CS5120 Homework Assignment #2

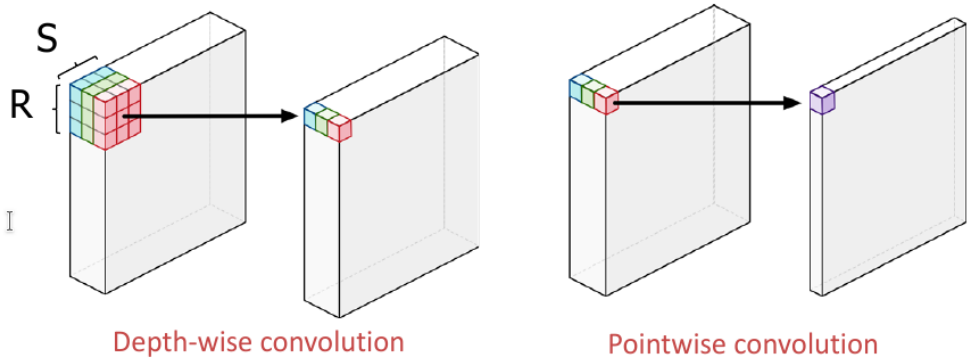
Name: 王領崧 Student ID: 107062107

1. Design Concept

此次要實作的東西為覆寫 lab1 中的 Lenet for CIFAR10 dataset。因此底下會分別介紹每一層 layer 的實作手法、如何 plot distribution、測試整個 testset accuracy，以及每種 Approach method 中特別新增的 function。

Convolution

convolution 就是參考老師上課的投影片，我使用的是 Depth-wise Separable Convolution。如下圖所示，先對每一層做 2D 的 convolution，接著使用 pointwise convolution 進行壓縮。



程式實作方法大概就是 6 層 for loop。
最外層的是 kernel number，表示正在對第幾個 kernel 進行convolution。
接下來是 channel，表示對 input 的第幾層 (總共有RGB 3 層)進行 convolution。
然後剩下的 4 層 for loop，就是定位 input & kernel 的位置與 sliding window 的移動，這樣就完成 depth-wise convolution 的部分。
pointwise convolution 也是同理，利用 for loop來掃過每個位置，然後再把 z 軸方向的 channels 全部加總即可。

```
output = np.zeros((number, tmp_width, tmp_height))
# 2D conv & pointwise
for num in range(number):
  tmp = np.zeros((channels, tmp_width, tmp_height))
  for channel in range(channels):
    for w in range(tmp_width):
      for h in range(tmp_height):
        for kw in range(kernel_width):
          for kh in range(kernel_height):
            result = input[channel, w+kw, h+kh] * weight[num, channel, kw, kh]
            tmp[channel, w, h] += result
            if tmp[channel, w, h] < - (pow(2, bitWidth)):
              tmp[channel, w, h] = - (pow(2, bitWidth))
            elif tmp[channel, w, h] > (pow(2, bitWidth)) - 1:
              tmp[channel, w, h] = (pow(2, bitWidth)) - 1
            #partial[idx] = result
            idx += 1

  for w in range(tmp_width):
    for h in range(tmp_height):
      for channel in range(channels):
        result = tmp[channel, w, h]
        output[num, w, h] += result
        if output[num, w, h] < - (pow(2, bitWidth)):
          output[num, w, h] = - (pow(2, bitWidth))
        elif output[num, w, h] > (pow(2, bitWidth)) - 1:
          output[num, w, h] = (pow(2, bitWidth)) - 1
        #partial[idx] = result
        idx += 1
```

MaxPooling

因為一開始不確定 numba 有沒有支援 torch 的 maxpooling function，因此就自己手刻了一個。想法蠻單純的，就是移動一個 2*2 大小的 sliding window，然後使用 np.max() function，尋找出 window 裡面最大的值，依序尋遍整個 activations。

```
def maxpool(input):
  channels = int(input.shape[0])
  width = int(input.shape[1])
  height = int(input.shape[2])
  size = 2;
  output = np.zeros((channels, int(width/size), int(height/size)))

  for channel in range(channels):
    for w in range(width/size):
      for h in range(height/size):
        output[channel, w, h] = np.max(input[channel, w*size:w*size+size, h*size:h*size+size])

  return output
```

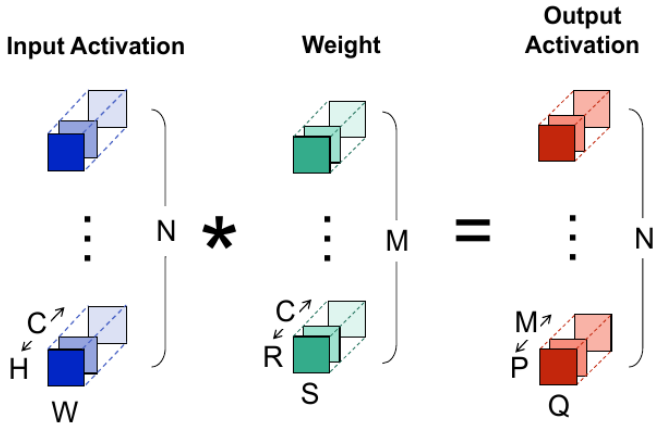
ReLU

ReLU activation function 利用 python 方便的語法，在 array 中寫入判斷式，對每一個 element 檢查其值是否 < 0，如果是的話就統一變成 0，反之則維持原本的數值。

```
x[x < 0] = 0
```

FC

Fully Connected 的概念與講義上的圖一樣，但是因為我們的 input activations 已經先 flatten 過了，所以 N = 1，只有一個 1D 的 IA，因此就是跟不同的 weight 做簡單的矩陣乘法，產生出相對應位置的 output。




```
for dp in range(depth):
  for h in range(height):
    result = input[h] * weight[dp, h]
    output[dp] += result
    if output[dp] < - (pow(2, bitWidth)):
      output[dp] = - (pow(2, bitWidth))
    elif output[dp] > (pow(2, bitWidth)) - 1:
      output[dp] = (pow(2, bitWidth)) - 1
    #partial[idx] = result
    idx += 1

for dp in range(depth):

  output[dp] += bias[dp]
  if output[dp] < - (pow(2, bitWidth)):
    output[dp] = - (pow(2, bitWidth))
  elif output[dp] > (pow(2, bitWidth)) - 1:
    output[dp] = (pow(2, bitWidth)) - 1
  #partial[idx] = bias[dp]
  idx += 1
```

第三層的 FC 因為需要加入 bias，所以單獨寫一個 function 出來，但與上面的 FC 差異並不大，只要在矩陣乘法後，增加將結果 (size = 10) 依序與 bias (size = 10) 相加的步驟就完成了。

Test Accuracy

test accuracy 主要就是改寫 lab1 中 test function，然而因為一開始沒有考慮 batch size 的關

係，我在餵圖片的時候還是一張一張的餵，然後重複4次去消化一次的 batch，沒有用 for loop 去實作是因為發現當要把 partial sums 合再一起的時候，如果寫在 for loop 中重複呼叫，會讓程式執行時間拉長不少，因此最後選擇比較暴力的寫法實作。

```
for data in dataloader:
  images, labels = data
  images = images.numpy()
  labels = labels.numpy()
  predict = np.zeros((4))
  x1, p1 = model(images[0], input_scale, conv1_weight, conv1_output_scale, conv2_weight,
    conv2_output_scale, fc1_weight, fc1_output_scale, fc2_weight, fc2_output_scale,
    fc3_weight, fc3_bias, fc3_output_scale, bitWidth)
  x2, p2 = model(images[1], input_scale, conv1_weight, conv1_output_scale, conv2_weight,
    conv2_output_scale, fc1_weight, fc1_output_scale, fc2_weight, fc2_output_scale,
    fc3_weight, fc3_bias, fc3_output_scale, bitWidth)
  x3, p3 = model(images[2], input_scale, conv1_weight, conv1_output_scale, conv2_weight,
    conv2_output_scale, fc1_weight, fc1_output_scale, fc2_weight, fc2_output_scale,
    fc3_weight, fc3_bias, fc3_output_scale, bitWidth)
  x4, p4 = model(images[3], input_scale, conv1_weight, conv1_output_scale, conv2_weight,
    conv2_output_scale, fc1_weight, fc1_output_scale, fc2_weight, fc2_output_scale,
    fc3_weight, fc3_bias, fc3_output_scale, bitWidth)

  partial = np.concatenate((partial, p1, p2, p3, p4))
  predict[0] = np.argmax(x1)
  predict[1] = np.argmax(x2)
  predict[2] = np.argmax(x3)
  predict[3] = np.argmax(x4)

  total += labels.shape[0]

  correct += np.sum(predict == labels)
  if max_samples:
    n_inferences += images.shape[0]
    if n_inferences > max_samples:
        break
```

Plot distribution

因為 partial sums 為累加完的 OA，因此在每一次呼叫 model function，我都會把這張 image 的每一層 OA 都連接起來再回傳，接著在 test function 有一個 partial array 儲存每一張 image 的 return value，最後跑完 10000張的 testset 後，再利用助教於討論區提示的加速方法，將整個 dataset partial sums plot 出來，並且印出 (min, max) 來得到整個的 range。

```
from matplotlib import pyplot as plt

def plotPartialSum(data):
  counts, bins = np.histogram(data)
  plt.hist(bins[:-1], bins, weights=counts)

  print("max: ", max(data))
  print("min: ", min(data))
```

OverFlow

overflow 發生的原因在於當我們把 bit-width 縮小時，很可能在累加 OA 的過程中，超過了 bit 能表示的範圍，所以判定 overflow 的方法，就是在每一層 range clip 的過程中，判斷累加的 OA 是否會超過 [-2^ bit-width, 2^bit-width-1]，如果會的話就是 overflow，反之則沒有 overflow 的產生，最後在將每一層的結果 OR 起來，就是最終答案。

限制**bit-width**

此次 lab 最重要的部分，實作方法也不難，就是將 for loop 中每一次累加出來的 output(*OA*)，進行一次限縮的 bit-width 的動作 (bit-width 是從外面傳進來的 parameter)，因為擔心 numba 沒有支援加速，因此自己寫了 if-else statement 來做判斷。

```
for dp in range(depth):
    for h in range(height):
        result = input[h] * weight[dp, h]
        output[dp] += result
        if output[dp] < -(pow(2, bitWidth)):
            output[dp] = -(pow(2, bitWidth))
            overflow = 1
        elif output[dp] > (pow(2, bitWidth)) - 1:
            output[dp] = (pow(2, bitWidth)) - 1
            overflow = 1
        #partial[idx] = result
        add += 1
        mul += 1
```

2. Simulation and Discussion

Approach A

testset & trainset images output

lab1 output

```
testset 1300: tensor([[-11., -6., 8., 27., 5., 14., 4., 7., -15., -15.]],
       device='cuda:0', grad_fn=<ClampBackward>)
testset 3108: tensor([[-1., -7., 4., 2., 7., -3., -7., -3., 3., 5.]], device='cuda:0',
       grad_fn=<ClampBackward>)
trainset 21280: tensor([[ 28., 88., -13., -31., -61., -30., -57., -42., 36., 81.]],
       device='cuda:0', grad_fn=<ClampBackward>)
trainset 30702: tensor([[-16., 25., -4., 19., -22., 0., 4., -2., -30., 41.]],
       device='cuda:0', grad_fn=<ClampBackward>)
```

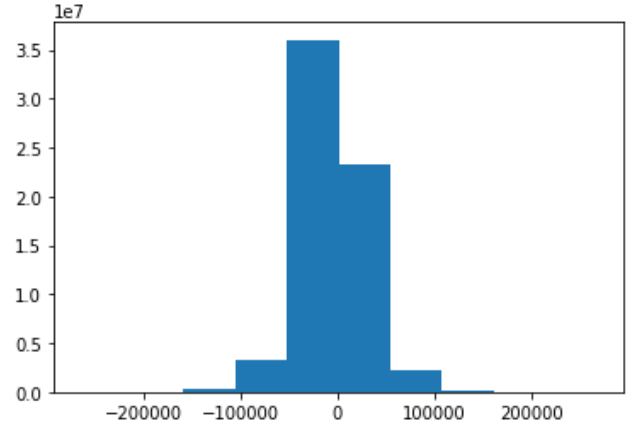
lab2 output

```
overflow: False
testset 1300: [-11. -6. 8. 27. 5. 14. 4. 7. -15. -15.]
overflow: False
testset 3108: [-1. -7. 4. 2. 7. -3. -7. -3. 3. 5.]
overflow: False
testset 21280: [ 28. 88. -13. -31. -61. -30. -57. -42. 36. 81.]
overflow: False
testset 30702: [-16. 25. -4. 19. -22. 0. 4. -2. -30. 41.]
```

圖一為利用 lab1 model 所跑出來的 4 張圖片的 output。圖二為此次作業所跑出來的 output，可以看到結果是一致的，由此驗證實作的 conv, maxpooling, FC, ReLU 是正確的，而且都沒有 overflow 的產生，表示所使用的 bit-width 是可以涵蓋以上四張圖的 range。

plot the distribution of all layers

max: 267554.0
min: -265304.0
bit-width: 32 -> accuracy : 53.4



由 distribution chart 可以看到，partial sums 大致都分布在 [-110000, 110000] 中，只有極少部分是在[-200000, -110000] 和 [110000, 200000] 之中，然而從 (min, max) 中，卻發現 range 竟然是從 (-260000, 260000)左右，顯示出 partial sums 存在 outlier 的問題。
同時也發現 partial sums 的 bit-width可以下降，因為原本 32-bit precision 可以表示 [-2147483648, 2147483647]，但顯然的我們只需要 18, 19 bit-width 就可以表示整個 range。

reduce the bit-width

lab1 accuracy

```
Accuracy of the network on the test images after all the weights and the bias are quantized: 53.41%
```

lab2 accuracy

```
bit-width: 32  ->  accuracy :  53.4
bit-width:  21  -> accuracy :  53.4
bit-width:  20  -> accuracy :  53.4
bit-width:  19  -> accuracy :  53.4
bit-width:  18  -> accuracy :  53.48
bit-width:  17  -> accuracy :  52.02
bit-width:  16  -> accuracy :  45.21
bit-width:  15  -> accuracy :  24.48
```

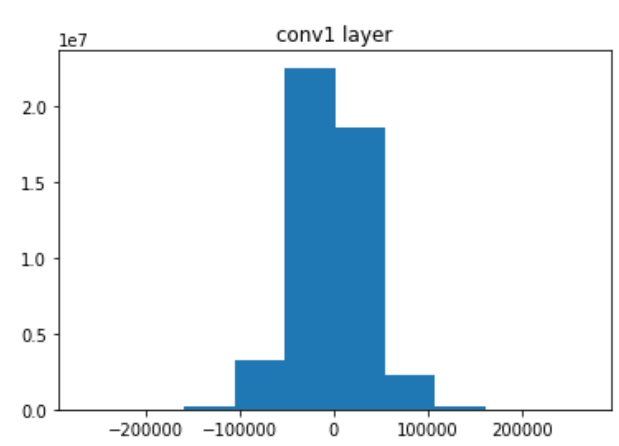
圖一為 lab1 model 所得到的 accuracy，圖二為此次作業在不同 bit-width 情況下得到的 accuracy，可以發現大概在 bit-width = 17 的時候，accuracy 就開始下降，因此在不損失 accuracy 的情況下，我們只能將 bit-width 降至18，這也與上一小題的討論相符。

Approach B

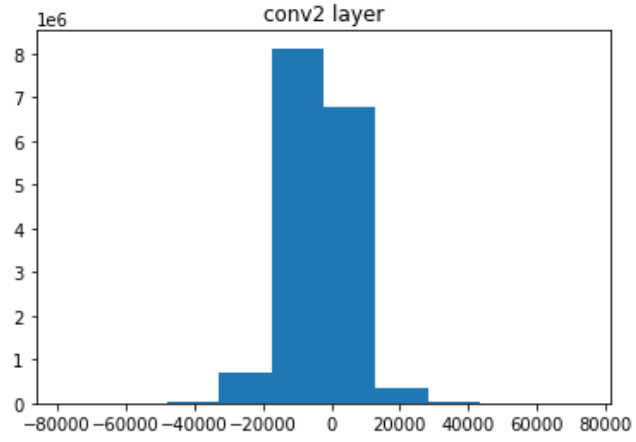
```
bit-width: 32  ->  accuracy :  53.4
bit-width:  21  -> accuracy :  53.4
bit-width:  20  -> accuracy :  53.4
bit-width:  19  -> accuracy :  53.4
bit-width:  18  -> accuracy :  53.48
bit-width:  17  -> accuracy :  52.02
bit-width:  16  -> accuracy :  45.21
bit-width:  15  -> accuracy :  24.48
```

由 Approach A 的結果可以得知，bit-width = 18 大概就是最小的 bit-width 去涵蓋整個 partial sum 的 range，因此當 bit-width 降至 17，accuracy下降了大約 1% (1.4%)，之後在往下縮 bit-width，accuracy 下降速度會越來越快，推測應該是因為犧牲到越來越多 partial sum 的精準度 (也就是有被 clamp 到)。
因此由結果圖來看，在 accuracy 允許下降 1% 左右的情況下，可以將 bit-width 降至 17。
Notes: 我覺得 19 → 18 accuracy 會微微升高的原因是因為我們排除掉了非常少部分的 outlier，讓整個 model 不會受到極值的影響，進而使得準確率有一點點的上升。

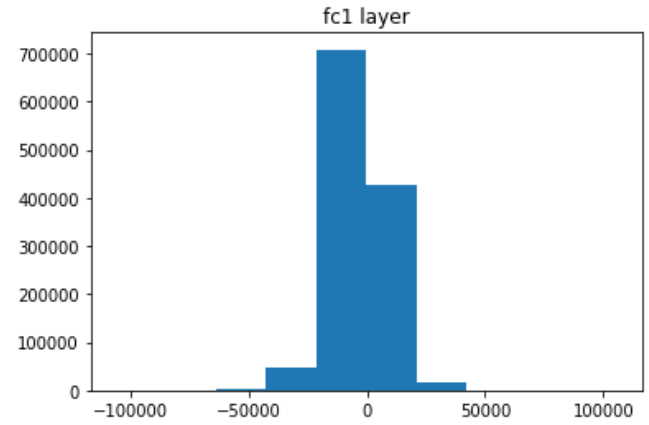
Approach C



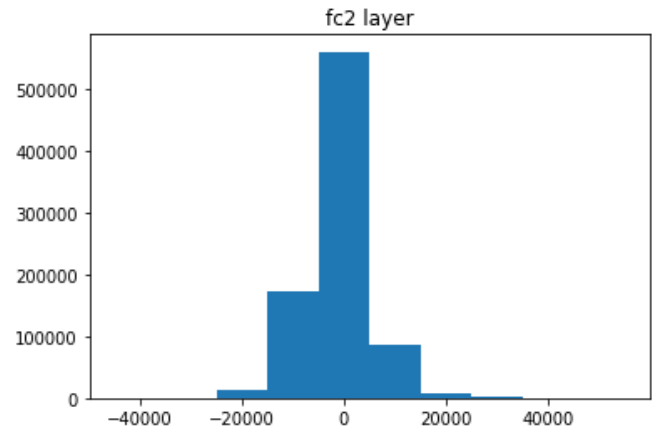

```
max:  267554.0
min:  -265304.0
```



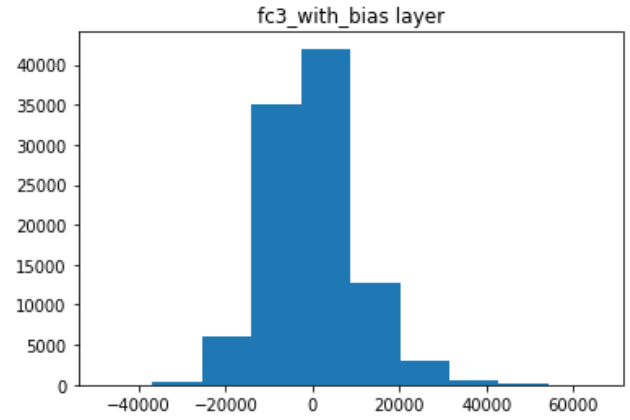

```
max:  74008.0
min:  -78578.0
```




```
max:  105713.0
min:  -106446.0
```




```
max:  54765.0
min:  -44795.0
```




```
max:  65617.0
min:  -48264.0
bit-width: 32  ->  accuracy :  53.4
```

由上面的 distribution chart 可以發現各個 layer 間的 range 差距很大，同樣依照 Approach A 的概念，從最小 bit-width 能包含整個 layer 的 range 開始，土法煉鋼的往下嘗試，看看會不會因為去除一些 outlier 使得 accuracy 提高，抑或者是直接下降。
最後測試的結果如下，最好的 accuracy 可以到 53.5 (53.4 → 53.5)，左下圖。但如果要看最低 bit-width 的話，則是右下圖這樣， accuracy (53.47) 雖然有所下降，但是仍然比初始的 53.4 高。右下圖那個已經是 minimum bit-width，如果調低其中一個 layer 的 bit-width，都會使得 accuracy 降到 53.2 以下。

```
accuracy :  53.5
conv1_bitWidth:  19
conv2_bitWidth:  16
fc1_bitWidth:  17
fc2_bitWidth:  16
fc3_bitWidth:  17
```

```
accuracy :  53.47
conv1_bitWidth:  18
conv2_bitWidth:  16
fc1_bitWidth:  17
fc2_bitWidth:  16
fc3_bitWidth:  17
```

observation with train set
嘗試跑過 trainset distribtution，然而不知道是什麼原因 (懷疑可能是 50000張有點過於龐大)，每次都跑超過一個小時，然後直接被 colab 禁用 GPU，因此沒有產出任何結果 :(
不過我有將 plot trainloader 的 code 附在上面，或許助教用 server 可以跑得動。

Energy function

```
layer 1 - the number of additions:  366912
layer 1 - the number of multiplication:  352800
layer 2 - the number of additions:  249600
layer 2 - the number of multiplication:  240000
layer 3 - the number of additions:  48000
layer 3 - the number of multiplication:  48000
layer 4 - the number of additions:  10080
layer 4 - the number of multiplication:  10080
layer 5 - the number of additions:  850
layer 5 - the number of multiplication:  840
```

$$E_W = s_{mul} \times N_{mul} + s_{add} \times N_{add},$$

$$s_{mul} = 64 \times \left(\frac{B_{mul}}{8}\right)^2, s_{add} = B_{add},$$

上圖為 Convolution & FC layer 所需要進行的加法與乘法次數
搭配助教在 spec 中提供的算式，計算出下面的結果 (算式的部分因為過於冗長，就省略不寫了)
Approach A (bit-width: 18)： 223,315,236
Approach B (bit-width: 17)：199,829,594
Approach C (conv1: 18, conv2: 16, fc1: 17, fc2:16, fc3:17)：204,032,186

三者比較之下，可以看到 Approach B 的 energy 是最少的，雖然直覺來說會覺得 Approach C 最低，然而仔細想想會發現，雖然 Approach C 的 bit-width 是最 tight，可是最大那層(conv1)的乘加計算量比 Approach B 還高一個 bit，所以就算底下的 bit-width 在這麼縮減，對於 energy reduction 也是不及最 significant 的那一層。這也給了我一點啟發，在 improveperformance 的時候，有時候只要盡全力去優化運算量最龐大的那一層就好，底下的其他層，儘管花了很多心力與時間進行優化，效果反而遠遠不及最上層做一個簡單縮減 bit-width 的動作。

遇到的困難

這次的作業雖然有提早開始做，但仍然花了整整4天的清明連假在努力趕工。可能是一開始在 partial sums 的認知上與 spec 要求的有所落差吧，在 plot & reduce-bit 的部份花了很多的時間與心力調整，另外在 numba library 的使用上，也因為誤用了某些 function，導致程式不但沒有加速，反而變得更慢，這邊也是花了不少時間進行 debug 的動作，感覺這次的作業不算特別的難，只能說是自己的經驗不足，在很多地方繳了不少的「學費」。

另外一點是 plot partial sums distribution，明明每個矩陣運算的 function 都有加速到，但跑一個 testset 還是需要花上 4 min 30 sec 左右，trainset 更扯，跑到 colab 直接不給我使用 GPU，也是沒有跑出最終結果。

3. Summary

lab2 對 LeNet network layer 進行改寫，原本使用 pytorch 內建的 function，我們依序手刻 convolution, ReLU, Maxpooling, Fully connected layer，這樣讓我們能觀察 partial sums distribution，進一步限縮 partial sums 在某個 bit-width 間，並且檢測是否有 overflow 的情況發生，以及計算 Energy function。我們也利用 3 種不同的 approach method，了解到降低 Energy 最有效的方法就是優先對於最龐大計算量的部分進行優化，如果有餘力再對小細節進行處理。

很感謝這次一同討論的學長們，幫我釐清蠻多的觀念，也感謝助教很用心且快速的回答討論區上面的問題，讓也有同樣問題的我，能往正確的道路上前進。不過希望之後的作業能少一點 training 的部份，實在是沒有很喜歡花很多時間在等待 model training 與 plot 之類的東西啊~

4. 執行方法

請助教將我的 .py 檔案以及 input, weight, scale...放在同一個資料夾，再執行檔案

另外，我的程式跑一次 plot distribution 需要大約 4-5 min，請助教有個心理準備，非常不好意思。