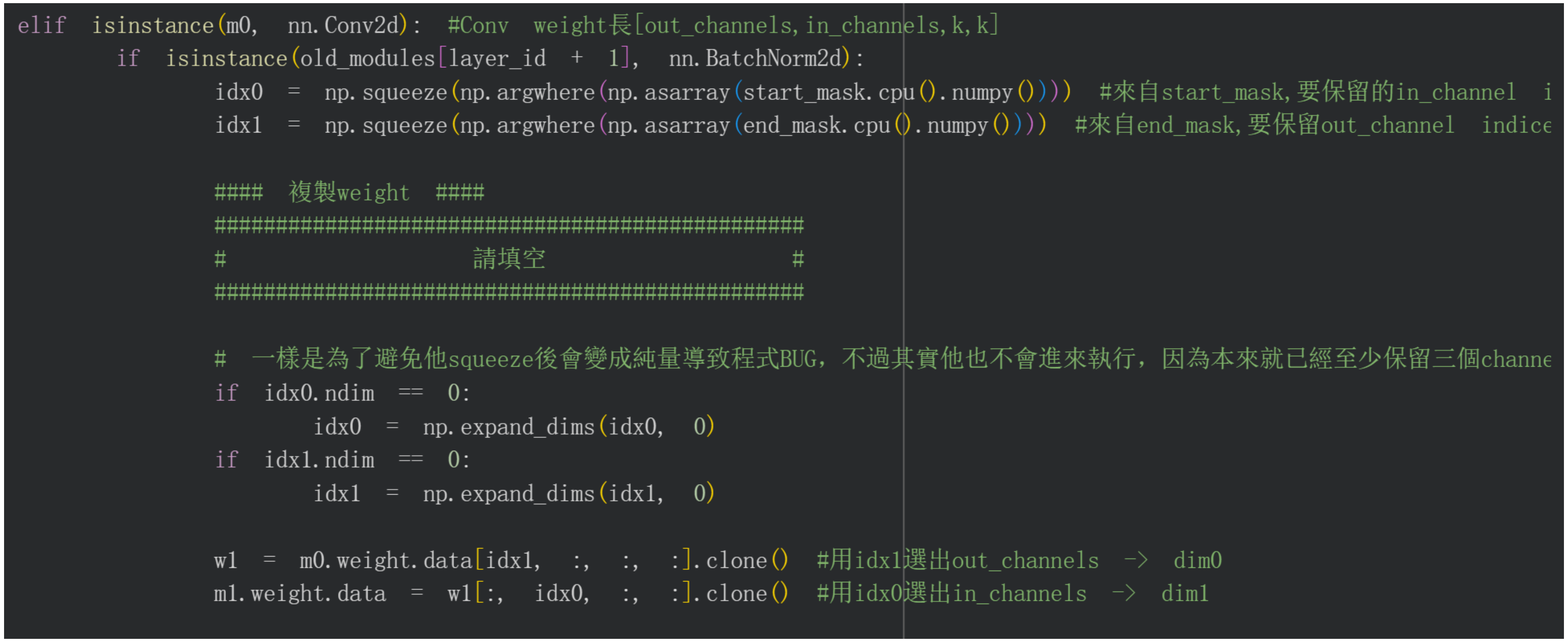


```
if  isinstance(m0,  nn.BatchNorm2d):
        bn_count  +=  1
        ####  找出遮罩中非零元素的index  ####
        ################################################
        #                    請填空                    #
        ################################################

        #  np.argwhere  會找出  end_mask  中不為0的indices  e.g.[0,1,0,1,1]  ->  回傳{[1],[3],[4]}  ->  shape=(3,1)
        #  np.squeeze  會把dimention=1的remove
        idx1  =  np.squeeze(np.argwhere(np.asarray(end_mask.cpu().numpy())))
        if  idx1.ndim  ==  0:
                idx1  =  np.expand_dims(idx1,  0)
```



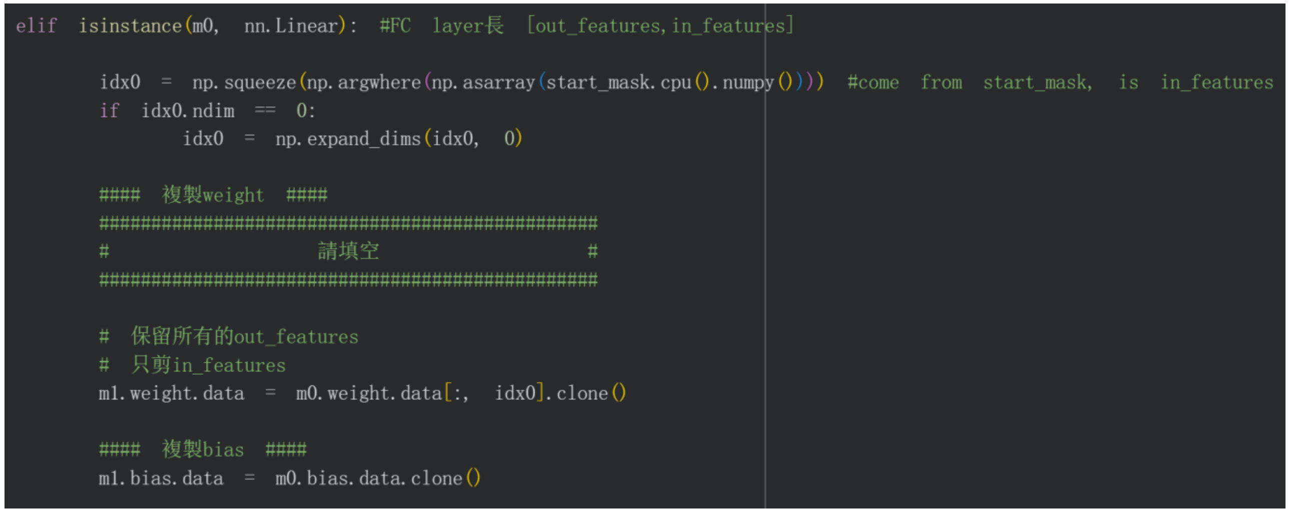
```
####  複製weight,  bias,  running  mean,and  running  variance  ####
################################################
#                    請填空                    #
################################################

#把weight  bias  running_mean  running_variance複製到過去(mask過後的複製過去)
m1.weight.data  =  m0.weight.data[idx1].clone()
m1.bias.data  =  m0.bias.data[idx1].clone()
m1.running_mean  =  m0.running_mean[idx1].clone()
m1.running_var  =  m0.running_var[idx1].clone()

layer_id_in_cfg  +=  1
start_mask  =  end_mask.clone()
if  layer_id_in_cfg  <  len(cfg_mask):
        end_mask  =  cfg_mask[layer_id_in_cfg]
```

找出idx=1 這些是要保留的channel,然後去做對應的mapping



```
elif isinstance(m0, nn.Conv2d): #Conv weight長[out_channels,in_channels,k,k]
    if isinstance(old_modules[layer_id + 1], nn.BatchNorm2d):
        idx0 = np.squeeze(np.argwhere(np.asarray(start_mask.cpu().numpy()))) #來自start_mask,要保留的in_channel i
        idx1 = np.squeeze(np.argwhere(np.asarray(end_mask.cpu().numpy()))) #來自end_mask,要保留out_channel indice

        #### 複製weight ####
        ################################################
        #                     請填空                   #
        ################################################

        # 一樣是為了避免他squeeze後會變成純量導致程式BUG，不過其實他也不會進來執行，因為本來就已經至少保留三個channe
        if idx0.ndim == 0:
            idx0 = np.expand_dims(idx0, 0)
        if idx1.ndim == 0:
            idx1 = np.expand_dims(idx1, 0)

        w1 = m0.weight.data[idx1, :, :, :].clone() #用idx1選出out_channels -> dim0
        m1.weight.data = w1[:, idx0, :, :].clone() #用idx0選出in_channels -> dim1
```

用前一層BN儲存的start_mask找出idx0 用下一層的end_mask找出idx1
然後去做mapping, FC層也差不多概念



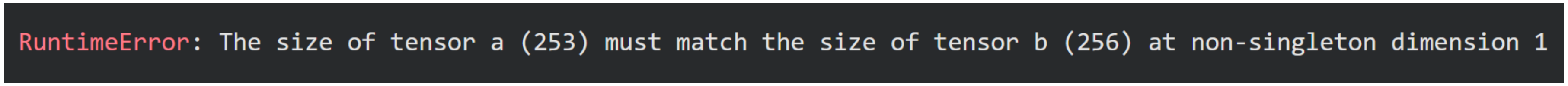
```
elif isinstance(m0, nn.Linear): #FC layer長 [out_features,in_features]

    idx0 = np.squeeze(np.argwhere(np.asarray(start_mask.cpu().numpy()))) #come from start_mask, is in_features
    if idx0.ndim == 0:
        idx0 = np.expand_dims(idx0, 0)

    #### 複製weight ####
    ################################################
    #                  請填空                      #
    ################################################

    # 保留所有的out_features
    # 只剪in_features
    m1.weight.data = m0.weight.data[:, idx0].clone()

    #### 複製bias ####
    m1.bias.data = m0.bias.data.clone()
```

- Please think about why it prompts you to fix the input and output channel numbers of each bottleneck to the channel numbers before pruning. What could happen if they are not fixed, and why?(5%)

因為在ResNet Network中，以ResNet50舉例，總共16個bottleneck
而其中有四個是projection shortcut,這個部分是因為跨維度(或channel)必須要做的處理

而最重要的是剩下的12bottleneck是”identity shortcut”
因為prune之後，本來channel都對的好好的，被prune完變成維度不一樣
這時候如果要在運算(其實就是矩陣要維度一樣才能相乘)就會有問題
那如果要他能夠順利進行，這個時候identity shortcut就會強制變成projection shortcut

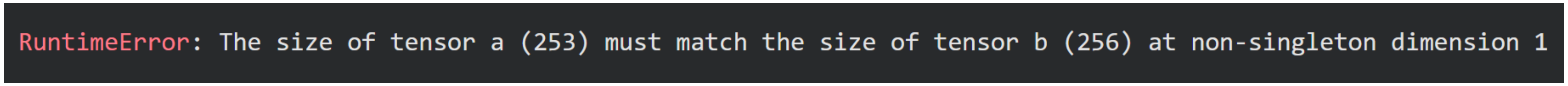
這就是為什麼必須要去做fixed input and output channel，不然prune完之後會直接把整個模型的網路架構大改，下面是我一開始沒有做好遇到的問題



```
RuntimeError: The size of tensor a (253) must match the size of tensor b (256) at non-singleton dimension 1
```

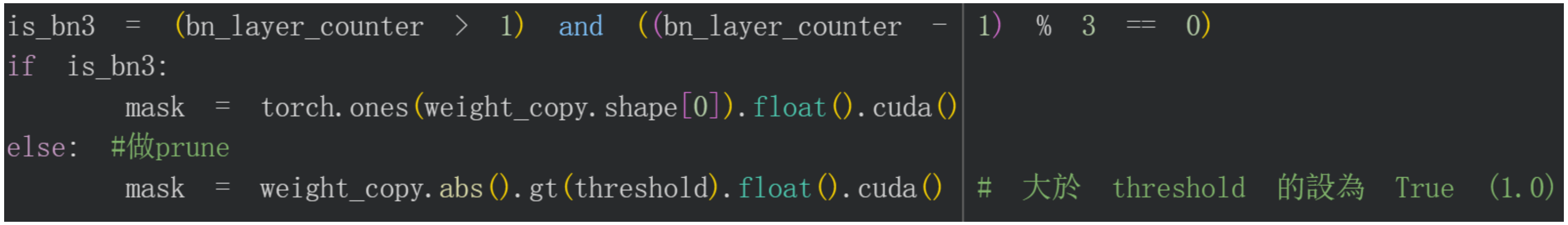
可以看到因為為了維持本來該是identity shortcut的結構
然後我在prune完之後channel改變，這時候如果仍然維持這樣就只能改成
Projection shortcut才能繼續進行，但這就已經破壞整個模型的架構了

- Describe the problems you encountered and how you solved them (5%)



```
RuntimeError: The size of tensor a (253) must match the size of tensor b (256) at non-singleton dimension 1
```

這個問題一開始困擾我很久，我原本想說可不可以讓他雖然被prune
但是把weight補0，讓他做個無效運算，但顯然我太天真了



```
is_bn3 = (bn_layer_counter > 1) and ((bn_layer_counter - 1) % 3 == 0)
if is_bn3:
    mask = torch.ones(weight_copy.shape[0]).float().cuda()
else: #做prune
    mask = weight_copy.abs().gt(threshold).float().cuda() # 大於 threshold 的設為 True (1.0)
```

我最後的解法是，跳過這一層，不要去對他做prune，這樣就能對齊channel size了