

## Implement Darknet on FPGA

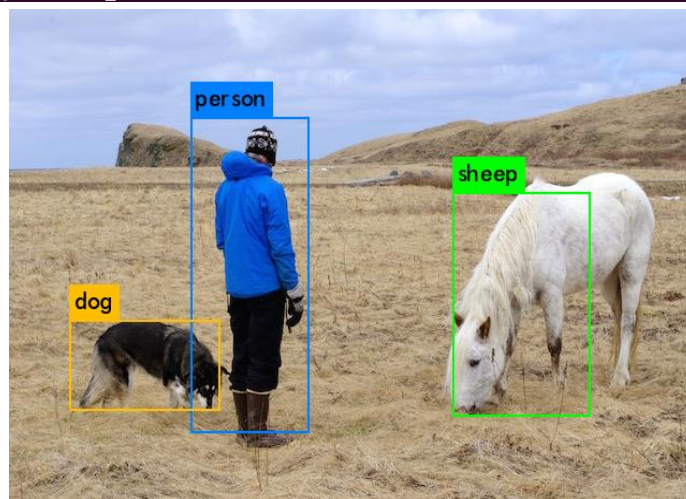
### 1 Darknet

Darknet is a current open source deep learning framework, it is lightweight and is easy to be deployed. Darknet is very convenient to realize object detection, classification and etc. Like Caffe, Darknet is based on C, and it is easy to modify its source code.

### 2 Using Darknet to realize object detection

Darknet supports CPU and GPU, change the Makefile according to your condition. Here is a example of running darknet on CPU. It's the yolo algorithms and the input image is person.jpg. As can be seen, it will output the confidence of each object and the total detection time: sheep(60%),person(73%),dog(53%),time(2.88s). The object is surrounded by the bounding box.

```
zlm@zlm: ~/darknet
zlm@zlm:~$ cd darknet/
zlm@zlm:~/darknet$ ./darknet detector test cfg/voc.data cfg/yolov2-tiny-voc.cfg
g ./tiny-yolo-voc.weights ./data/person.jpg
layer   filters  size  input  output
0 conv   16  3 x 3 / 1  416 x 416 x 3  -> 416 x 416 x 16  0.150 BFL
OPs
1 max    2  2 x 2 / 2  416 x 416 x 16  -> 208 x 208 x 16
2 conv   32  3 x 3 / 1  208 x 208 x 16  -> 208 x 208 x 32  0.399 BFL
OPs
3 max    2  2 x 2 / 2  208 x 208 x 32  -> 104 x 104 x 32
4 conv   64  3 x 3 / 1  104 x 104 x 32  -> 104 x 104 x 64  0.399 BFL
OPs
5 max    2  2 x 2 / 2  104 x 104 x 64  -> 52 x 52 x 64
6 conv  128  3 x 3 / 1   52 x 52 x 64  -> 52 x 52 x 128  0.399 BFL
OPs
7 max    2  2 x 2 / 2   52 x 52 x 128  -> 26 x 26 x 128
8 conv  256  3 x 3 / 1   26 x 26 x 128  -> 26 x 26 x 256  0.399 BFL
OPs
9 max    2  2 x 2 / 2   26 x 26 x 256  -> 13 x 13 x 256
10 conv  512  3 x 3 / 1   13 x 13 x 256  -> 13 x 13 x 512  0.399 BFL
OPs
11 max   2  2 x 2 / 1   13 x 13 x 512  -> 13 x 13 x 512
12 conv 1024  3 x 3 / 1   13 x 13 x 512  -> 13 x 13 x 1024 1.595 BFL
OPs
13 conv 1024  3 x 3 / 1   13 x 13 x 1024 -> 13 x 13 x 1024 3.190 BFL
OPs
14 conv  125  1 x 1 / 1   13 x 13 x 1024 -> 13 x 13 x 125  0.043 BFL
OPs
15 detection
mask_scale: Using default '1.000000'
Loading weights from ./tiny-yolo-voc.weights...Done!
./data/person.jpg: Predicted in 2.883106 seconds.
sheep: 60%
person: 73%
dog: 53%
zlm@zlm:~/darknet$
```



### 3 Source Code Analysis

```
~/darknet/src/convolutional_layer.c - Sublime Text (UNREGISTERED)
gemmm_nn.c x convolutional_layer.c x
443 }
444
445 void forward_convolutional_layer(convolutional_layer l, network net)
446 {
447     int i, j;
448
449     fill_cpu(l.outputs*l.batch, 0, l.output, 1);
450
451     if(l.xnor){
452         binarize_weights(l.weights, l.n, l.c/l.groups*l.size*l.size, l.binary_weights);
453         swap_binary(&l);
454         binarize_cpu(net.input, l.c*l.h*l.w*l.batch, l.binary_input);
455         net.input = l.binary_input;
456     }
457
458     int m = l.n/l.groups;
459     int k = l.size*l.size*l.c/l.groups;
460     int n = l.out_w*l.out_h;
461     for(i = 0; i < l.batch; ++i){
462         for(j = 0; j < l.groups; ++j){
463             float *a = l.weights + j*l.nweights/l.groups;
464             float *b = net.workspace;
465             float *c = l.output + (i*l.groups + j)*n*m;
466
467             im2col_cpu(net.input + (i*l.groups + j)*l.c/l.groups*l.h*l.w,
468                 l.c/l.groups, l.h, l.w, l.size, l.stride, l.pad, b);
469             gemm(0,0,m,n,k,1,a,k,b,n,1,c,n);
470         }
471     }
472
473     if(l.batch_normalize){
474         forward_batchnorm_layer(l, net);
475     } else {
476         add_bias(l.output, l.biases, l.batch, l.n, l.out_h*l.out_w);
477     }
478
479     activate_array(l.output, l.outputs*l.batch, l.activation);
480     if(l.binary || l.xnor) swap_binary(&l);
481 }
}

~/darknet/src/gemm.c - Sublime Text (UNREGISTERED)
gemmm_nn.c x convolutional_layer.c x gemm.c x
65 void gemm(int TA, int TB, int M, int N, int K, float ALPHA,
66     float *A, int lda,
67     float *B, int ldb,
68     float BETA,
69     float *C, int ldc)
70 {
71     gemm_cpu(TA, TB, M, N, K, ALPHA, A, lda, B, ldb, BETA, C, ldc);
72 }
73
74 void gemm_nn(int M, int N, int K, float ALPHA,
75     float *A, int lda,
76     float *B, int ldb,
77     float *C, int ldc)
78 {
79     int i,j,k;
80     #pragma omp parallel for
81     for(i = 0; i < M; ++i){
82         for(k = 0; k < K; ++k){
83             register float A_PART = ALPHA*A[i*lda+k];
84             for(j = 0; j < N; ++j){
85                 C[i*ldc+j] += A_PART*B[k*ldb+j];
86             }
87         }
88     }
89 }
90
91 void gemm_nt(int M, int N, int K, float ALPHA,
92     float *A, int lda,
93     float *B, int ldb,
94     float *C, int ldc)
95 {
96 }
97
98 void gemm_tn(int M, int N, int K, float ALPHA,
99     float *A, int lda,
100     float *B, int ldb,
101     float *C, int ldc)
102 {
103 }
104
105 void gemm_tt(int M, int N, int K, float ALPHA,
106     float *A, int lda,
107     float *B, int ldb,
108     float *C, int ldc)
109 {
110 }
111
112 void gemm_cpu(int TA, int TB, int M, int N, int K, float ALPHA,
113     float *A, int lda,
114     float *B, int ldb,
115     float BETA,
116     float *C, int ldc)
117 {
118     //printf("cpu: %d %d %d %d %d %d %d %d %d %d\n", TA, TB, M, N, K, ALPHA, lda, ldb, BETA, ldc);
119     int i, j;
120     for(i = 0; i < M; ++i){
121         for(j = 0; j < N; ++j){
122             C[i*ldc+j] *= BETA;
123         }
124     }
125     if(!TA && !TB)
126         gemm_nn(M, N, K, ALPHA, A, lda, B, ldb, C, ldc);
127     else if(TA && !TB)
128         gemm_tn(M, N, K, ALPHA, A, lda, B, ldb, C, ldc);
129     else if(!TA && TB)
130         gemm_nt(M, N, K, ALPHA, A, lda, B, ldb, C, ldc);
131     else
132         gemm_tt(M, N, K, ALPHA, A, lda, B, ldb, C, ldc);
133 }
```

In CNNs, there are Conv-layers, pooling layers and fully connected layers, and the Conv-layers are computational intensive. Here we mainly modify the source code of Conv-layers. In Darknet, the source code of Conv-layers is in convolutional\_layer.c. And here we talk about running CNN on fpga, it mainly means we run the inference phase on fpga. In inference phase, Darknet mainly uses this function: forward\_convolutional\_layer(), it completes the matrix multiplication which is input matrix A \* input matrix B=output matrix C. And the gemm() function

in `forward_convolutional_layer()` realizes this multiplication. The parameters in the `gemm()` function specifically refer to TA: whether matrix A is transposed, TB: whether matrix B is transposed, M: row numbers of matrix A, N: column numbers of matrix B, K: column numbers of matrix B, \*A: input Matrix A, \*B: input matrix B, float \*C: output matrix C, lda: columns of A, ldb: columns of B, ldc: columns of C. `gemm()` function realizes  $A(M,K) \cdot B(K,N) = C(M,N)$  operation. It can be seen that the `gemm()` function further calls `gemm_cpu()`, and `gemm_cpu()` function further calls `gemm_nn()`, `gemm_tn()`, `gemm_nt()` and `gemm_tt()`, `gemm_nn()` is used by default in darknet. The `gemm_nn()` function is called every time the convolution layer is running, so this is the core function of the convolutional layer matrix operation. When transforming to fpga, modify the `gemm_nn()` function to the `kernel.cl` function, and let fpga be responsible for this part of the operation.

#### 4 Modify Source Code

As shown in the figure below, I first replace the original `gemm()` function with `gemm_fpga()` in the initial `convolutional_layer.c`, and realize `gemm_fpga()` in `gemm.c`.

```

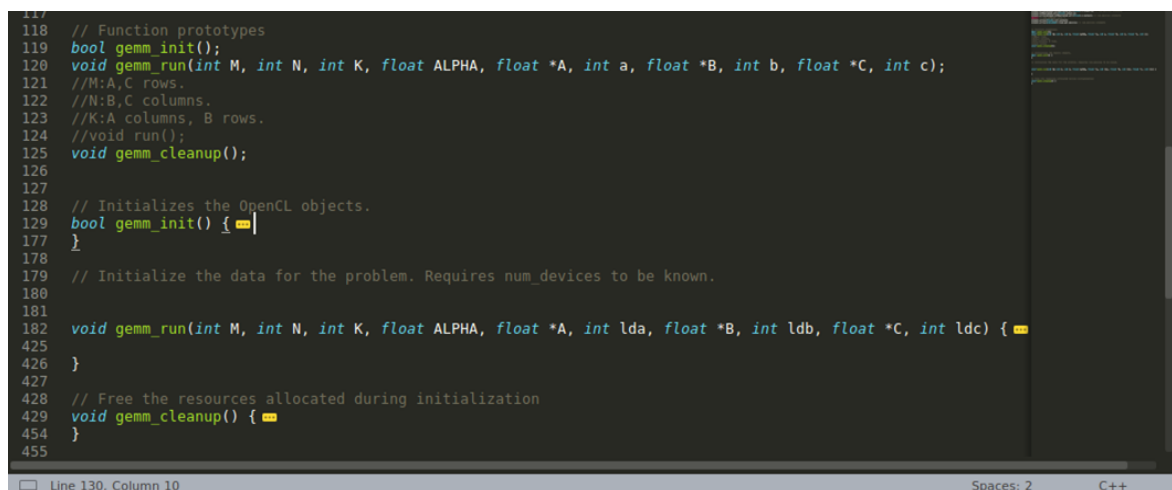
505         if (l.size == 1) {
506             b = im;
507         } else {
508             im2col_cpu(im, l.c/l.groups, l.h, l.w, l.size, l.stride, l.pad, b);
509         }
510         //gemm(0,0,m,n,k,1,a,k,b,n,1,c,n);
511         gemm_fpga(0,0,m,n,k,1,a,k,b,n,1,c,n);
512     }
513 }
514
515

```

Adding two interfaces `gemm_fpga()` and `gemm_nn_fpga()` to `gemm.c` as shown below. All parameters in `gemm_fpga()` are the same as `gemm()`, then `gemm_fpga()` calls `gemm_nn_fpga()`, while `gemm_nn_fpga()` further calls `gemm_run()`, and `gemm_run()` uses `fpga` to replace the matrix operation taken by `gemm_nn()` before.

#### 4.1 `gemm_run()`

The `gemm_run()` is responsible for realising the matrix operation of  $A*B=C$  on fpga. This function is implemented with `opencl`. I modify the official matrix multiplication example to realize `gemm_run()`. As shown in the figure below, I modify the original host program to get `gemm_fpga.cpp`, which mainly contains three interfaces, `gemm_init()`, `gemm_run()` and `gemm_cleanup()`. `Gemm_init()` is responsible for some initialization of `opencl`, `gemm_cleanup()` is responsible for the release of resources, and `gemm_run()` is mainly responsible for matrix operations.



```

117
118 // Function prototypes
119 bool gemm_init();
120 void gemm_run(int M, int N, int K, float ALPHA, float *A, int a, float *B, int b, float *C, int c);
121 //M:A,C rows.
122 //N:B,C columns.
123 //K:A columns, B rows.
124 //void run();
125 void gemm_cleanup();
126
127
128 // Initializes the OpenCL objects.
129 bool gemm_init() {
130 }
131
132 // Initialize the data for the problem. Requires num_devices to be known.
133
134 void gemm_run(int M, int N, int K, float ALPHA, float *A, int lda, float *B, int ldb, float *C, int ldc) {
135 }
136
137 // Free the resources allocated during initialization
138 void gemm_cleanup() {
139 }
140
141

```

The actual implementation of these three functions is much complicated. You can refer to the `gemm_fpga.cpp` which I already completed. Here only some of the code in the `gemm_run()` function is explained. As shown in the figure, this section passes parameters of `gemm_run()` in host file to the kernel file, and the parameters should in order.



```

status = clSetKernelArg(kernel[i], argi++, sizeof(A_height), &A_height);
checkError(status, "Failed to set argument %d", argi - 1);

status = clSetKernelArg(kernel[i], argi++, sizeof(B_width), &B_width);
checkError(status, "Failed to set argument %d", argi - 1);

status = clSetKernelArg(kernel[i], argi++, sizeof(A_width), &A_width);
checkError(status, "Failed to set argument %d", argi - 1);

status = clSetKernelArg(kernel[i], argi++, sizeof(ALPHA), &ALPHA);
checkError(status, "Failed to set argument %d", argi - 1);

status = clSetKernelArg(kernel[i], argi++, sizeof(cl_mem), &input_a_buf[i]);
checkError(status, "Failed to set argument %d", argi - 1);

status = clSetKernelArg(kernel[i], argi++, sizeof(lda), &lda);
checkError(status, "Failed to set argument %d", argi - 1);

status = clSetKernelArg(kernel[i], argi++, sizeof(cl_mem), &input_b_buf[i]);
checkError(status, "Failed to set argument %d", argi - 1);

status = clSetKernelArg(kernel[i], argi++, sizeof(ldb), &ldb);
checkError(status, "Failed to set argument %d", argi - 1);

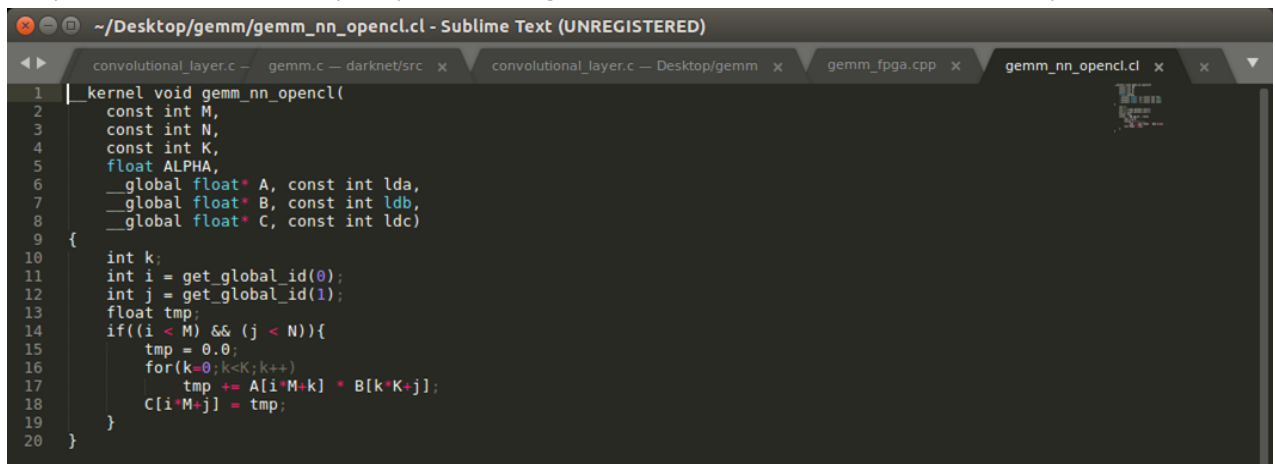
status = clSetKernelArg(kernel[i], argi++, sizeof(cl_mem), &output_buf[i]);
checkError(status, "Failed to set argument %d", argi - 1);

status = clSetKernelArg(kernel[i], argi++, sizeof(ldc), &ldc);
checkError(status, "Failed to set argument %d", argi - 1);

```

## 4.2 Kernel function

Here, the implementation of the kernel function is directly converted from the book. This implementation is relatively simple, according to the latter test, this kernel function may has a



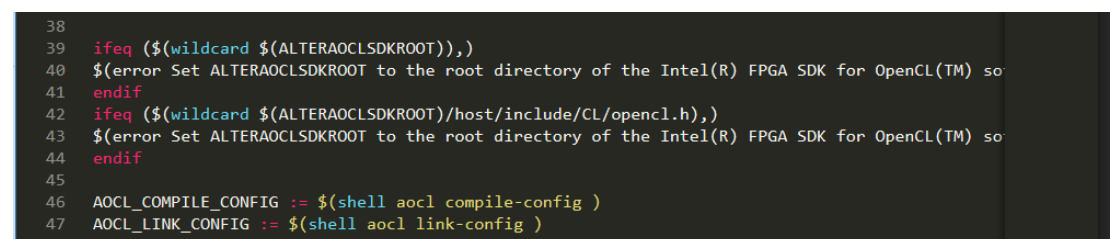
```
1 kernel void gemm_nn_opengl(
2     const int M,
3     const int N,
4     const int K,
5     float ALPHA,
6     __global float* A, const int lda,
7     __global float* B, const int ldb,
8     __global float* C, const int ldc)
9 {
10     int k;
11     int i = get_global_id(0);
12     int j = get_global_id(1);
13     float tmp;
14     if((i < M) && (j < N)){
15         tmp = 0.0;
16         for(k=0; k<K; k++){
17             tmp += A[i*M+k] * B[k*N+j];
18         }
19         C[i*M+j] = tmp;
20     }
```

little problem. After the kernel function is written, I compile it and generates the **aocx** file. **This compiling step will take about 2 hours.**

## 5 Compile

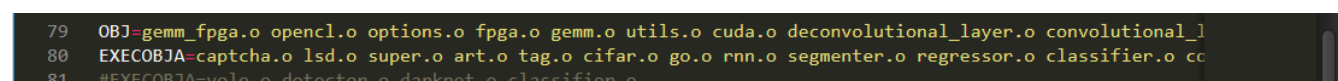
Place the written `gemm_fpga.cpp` and its corresponding header file `gemm_fpga.h` in the `darknet/src` directory, and place the kernel `aocx` file in the `darknet/` directory. Then we need modify the Makefile, as shown below.

First check the openCl path:



```
38
39 ifeq ($(wildcard $(ALTERAOCLSDKROOT)),)
40 $(error Set ALTERAOCLSDKROOT to the root directory of the Intel(R) FPGA SDK for OpenCL(TM) so
41 endif
42 ifeq ($(wildcard $(ALTERAOCLSDKROOT)/host/include/CL/opencl.h),)
43 $(error Set ALTERAOCLSDKROOT to the root directory of the Intel(R) FPGA SDK for OpenCL(TM) so
44 endif
45
46 AOCL_COMPILE_CONFIG := $(shell aocl compile-config )
47 AOCL_LINK_CONFIG := $(shell aocl link-config )
48
```

Then add `gemm_fpga.o` on OBJ, this aims to compile the previously written `gemm_fpga.cpp` to an executable file.



```
79 OBJ+=gemm_fpga.o opengl.o options.o fpga.o gemm.o utils.o cuda.o deconvolutional_layer.o convolutional_l
80 EXECCOBJA=captcha.o lsd.o super.o art.o tag.o cifar.o go.o rnn.o segmenter.o regressor.o classifier.o cc
81 #EXECCOBJA=vol.o detector.o darknet.o classifier.o
```

Finally, adding openCl path at compile time, and add a new line:

```
$(OBJDIR)%.o: %.cpp $(DEPS)
$(CC) $(COMMON) $(CPPFLAGS) $(CXXFLAGS) $(AOCL_COMPILE_CONFIG) -c $< -o $@
$(AOCL_LINK_CONFIG) $(foreach L,$(LIBS),-l$L),
```

this line is responsible for compiling `gemm_fpga.cpp` into `gemm_fpga.o`, because the original Makefile can only compile the `.c` file, and the `.cpp` file can be compiled after the addition. In addition, the compiler in Makefile should be replaced by `g++` before `gcc`, because `opencl` is based on `g++`, `darknet` is based on `gcc`.

**I 'm not going to show the other little changes in Darknet, please refer to the project on Github(such as add head file to `darknet.h` and etc.).**

## 6 Test

Test results are are shown below.

After running the test command, `fpga` can be called normally and the `fpga` information is

printed, this means darknet has loaded fpga.

However, it produced -nan error in the eighth layer, it might be that the operation value is too large, resulting in the final failure and only output execution time. This should be a problem in the kernel function, and because the entire process is still rough, the execution time is still slow. After further writing a better kernel function, it should be able to output the target information normally and achieve target detection.

```
root@046099:/home/test/darknet
[root@046099 darknet]#
[root@046099 darknet]#
[root@046099 darknet]# ./darknet detector test cfg/voc.data cfg/yolov2-tiny-voc.
cfg ./tiny-yolo-voc.weights ./data/person.jpg
Initializing OpenCL
Platform: Intel(R) FPGA SDK for OpenCL(TM)
Using 1 device(s)
  fa510q : Arria 10 Reference Platform (acla10_ref0)
Using AOXC: gemm_nn_opencl.aocx
Reprogramming device [0] with handle 1
layer   filters   size   input   output
0 conv   16   3 x 3 / 1   228 x 228 x 3   -> 228 x 228 x 16   0.045 BFL
OPS
1 max     2   2 x 2 / 2   228 x 228 x 16   -> 114 x 114 x 16
2 conv    32  3 x 3 / 1   114 x 114 x 16   -> 114 x 114 x 32   0.120 BFL
OPS
3 max     2   2 x 2 / 2   114 x 114 x 32   -> 57 x 57 x 32
4 conv    64  3 x 3 / 1   57 x 57 x 32     -> 57 x 57 x 64     0.120 BFL
OPS
5 max     2   2 x 2 / 2   57 x 57 x 64     -> 28 x 28 x 64
6 conv   128  3 x 3 / 1   28 x 28 x 64     -> 28 x 28 x 128    0.116 BFL
OPS
7 max     2   2 x 2 / 2   28 x 28 x 128    -> 14 x 14 x 128
8 conv   256  3 x 3 / 1   14 x 14 x 128    -> 14 x 14 x 256    0.116 BFL
OPS
9 max     2   2 x 2 / 2   14 x 14 x 256    -> 7 x 7 x 256
10 conv  512  3 x 3 / 1   7 x 7 x 256      -> 7 x 7 x 512      0.116 BFL
OPS
11 max    2   2 x 2 / 1   7 x 7 x 512      -> 7 x 7 x 512
12 conv 1024  3 x 3 / 1   7 x 7 x 512      -> 7 x 7 x 1024     0.462 BFL
OPS
13 conv 1024  3 x 3 / 1   7 x 7 x 1024     -> 7 x 7 x 1024     0.925 BFL
OPS
14 conv 125  1 x 1 / 1   7 x 7 x 1024     -> 7 x 7 x 125      0.013 BFL
OPS
15 detection
mask_scale: Using default '1.000000'
Loading weights from ./tiny-yolo-voc.weights...Done!
Matrix sizes:
  A: 16 x 27
  B: 27 x 51984
  C: 16 x 51984
Generating input matrices
Launching for device 0 (global size: 16, 51984)
Time: 65.570 ms
Kernel time (device 0): 65.507 ms
Throughput: 0.68 GFLOPS
device:0
output_size:8
C_size:8
output1: -0.183877
Matrix sizes:
  A: 32 x 144
  C_size:8
output1: 492.317
Matrix sizes:
  A: 1024 x 9216
  B: 9216 x 49
  C: 1024 x 49
Generating input matrices
Launching for device 0 (global size: 1024, 49)
Time: 3348.849 ms
Kernel time (device 0): 3348.777 ms
Throughput: 0.28 GFLOPS
device:0
output_size:8
C_size:8
output1: nan
Matrix sizes:
  A: 125 x 1024
  B: 1024 x 49
  C: 125 x 49
Generating input matrices
Launching for device 0 (global size: 125, 49)
Time: 18.476 ms
Kernel time (device 0): 18.414 ms
Throughput: 0.68 GFLOPS
device:0
output_size:8
C_size:8
output1: nan
./data/person.jpg: Predicted in 6.662858 seconds.
[root@046099 darknet]#
```