**Yolo\_layer.c中**

layer make\_yolo\_layer

l.outputs = h\*w\*n\*(classes + 4 + 1 + 1);//添加一个角度预测

l.truth\_size = 4 + 2 +1; //truth\_size也要加一

l.c = n\*(classes + 4 + 1+2);//rotate\_yolo -2

ious delta\_yolo\_box

if (delta[index + 0 \* stride] || delta[index + 1 \* stride] || delta[index + 2 \* stride] || delta[index + 3 \* stride] || delta[index + 4 \* stride]) {

(\*rewritten\_bbox)++;

}//rotate

float tr1 = truth.r1;//rotate\_yolo

float tr2 = truth.r2;//rotate\_yolo

delta[index + 4 \* stride] += tr1 - x[index + 4 \* stride];//rotate\_yolo

delta[index + 5 \* stride] += tr2 - x[index + 5 \* stride];//rotate\_yolo

*printf*("tr1 = %f, pred1 = %f, delta\_angle1 = %f\n", tr1, x[index + 4 \* stride],delta[index + 4 \* stride]);//rotate\_yolo

*printf*("tr2 = %f, pred2 = %f, delta\_angle2 = %f\n", tr2, x[index + 5 \* stride], delta[index + 5 \* stride]);//rotate\_yolo

void averages\_yolo\_deltas

delta[box\_index + 4 \* stride] /= classes\_in\_one\_box;//rotate

delta[box\_index + 5 \* stride] /= classes\_in\_one\_box;//rotate

static int entry\_index

{

int n = location / (l.w\*l.h);

int loc = location % (l.w\*l.h);

return batch\*l.outputs + n\*l.w\*l.h\*(4+l.classes+1+2) + entry\*l.w\*l.h + loc;//rotate

}

void resize\_yolo\_layer(layer \*l, int w, int h)

l->outputs = h\*w\*l->n\*(l->classes + 4 + 1+1);//rotate //l->classes + 4 + 1

void forward\_yolo\_layer(const layer l, network\_state state)

const int class\_index = entry\_index(l, b, n\*l.w\*l.h + j\*l.w + i, 4+1 + 2);//rotate

const int obj\_index = entry\_index(l, b, n\*l.w\*l.h + j\*l.w + i, 4+2);//rotate

no\_iou\_loss\_delta[index + 4 \* stride] = 0;//rotate\_yolo

no\_iou\_loss\_delta[index + 5 \* stride] = 0;//rotate\_yolo

int class\_id = state.truth[t\*l.truth\_size + b\*l.truths + 6];//rotate\_yolo

int class\_index = entry\_index(l, b, n\*l.w\*l.h + j\*l.w + i, 4 + 2);//rotate\_yolo

int obj\_index = entry\_index(l, b, mask\_n\*l.w\*l.h + j\*l.w + i, 6);//rotate\_yolo

box get\_yolo\_box

b.r1 = x[index + 4 \* stride];//rotate\_yolo

b.r2 = x[index + 5 \* stride];//rotate\_yolo

void forward\_yolo\_layer\_gpu

index = entry\_index(l, b, n\*l.w\*l.h, 4+2);//rotate\_yolo

void forward\_yolo\_layer

index = entry\_index(l, b, n\*l.w\*l.h, 4+2);//rotate\_yolo

int get\_yolo\_detections

int obj\_index = entry\_index(l, 0, n\*l.w\*l.h + i, 4+2);//rotate\_yolo

int class\_index = entry\_index(l, 0, n\*l.w\*l.h + i, 4 + 1+2 + j);//rotate\_yolo

**data.c中添加**

#include<math.h>//rotate\_yolo

box\_label \*read\_boxes

float x, y, h, w, r1,r2;//rotate\_yolo

while (*fscanf*(file, "%d %f %f %f %f %f", &id, &x, &y, &w, &h, &r) == 6) {//rotate\_yolo

int fill\_truth\_detection

float x, y, w, h, r1,r2;//rotate\_yolo

r1 = boxes[i].r1;//rotate\_yolo

r2 = boxes[i].r2;//rotate\_yolo

truth[(i-sub)\*truth\_size +4] = r1;//rotate\_yolo

truth[(i - sub) \* truth\_size + 5] = r2;//rotate\_yolo

**darknet.h中添加**

typedef struct box {

float x, y, w, h,r1,r2;//rotate\_yolo

} box;

typedef struct box\_label {

int id;

int track\_id;

//float x, y, w, h;

float x, y, w, h, r1,r2;//rotate\_yolo

float left, right, top, bottom;

} box\_label;

**box.c文件中添加**

box float\_to\_box\_stride(float \*f, int stride)

{

box b = { 0 };

b.x = f[0];

b.y = f[1 \* stride];

b.w = f[2 \* stride];

b.h = f[3 \* stride];

b.r1 = f[4 \* stride];//rotate\_yolo

b.r2 = f[5 \* stride];//rotate\_yolo

return b;

}

**Image.c中添加**

image load\_yolo\_image\_gray(char\* filename, int w, int h)//rotate\_yolo

{

return load\_image(filename, w, h, 1);

}

image \*\*load\_alphabet()

{

int i, j;

const int nsize = 8;

image\*\* alphabets = (image\*\*)xcalloc(nsize, sizeof(image\*));

for(j = 0; j < nsize; ++j){

alphabets[j] = (image\*)xcalloc(128, sizeof(image));

for(i = 32; i < 127; ++i){

char buff[256];

*sprintf*(buff, "C:/Users/zheng/Desktop/GPU\_darknet-master/darknet-master/build/darknet/x64/data/labels/%d\_%d.png", i, j);

alphabets[j][i] = load\_image\_gray(buff, 0, 0);//rotate\_yolo

}

}

return alphabets;

}

**针对YOLOv4的角度预测修改：**

ious delta\_yolo\_box(box truth, float \*x, float \*biases, int n, int index, int i, int j, int lw, int lh, int w, int h, float \*delta, float scale, int stride, float iou\_normalizer, IOU\_LOSS iou\_loss, int accumulate, float max\_delta, int \*rewritten\_bbox)

{

if (delta[index + 0 \* stride] || delta[index + 1 \* stride] || delta[index + 2 \* stride] || delta[index + 3 \* stride] || delta[index + 4 \* stride] || delta[index + 5 \* stride]) {

(\*rewritten\_bbox)++;

}//rotate

ious all\_ious = { 0 };

// i - step in layer width

// j - step in layer height

// Returns a box in absolute coordinates

box pred = get\_yolo\_box(x, biases, n, index, i, j, lw, lh, w, h, stride);

all\_ious.iou = box\_iou(pred, truth);

all\_ious.giou = box\_giou(pred, truth);

all\_ious.diou = box\_diou(pred, truth);

all\_ious.ciou = box\_ciou(pred, truth);

// avoid nan in dx\_box\_iou

if (pred.w == 0) { pred.w = 1.0; }

if (pred.h == 0) { pred.h = 1.0; }

if (iou\_loss == MSE) // old loss

{

float tx = (truth.x\*lw - i);

float ty = (truth.y\*lh - j);

float tw = log(truth.w\*w / biases[2 \* n]);

float th = log(truth.h\*h / biases[2 \* n + 1]);

float tr1 = truth.r1;//rotate\_yolo

float tr2 = truth.r2;//rotate\_yolo

//printf(" tx = %f, ty = %f, tw = %f, th = %f \n", tx, ty, tw, th);

//printf(" x = %f, y = %f, w = %f, h = %f \n", x[index + 0 \* stride], x[index + 1 \* stride], x[index + 2 \* stride], x[index + 3 \* stride]);

// accumulate delta

delta[index + 0 \* stride] += scale \* (tx - x[index + 0 \* stride]) \* iou\_normalizer;

delta[index + 1 \* stride] += scale \* (ty - x[index + 1 \* stride]) \* iou\_normalizer;

delta[index + 2 \* stride] += scale \* (tw - x[index + 2 \* stride]) \* iou\_normalizer;

delta[index + 3 \* stride] += scale \* (th - x[index + 3 \* stride]) \* iou\_normalizer;

delta[index + 4 \* stride] += scale \* (tr1 - x[index + 4 \* stride]);//rotate\_yolo

delta[index + 5 \* stride] += scale \* (tr2 - x[index + 5 \* stride]);//rotate\_yolo

*printf*("tr1 = %f, pred1 = %f, delta\_cos = %f\n", tr1, x[index + 4 \* stride],delta[index + 4 \* stride]);//rotate\_yolo

*printf*("tr2 = %f, pred2 = %f, delta\_sin = %f\n", tr2, x[index + 5 \* stride], delta[index + 5 \* stride]);//rotate\_yolo

}

else {

// https://github.com/generalized-iou/g-darknet

// https://arxiv.org/abs/1902.09630v2

// https://giou.stanford.edu/

float tr1 = truth.r1;//rotate\_yolo

float tr2 = truth.r2;//rotate\_yolo

all\_ious.dx\_iou = dx\_box\_iou(pred, truth, iou\_loss);

// jacobian^t (transpose)

//float dx = (all\_ious.dx\_iou.dl + all\_ious.dx\_iou.dr);

//float dy = (all\_ious.dx\_iou.dt + all\_ious.dx\_iou.db);

//float dw = ((-0.5 \* all\_ious.dx\_iou.dl) + (0.5 \* all\_ious.dx\_iou.dr));

//float dh = ((-0.5 \* all\_ious.dx\_iou.dt) + (0.5 \* all\_ious.dx\_iou.db));

// jacobian^t (transpose)

float dx = all\_ious.dx\_iou.dt;

float dy = all\_ious.dx\_iou.db;

float dw = all\_ious.dx\_iou.dl;

float dh = all\_ious.dx\_iou.dr;

// predict exponential, apply gradient of e^delta\_t ONLY for w,h

dw \*= *exp*(x[index + 2 \* stride]);

dh \*= *exp*(x[index + 3 \* stride]);

// normalize iou weight

dx \*= iou\_normalizer;

dy \*= iou\_normalizer;

dw \*= iou\_normalizer;

dh \*= iou\_normalizer;

dx = fix\_nan\_inf(dx);

dy = fix\_nan\_inf(dy);

dw = fix\_nan\_inf(dw);

dh = fix\_nan\_inf(dh);

if (max\_delta != *FLT\_MAX*) {

dx = clip\_value(dx, max\_delta);

dy = clip\_value(dy, max\_delta);

dw = clip\_value(dw, max\_delta);

dh = clip\_value(dh, max\_delta);

}

if (!accumulate) {

delta[index + 0 \* stride] = 0;

delta[index + 1 \* stride] = 0;

delta[index + 2 \* stride] = 0;

delta[index + 3 \* stride] = 0;

}

// accumulate delta

delta[index + 0 \* stride] += dx;

delta[index + 1 \* stride] += dy;

delta[index + 2 \* stride] += dw;

delta[index + 3 \* stride] += dh;

delta[index + 4 \* stride] += scale \* (tr1 - x[index + 4 \* stride]);//rotate\_yolo

delta[index + 5 \* stride] += scale \* (tr2 - x[index + 5 \* stride]);//rotate\_yolo

*printf*("tr1 = %f, pred1 = %f, delta\_cos = %f\n", tr1, x[index + 4 \* stride], delta[index + 4 \* stride]);//rotate\_yolo

*printf*("tr2 = %f, pred2 = %f, delta\_sin = %f\n", tr2, x[index + 5 \* stride], delta[index + 5 \* stride]);//rotate\_yolo

}