

This is a sample of the report, but applicable for all homework.

[113062575] [徐義鈞] This is for double verification.

Don't copy the problem statement, just write the answer.

Please write down the question number in unit of sub-question.

Please write down the sub-question number even if you don't know how to solve it.

Part1

(1.3.1)

Layer (type:depth-idx)	Output Shape	Param #
Net	[1, 10]	--
└─Sequential: 1-1	[1, 6, 28, 28]	--
└─Conv2d: 2-1	[1, 6, 28, 28]	150
└─ReLU: 2-2	[1, 6, 28, 28]	--
└─Sequential: 1-2	[1, 6, 14, 14]	--
└─MaxPool2d: 2-3	[1, 6, 14, 14]	--
└─Sequential: 1-3	[1, 16, 10, 10]	--
└─Conv2d: 2-4	[1, 16, 10, 10]	2,400
└─ReLU: 2-5	[1, 16, 10, 10]	--
└─Sequential: 1-4	[1, 16, 5, 5]	--
└─MaxPool2d: 2-6	[1, 16, 5, 5]	--
└─Sequential: 1-5	[1, 120, 1, 1]	--
└─Conv2d: 2-7	[1, 120, 1, 1]	48,000
└─ReLU: 2-8	[1, 120, 1, 1]	--
└─Sequential: 1-6	[1, 84]	--
└─Linear: 2-9	[1, 84]	10,080
└─ReLU: 2-10	[1, 84]	--
└─Sequential: 1-7	[1, 10]	--
└─Linear: 2-11	[1, 10]	840

Total params: 61,470

Trainable params: 61,470

Non-trainable params: 0

Total mult-adds (Units.MEGABYTES): 0.42

Input size (MB): 0.00

Forward/backward pass size (MB): 0.05

Params size (MB): 0.25

Estimated Total Size (MB): 0.30

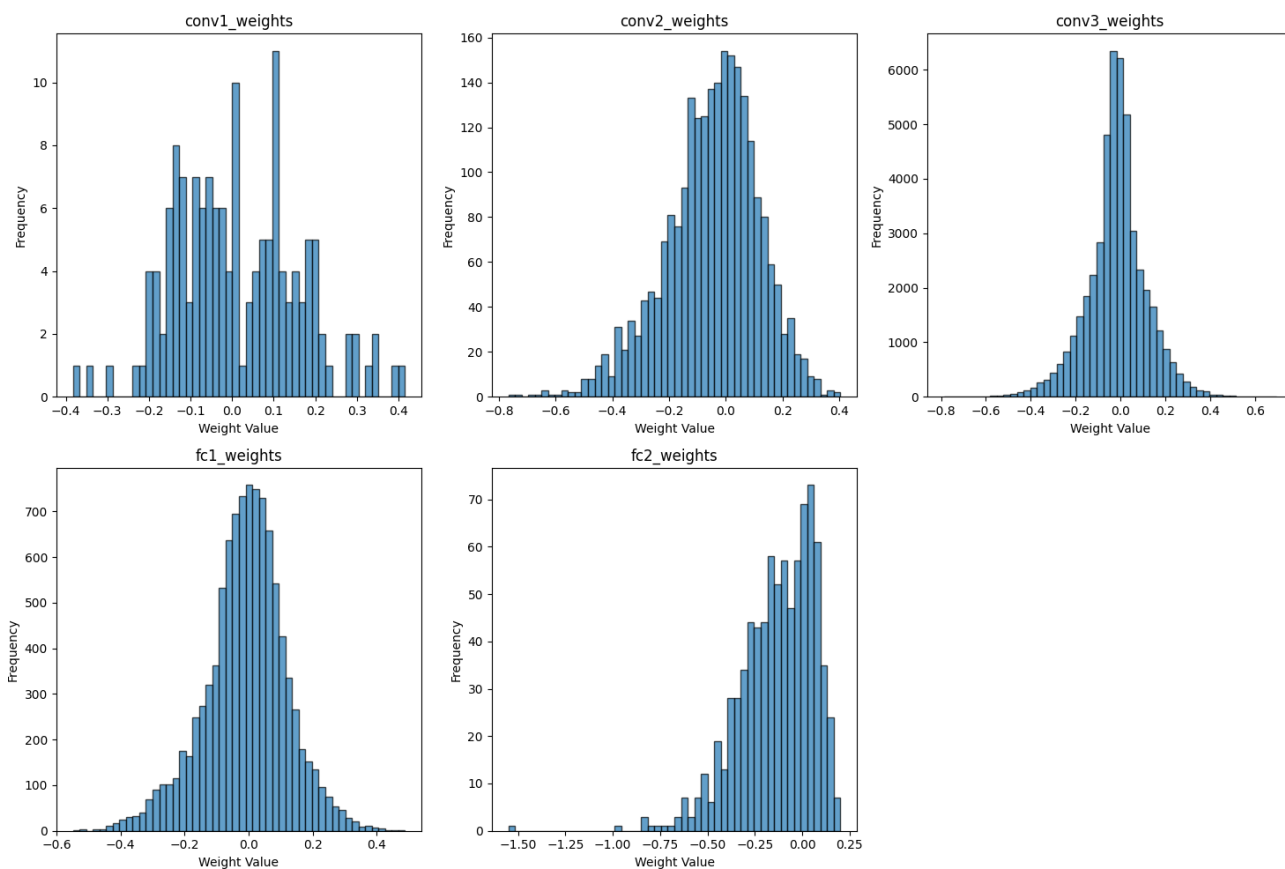
(1.3.2)

	type	input activation size	output activation size	activation function
conv1	convolution	1*32*32	6*28*28	ReLU
maxpool2	pooling	6*28*28	6*14*14	
conv3	convolution	6*14*14	16*10*10	ReLU
maxpool4	pooling	16*10*10	16*5*5	
conv5	convolution	16*5*5	120*1*1	ReLU
fc6	fully-connected	120	84	ReLU
output	fully-connected	84	10	

(1.3.3)

可以，藉由修改input / ouput activation size 即可正常運作。但是會使參數量顯著提升，影響模型效能，對於影像識別的表現可能會下降。

(2.1.1)



(2.2.1)

我使用symmetric quantization, 具體計算為: $\max(\text{abs}(\text{weights})) / 255$

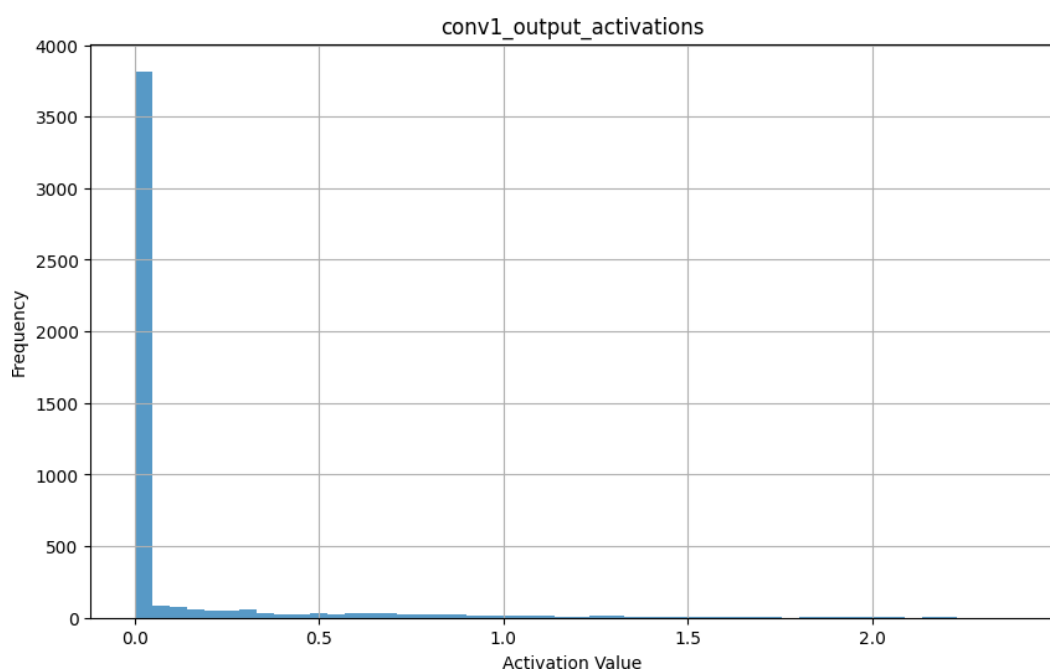
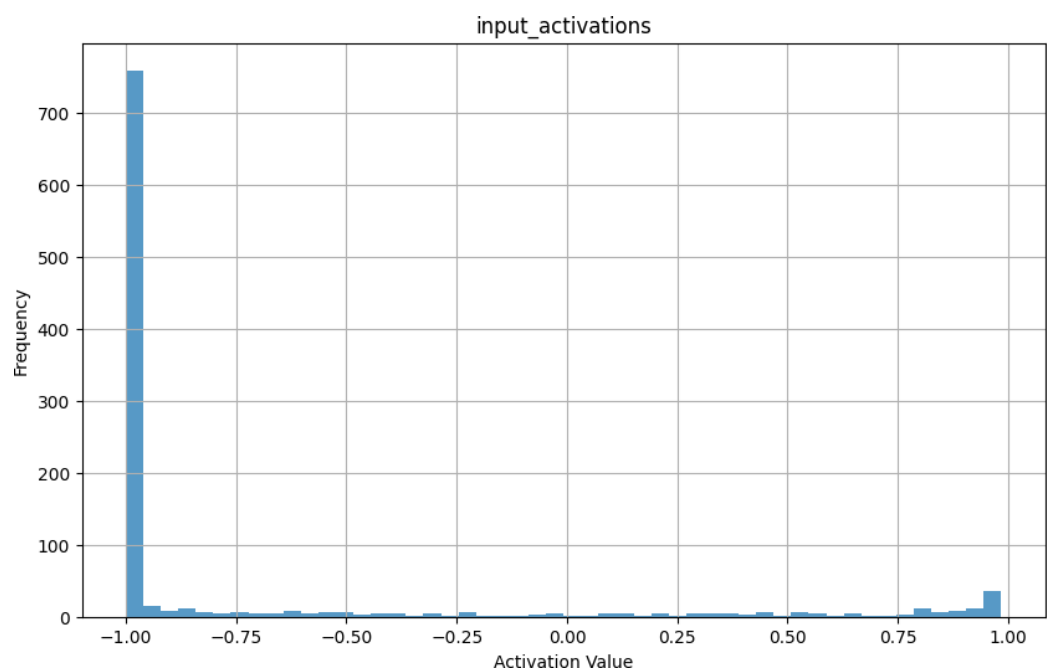
(2.2.2)

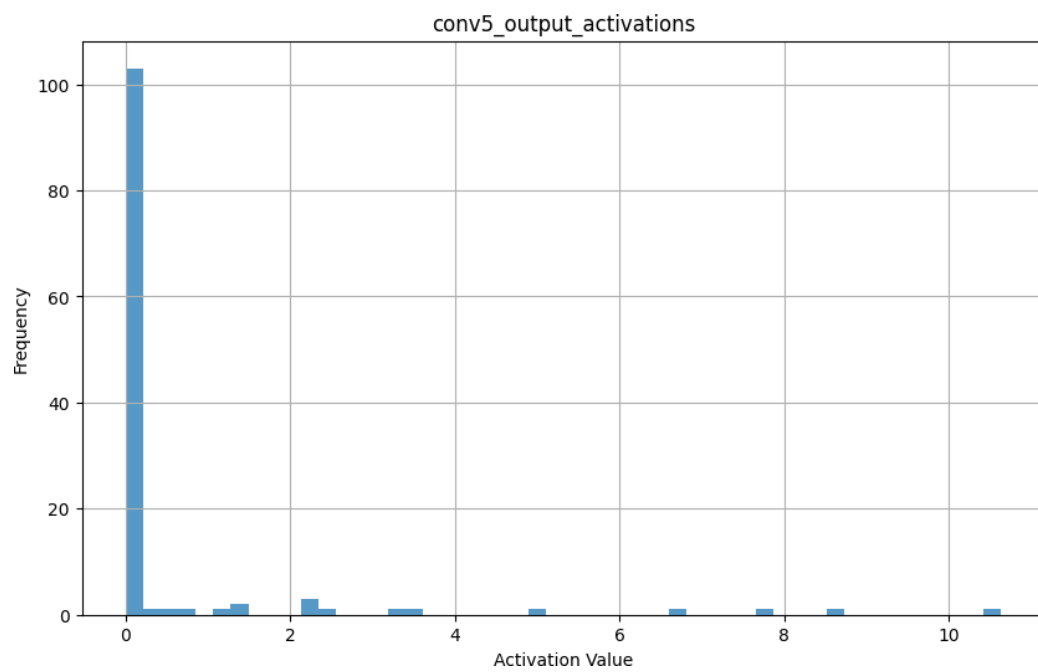
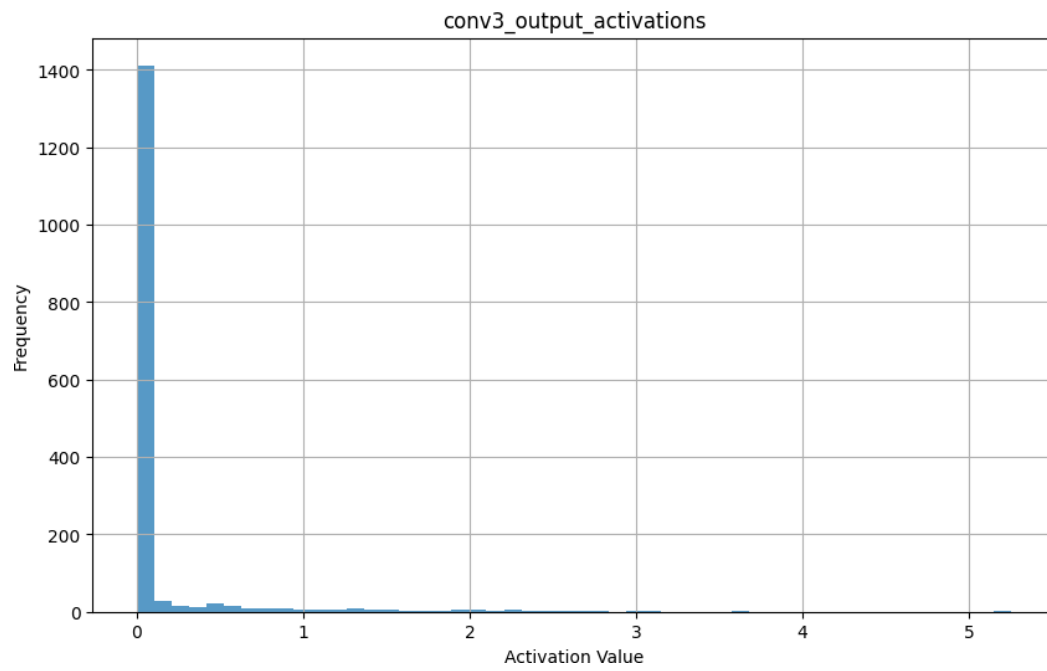
Accuracy of the network on the test images: 98.69%

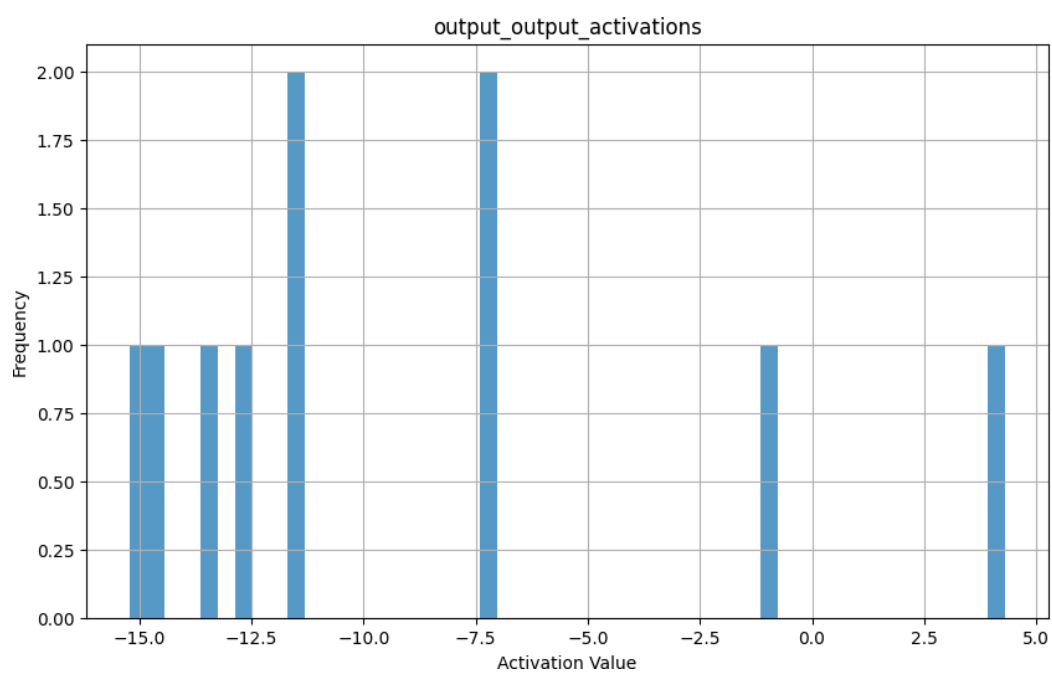
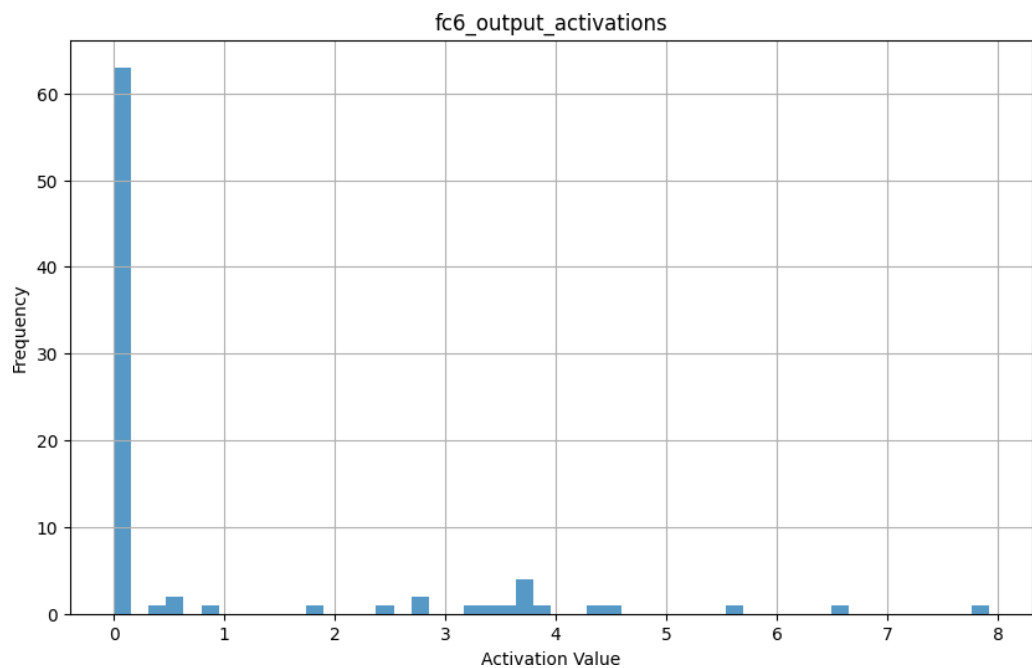
Accuracy of the network after quantizing all weights: 98.68%

Accuracy degradation is 0.01%

(2.3.1)







(2.4.1)

SI , $SWconv1$, and $SOconv1$ 這三者的計算方式概念都一樣，以 SI 為例：
 $\max(\text{abs}(\text{inputactivations})) / 255$ ，其他兩者依此類推。

(2.4.2)

$$Oconv1q = \frac{SWconv1 * SI}{SOconv1} * (Wconv1q * Iq)$$

$$M_1 = \frac{SWconv1 * SI}{SOconv1}$$

(2.4.3)

$$O_{conv3q} = \frac{SW_{conv3} * SO_{conv1}}{SO_{conv3}} * (W_{conv3q} * O_{conv1q})$$

(2.4.4)

$$Mn = \frac{SW_n * SI}{SO_n}, \text{ if } n = 1$$
$$= \frac{SW_n * SO_{n-1}}{SO_n}, \text{ otherwise}$$

(2.4.6)

使用floor在硬體層面比較容易實作，相比於round還需要去設計硬體考慮carry-in carry-out

(2.4.7)

由於output_scale為小於1的浮點數， $x * \text{output_scale}$ 為浮點數乘法運算較難實現。而 $\text{round}(1/\text{output_scale})$ 為整數，則 $x/\text{round}(1/\text{output_scale})$ 為整數除法運算，硬體上較容易實現

(2.5.1)

$$S_\beta = SW * SO_{n-1}$$

(3.1.1)

QAT訓練時就考量quantization，模型學會在該條件下調整權重減少quantization error，故相比於PTQ訓練後才去做quantization，能有更高的accuracy

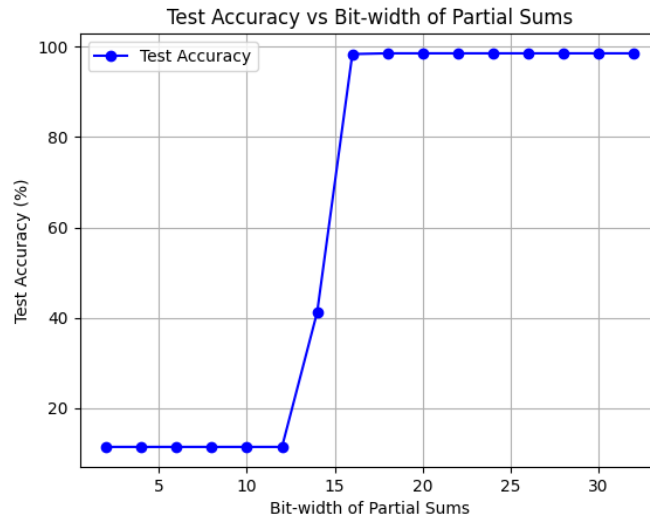
(3.1.2)

quant: 將 floating-point 的 weight 轉成 quantized values (e.g., 8-bit integer values)

dequant: 將 quantized values 轉回 floating-point 以供後續的layer運算。

Part2

(2.1.1)



(2.1.2)

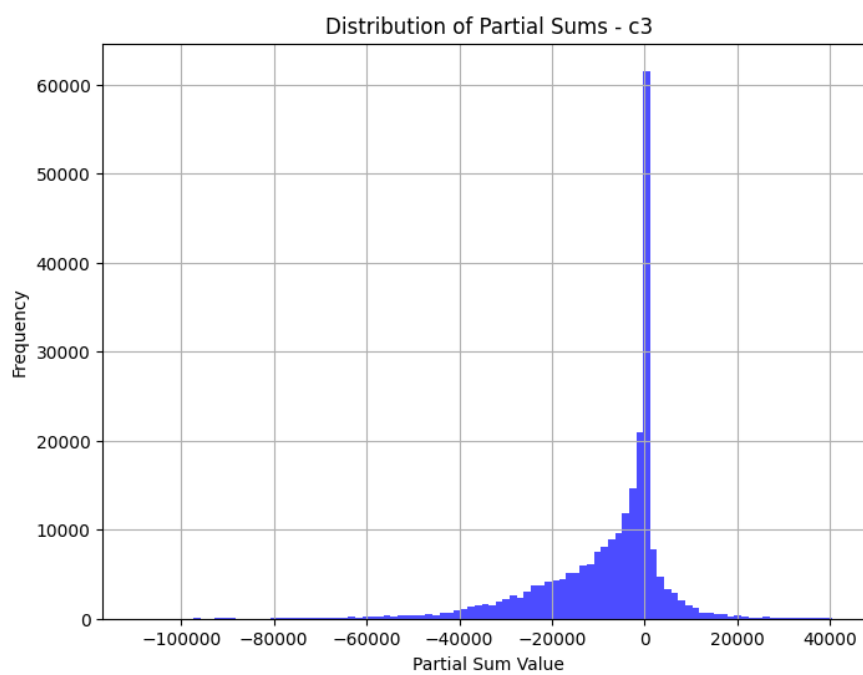
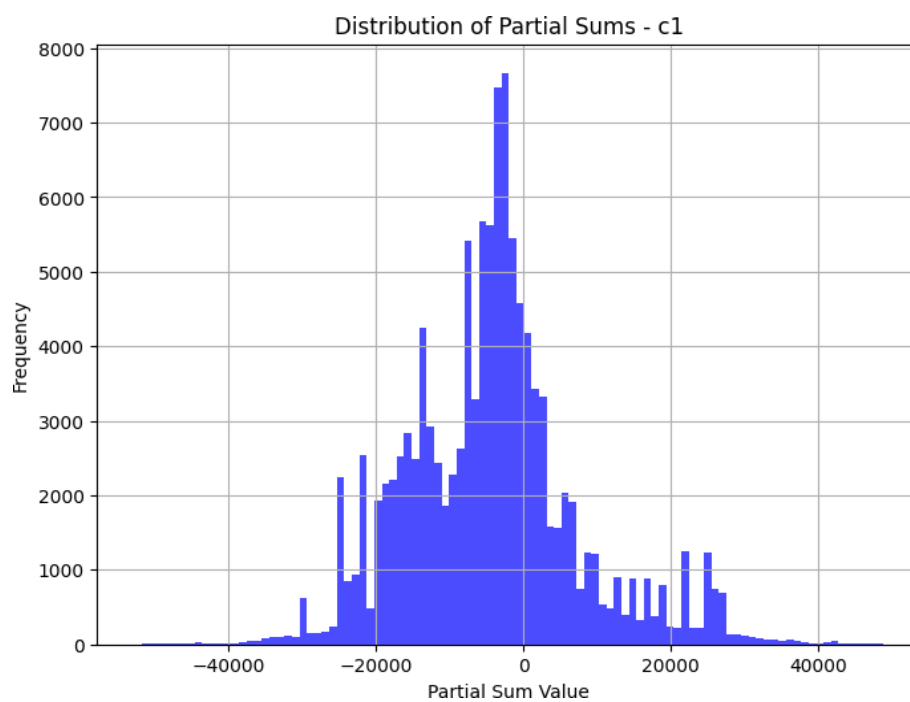
```

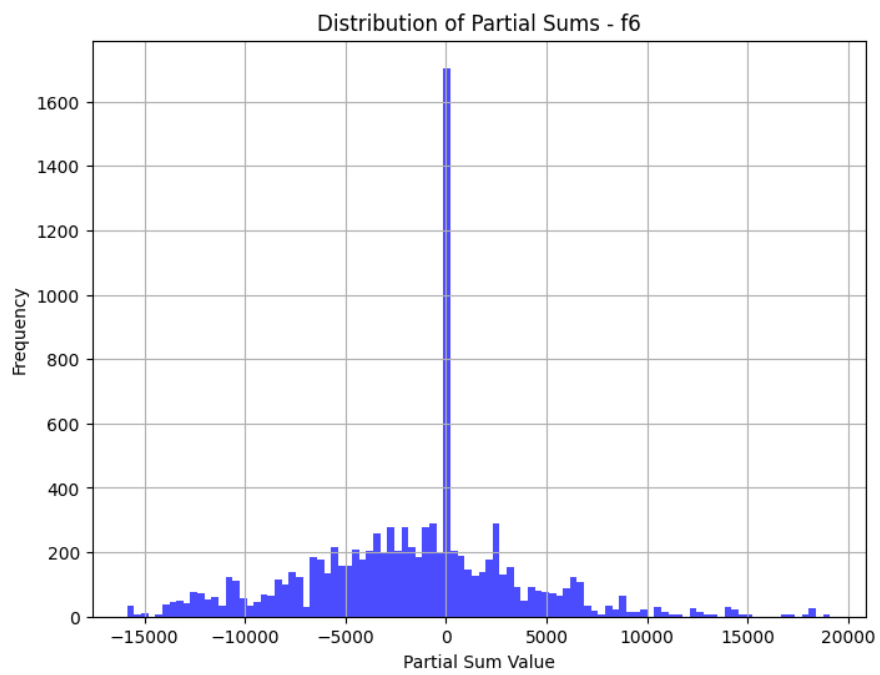
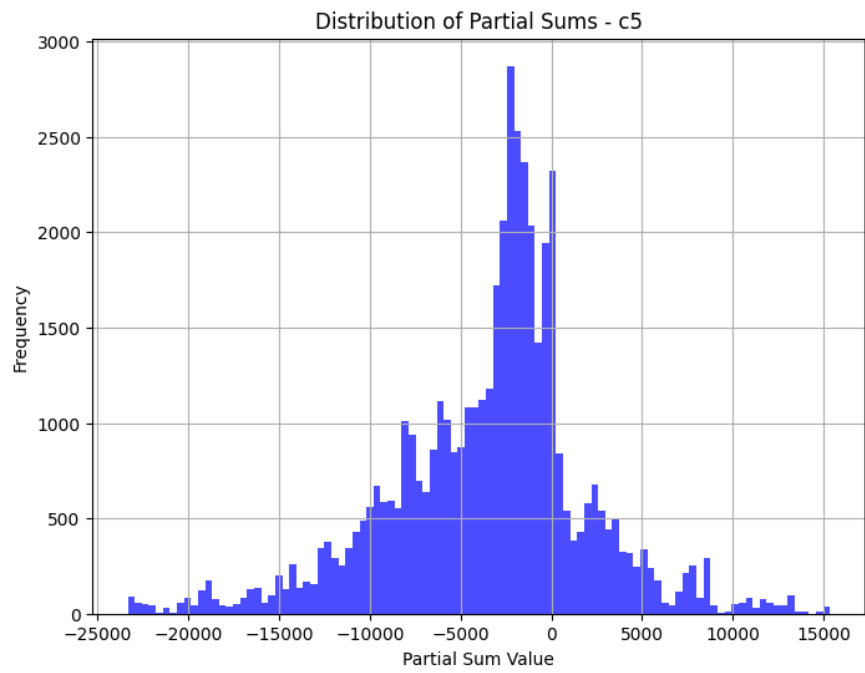
bit: 2
bit-width range: (-2, 1)
Accuracy: 11.35%
bit: 4
bit-width range: (-8, 7)
Accuracy: 11.35%
bit: 6
bit-width range: (-32, 31)
Accuracy: 11.35%
bit: 8
bit-width range: (-128, 127)
Accuracy: 11.35%
bit: 10
bit-width range: (-512, 511)
Accuracy: 11.35%
bit: 12
bit-width range: (-2048, 2047)
Accuracy: 11.35%
bit: 14
bit-width range: (-8192, 8191)
Accuracy: 41.22%
bit: 16
bit-width range: (-32768, 32767)
Accuracy: 98.41%
bit: 18
bit-width range: (-131072, 131071)
Accuracy: 98.58%
bit: 20
bit-width range: (-524288, 524287)
Accuracy: 98.58%
bit: 22
bit-width range: (-2097152, 2097151)
Accuracy: 98.58%
bit: 24
bit-width range: (-8388608, 8388607)
Accuracy: 98.58%
bit: 26

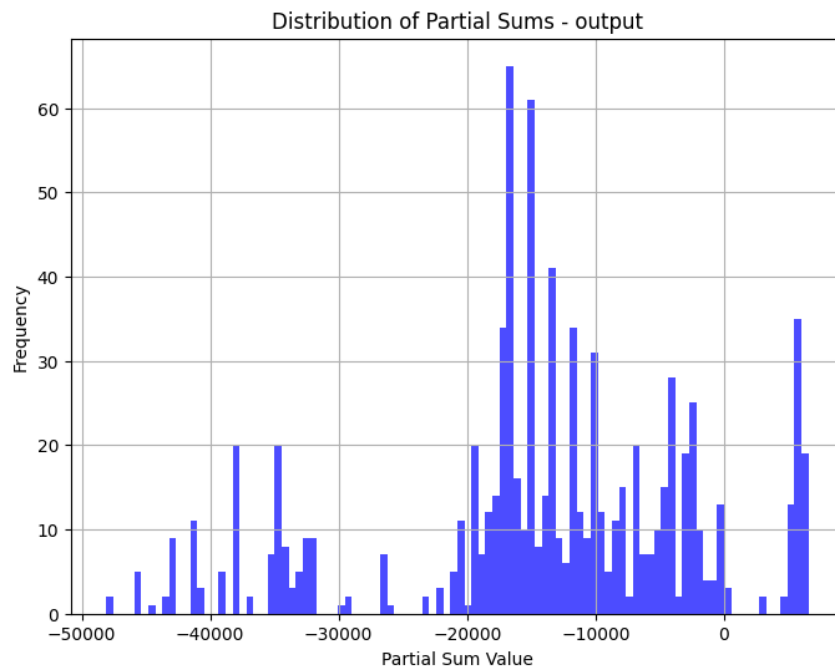
```

smallest bit-width of partial sums that maintains the same accuracy = 18

(2.2.1)







	min	max	standard deviation
C1	-52685	48834	11930.04150
C3	-109360	42057	13442.25206
C5	-23325	15341	5500.06614
F6	-15866	19112	5324.82541
Output	-48103	6569	11984.27439

(2.2.2)

	minimum bit-width
C1	18
C3	18
C5	16
F6	16
Output	16

Accuracy: 98.58%

(3.1.1)

根據程式碼 Batch Size = 4

```

for n in range(N):
    for m in range(out_channels):
        for p in range(P):
            for q in range(Q):
                x_out[n][m][p][q] = 0
                for r in range(kernel_size):
                    for s in range(kernel_size):
                        for c in range(C):
                            h = p * stride + r
                            w = q * stride + s
                            x_out[n][m][p][q] += x[n][c][h][w] * weights[m][c][r][s]

                if(x_out[n][m][p][q] < psum_range[0]):
                    x_out[n][m][p][q] = psum_range[0]
                elif(x_out[n][m][p][q] > psum_range[1]):
                    x_out[n][m][p][q] = psum_range[1]

                if psum_record:
                    psum_record_list.append(x_out[n][m][p][q])

```

Conv2d 迴圈次數 $N * M * P * Q * R * S * C$

最內層 3 次 mul, 3 次 add

$$C1 = 4 * 6 * 28 * 28 * 5 * 5 * 1 = 470400$$

$$C3 = 4 * 16 * 10 * 10 * 5 * 5 * 6 = 960000$$

$$C5 = 4 * 120 * 1 * 1 * 5 * 5 * 16 = 192000$$

```

for h in range(H):
    for c in range(C):
        x_out[h][c] = 0
        for w in range(W):
            x_out[h][c] += x[h][w] * weights[c][w]
            # clamp the partial sum to the specified range
            if x_out[h][c] < psum_range[0]:
                x_out[h][c] = psum_range[0]
            elif x_out[h][c] > psum_range[1]:
                x_out[h][c] = psum_range[1]

            if psum_record:
                psum_record_list.append(x_out[h][c])

if weightsBias is not None:
    x_out += weightsBias

```

Linear 迴圈次數 $H * C * W$

$$F6 = 4 * 84 * 120 = 40320$$

$$\text{Output} = 4 * 10 * 84 = 3360$$

最內層 1 次 mul, 1 次 add

$H * C$ 個 elements 要加 bias

	Nmul	Nadd	Bmul	Smul	Badd	Sadd	EW
C1	1411200	1411200	8	64	18	18	115718400
C3	2880000	2880000	8	64	18	18	236160000
C5	576000	576000	8	64	18	18	47232000
F6	40320	40320	8	64	18	18	3306240
Output	3360	3400	8	64	18	18	276240

Overall $E_w = 402,692,880$

(3.1.2)

	Nmul	Nadd	Bmul	Smul	Badd	Sadd	E_w
C1	1411200	1411200	8	64	18	18	115718400
C3	2880000	2880000	8	64	16	16	236160000
C5	576000	576000	8	64	16	16	46080000
F6	40320	40320	8	64	16	16	3225600
Output	3360	3400	8	64	16	16	269440

Overall $E_w = 401,453,440$

