QEMU 上的 YOLO 算法

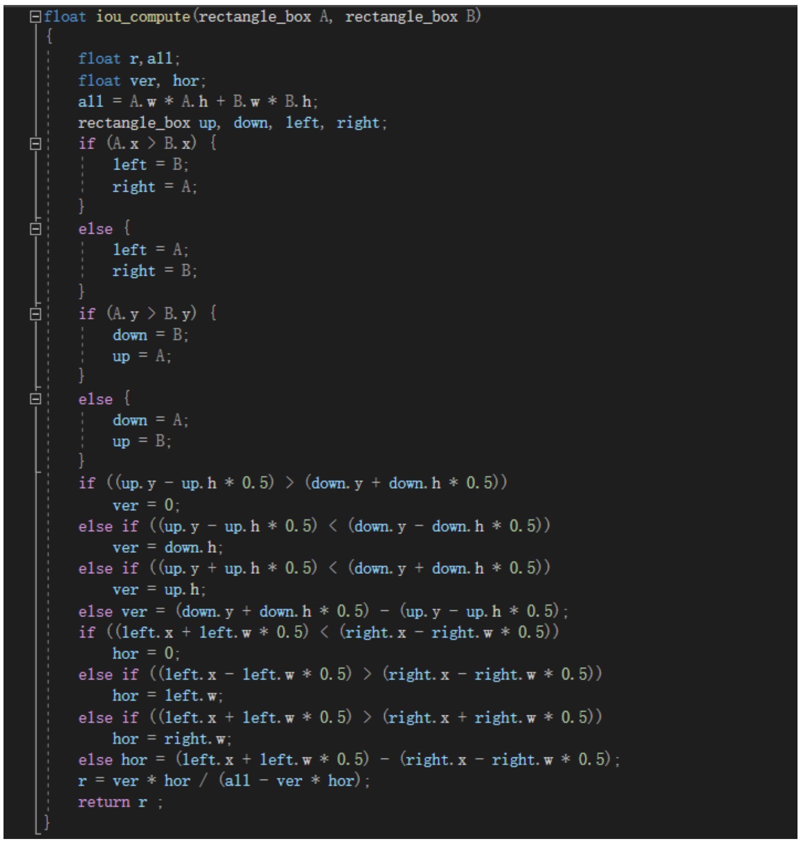
1、实验目的

了解目标识别领域的基本算法。通过编写 YOLO 算法网络层的部分核心函数，了解 YOLO 算法的基本原理。通过在 RISCV QEMU 上运行 YOLO 算法，熟悉 C 语言的嵌入式调试和仿真。

2、实验步骤（包括代码、实验结果、数据记录、截图等）

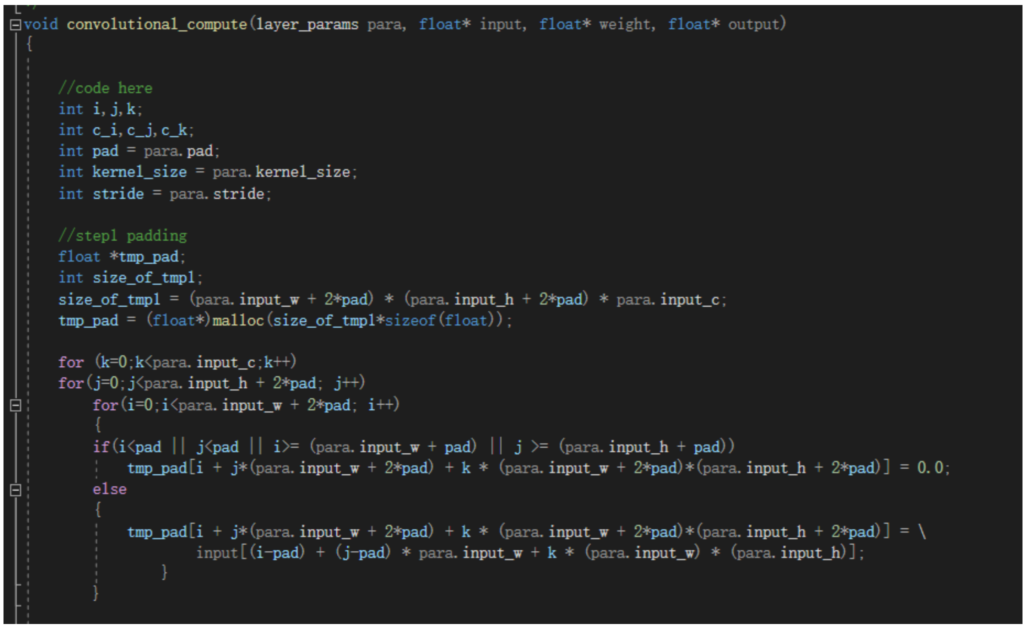
（这里先贴上代码和操作，分析在 part 3 里）

1) IOU 函数的实现

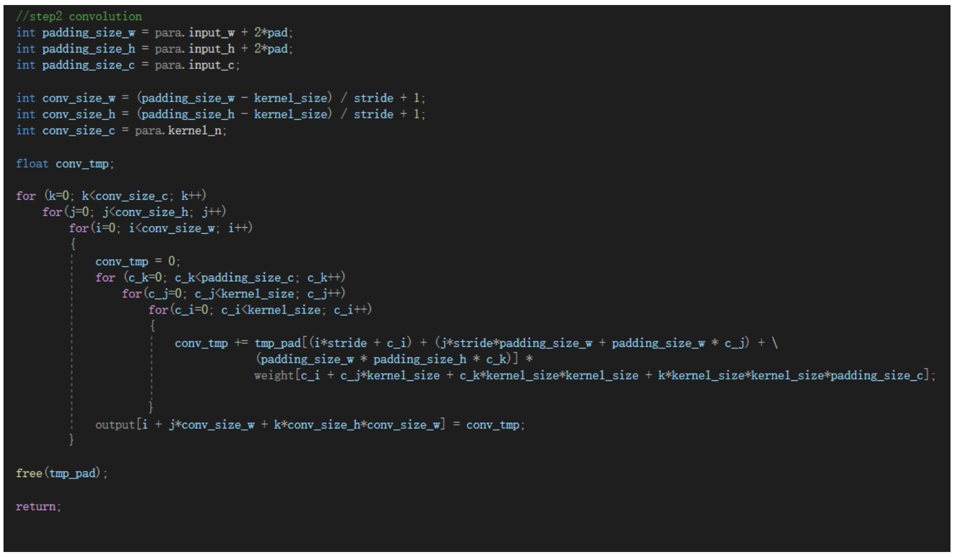


```c
float iou_compute(rectangle_box A, rectangle_box B)
{
    float r,all;
    float ver, hor;
    all = A.w * A.h + B.w * B.h;
    rectangle_box up, down, left, right;
    if (A.x > B.x) {
        left = B;
        right = A;
    }
    else {
        left = A;
        right = B;
    }
    if (A.y > B.y) {
        down = B;
        up = A;
    }
    else {
        down = A;
        up = B;
    }
    if ((up.y - up.h * 0.5) > (down.y + down.h * 0.5))
        ver = 0;
    else if ((up.y - up.h * 0.5) < (down.y - down.h * 0.5))
        ver = down.h;
    else if ((up.y + up.h * 0.5) < (down.y + down.h * 0.5))
        ver = up.h;
    else ver = (down.y + down.h * 0.5) - (up.y - up.h * 0.5);
    if ((left.x + left.w * 0.5) < (right.x - right.w * 0.5))
        hor = 0;
    else if ((left.x - left.w * 0.5) > (right.x - right.w * 0.5))
        hor = left.w;
    else if ((left.x + left.w * 0.5) > (right.x + right.w * 0.5))
        hor = right.w;
    else hor = (left.x + left.w * 0.5) - (right.x - right.w * 0.5);
    r = ver * hor / (all - ver * hor);
    return r ;
}
```

2) 卷积函数的实现

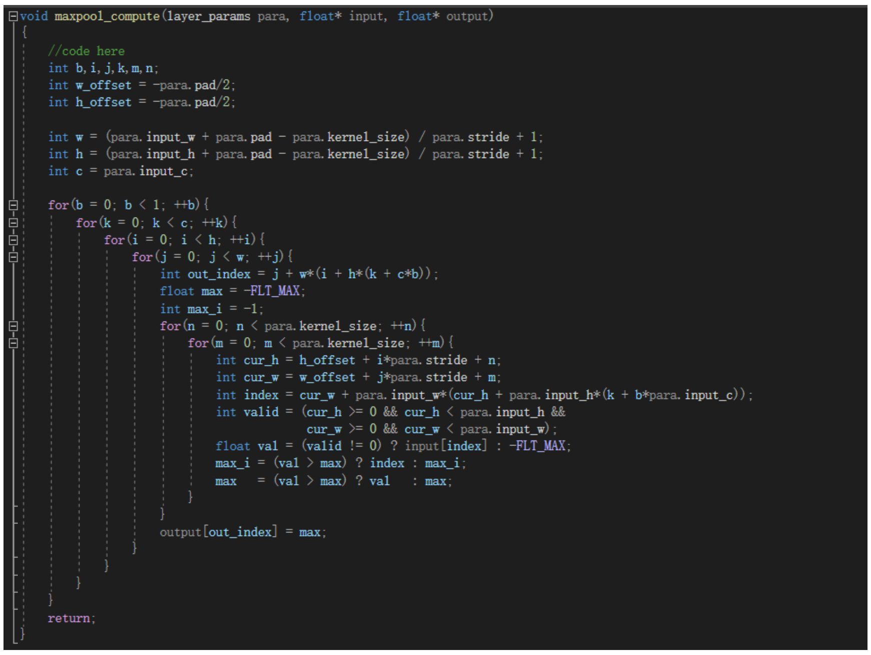


```c
void convolutional_compute(layer_params para, float* input, float* weight, float* output)
{


    //code here
    int i,j,k;
    int c_i,c_j,c_k;
    int pad = para.pad;
    int kernel_size = para.kernel_size;
    int stride = para.stride;

    //step1 padding
    float *tmp_pad;
    int size_of_tmpl;
    size_of_tmpl = (para.input_w + 2*pad) * (para.input_h + 2*pad) * para.input_c;
    tmp_pad = (float*)malloc(size_of_tmpl*sizeof(float));

    for (k=0;k<para.input_c;k++)
    for(j=0;j<para.input_h + 2*pad; j++)
        for(i=0;i<para.input_w + 2*pad; i++)
        {
        if(i<pad || j<pad || i>= (para.input_w + pad) || j >= (para.input_h + pad))
            tmp_pad[i + j*(para.input_w + 2*pad) + k * (para.input_w + 2*pad)*(para.input_h + 2*pad)] = 0.0;
        else
        {
            tmp_pad[i + j*(para.input_w + 2*pad) + k * (para.input_w + 2*pad)*(para.input_h + 2*pad)] = \
                    input[(i-pad) + (j-pad) * para.input_w + k * (para.input_w) * (para.input_h)];
                }
        }
```



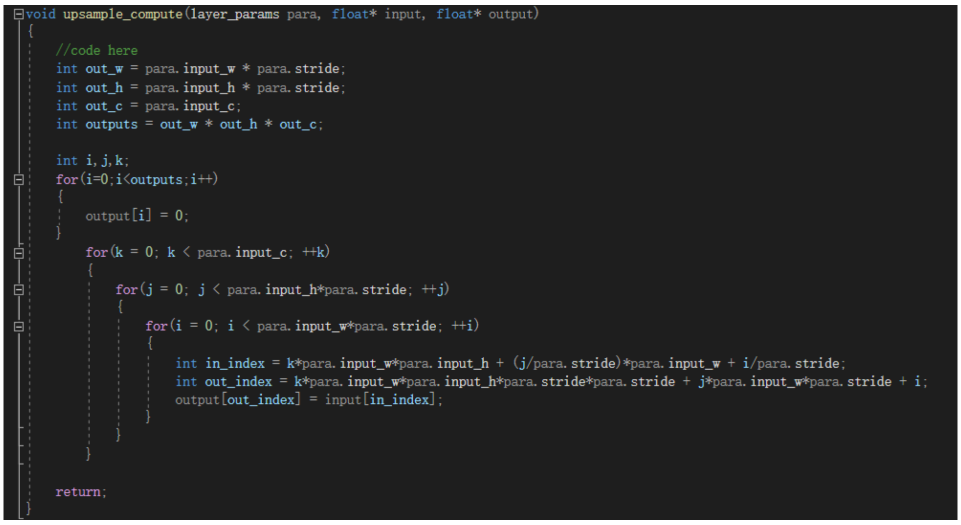
```
//step2 convolution
int padding_size_w = para.input_w + 2*pad;
int padding_size_h = para.input_h + 2*pad;
int padding_size_c = para.input_c;

int conv_size_w = (padding_size_w - kernel_size) / stride + 1;
int conv_size_h = (padding_size_h - kernel_size) / stride + 1;
int conv_size_c = para.kernel_n;

float conv_tmp;

for (k=0; k<conv_size_c; k++)
    for(j=0; j<conv_size_h; j++)
        for(i=0; i<conv_size_w; i++)
        {
            conv_tmp = 0;
            for (c_k=0; c_k<padding_size_c; c_k++)
                for(c_j=0; c_j<kernel_size; c_j++)
                    for(c_i=0; c_i<kernel_size; c_i++)
                    {
                        conv_tmp += tmp_pad[(i*stride + c_i) + (j*stride*padding_size_w + padding_size_w * c_j) + \
                                (padding_size_w * padding_size_h * c_k)] *
                                weight[c_i + c_j*kernel_size + c_k*kernel_size*kernel_size + k*kernel_size*kernel_size*padding_size_c];

                    }
            output[i + j*conv_size_w + k*conv_size_h*conv_size_w] = conv_tmp;
        }

free(tmp_pad);

return;
```

3) Maxpool 函数的实现



```
void maxpool_compute(layer_params para, float* input, float* output)
{
    //code here
    int b,i,j,k,m,n;
    int w_offset = -para.pad/2;
    int h_offset = -para.pad/2;

    int w = (para.input_w + para.pad - para.kernel_size) / para.stride + 1;
    int h = (para.input_h + para.pad - para.kernel_size) / para.stride + 1;
    int c = para.input_c;

    for(b = 0; b < 1; ++b){
        for(k = 0; k < c; ++k){
            for(i = 0; i < h; ++i){
                for(j = 0; j < w; ++j){
                    int out_index = j + w*(i + h*(k + c*b));
                    float max = -FLT_MAX;
                    int max_i = -1;
                    for(n = 0; n < para.kernel_size; ++n){
                        for(m = 0; m < para.kernel_size; ++m){
                            int cur_h = h_offset + i*para.stride + n;
                            int cur_w = w_offset + j*para.stride + m;
                            int index = cur_w + para.input_w*(cur_h + para.input_h*(k + b*para.input_c));
                            int valid = (cur_h >= 0 && cur_h < para.input_h &&
                                         cur_w >= 0 && cur_w < para.input_w);
                            float val = (valid != 0) ? input[index] : -FLT_MAX;
                            max_i = (val > max) ? index : max_i;
                            max   = (val > max) ? val   : max;
                        }
                    }
                    output[out_index] = max;
                }
            }
        }
    }
    return;
}
```

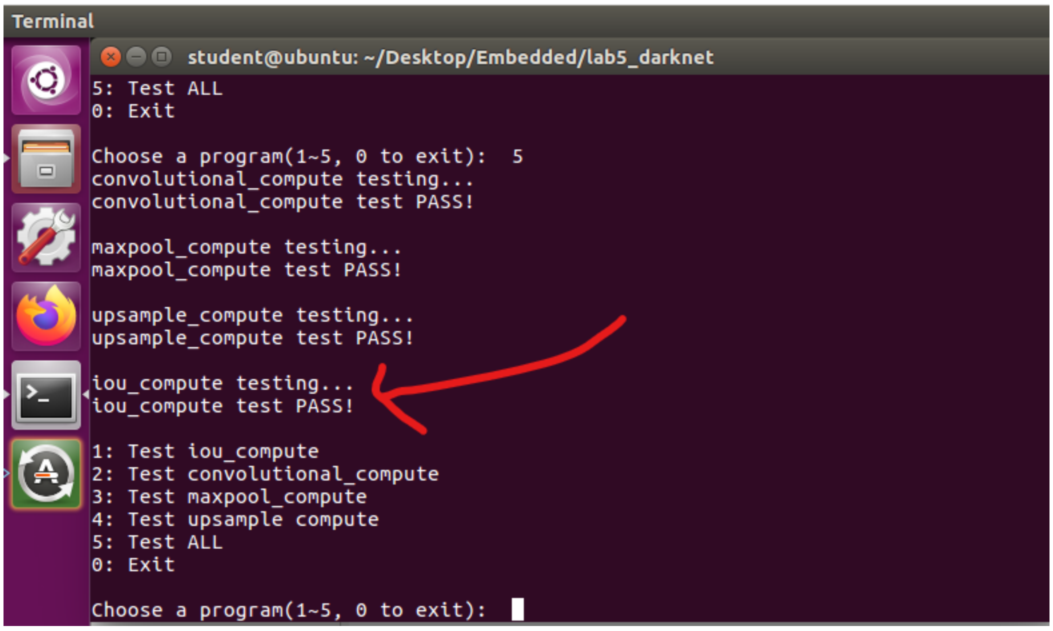
4) Upsample 函数的实现



```
void upsample_compute(layer_params para, float* input, float* output)
{
    //code here
    int out_w = para.input_w * para.stride;
    int out_h = para.input_h * para.stride;
    int out_c = para.input_c;
    int outputs = out_w * out_h * out_c;

    int i,j,k;
    for(i=0;i<outputs;i++)
    {
        output[i] = 0;
    }
        for(k = 0; k < para.input_c; ++k)
        {
            for(j = 0; j < para.input_h*para.stride; ++j)
            {
                for(i = 0; i < para.input_w*para.stride; ++i)
                {
                    int in_index = k*para.input_w*para.input_h + (j/para.stride)*para.input_w + i/para.stride;
                    int out_index = k*para.input_w*para.input_h*para.stride*para.stride + j*para.input_w*para.stride + i;
                    output[out_index] = input[in_index];
                }
            }
        }

    return;
}
```

5) 编写的函数在 work_test 上的测试结果

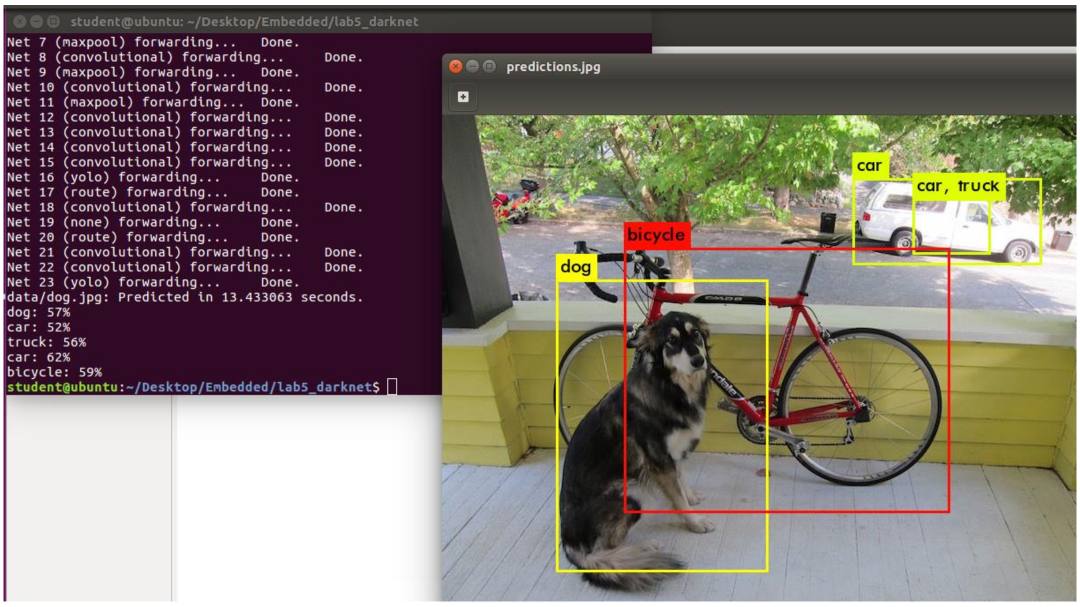
在 work 文件夹下编译，lab5_darknet 文件夹下运行可执行文件 work_test.



```
student@ubuntu: ~/Desktop/Embedded/lab5_darknet
5: Test ALL
0: Exit

Choose a program(1~5, 0 to exit):  5
convolutional_compute testing...
convolutional_compute test PASS!

maxpool_compute testing...
maxpool_compute test PASS!

upsample_compute testing...
upsample_compute test PASS!

iou_compute testing...
iou_compute test PASS!

1: Test iou_compute
2: Test convolutional_compute
3: Test maxpool_compute
4: Test upsample compute
5: Test ALL
0: Exit

Choose a program(1~5, 0 to exit): 
```

6) YOLO 算法在 x86 环境上的运行结果

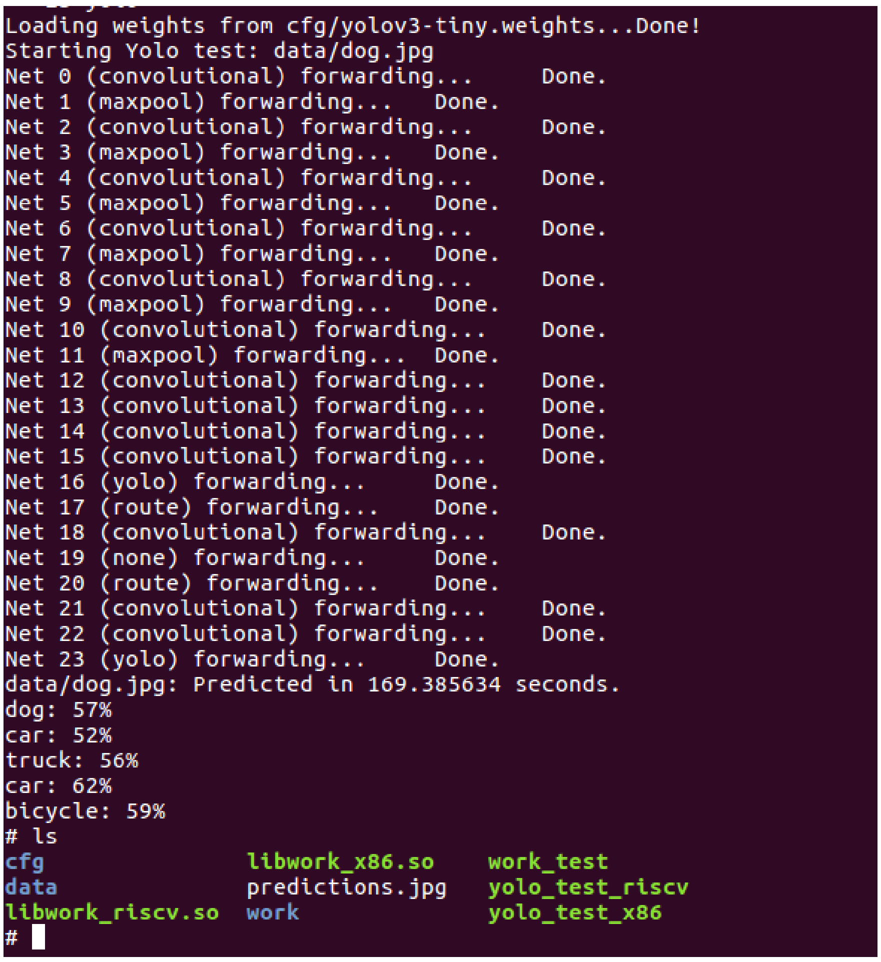
lab5_darknet 文件夹下运行可执行文件 yolo_test_x86.



```
student@ubuntu: ~/Desktop/Embedded/lab5_darknet
Net 7 (maxpool) forwarding...   Done.
Net 8 (convolutional) forwarding...     Done.
Net 9 (maxpool) forwarding...   Done.
Net 10 (convolutional) forwarding...    Done.
Net 11 (maxpool) forwarding...  Done.
Net 12 (convolutional) forwarding...    Done.
Net 13 (convolutional) forwarding...    Done.
Net 14 (convolutional) forwarding...    Done.
Net 15 (convolutional) forwarding...    Done.
Net 16 (yolo) forwarding...     Done.
Net 17 (route) forwarding...    Done.
Net 18 (convolutional) forwarding...    Done.
Net 19 (none) forwarding...     Done.
Net 20 (route) forwarding...    Done.
Net 21 (convolutional) forwarding...    Done.
Net 22 (convolutional) forwarding...    Done.
Net 23 (yolo) forwarding...     Done.
data/dog.jpg: Predicted in 13.433063 seconds.
dog: 57%
car: 52%
truck: 56%
car: 62%
bicycle: 59%
student@ubuntu:~/Desktop/Embedded/lab5_darknet$ 
```

7) YOLO 算法在 RISCV QEMU 上的运行结果

登入实验三搭载的 Linux 系统，使用 lftp 传输文件后运行即可



```
Loading weights from cfg/yolov3-tiny.weights...Done!
Starting Yolo test: data/dog.jpg
Net 0 (convolutional) forwarding...     Done.
Net 1 (maxpool) forwarding...   Done.
Net 2 (convolutional) forwarding...     Done.
Net 3 (maxpool) forwarding...   Done.
Net 4 (convolutional) forwarding...     Done.
Net 5 (maxpool) forwarding...   Done.
Net 6 (convolutional) forwarding...     Done.
Net 7 (maxpool) forwarding...   Done.
Net 8 (convolutional) forwarding...     Done.
Net 9 (maxpool) forwarding...   Done.
Net 10 (convolutional) forwarding...    Done.
Net 11 (maxpool) forwarding...  Done.
Net 12 (convolutional) forwarding...    Done.
Net 13 (convolutional) forwarding...    Done.
Net 14 (convolutional) forwarding...    Done.
Net 15 (convolutional) forwarding...    Done.
Net 16 (yolo) forwarding...     Done.
Net 17 (route) forwarding...    Done.
Net 18 (convolutional) forwarding...    Done.
Net 19 (none) forwarding...     Done.
Net 20 (route) forwarding...    Done.
Net 21 (convolutional) forwarding...    Done.
Net 22 (convolutional) forwarding...    Done.
Net 23 (yolo) forwarding...     Done.
data/dog.jpg: Predicted in 169.385634 seconds.
dog: 57%
car: 52%
truck: 56%
car: 62%
bicycle: 59%
# ls
cfg                 libwork_x86.so    work_test
data                predictions.jpg   yolo_test_riscv
libwork_riscv.so    work              yolo_test_x86
# 
```

3、实验分析和总结

1)：将交集分成水平方向和竖直方向两个层面去考虑，再区分出两个矩形的上下左右关系，就可以通过两个矩形“没有交集”、“包含”、“交叉”三种情形确定两个方向上的交集，相乘即得面积。

2)：首先在宽度为 w , h , c 的原始数据的长、宽维度上前后填充两个 pad 生成用于卷积的三维数组，之后逐个访问定下的卷积初始点，便可进行卷积。

3)：对于一个给定的数据样本“池”，找出其中的最大值，减少数据量

4)：将采样图片得到的每个像素点扩展成 stride * stride 数据，匹配卷积尺寸

5)：运行 Linux 系统时分配了 2G 的空间用来存储相关的资源，因为对图片的采样、卷积计算是非常消耗存储空间的，此外注意到在 Linux 系统上运行的时间明显长于在 Ubuntu 环境 x86 系统上的时间，表明仿真系统还是比实际系统在性能上有所欠缺。

4、实验收获、存在问题、改进措施或建议等

本次实验从了解 CNN 卷积神经网络的工作原理开始、自行编写了实现交并比函数和卷积函数的算法，并在 x86 和 Linux 环境中分别验证了其功能，由此对于“AI 识别”有了算法实现层面的理解。