Report - Lab5

Siyu Meng sm2659

Baseline runtime: ~ 128s

```
Total Accuracy: 100.000000%

real    2m28.820s
user    2m28.671s
sys     0m0.024s
```

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 87.94    126.88   126.88    20000     6.34     6.34  linear
  8.79    139.56    12.68    10000     1.27     1.27  matmul
  1.36    141.52     1.96    10000     0.20     1.65  convolution_im2col
  0.64    142.45     0.93    10000     0.09     0.09  col2im
  0.51    143.18     0.73    10000     0.07     0.07  flatten
  0.38    143.73     0.55    10000     0.06     0.06  im2col
  0.24    144.07     0.34 217600000     0.00     0.00  relu
  0.06    144.16     0.09                             _init
  0.03    144.21     0.04    10000     0.00     0.00  applyRelu
  0.03    144.25     0.04    10000     0.00     0.00  destroyConvOutput
  0.01    144.26     0.01    10000     0.00     0.00  kernel_flatten
  0.01    144.27     0.01        1    10.00    10.00  loadImages
  0.01    144.28     0.01                             main
  0.00    144.28     0.00    10000     0.00    14.42  forwardPass
  0.00    144.28     0.00    10000     0.00     0.00  predict
  0.00    144.28     0.00    10000     0.00     0.00  softmax
```

With SIMD instruction (AVX2) in linear function: ~78s

```
Total Accuracy: 100.000000%

real    1m18.087s
user    1m18.002s
sys     0m0.012s
```

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 57.44     38.42    38.42    20000     1.92     1.92  simd_linear
 18.67     50.91    12.49    10000     1.25     1.25  matmul
 17.70     62.75    11.84    10000     1.18     2.67  convolution_im2col
  1.52     63.77     1.02    10000     0.10     0.10  col2im
  1.34     64.67     0.90 217600000     0.00     0.00  relu
  1.24     65.50     0.83    10000     0.08     0.08  flatten
  1.19     66.29     0.80    10000     0.08     0.08  applyRelu
  0.67     66.74     0.45    10000     0.04     0.04  im2col
  0.15     66.84     0.10                             _init
  0.04     66.87     0.03    10000     0.00     0.00  destroyConvOutput
  0.01     66.88     0.01    10000     0.00     6.68  forwardPass
  0.01     66.89     0.01    10000     0.00     0.00  kernel_flatten
  0.00     66.89     0.00  1380000     0.00     0.00  reduce_avx
  0.00     66.89     0.00    10000     0.00     0.00  predict
  0.00     66.89     0.00    10000     0.00     0.00  softmax
```

With SIMD linear function + compiler optimization (-O3): ~13.5s

```
Total Accuracy: 100.000000%

real    0m13.539s
user    0m13.497s
sys     0m0.017s
```

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  us/call  us/call  name
 65.59      5.87     5.87    20000   293.50   293.50  simd_linear
 24.25      8.04     2.17    10000   217.00   217.00  matmul
  4.92      8.48     0.44    10000    44.00   300.00  convolution_im2col
  3.02      8.75     0.27 216320000     0.00     0.00  relu
  1.12      8.85     0.10    10000    10.00    10.00  col2im
  0.89      8.93     0.08    10000     8.00   895.00  forwardPass
  0.22      8.95     0.02    10000     2.00     2.00  kernel_flatten
  0.00      8.95     0.00    10000     0.00     0.00  applyRelu
  0.00      8.95     0.00    10000     0.00     0.00  softmax
```

- Final Speed up: baseline / optimized = 128s / 13.5s = 9.4
- First, I used gprof to profile the runtime breakdown of the functions. From the first profiling image, I found that linear calls many times and occupy lots of time compared with other functions. Therefore, I focused on optimizing the linear function.
- I googled online according to the options provided in the lab5 assignment and found vectorizing linear multiplication and addition by using SIMD instruction such as AVX is appropriate. I used mm512 at first time but when I checked the system with lscpu, I found we needed to use avx2 rather than avx512f. Therefore I change to use mm256 and compile with -mavx2 successfully.

- Mm256 can hold 8 single-precision (float) values and each avx2 instruction operates on all elements of the vectors simultaneously. SIMD instruction allows computers to process multiple elements in parallel, using a single instruction.
- The runtime with SIMD optimization improves around 128s / 78s = 1.6.
- In order to further optimize and improve the runtime, I tried with different optimization compile flags such as -O2, -O3. Compared to these two flags, they only have little difference around 0.3s, but -O3 is still the faster one.
- -O3 compiler optimization flags include loop unrolling, vectorization, instruction scheduling, and data flow analysis. All of them will reduce loop overhead and increase the instruction-level parallelism. Loop unrolling that replicates the loop body multiple times combined with SIMD instruction will largely improve the performance.
- Finally, the runtime speedup after SIMD and compiler flag is around 128s / 13.5s = 9.4