Advanced Arch Wet-HW2

Chris Shakkour, 208157826, christian.s@campus.technion.ac.il

Nadi Najjar, 211610704, nadi.najjar@campus.technion.ac.il

Contents

Question 1 – Getting started with PyTourch

CIFAR-10: 10 classes each with 6000 images. 5000 training and 1000 test per class

Training result:

```
Test: [ 0/79]     Time  0.146 ( 0.146)   Loss 1.7744e+00 (1.7744e+00)
      Acc@1  37.50 ( 37.50)  Acc@5  86.72 ( 86.72)
 * Acc@1 35.710 Acc@5 85.310
```

Part 1

CIFAR-100: 100 classes each with 600 images. 500 training and 100 test per class.

To enable the CIFAR-100 in python conduct the following.

1. Change the label names in the classes string set now we have 100 labels.

```
classes = ("apple", "aquarium_fish", "baby", "bear", "beaver"…
```

2. Changing the output channels on the last fully connected layer from 10 classes to 100 classes.

```
self.fc3 = nn.Linear(84, 100) # *Now we have 100 classes to clasify
```

Training results:

```
Test: [ 0/79]     Time  0.157 ( 0.157)   Loss 4.5903e+00 (4.5903e+00)
      Acc@1   0.78 (  0.78)  Acc@5   3.12 (  3.12)
 * Acc@1 1.400 Acc@5 7.140
```

Part 2

ID1 = 208157826

ID2 = 211610704

Conv1Kernal = Max(ID1[:]) = 8

Conv2Kernel = 9-Min(ID2[:]) = 9

Epochs = Max(7*8, 30) = 56

After 56 Epocs the last epoch reports the following on the test bach.

```
Test: [ 0/79]     Time  0.141 ( 0.141)   Loss 4.6086e+00 (4.6086e+00)
      Acc@1   0.78 (  0.78)  Acc@5   2.34 (  2.34)
 * Acc@1 1.030 Acc@5 4.980
```

Part 3

a. Conv architecture

```
-I- conv1   IC:3        OC:6        Kernel:8x8       stride:1
-I- pool    IC:none,    OC:none,    Kernel:2x2,      stride:2
-I- conv2   IC:6,       OC:16,      Kernel:9x9,      stride:1
-I- fc1     IC:64,      OC:120,     Kernel:none,     stride:none
-I- fc2     IC:120,     OC:84,      Kernel:none,     stride:none
-I- fc3     IC:84,      OC:100,     Kernel:none,     stride:none
```

b. memory footprint for each layer during inference (activations and weights):

the number of weights is displayed below for each layer, the multiplication of each element in the torch.size list. To get the footprint we need to multiply this number with the size of our data representation, in FP32 we have 32 bits for each weight

```
conv1:      Tensor size is: torch.Size([6, 3, 8, 8])
conv2:      Tensor size is: torch.Size([16, 6, 9, 9])
fc1:        Tensor size is: torch.Size([120, 64])
fc2:        Tensor size is: torch.Size([84, 120])
fc3         Tensor size is: torch.Size([100, 84])
```

```
Sum of weights = (6*3*8*8)+(16*6*9*9)+(120*64)+(84*120)+(100*84)
```

```
Sum of weights = 35088
```

Memory footprint = (35088 * 32bits)/8bits = 140,352 Byte

c. memory footprint for each layer during backpropagation (activation and weights):

to calculate the memory footprint for a single batch we do the following, for each weight and bias we need 3 memory elements according to the backpropagation algorithm, and sum this up with the number of activations times 2 times the batch size and we gut the number of memory footprint.

```
Sum of weights =35088
```

```
Sum of biases =fc3(100)+fc2(84)+fc1(120)+conv2(64)+conv1(6*25*25)=4,118
```

```
Sum of activations
=fc3(100)+fc2(84)+fc1(120)+conv2(64)+conv1(6*25*25)=4,118
```

```
Batch size = 128
```

```
Memory footprint = (35088 + 4,118)*3 + 4,118*2*128 = 1,171,826 elements
```

```
Memory footprint =1,171,826*4 =4,687,304 bytes =~4,577 Kbyte =~4 MByte
```

d. computations for each layer during inference:

```
-I- conv1   IC:3        OC:6        Kernel:8x8        stride:1
-I- pool    IC:none,    OC:none,    Kernel:2x2,       stride:2
-I- conv2   IC:6,       OC:16,      Kernel:9x9,       stride:1
-I- fc1     IC:64,      OC:120,     Kernel:none,      stride:none
-I- fc2     IC:120,     OC:84,      Kernel:none,      stride:none
-I- fc3     IC:84,      OC:100,     Kernel:none,      stride:none
```

MAC operations per convolutional layer:

C_OUT * ((K * K) * C_IN * (H_OUT * W_OUT))

MAC operations for a Pool layer:

H_OUT * W_OUT * (2 * 2) * C_IN

MAC operations per Fully connected layer:

C_IN * C_OUT

Conv1: 6*8*8*3*25*25 = 720,000

Pool : 13*13*2*2*6 = 4,056

Conv2 : 6*9*9*16*2*2 = 31,104

Fc1 : 64*120 = 7680

Fc2 : 120*84 = 10,080

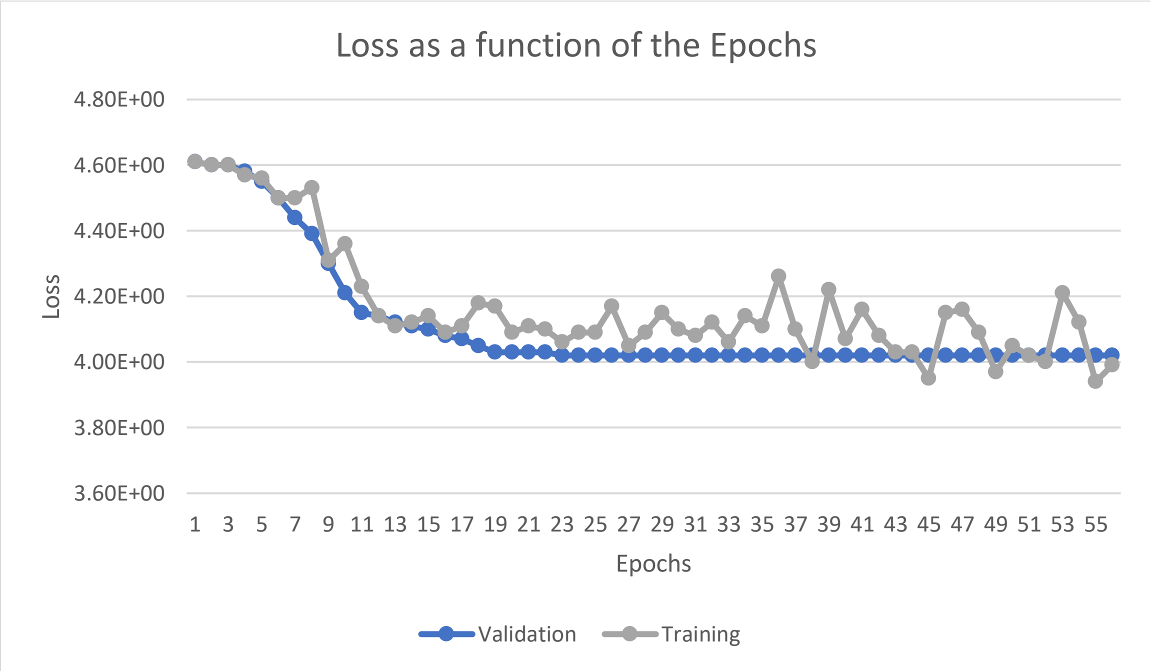
Fc3 : 84*100 = 8,400

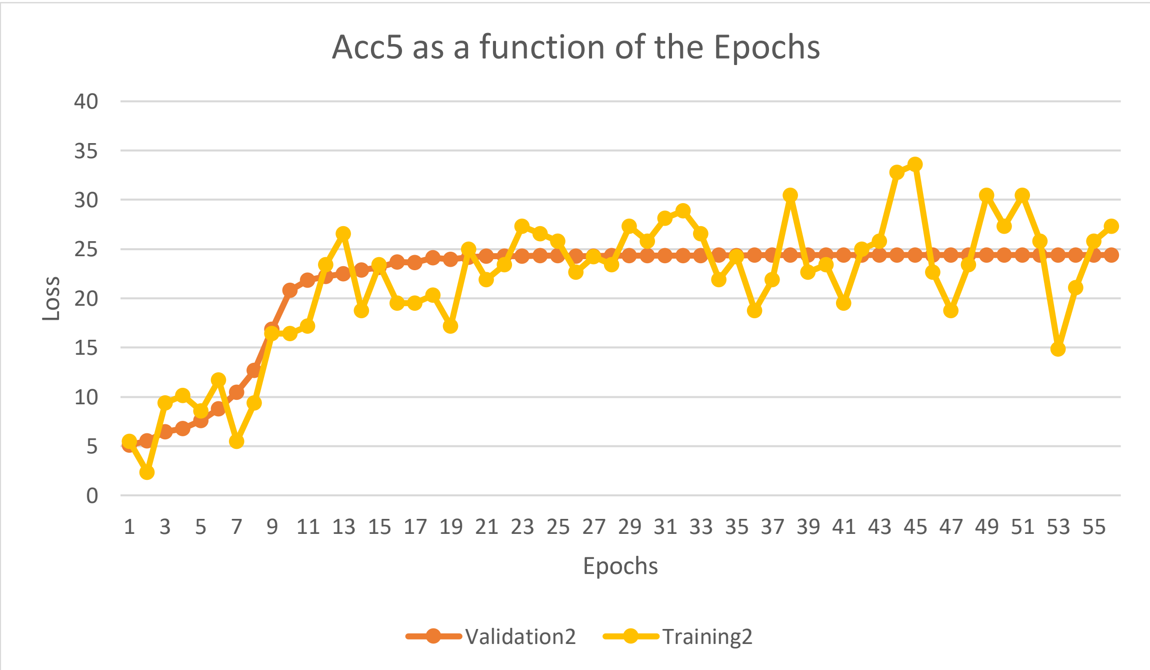
Overall, MAC operations= 781,320

e. training and validation errors as a function of epochs:

Epoch	Validation Loss	Validation Acc5	Training Loss	Training Acc5
1	4.61E+00	5.11	4.61E+00	5.47
2	4.60E+00	5.51	4.60E+00	2.34
3	4.60E+00	6.44	4.60E+00	9.38
4	4.58E+00	6.78	4.57E+00	10.16
5	4.55E+00	7.59	4.56E+00	8.59
6	4.50E+00	8.77	4.50E+00	11.72
7	4.44E+00	10.47	4.50E+00	5.47
8	4.39E+00	12.7	4.53E+00	9.38
9	4.30E+00	16.88	4.31E+00	16.41
10	4.21E+00	20.82	4.36E+00	16.41
11	4.15E+00	21.82	4.23E+00	17.19
12	4.14E+00	22.23	4.14E+00	23.44
13	4.12E+00	22.5	4.11E+00	26.56
14	4.11E+00	22.87	4.12E+00	18.75
15	4.10E+00	23.12	4.14E+00	23.44
16	4.08E+00	23.67	4.09E+00	19.53
17	4.07E+00	23.64	4.11E+00	19.53
18	4.05E+00	24.1	4.18E+00	20.31
19	4.03E+00	23.94	4.17E+00	17.19
20	4.03E+00	24.19	4.09E+00	25
21	4.03E+00	24.26	4.11E+00	21.88
22	4.03E+00	24.27	4.10E+00	23.44
23	4.02E+00	24.26	4.06E+00	27.34
24	4.02E+00	24.35	4.09E+00	26.56
25	4.02E+00	24.31	4.09E+00	25.78
26	4.02E+00	24.26	4.17E+00	22.66
27	4.02E+00	24.26	4.05E+00	24.22
28	4.02E+00	24.32	4.09E+00	23.44
29	4.02E+00	24.34	4.15E+00	27.34
30	4.02E+00	24.34	4.10E+00	25.78
31	4.02E+00	24.36	4.08E+00	28.12
32	4.02E+00	24.36	4.12E+00	28.91
33	4.02E+00	24.36	4.06E+00	26.56
34	4.02E+00	24.37	4.14E+00	21.88
35	4.02E+00	24.36	4.11E+00	24.22
36	4.02E+00	24.38	4.26E+00	18.75
37	4.02E+00	24.38	4.10E+00	21.88
38	4.02E+00	24.38	4.00E+00	30.47
39	4.02E+00	24.39	4.22E+00	22.66
40	4.02E+00	24.4	4.07E+00	23.44
41	4.02E+00	24.4	4.16E+00	19.53
42	4.02E+00	24.4	4.08E+00	25
43	4.02E+00	24.4	4.03E+00	25.78
44	4.02E+00	24.4	4.03E+00	32.81
45	4.02E+00	24.4	3.95E+00	33.59
46	4.02E+00	24.4	4.15E+00	22.66
47	4.02E+00	24.4	4.16E+00	18.75

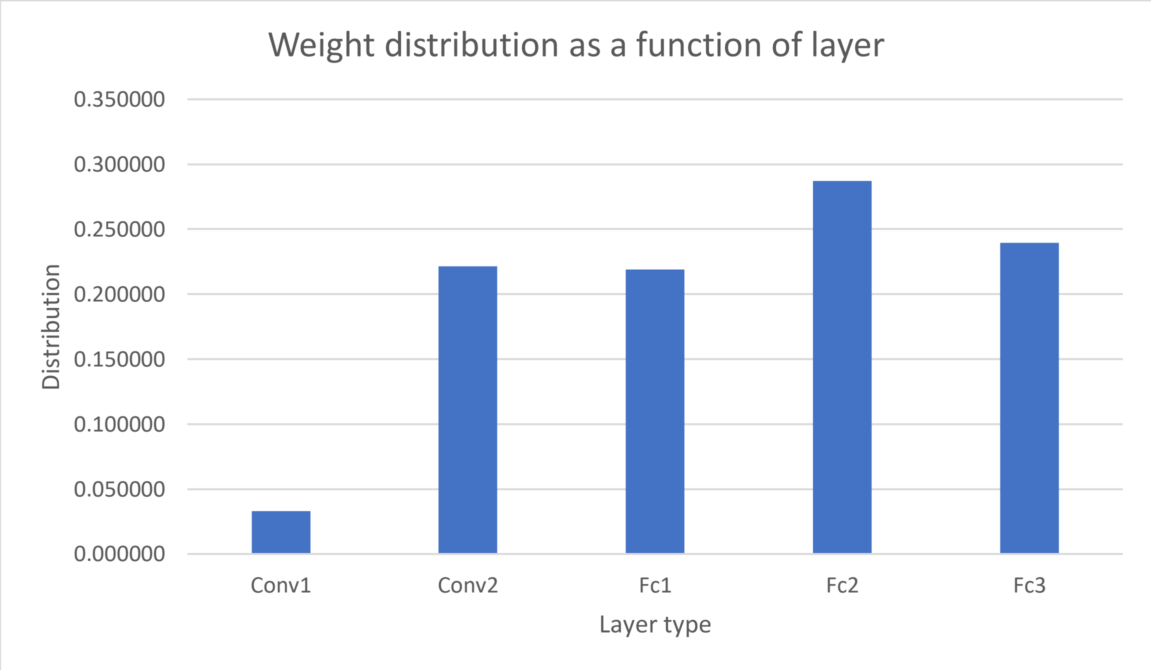
48	4.02E+00	24.4	4.09E+00	23.44
49	4.02E+00	24.4	3.97E+00	30.47
50	4.02E+00	24.4	4.05E+00	27.34
51	4.02E+00	24.4	4.02E+00	30.47
52	4.02E+00	24.4	4.00E+00	25.78
53	4.02E+00	24.4	4.21E+00	14.84
54	4.02E+00	24.4	4.12E+00	21.09
55	4.02E+00	24.4	3.94E+00	25.78
56	4.02E+00	24.4	3.99E+00	27.34





Part 4

Layer	Structure	Weights count
Conv1	[6, 3, 8, 8]	1,152
Conv2	[16, 6, 9, 9]	7,776
Fc1	[120, 64]	7,680
Fc2	[84, 120]	10,080
Fc3	[100, 84]	8,400



Question 2 – Quantization

Part 1

Following accuracy without any quantization:

```
Test: [ 0/79]     Time  0.168 ( 0.168)    Loss 4.0791e+00 (4.0791e+00)
     Acc@1   6.25 (  6.25)  Acc@5  25.78 ( 25.78)
 * Acc@1 6.820 Acc@5 22.360
```

With Quantization pre layer we get the following 16 iterations:

```
Quantizing weights to 1-bit
Test: [ 0/79]     Time  0.416 ( 0.416)    Loss 8.7914e+03 (8.7914e+03)
     Acc@1   7.81 (  7.81)  Acc@5  26.56 ( 26.56)
 * Acc@1 8.210 Acc@5 26.020
```

```
Quantizing weights to 2-bit
Test: [ 0/79]     Time  1.073 ( 1.073)    Loss 5.8816e+01 (5.8816e+01)
     Acc@1   0.78 (  0.78)  Acc@5   5.47 (  5.47)
 * Acc@1 0.460 Acc@5 3.120
```

```
Quantizing weights to 3-bit
Test: [ 0/79]     Time  0.173 ( 0.173)    Loss 4.6033e+00 (4.6033e+00)
     Acc@1   0.78 (  0.78)  Acc@5   7.03 (  7.03)
 * Acc@1 1.000 Acc@5 5.000
```

```
Quantizing weights to 4-bit
Test: [ 0/79]     Time  0.176 ( 0.176)    Loss 8.5914e+05 (8.5914e+05)
     Acc@1   7.81 (  7.81)  Acc@5  26.56 ( 26.56)
 * Acc@1 8.220 Acc@5 25.920
```

```
Quantizing weights to 5-bit
Test: [ 0/79]     Time  0.232 ( 0.232)    Loss 2.1380e+06 (2.1380e+06)
     Acc@1   7.81 (  7.81)  Acc@5  26.56 ( 26.56)
 * Acc@1 8.220 Acc@5 25.890
```

```
Quantizing weights to 6-bit
Test: [ 0/79]     Time  0.891 ( 0.891)    Loss 6.6788e+04 (6.6788e+04)
     Acc@1   7.81 (  7.81)  Acc@5  26.56 ( 26.56)
 * Acc@1 8.200 Acc@5 25.930
```

```
Quantizing weights to 7-bit
Test: [ 0/79]     Time  0.400 ( 0.400)    Loss 2.8150e+05 (2.8150e+05)
     Acc@1   7.81 (  7.81)  Acc@5  26.56 ( 26.56)
 * Acc@1 8.200 Acc@5 25.940
```

```
Quantizing weights to 8-bit
Test: [ 0/79]     Time  0.659 ( 0.659)    Loss 1.6237e+07 (1.6237e+07)
     Acc@1   7.81 (  7.81)  Acc@5  26.56 ( 26.56)
 * Acc@1 8.210 Acc@5 25.910
```

```
Quantizing weights to 9-bit
Test: [ 0/79]     Time  0.988 ( 0.988)    Loss 2.7498e+07 (2.7498e+07)
     Acc@1   7.81 (  7.81)  Acc@5  26.56 ( 26.56)
 * Acc@1 8.210 Acc@5 25.920
```

```
Quantizing weights to 10-bit
Test: [ 0/79]     Time  0.231 ( 0.231)    Loss 4.6212e+06 (4.6212e+06)
      Acc@1   7.81 (  7.81)  Acc@5  26.56 ( 26.56)
 * Acc@1 8.220 Acc@5 25.880
```

```
Quantizing weights to 11-bit
Test: [ 0/79]     Time  0.173 ( 0.173)    Loss 9.0100e+06 (9.0100e+06)
      Acc@1   7.81 (  7.81)  Acc@5  26.56 ( 26.56)
 * Acc@1 8.220 Acc@5 25.890
```

```
Quantizing weights to 12-bit
Test: [ 0/79]     Time  0.453 ( 0.453)    Loss 1.0210e+08 (1.0210e+08)
      Acc@1   7.81 (  7.81)  Acc@5  26.56 ( 26.56)
 * Acc@1 8.210 Acc@5 25.890
```

```
Quantizing weights to 13-bit
Test: [ 0/79]     Time  0.297 ( 0.297)    Loss 1.4790e+08 (1.4790e+08)
      Acc@1   7.81 (  7.81)  Acc@5  26.56 ( 26.56)
 * Acc@1 8.210 Acc@5 25.870
```

```
Quantizing weights to 14-bit
Test: [ 0/79]     Time  0.174 ( 0.174)    Loss 4.4286e+07 (4.4286e+07)
      Acc@1   7.81 (  7.81)  Acc@5  26.56 ( 26.56)
 * Acc@1 8.210 Acc@5 25.920
```

```
Quantizing weights to 15-bit
Test: [ 0/79]     Time  0.171 ( 0.171)    Loss 6.8425e+07 (6.8425e+07)
      Acc@1   7.81 (  7.81)  Acc@5  26.56 ( 26.56)
 * Acc@1 8.210 Acc@5 25.910
```

```
Quantizing weights to 16-bit
Test: [ 0/79]     Time  0.169 ( 0.169)    Loss 3.9046e+08 (3.9046e+08)
      Acc@1   7.81 (  7.81)  Acc@5  26.56 ( 26.56)
 * Acc@1 8.210 Acc@5 25.890
```

Part 2

When the quantization factor is done for each OC in a layer:

```
Quantizing weights to 1-bit
Test: [ 0/79]     Time  0.172 ( 0.172)    Loss 3.5386e+04 (3.5386e+04)
      Acc@1   6.25 (  6.25)  Acc@5  26.56 ( 26.56)
 * Acc@1 6.620 Acc@5 22.390
```

```
Quantizing weights to 2-bit
Test: [ 0/79]     Time  0.169 ( 0.169)    Loss 3.1161e+02 (3.1161e+02)
      Acc@1   1.56 (  1.56)  Acc@5   2.34 (  2.34)
 * Acc@1 0.750 Acc@5 3.800
```

```
Quantizing weights to 3-bit
Test: [ 0/79]     Time  0.682 ( 0.682)    Loss 4.6033e+00 (4.6033e+00)
      Acc@1   0.78 (  0.78)  Acc@5   7.03 (  7.03)
 * Acc@1 1.000 Acc@5 5.000
```

```
Quantizing weights to 4-bit
Test: [ 0/79]     Time  0.789 ( 0.789)    Loss 3.4556e+06 (3.4556e+06)
      Acc@1   6.25 (  6.25)  Acc@5  25.78 ( 25.78)
 * Acc@1 6.550 Acc@5 22.330
```

```
Quantizing weights to 5-bit
Test: [ 0/79]     Time  0.172 ( 0.172)    Loss 8.5986e+06 (8.5986e+06)
      Acc@1   6.25 (  6.25)  Acc@5  25.78 ( 25.78)
 * Acc@1 6.550 Acc@5 22.340
```

```
Quantizing weights to 6-bit
Test: [ 0/79]     Time  0.392 ( 0.392)    Loss 2.6871e+05 (2.6871e+05)
      Acc@1   6.25 (  6.25)  Acc@5  26.56 ( 26.56)
 * Acc@1 6.600 Acc@5 22.370
```

```
Quantizing weights to 7-bit
Test: [ 0/79]     Time  0.172 ( 0.172)    Loss 1.1323e+06 (1.1323e+06)
      Acc@1   6.25 (  6.25)  Acc@5  26.56 ( 26.56)
 * Acc@1 6.570 Acc@5 22.330
```

```
Quantizing weights to 8-bit
Test: [ 0/79]     Time  0.162 ( 0.162)    Loss 6.5295e+07 (6.5295e+07)
      Acc@1   6.25 (  6.25)  Acc@5  25.78 ( 25.78)
 * Acc@1 6.530 Acc@5 22.320
```

```
Quantizing weights to 9-bit
Test: [ 0/79]     Time  0.163 ( 0.163)    Loss 1.1058e+08 (1.1058e+08)
      Acc@1   6.25 (  6.25)  Acc@5  25.78 ( 25.78)
 * Acc@1 6.530 Acc@5 22.320
```

```
Quantizing weights to 10-bit
Test: [ 0/79]     Time  0.204 ( 0.204)    Loss 1.8585e+07 (1.8585e+07)
      Acc@1   6.25 (  6.25)  Acc@5  25.78 ( 25.78)
 * Acc@1 6.530 Acc@5 22.340
```

```
Quantizing weights to 11-bit
Test: [ 0/79]     Time  0.160 ( 0.160)    Loss 3.6234e+07 (3.6234e+07)
      Acc@1   6.25 (  6.25)  Acc@5  25.78 ( 25.78)
 * Acc@1 6.530 Acc@5 22.330
```

```
Quantizing weights to 12-bit
Test: [ 0/79]     Time  0.190 ( 0.190)    Loss 4.1056e+08 (4.1056e+08)
      Acc@1   6.25 (  6.25)  Acc@5  25.78 ( 25.78)
 * Acc@1 6.520 Acc@5 22.320
```

```
Quantizing weights to 13-bit
Test: [ 0/79]     Time  0.168 ( 0.168)    Loss 5.9471e+08 (5.9471e+08)
      Acc@1   6.25 (  6.25)  Acc@5  25.78 ( 25.78)
 * Acc@1 6.520 Acc@5 22.320
```

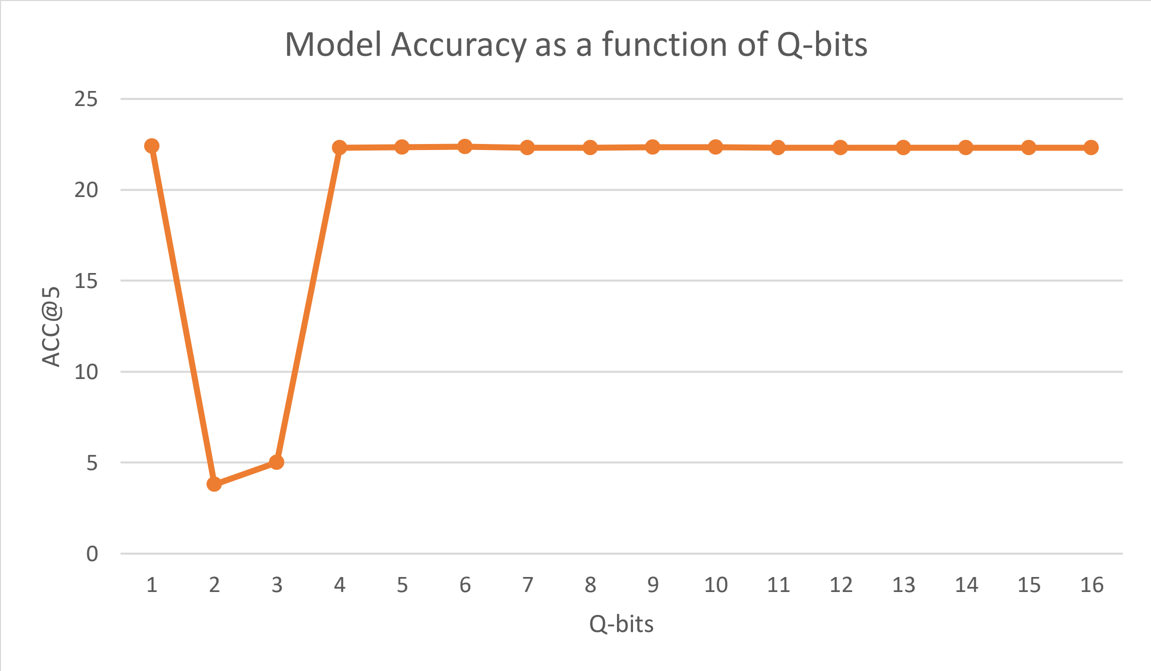
```
Quantizing weights to 14-bit
Test: [ 0/79]     Time  0.171 ( 0.171)    Loss 1.7809e+08 (1.7809e+08)
      Acc@1   6.25 (  6.25)  Acc@5  25.78 ( 25.78)
 * Acc@1 6.530 Acc@5 22.320
```

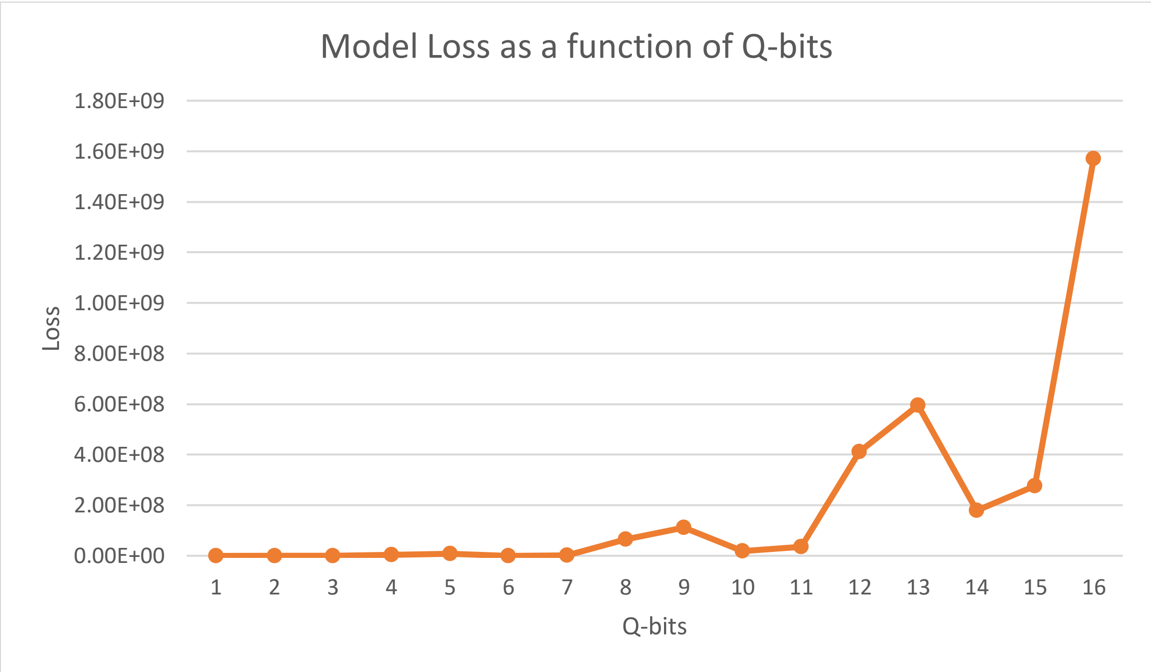
```
Quantizing weights to 15-bit
Test: [ 0/79]     Time  0.161 ( 0.161)    Loss 2.7515e+08 (2.7515e+08)
      Acc@1   6.25 (  6.25)  Acc@5  25.78 ( 25.78)
 * Acc@1 6.530 Acc@5 22.320
```

```
Quantizing weights to 16-bit
Test: [ 0/79]     Time  0.155 ( 0.155)    Loss 1.5700e+09 (1.5700e+09)
      Acc@1   6.25 (  6.25)  Acc@5  25.78 ( 25.78)
 * Acc@1 6.520 Acc@5 22.310
```

Num of Q_BITS	ACC@5	Loss
1	22.390	3.5386e+04
2	3.800	3.1161e+02
3	5.000	4.6033e+00
4	22.330	3.4556e+06
5	22.340	8.5986e+06
6	22.370	2.6871e+05
7	22.330	1.1323e+06
8	22.320	6.5295e+07
9	22.340	1.1058e+08
10	22.340	1.8585e+07
11	22.330	3.6234e+07
12	22.320	4.1056e+08
13	22.320	5.9471e+08
14	22.320	1.7809e+08
15	22.320	2.7515e+08
16	22.310	1.5700e+09

by theory we expect that the accuracy degrades the more we quantize the original weights meaning the less bit representation each weight has. But for the loss we expect a rise in the loss function since now each weight is changed since the training hence this change will be summed up along the way when forward propagating hence leading to bigger differences from the expected labels.





Part 3

The memory footprint in inference when Q_BITS={2, 4, 8} compared to FP32 is:

```
Sum of weights = 35088
```

Memory footprint(2bits) = (35088 * 32bits)/8bits = 140,352 Byte

Memory footprint(2bits) = (35088 * 2bits)/8bits = 8,772 Byte

Memory footprint(4bits) = (35088 * 4bits)/8bits = 17,544 Byte

Memory footprint(8bits) = (35088 * 8bits)/8bits = 35,088 Byte

Question 3 – Pruning

Following results before pruning for reference.

```
Test: [ 0/79]     Time  0.160 ( 0.160)    Loss 4.0176e+00 (4.0176e+00)
     Acc@1  10.94 ( 10.94)  Acc@5  29.69 ( 29.69)
 * Acc@1 7.630 Acc@5 24.400
```

Part 1

The implementation can be found in the notebook, the algorithm is as follows:

1. For any layer we put all the weights in their absolute value in a list, this is done by iterating over the tensor and appending the float values into a list
2. Then we sort this list of floats.
3. Then we find the threshold for which any weight is smaller than this number is in the 20% percent weights closest to zero, this is found by taking the number in the sorted list that is in the 20% percent index in the list meaning threshold = sorted_weight_list[list_length/5]
4. Then we iterate again over the tensor zeroing every element smaller than the threshold.

For 20% we get the following results:

```
Test: [ 0/79]     Time  0.154 ( 0.154)    Loss 4.0087e+00 (4.0087e+00)
     Acc@1   9.38 (  9.38)  Acc@5  28.91 ( 28.91)
 * Acc@1 7.000 Acc@5 23.850
```

Part 2

Pruning 10% of the weights:

```
Test: [ 0/79]     Time  0.157 ( 0.157)    Loss 4.0165e+00 (4.0165e+00)
     Acc@1  10.16 ( 10.16)  Acc@5  30.47 ( 30.47)
 * Acc@1 7.670 Acc@5 24.400
```

Pruning 20% of the weights:

```
Test: [ 0/79]     Time  0.154 ( 0.154)    Loss 4.0087e+00 (4.0087e+00)
     Acc@1   9.38 (  9.38)  Acc@5  28.91 ( 28.91)
 * Acc@1 7.000 Acc@5 23.850
```

Pruning 30% of the weights:

```
Test: [ 0/79]     Time  0.147 ( 0.147)    Loss 4.0147e+00 (4.0147e+00)
     Acc@1  10.16 ( 10.16)  Acc@5  27.34 ( 27.34)
 * Acc@1 6.150 Acc@5 22.280
```

Pruning 40% of the weights:

```
Test: [ 0/79]     Time  0.136 ( 0.136)    Loss 4.0700e+00 (4.0700e+00)
     Acc@1  10.16 ( 10.16)  Acc@5  28.91 ( 28.91)
 * Acc@1 5.480 Acc@5 20.230
```

Pruning 50% of the weights:

```
Test: [ 0/79]     Time  0.139 ( 0.139)    Loss 4.2187e+00 (4.2187e+00)
     Acc@1   3.91 (  3.91)  Acc@5  22.66 ( 22.66)
 * Acc@1 4.550 Acc@5 18.480
```

Pruning 60% of the weights:

```
Test: [ 0/79]     Time  0.138 ( 0.138)    Loss 4.4327e+00 (4.4327e+00)
     Acc@1   4.69 (  4.69)  Acc@5  14.06 ( 14.06)
 * Acc@1 3.970 Acc@5 14.280
```

Pruning 70% of the weights:

```
Test: [ 0/79]     Time  0.149 ( 0.149)    Loss 4.6275e+00 (4.6275e+00)
     Acc@1   2.34 (  2.34)  Acc@5  11.72 ( 11.72)
 * Acc@1 3.430 Acc@5 12.800
```

Pruning 80% of the weights:

```
Test: [ 0/79]     Time  0.160 ( 0.160)    Loss 4.5677e+00 (4.5677e+00)
     Acc@1   1.56 (  1.56)  Acc@5   8.59 (  8.59)
 * Acc@1 2.740 Acc@5 12.300
```

Pruning 90% of the weights:

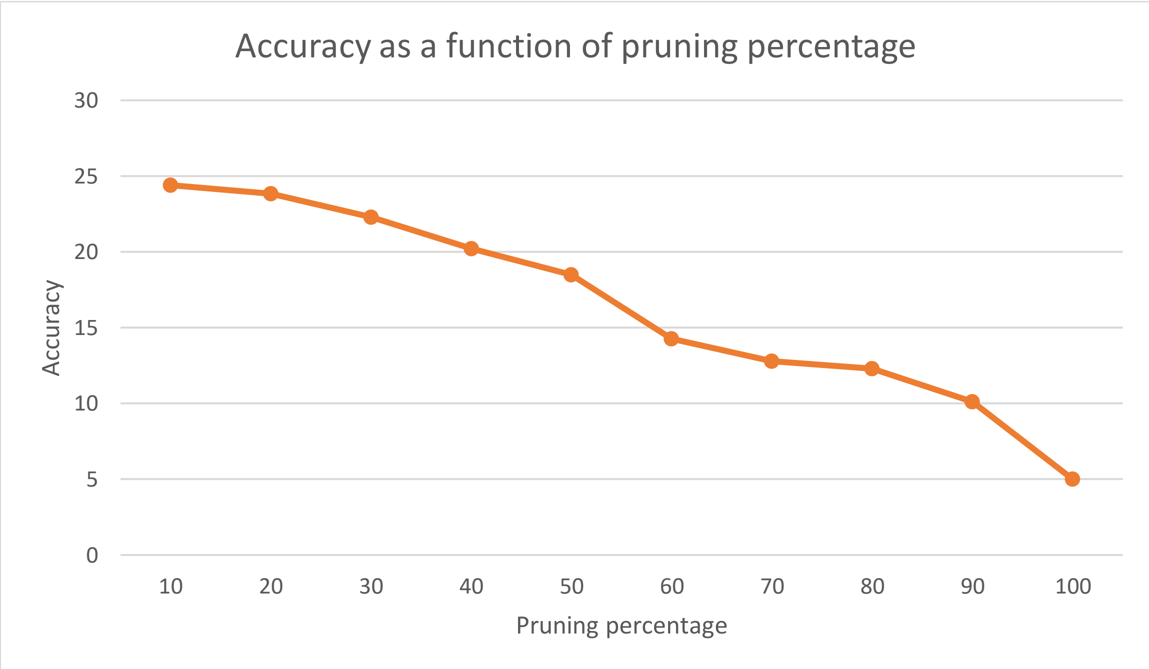
```
Test: [ 0/79]     Time  0.137 ( 0.137)    Loss 4.5659e+00 (4.5659e+00)
     Acc@1   3.91 (  3.91)  Acc@5  14.06 ( 14.06)
 * Acc@1 2.980 Acc@5 10.110
```

Pruning 100% of the weights (completely Zero weights tensors):

```
Test: [ 0/79]     Time  0.165 ( 0.165)    Loss 4.6126e+00 (4.6126e+00)
     Acc@1   0.78 (  0.78)  Acc@5   6.25 (  6.25)
 * Acc@1 1.000 Acc@5 5.000
```

Following results table:

Pruning percentage	Accuracy for K=5
10	24.400
20	23.850
30	22.280
40	20.230
50	18.480
60	14.280
70	12.800
80	12.300
90	10.110
100	5.000



The above plot is expected since the more we zero out weights the more we defect the trained model that we worked hard to train at first place, one thing that we can conclude is that for low pruning percentage like 10-20% we didn't see a major degradation in the model accuracy.

Part 3

For this we need to calculate the overall MAC operations in a single feed forward loop.

```
-I- conv1   IC:3         OC:6         Kernel:8x8          stride:1
-I- pool    IC:none,     OC:none,     Kernel:2x2,         stride:2
-I- conv2   IC:6,        OC:16,       Kernel:9x9,         stride:1
-I- fc1     IC:64,       OC:120,      Kernel:none,        stride:none
-I- fc2     IC:120,      OC:84,       Kernel:none,        stride:none
-I- fc3     IC:84,       OC:100,      Kernel:none,        stride:none
```

MAC operations per convolutional layer:

C_OUT * ((K * K) * C_IN * (H_OUT * W_OUT))

MAC operations for a Pool layer:

H_OUT * W_OUT * (2 * 2) * C_IN

MAC operations per Fully connected layer:

C_IN * C_OUT

Conv1: 6*8*8*3*25*25 = 720,000

Pool : 13*13*2*2*6 = 4,056

Conv2 : 6*9*9*16*2*2 = 31,104

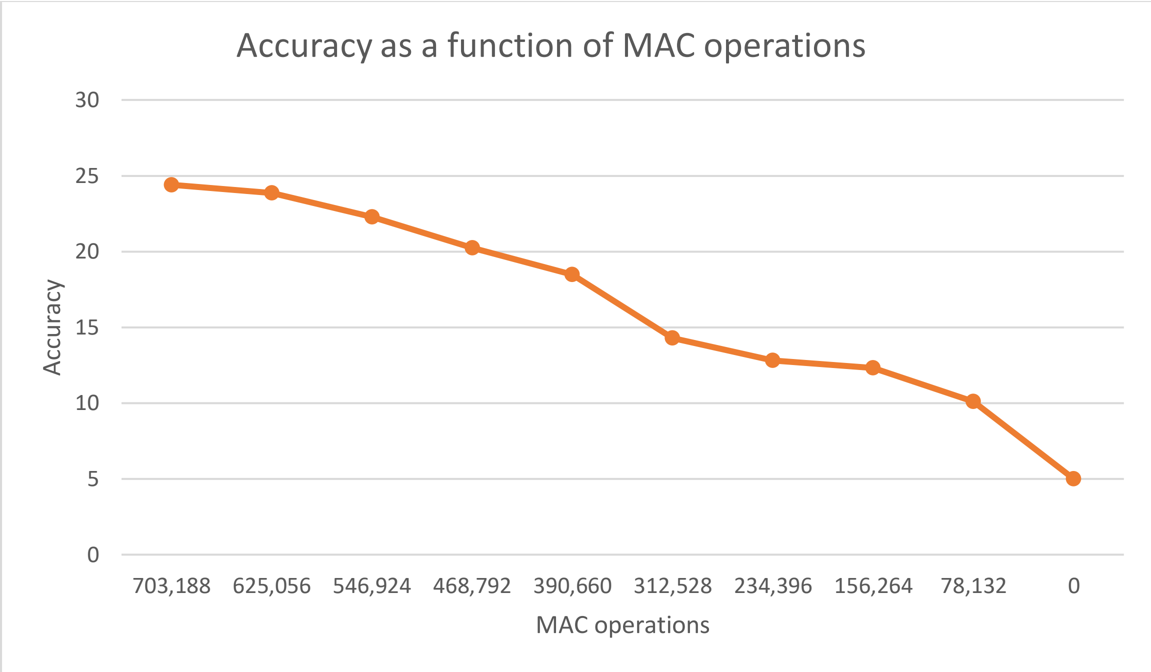
Fc1 : 64*120 = 7680

Fc2 : 120*84 = 10,080

Fc3 : 84*100 = 8,400

Overall, MAC operations= 781,320

Pruning percentage	MAC operations	MAC Result	Accuracy for K=5
10	781,320 * 0.9	703,188	24.400
20	781,320 * 0.8	625,056	23.850
30	781,320 * 0.7	546,924	22.280
40	781,320 * 0.6	468,792	20.230
50	781,320 * 0.5	390,660	18.480
60	781,320 * 0.4	312,528	14.280
70	781,320 * 0.3	234,396	12.800
80	781,320 * 0.2	156,264	12.300
90	781,320 * 0.1	78,132	10.110
100	781,320 * 0	0	5.000



Accuracy as a function of MAC operations
Accuracy
30
25
20
15
10
5
0
703,188
625,056
546,924
468,792
390,660
312,528
234,396
156,264
78,132
0
MAC operations

Part 4

Following part 3 we calculate the overall MAC operation in a single forward pass by multiplying the percentage of non-zero weights in the overall MAC operation of the layer

Overall, MAC operations = 720,000 + 4,056 + 31,104 + 7,680 + 10,080 + 8,400

- Pruning conv1 layer with a range of percentage:

```
Pruning weights with 10.0% of layer conv1
Test: [ 0/79]     Time  0.165 ( 0.165)    Loss 4.0374e+00 (4.0374e+00)
      Acc@1  10.94 ( 10.94)  Acc@5  29.69 ( 29.69)
 * Acc@1 7.570 Acc@5 24.380
```

```
Pruning weights with 20.0% of layer conv1
Test: [ 0/79]     Time  0.155 ( 0.155)    Loss 4.0374e+00 (4.0374e+00)
      Acc@1  10.94 ( 10.94)  Acc@5  29.69 ( 29.69)
 * Acc@1 7.570 Acc@5 24.380
```

```
Pruning weights with 30.0% of layer conv1
Test: [ 0/79]     Time  0.151 ( 0.151)    Loss 4.0374e+00 (4.0374e+00)
      Acc@1  10.94 ( 10.94)  Acc@5  29.69 ( 29.69)
 * Acc@1 7.570 Acc@5 24.380
```

```
Pruning weights with 40.0% of layer conv1
Test: [ 0/79]     Time  0.148 ( 0.148)    Loss 4.0374e+00 (4.0374e+00)
      Acc@1  10.94 ( 10.94)  Acc@5  29.69 ( 29.69)
 * Acc@1 7.570 Acc@5 24.380
```

```
Pruning weights with 50.0% of layer conv1
Test: [ 0/79]     Time  0.142 ( 0.142)    Loss 4.0374e+00 (4.0374e+00)
      Acc@1  10.94 ( 10.94)  Acc@5  29.69 ( 29.69)
 * Acc@1 7.570 Acc@5 24.380
```

```
Pruning weights with 60.0% of layer conv1
Test: [ 0/79]     Time  0.156 ( 0.156)    Loss 4.0396e+00 (4.0396e+00)
      Acc@1  10.94 ( 10.94)  Acc@5  32.03 ( 32.03)
 * Acc@1 7.660 Acc@5 24.360
```

```
Pruning weights with 70.0% of layer conv1
Test: [ 0/79]     Time  0.145 ( 0.145)    Loss 4.0567e+00 (4.0567e+00)
      Acc@1  11.72 ( 11.72)  Acc@5  30.47 ( 30.47)
 * Acc@1 7.520 Acc@5 24.310
```

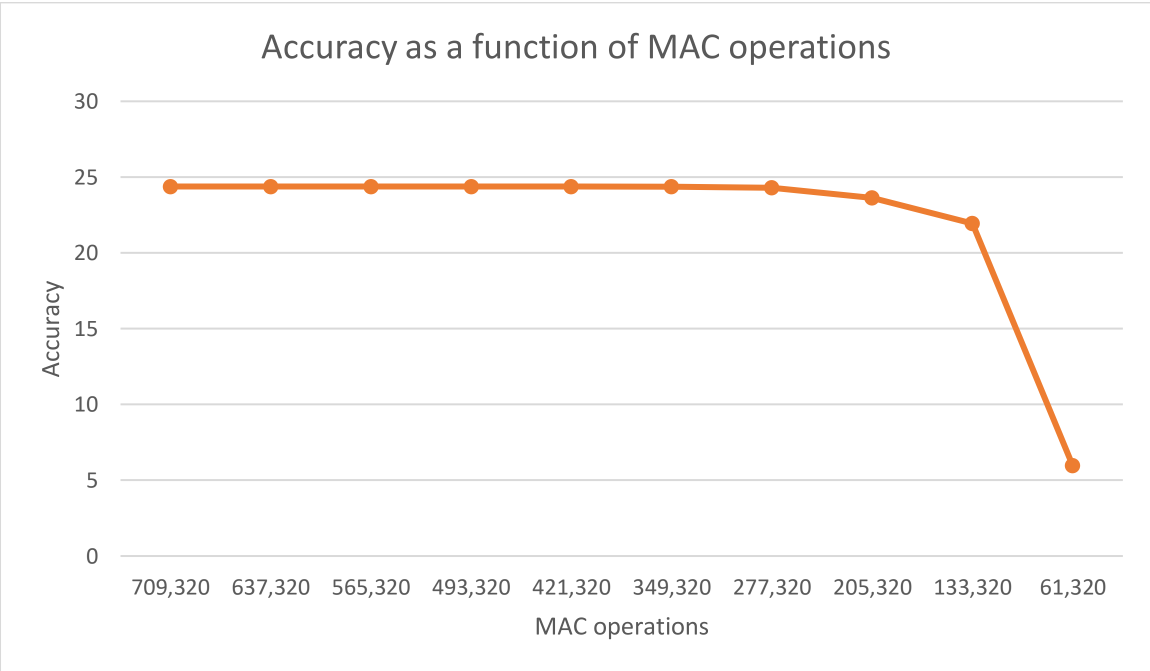
```
Pruning weights with 80.0% of layer conv1
Test: [ 0/79]     Time  0.137 ( 0.137)    Loss 4.1359e+00 (4.1359e+00)
      Acc@1  12.50 ( 12.50)  Acc@5  24.22 ( 24.22)
 * Acc@1 6.920 Acc@5 23.630
```

```
Pruning weights with 90.0% of layer conv1
Test: [ 0/79]     Time  0.153 ( 0.153)    Loss 4.2440e+00 (4.2440e+00)
      Acc@1  10.94 ( 10.94)  Acc@5  24.22 ( 24.22)
 * Acc@1 5.840 Acc@5 21.930
```

```
Pruning weights with 100.0% of layer conv1
Test: [ 0/79]     Time  0.145 ( 0.145)    Loss 4.6185e+00 (4.6185e+00)
      Acc@1   1.56 (  1.56)  Acc@5   7.03 (  7.03)
 * Acc@1 1.000 Acc@5 5.960
```

Overall, MAC operations = 720,000*(1-Percentage) + 4,056 + 31,104 + 7,680 + 10,080 + 8,400

Pruning percentage	MAC operations	Accuracy for K=5
10	709,320	24.380
20	637,320	24.380
30	565,320	24.380
40	493,320	24.380
50	421,320	24.380
60	349,320	24.360
70	277,320	24.310
80	205,320	23.630
90	133,320	21.930
100	61,320	5.960



- Pruning conv2 layer with a range of percentage:

```
Pruning weights with 10.0% of layer conv2
Test: [ 0/79]	Time  0.147 ( 0.147)	Loss 4.0385e+00 (4.0385e+00)
	Acc@1  10.94 ( 10.94)	Acc@5  28.91 ( 28.91)
 * Acc@1 7.580 Acc@5 24.390
```

```
Pruning weights with 20.0% of layer conv2
Test: [ 0/79]	Time  0.170 ( 0.170)	Loss 4.0368e+00 (4.0368e+00)
	Acc@1  11.72 ( 11.72)	Acc@5  28.12 ( 28.12)
 * Acc@1 7.590 Acc@5 24.380
```

```
Pruning weights with 30.0% of layer conv2
Test: [ 0/79]	Time  0.162 ( 0.162)	Loss 4.0318e+00 (4.0318e+00)
	Acc@1  10.94 ( 10.94)	Acc@5  27.34 ( 27.34)
 * Acc@1 7.770 Acc@5 24.280
```

```
Pruning weights with 40.0% of layer conv2
Test: [ 0/79]	Time  0.149 ( 0.149)	Loss 4.0279e+00 (4.0279e+00)
	Acc@1  11.72 ( 11.72)	Acc@5  28.91 ( 28.91)
 * Acc@1 7.620 Acc@5 24.260
```

```
Pruning weights with 50.0% of layer conv2
Test: [ 0/79]	Time  0.155 ( 0.155)	Loss 4.0385e+00 (4.0385e+00)
	Acc@1  10.16 ( 10.16)	Acc@5  28.91 ( 28.91)
 * Acc@1 7.520 Acc@5 24.170
```

```
Pruning weights with 60.0% of layer conv2
Test: [ 0/79]	Time  0.151 ( 0.151)	Loss 4.0394e+00 (4.0394e+00)
	Acc@1  10.16 ( 10.16)	Acc@5  31.25 ( 31.25)
 * Acc@1 7.540 Acc@5 24.230
```

```
Pruning weights with 70.0% of layer conv2
Test: [ 0/79]	Time  0.147 ( 0.147)	Loss 4.0449e+00 (4.0449e+00)
	Acc@1  10.16 ( 10.16)	Acc@5  30.47 ( 30.47)
 * Acc@1 7.550 Acc@5 24.230
```

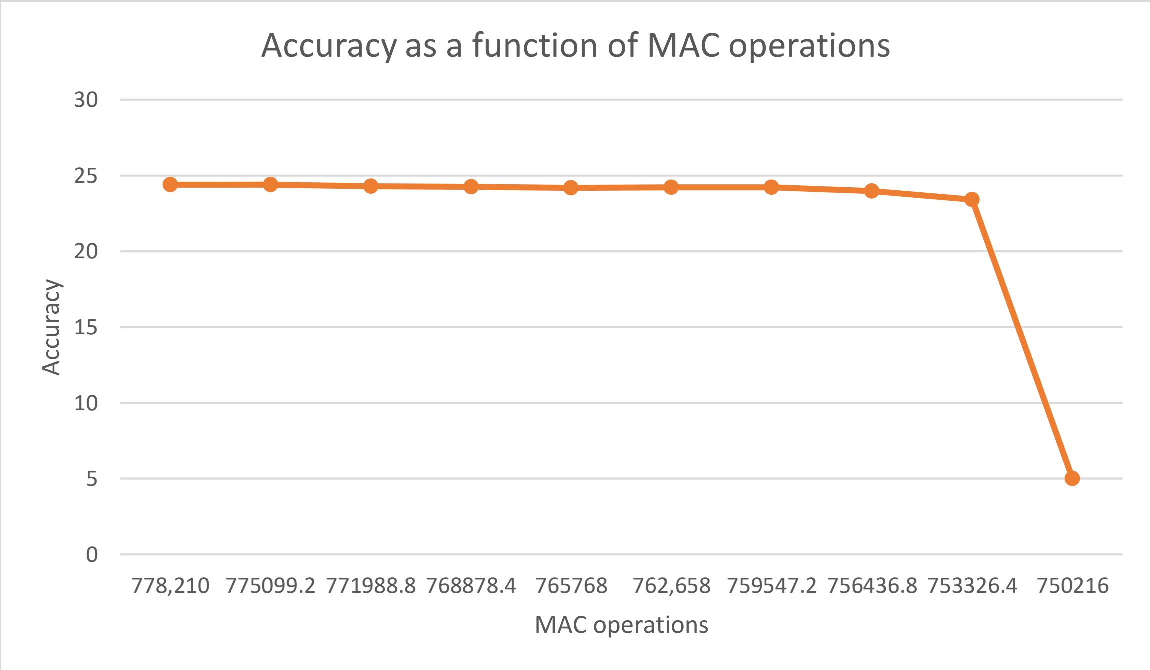
```
Pruning weights with 80.0% of layer conv2
Test: [ 0/79]	Time  0.156 ( 0.156)	Loss 4.0573e+00 (4.0573e+00)
	Acc@1   8.59 (  8.59)	Acc@5  28.91 ( 28.91)
 * Acc@1 7.300 Acc@5 23.960
```

```
Pruning weights with 90.0% of layer conv2
Test: [ 0/79]	Time  0.157 ( 0.157)	Loss 4.1085e+00 (4.1085e+00)
	Acc@1   8.59 (  8.59)	Acc@5  28.91 ( 28.91)
 * Acc@1 6.940 Acc@5 23.400
```

```
Pruning weights with 100.0% of layer conv2
Test: [ 0/79]	Time  0.152 ( 0.152)	Loss 4.6192e+00 (4.6192e+00)
	Acc@1   1.56 (  1.56)	Acc@5   8.59 (  8.59)
 * Acc@1 0.840 Acc@5 5.000
```

Overall, MAC operations = 720,000 + 4,056 + 31,104*(1-Percentage) + 7,680 + 10,080 + 8,400

Pruning percentage	MAC operations	Accuracy for K=5
10	778,210	24.390
20	775099.2	24.380
30	771988.8	24.280
40	768878.4	24.260
50	765768	24.170
60	762,658	24.230
70	759547.2	24.230
80	756436.8	23.960
90	753326.4	23.400
100	750216	5.000



- Pruning fc1 layer with a range of percentage:

```
Pruning weights with 10.0% of layer fc1
Test: [ 0/79]     Time  0.144 ( 0.144)    Loss 4.0385e+00 (4.0385e+00)
      Acc@1  10.16 ( 10.16)  Acc@5  28.12 ( 28.12)
 * Acc@1 7.580 Acc@5 24.430
```

```
Pruning weights with 20.0% of layer fc1
Test: [ 0/79]     Time  0.276 ( 0.276)    Loss 4.0280e+00 (4.0280e+00)
      Acc@1  11.72 ( 11.72)  Acc@5  31.25 ( 31.25)
 * Acc@1 7.550 Acc@5 24.350
```

```
Pruning weights with 30.0% of layer fc1
Test: [ 0/79]     Time  0.285 ( 0.285)    Loss 4.0215e+00 (4.0215e+00)
      Acc@1  10.16 ( 10.16)  Acc@5  30.47 ( 30.47)
 * Acc@1 7.520 Acc@5 24.380
```

```
Pruning weights with 40.0% of layer fc1
Test: [ 0/79]     Time  0.241 ( 0.241)    Loss 4.0233e+00 (4.0233e+00)
      Acc@1   8.59 (  8.59)  Acc@5  35.16 ( 35.16)
 * Acc@1 7.350 Acc@5 24.450
```

```
Pruning weights with 50.0% of layer fc1
Test: [ 0/79]     Time  0.253 ( 0.253)    Loss 4.0482e+00 (4.0482e+00)
      Acc@1  10.94 ( 10.94)  Acc@5  28.91 ( 28.91)
 * Acc@1 6.730 Acc@5 23.420
```

```
Pruning weights with 60.0% of layer fc1
Test: [ 0/79]     Time  0.159 ( 0.159)    Loss 4.1112e+00 (4.1112e+00)
      Acc@1   9.38 (  9.38)  Acc@5  25.00 ( 25.00)
 * Acc@1 6.030 Acc@5 22.290
```

```
Pruning weights with 70.0% of layer fc1
Test: [ 0/79]     Time  0.140 ( 0.140)    Loss 4.1941e+00 (4.1941e+00)
      Acc@1   9.38 (  9.38)  Acc@5  24.22 ( 24.22)
 * Acc@1 5.270 Acc@5 20.410
```

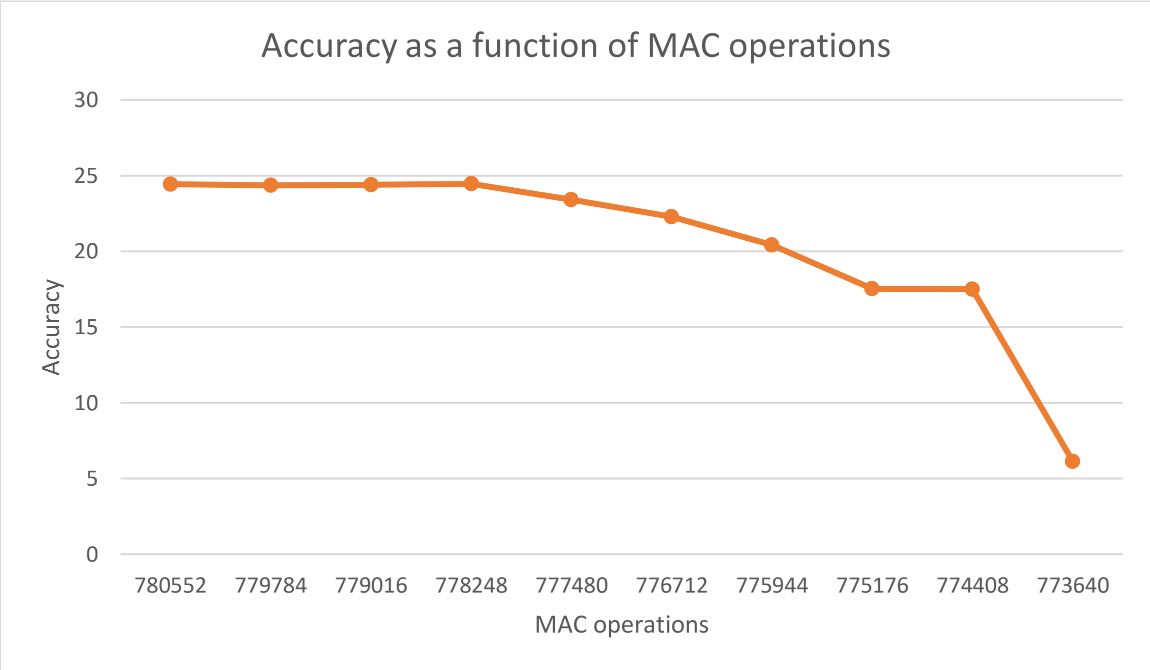
```
Pruning weights with 80.0% of layer fc1
Test: [ 0/79]     Time  0.156 ( 0.156)    Loss 4.2849e+00 (4.2849e+00)
      Acc@1   4.69 (  4.69)  Acc@5  17.19 ( 17.19)
 * Acc@1 4.750 Acc@5 17.540
```

```
Pruning weights with 90.0% of layer fc1
Test: [ 0/79]     Time  0.151 ( 0.151)    Loss 4.3055e+00 (4.3055e+00)
      Acc@1   4.69 (  4.69)  Acc@5  19.53 ( 19.53)
 * Acc@1 4.800 Acc@5 17.480
```

```
Pruning weights with 100.0% of layer fc1
Test: [ 0/79]     Time  0.158 ( 0.158)    Loss 4.6111e+00 (4.6111e+00)
      Acc@1   1.56 (  1.56)  Acc@5  10.16 ( 10.16)
 * Acc@1 1.220 Acc@5 6.140
```

Overall, MAC operations = 720,000 + 4,056 + 31,104 + 7,680*(1-Percentage) + 10,080 + 8,400

Pruning percentage	MAC operations	Accuracy for K=5
10	780552	24.430
20	779784	24.350
30	779016	24.380
40	778248	24.450
50	777480	23.420
60	776712	22.290
70	775944	20.410
80	775176	17.540
90	774408	17.480
100	773640	6.140



- Pruning fc2 layer with a range of percentage:

```
Pruning weights with 10.0% of layer fc2
Test: [ 0/79]     Time  0.172 ( 0.172)    Loss 4.0403e+00 (4.0403e+00)
      Acc@1  10.94 ( 10.94)  Acc@5  29.69 ( 29.69)
 * Acc@1 7.650 Acc@5 24.540
```

```
Pruning weights with 20.0% of layer fc2
Test: [ 0/79]     Time  0.172 ( 0.172)    Loss 4.0307e+00 (4.0307e+00)
      Acc@1   9.38 (  9.38)  Acc@5  32.03 ( 32.03)
 * Acc@1 7.630 Acc@5 24.670
```

```
Pruning weights with 30.0% of layer fc2
Test: [ 0/79]     Time  0.162 ( 0.162)    Loss 4.0618e+00 (4.0618e+00)
      Acc@1   8.59 (  8.59)  Acc@5  31.25 ( 31.25)
 * Acc@1 7.390 Acc@5 24.550
```

```
Pruning weights with 40.0% of layer fc2
Test: [ 0/79]     Time  0.162 ( 0.162)    Loss 4.0621e+00 (4.0621e+00)
      Acc@1  10.94 ( 10.94)  Acc@5  30.47 ( 30.47)
 * Acc@1 6.530 Acc@5 23.820
```

```
Pruning weights with 50.0% of layer fc2
Test: [ 0/79]     Time  0.159 ( 0.159)    Loss 4.0968e+00 (4.0968e+00)
      Acc@1   9.38 (  9.38)  Acc@5  31.25 ( 31.25)
 * Acc@1 5.820 Acc@5 22.530
```

```
Pruning weights with 60.0% of layer fc2
Test: [ 0/79]     Time  0.151 ( 0.151)    Loss 4.1841e+00 (4.1841e+00)
      Acc@1   7.03 (  7.03)  Acc@5  24.22 ( 24.22)
 * Acc@1 5.350 Acc@5 19.960
```

```
Pruning weights with 70.0% of layer fc2
Test: [ 0/79]     Time  0.173 ( 0.173)    Loss 4.1967e+00 (4.1967e+00)
      Acc@1   4.69 (  4.69)  Acc@5  22.66 ( 22.66)
 * Acc@1 5.790 Acc@5 19.510
```

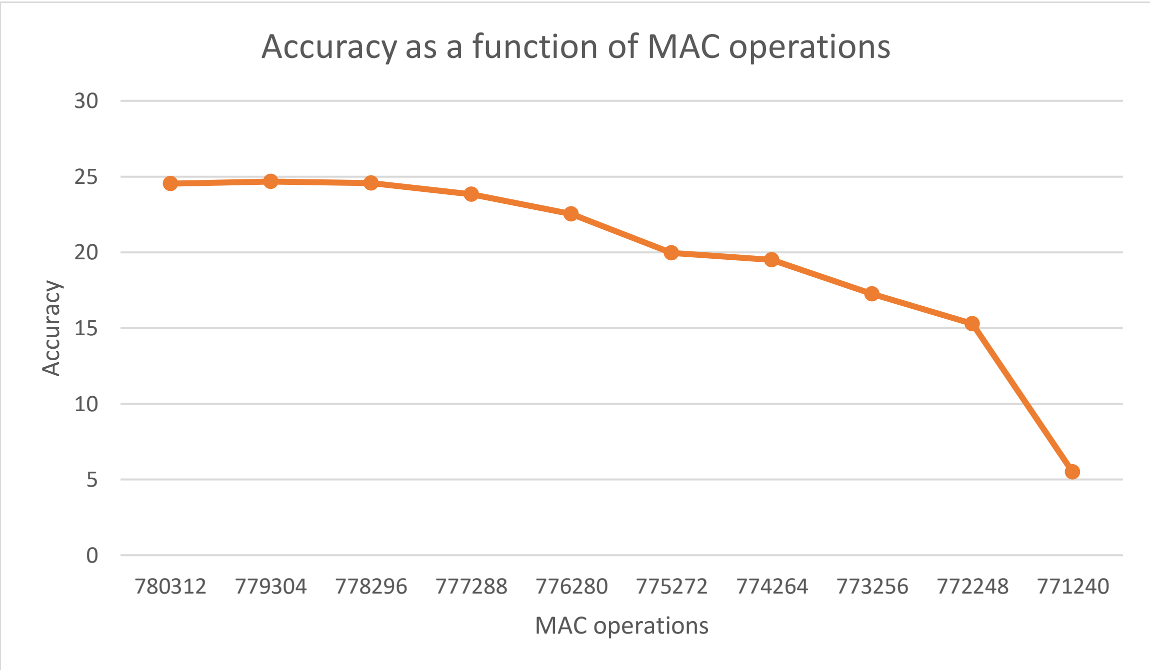
```
Pruning weights with 80.0% of layer fc2
Test: [ 0/79]     Time  0.150 ( 0.150)    Loss 4.2805e+00 (4.2805e+00)
      Acc@1   4.69 (  4.69)  Acc@5  16.41 ( 16.41)
 * Acc@1 4.900 Acc@5 17.240
```

```
Pruning weights with 90.0% of layer fc2
Test: [ 0/79]     Time  0.151 ( 0.151)    Loss 4.3426e+00 (4.3426e+00)
      Acc@1   3.91 (  3.91)  Acc@5  18.75 ( 18.75)
 * Acc@1 4.600 Acc@5 15.280
```

```
Pruning weights with 100.0% of layer fc2
Test: [ 0/79]     Time  0.153 ( 0.153)    Loss 4.6256e+00 (4.6256e+00)
      Acc@1   0.00 (  0.00)  Acc@5   5.47 (  5.47)
 * Acc@1 1.650 Acc@5 5.480
```

Overall, MAC operations = 720,000 + 4,056 + 31,104 + 7,680 + 10,080*(1-Percentage) + 8,400

Pruning percentage	MAC operations	Accuracy for K=5
10	780312	24.540
20	779304	24.670
30	778296	24.550
40	777288	23.820
50	776280	22.530
60	775272	19.960
70	774264	19.510
80	773256	17.240
90	772248	15.280
100	771240	5.480



- Pruning fc3 layer with a range of percentage:

```
Pruning weights with 10.0% of layer fc3
Test: [ 0/79]     Time  0.165 ( 0.165)    Loss 4.0310e+00 (4.0310e+00)
      Acc@1  10.94 ( 10.94)  Acc@5  32.03 ( 32.03)
 * Acc@1 7.630 Acc@5 24.550
```

```
Pruning weights with 20.0% of layer fc3
Test: [ 0/79]     Time  0.157 ( 0.157)    Loss 4.0414e+00 (4.0414e+00)
      Acc@1   9.38 (  9.38)  Acc@5  25.78 ( 25.78)
 * Acc@1 7.330 Acc@5 24.120
```

```
Pruning weights with 30.0% of layer fc3
Test: [ 0/79]     Time  0.148 ( 0.148)    Loss 4.0342e+00 (4.0342e+00)
      Acc@1  11.72 ( 11.72)  Acc@5  26.56 ( 26.56)
 * Acc@1 6.580 Acc@5 22.160
```

```
Pruning weights with 40.0% of layer fc3
Test: [ 0/79]     Time  0.149 ( 0.149)    Loss 4.0822e+00 (4.0822e+00)
      Acc@1   6.25 (  6.25)  Acc@5  25.78 ( 25.78)
 * Acc@1 6.200 Acc@5 20.860
```

```
Pruning weights with 50.0% of layer fc3
Test: [ 0/79]     Time  0.153 ( 0.153)    Loss 4.1653e+00 (4.1653e+00)
      Acc@1   7.03 (  7.03)  Acc@5  18.75 ( 18.75)
 * Acc@1 5.940 Acc@5 19.330
```

```
Pruning weights with 60.0% of layer fc3
Test: [ 0/79]     Time  0.164 ( 0.164)    Loss 4.3018e+00 (4.3018e+00)
      Acc@1   7.03 (  7.03)  Acc@5  15.62 ( 15.62)
 * Acc@1 4.430 Acc@5 16.400
```

```
Pruning weights with 70.0% of layer fc3
Test: [ 0/79]     Time  0.155 ( 0.155)    Loss 4.6253e+00 (4.6253e+00)
      Acc@1   1.56 (  1.56)  Acc@5  14.84 ( 14.84)
 * Acc@1 3.830 Acc@5 14.950
```

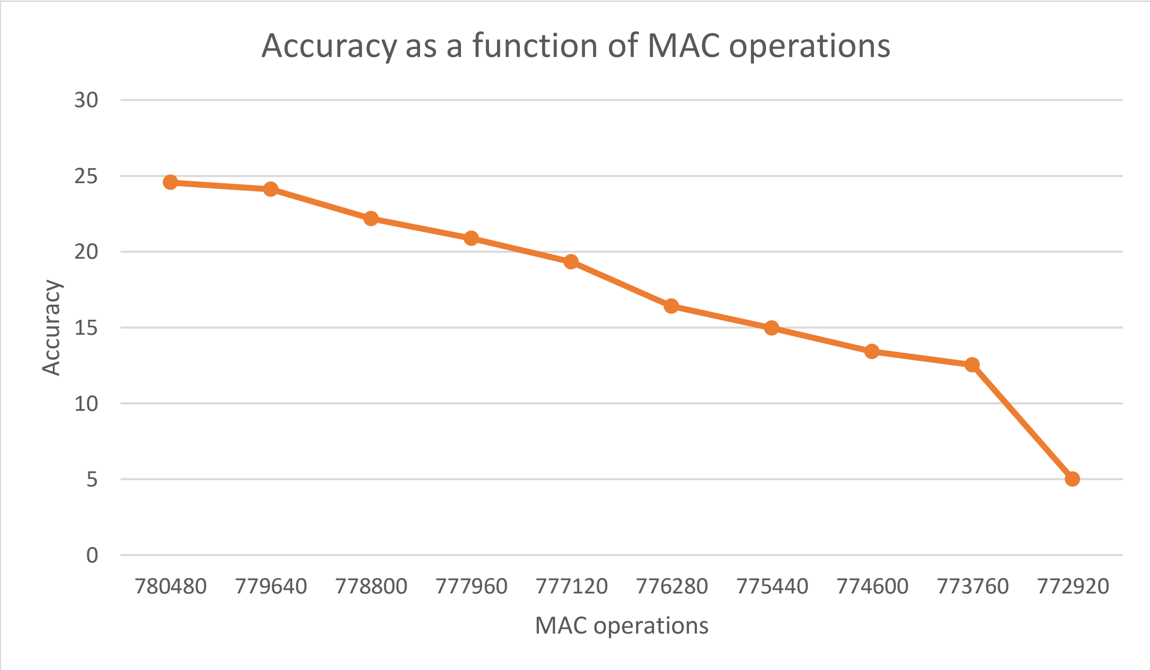
```
Pruning weights with 80.0% of layer fc3
Test: [ 0/79]     Time  0.146 ( 0.146)    Loss 4.8782e+00 (4.8782e+00)
      Acc@1   0.78 (  0.78)  Acc@5  13.28 ( 13.28)
 * Acc@1 3.120 Acc@5 13.420
```

```
Pruning weights with 90.0% of layer fc3
Test: [ 0/79]     Time  0.154 ( 0.154)    Loss 4.7374e+00 (4.7374e+00)
      Acc@1   3.12 (  3.12)  Acc@5  11.72 ( 11.72)
 * Acc@1 3.270 Acc@5 12.530
```

```
Pruning weights with 100.0% of layer fc3
Test: [ 0/79]     Time  0.155 ( 0.155)    Loss 4.6117e+00 (4.6117e+00)
      Acc@1   0.78 (  0.78)  Acc@5   6.25 (  6.25)
 * Acc@1 1.000 Acc@5 5.000
```

Overall, MAC operations = 720,000 + 4,056 + 31,104 + 7,680 + 10,080+ 8,400*(1-Percentage)

Pruning percentage	MAC operations	Accuracy for K=5
10	780480	24.550
20	779640	24.120
30	778800	22.160
40	777960	20.860
50	777120	19.330
60	776280	16.400
70	775440	14.950
80	774600	13.420
90	773760	12.530
100	772920	5.000



Conclusions:

The more we go deeper into the network the pruning becomes destructive in terms of accuracy as seen in the last three fc layers and in the same hand not profitable since the number of MAC operations is still very high due to the low weight ratio between fc layers that come in the end and conv layers that are present in the beginning of the network.

To sum up it is highly recommended to prune weights in layers with high computation rates compared to the rest of the layers for example in our network conv1 and conv2 layers. but also keep in mind that its better to prune layers in the beginning of the network to harm the accuracy as little as possible.

Part 5

Conventional Hardware may benefit from this pruning method since the pruned weights result in a zero multiplication hence the hardware could spare the cycles for each pruned weight computation in terms of latency and power consumption.

the benefits of pruning shown above are handled via software using sparsity methods and algorithms in todays hardware, conventional CPU’s and GPU’s