#include "box.h"

#include "utils.h"

#include <stdio.h>

#include <math.h>

#include <stdlib.h>

#ifndef M\_PI

#define M\_PI 3.141592

#endif

box float\_to\_box(float \*f)

{

box b;

b.x = f[0];

b.y = f[1];

b.w = f[2];

b.h = f[3];

return b;

}

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];

return b;

}

dbox derivative(box a, box b)

{

dbox d;

d.dx = 0;

d.dw = 0;

d.dy = 0;

d.dh = 0;

d.dx = a.x < b.x ? 1.0 : -1.0;

d.dy = a.y < b.y ? 1.0 : -1.0;

d.dw = a.w < b.w ? 1.0 : -1.0;

d.dh = a.h < b.h ? 1.0 : -1.0;

return d;

}

/\*

dbox derivative(box a, box b)

{

dbox d;

d.dx = 0;

d.dw = 0;

float l1 = a.x - a.w/2;

float l2 = b.x - b.w/2;

if (l1 > l2){

d.dx -= 1;

d.dw += .5;

}

float r1 = a.x + a.w/2;

float r2 = b.x + b.w/2;

if(r1 < r2){

d.dx += 1;

d.dw += .5;

}

if (l1 > r2) {

d.dx = -1;

d.dw = 0;

}

if (r1 < l2){

d.dx = 1;

d.dw = 0;

}

d.dy = 0;

d.dh = 0;

float t1 = a.y - a.h/2;

float t2 = b.y - b.h/2;

if (t1 > t2){

d.dy -= 1;

d.dh += .5;

}

float b1 = a.y + a.h/2;

float b2 = b.y + b.h/2;

if(b1 < b2){

d.dy += 1;

d.dh += .5;

}

if (t1 > b2) {

d.dy = -1;

d.dh = 0;

}

if (b1 < t2){

d.dy = 1;

d.dh = 0;

}

return d;

}

\*/

// where c is the smallest box that fully encompases a and b

boxabs box\_c(box a, box b) {

boxabs ba = { 0 };

ba.top = fmin(a.y - a.h / 2, b.y - b.h / 2);

ba.bot = fmax(a.y + a.h / 2, b.y + b.h / 2);

ba.left = fmin(a.x - a.w / 2, b.x - b.w / 2);

ba.right = fmax(a.x + a.w / 2, b.x + b.w / 2);

return ba;

}

// representation from x, y, w, h to top, left, bottom, right

boxabs to\_tblr(box a) {

boxabs tblr = { 0 };

float t = a.y - (a.h / 2);

float b = a.y + (a.h / 2);

float l = a.x - (a.w / 2);

float r = a.x + (a.w / 2);

tblr.top = t;

tblr.bot = b;

tblr.left = l;

tblr.right = r;

return tblr;

}

float overlap(float x1, float w1, float x2, float w2)

{

float l1 = x1 - w1/2;

float l2 = x2 - w2/2;

float left = l1 > l2 ? l1 : l2;

float r1 = x1 + w1/2;

float r2 = x2 + w2/2;

float right = r1 < r2 ? r1 : r2;

return right - left;

}

float box\_intersection(box a, box b)

{

float w = overlap(a.x, a.w, b.x, b.w);

float h = overlap(a.y, a.h, b.y, b.h);

if(w < 0 || h < 0) return 0;

float area = w\*h;

return area;

}

float box\_union(box a, box b)

{

float i = box\_intersection(a, b);

float u = a.w\*a.h + b.w\*b.h - i;

return u;

}

float box\_iou\_kind(box a, box b, IOU\_LOSS iou\_kind)

{

//IOU, GIOU, MSE, DIOU, CIOU

switch(iou\_kind) {

case IOU: return box\_iou(a, b);

case GIOU: return box\_giou(a, b);

case DIOU: return box\_diou(a, b);

case CIOU: return box\_ciou(a, b);

}

return box\_iou(a, b);

}

float box\_iou(box a, box b)

{

//return box\_intersection(a, b)/box\_union(a, b);

float I = box\_intersection(a, b);

float U = box\_union(a, b);

if (I == 0 || U == 0) {

return 0;

}

return I / U;

}

float box\_giou(box a, box b)

{

boxabs ba = box\_c(a, b);

float w = ba.right - ba.left;

float h = ba.bot - ba.top;

float c = w\*h;

float iou = box\_iou(a, b);

if (c == 0) {

return iou;

}

float u = box\_union(a, b);

float giou\_term = (c - u) / c;

#ifdef DEBUG\_PRINTS

printf(" c: %f, u: %f, giou\_term: %f\n", c, u, giou\_term);

#endif

return iou - giou\_term;

}

// https://github.com/Zzh-tju/DIoU-darknet

// https://arxiv.org/abs/1911.08287

float box\_diou(box a, box b)

{

boxabs ba = box\_c(a, b);

float w = ba.right - ba.left;

float h = ba.bot - ba.top;

float c = w \* w + h \* h;

float iou = box\_iou(a, b);

if (c == 0) {

return iou;

}

float d = (a.x - b.x) \* (a.x - b.x) + (a.y - b.y) \* (a.y - b.y);

float u = pow(d / c, 0.6);

float diou\_term = u;

#ifdef DEBUG\_PRINTS

printf(" c: %f, u: %f, riou\_term: %f\n", c, u, diou\_term);

#endif

return iou - diou\_term;

}

float box\_diounms(box a, box b, float beta1)

{

boxabs ba = box\_c(a, b);

float w = ba.right - ba.left;

float h = ba.bot - ba.top;

float c = w \* w + h \* h;

float iou = box\_iou(a, b);

if (c == 0) {

return iou;

}

float d = (a.x - b.x) \* (a.x - b.x) + (a.y - b.y) \* (a.y - b.y);

float u = pow(d / c, beta1);

float diou\_term = u;

#ifdef DEBUG\_PRINTS

printf(" c: %f, u: %f, riou\_term: %f\n", c, u, diou\_term);

#endif

return iou - diou\_term;

}

// https://github.com/Zzh-tju/DIoU-darknet

// https://arxiv.org/abs/1911.08287

float box\_ciou(box a, box b)

{

boxabs ba = box\_c(a, b);

float w = ba.right - ba.left;

float h = ba.bot - ba.top;

float c = w \* w + h \* h;

float iou = box\_iou(a, b);

if (c == 0) {

return iou;

}

float u = (a.x - b.x) \* (a.x - b.x) + (a.y - b.y) \* (a.y - b.y);

float d = u / c;

float ar\_gt = b.w / b.h;

float ar\_pred = a.w / a.h;

float ar\_loss = 4 / (M\_PI \* M\_PI) \* (atan(ar\_gt) - atan(ar\_pred)) \* (atan(ar\_gt) - atan(ar\_pred));

float alpha = ar\_loss / (1 - iou + ar\_loss + 0.000001);

float ciou\_term = d + alpha \* ar\_loss; //ciou

#ifdef DEBUG\_PRINTS

printf(" c: %f, u: %f, riou\_term: %f\n", c, u, ciou\_term);

#endif

return iou - ciou\_term;

}

dxrep dx\_box\_iou(box pred, box truth, IOU\_LOSS iou\_loss) {

boxabs pred\_tblr = to\_tblr(pred);

float pred\_t = fmin(pred\_tblr.top, pred\_tblr.bot);

float pred\_b = fmax(pred\_tblr.top, pred\_tblr.bot);

float pred\_l = fmin(pred\_tblr.left, pred\_tblr.right);

float pred\_r = fmax(pred\_tblr.left, pred\_tblr.right);

//dbox dover = derivative(pred,truth);

//dbox diouu = diou(pred, truth);

boxabs truth\_tblr = to\_tblr(truth);

#ifdef DEBUG\_PRINTS

printf("\niou: %f, giou: %f\n", box\_iou(pred, truth), box\_giou(pred, truth));

printf("pred: x,y,w,h: (%f, %f, %f, %f) -> t,b,l,r: (%f, %f, %f, %f)\n", pred.x, pred.y, pred.w, pred.h, pred\_tblr.top, pred\_tblr.bot, pred\_tblr.left, pred\_tblr.right);

printf("truth: x,y,w,h: (%f, %f, %f, %f) -> t,b,l,r: (%f, %f, %f, %f)\n", truth.x, truth.y, truth.w, truth.h, truth\_tblr.top, truth\_tblr.bot, truth\_tblr.left, truth\_tblr.right);

#endif

//printf("pred (t,b,l,r): (%f, %f, %f, %f)\n", pred\_t, pred\_b, pred\_l, pred\_r);

//printf("trut (t,b,l,r): (%f, %f, %f, %f)\n", truth\_tblr.top, truth\_tblr.bot, truth\_tblr.left, truth\_tblr.right);

dxrep ddx = {0};

float X = (pred\_b - pred\_t) \* (pred\_r - pred\_l);

float Xhat = (truth\_tblr.bot - truth\_tblr.top) \* (truth\_tblr.right - truth\_tblr.left);

float Ih = fmin(pred\_b, truth\_tblr.bot) - fmax(pred\_t, truth\_tblr.top);

float Iw = fmin(pred\_r, truth\_tblr.right) - fmax(pred\_l, truth\_tblr.left);

float I = Iw \* Ih;

float U = X + Xhat - I;

float S = (pred.x-truth.x)\*(pred.x-truth.x)+(pred.y-truth.y)\*(pred.y-truth.y);

float giou\_Cw = fmax(pred\_r, truth\_tblr.right) - fmin(pred\_l, truth\_tblr.left);

float giou\_Ch = fmax(pred\_b, truth\_tblr.bot) - fmin(pred\_t, truth\_tblr.top);

float giou\_C = giou\_Cw \* giou\_Ch;

//float IoU = I / U;

//#ifdef DEBUG\_PRINTS

//printf("X: %f", X);

//printf(", Xhat: %f", Xhat);

//printf(", Ih: %f", Ih);

//printf(", Iw: %f", Iw);

//printf(", I: %f", I);

//printf(", U: %f", U);

//printf(", IoU: %f\n", I / U);

//#endif

//Partial Derivatives, derivatives

float dX\_wrt\_t = -1 \* (pred\_r - pred\_l);

float dX\_wrt\_b = pred\_r - pred\_l;

float dX\_wrt\_l = -1 \* (pred\_b - pred\_t);

float dX\_wrt\_r = pred\_b - pred\_t;

// UNUSED

//// Ground truth

//float dXhat\_wrt\_t = -1 \* (truth\_tblr.right - truth\_tblr.left);

//float dXhat\_wrt\_b = truth\_tblr.right - truth\_tblr.left;

//float dXhat\_wrt\_l = -1 \* (truth\_tblr.bot - truth\_tblr.top);

//float dXhat\_wrt\_r = truth\_tblr.bot - truth\_tblr.top;

// gradient of I min/max in IoU calc (prediction)

float dI\_wrt\_t = pred\_t > truth\_tblr.top ? (-1 \* Iw) : 0;

float dI\_wrt\_b = pred\_b < truth\_tblr.bot ? Iw : 0;

float dI\_wrt\_l = pred\_l > truth\_tblr.left ? (-1 \* Ih) : 0;

float dI\_wrt\_r = pred\_r < truth\_tblr.right ? Ih : 0;

// derivative of U with regard to x

float dU\_wrt\_t = dX\_wrt\_t - dI\_wrt\_t;

float dU\_wrt\_b = dX\_wrt\_b - dI\_wrt\_b;

float dU\_wrt\_l = dX\_wrt\_l - dI\_wrt\_l;

float dU\_wrt\_r = dX\_wrt\_r - dI\_wrt\_r;

// gradient of C min/max in IoU calc (prediction)

float dC\_wrt\_t = pred\_t < truth\_tblr.top ? (-1 \* giou\_Cw) : 0;

float dC\_wrt\_b = pred\_b > truth\_tblr.bot ? giou\_Cw : 0;

float dC\_wrt\_l = pred\_l < truth\_tblr.left ? (-1 \* giou\_Ch) : 0;

float dC\_wrt\_r = pred\_r > truth\_tblr.right ? giou\_Ch : 0;

float p\_dt = 0;

float p\_db = 0;

float p\_dl = 0;

float p\_dr = 0;

if (U > 0 ) {

p\_dt = ((U \* dI\_wrt\_t) - (I \* dU\_wrt\_t)) / (U \* U);

p\_db = ((U \* dI\_wrt\_b) - (I \* dU\_wrt\_b)) / (U \* U);

p\_dl = ((U \* dI\_wrt\_l) - (I \* dU\_wrt\_l)) / (U \* U);

p\_dr = ((U \* dI\_wrt\_r) - (I \* dU\_wrt\_r)) / (U \* U);

}

// apply grad from prediction min/max for correct corner selection

p\_dt = pred\_tblr.top < pred\_tblr.bot ? p\_dt : p\_db;

p\_db = pred\_tblr.top < pred\_tblr.bot ? p\_db : p\_dt;

p\_dl = pred\_tblr.left < pred\_tblr.right ? p\_dl : p\_dr;

p\_dr = pred\_tblr.left < pred\_tblr.right ? p\_dr : p\_dl;

if (iou\_loss == GIOU) {

if (giou\_C > 0) {

// apply "C" term from gIOU

p\_dt += ((giou\_C \* dU\_wrt\_t) - (U \* dC\_wrt\_t)) / (giou\_C \* giou\_C);

p\_db += ((giou\_C \* dU\_wrt\_b) - (U \* dC\_wrt\_b)) / (giou\_C \* giou\_C);

p\_dl += ((giou\_C \* dU\_wrt\_l) - (U \* dC\_wrt\_l)) / (giou\_C \* giou\_C);

p\_dr += ((giou\_C \* dU\_wrt\_r) - (U \* dC\_wrt\_r)) / (giou\_C \* giou\_C);

}

if (Iw<=0||Ih<=0) {

p\_dt = ((giou\_C \* dU\_wrt\_t) - (U \* dC\_wrt\_t)) / (giou\_C \* giou\_C);

p\_db = ((giou\_C \* dU\_wrt\_b) - (U \* dC\_wrt\_b)) / (giou\_C \* giou\_C);

p\_dl = ((giou\_C \* dU\_wrt\_l) - (U \* dC\_wrt\_l)) / (giou\_C \* giou\_C);

p\_dr = ((giou\_C \* dU\_wrt\_r) - (U \* dC\_wrt\_r)) / (giou\_C \* giou\_C);

}

}

float Ct = fmin(pred.y - pred.h / 2,truth.y - truth.h / 2);

float Cb = fmax(pred.y + pred.h / 2,truth.y + truth.h / 2);

float Cl = fmin(pred.x - pred.w / 2,truth.x - truth.w / 2);

float Cr = fmax(pred.x + pred.w / 2,truth.x + truth.w / 2);

float Cw = Cr - Cl;

float Ch = Cb - Ct;

float C = Cw \* Cw + Ch \* Ch;

float dCt\_dx = 0;

float dCt\_dy = pred\_t < truth\_tblr.top ? 1 : 0;

float dCt\_dw = 0;

float dCt\_dh = pred\_t < truth\_tblr.top ? -0.5 : 0;

float dCb\_dx = 0;

float dCb\_dy = pred\_b > truth\_tblr.bot ? 1 : 0;

float dCb\_dw = 0;

float dCb\_dh = pred\_b > truth\_tblr.bot ? 0.5: 0;

float dCl\_dx = pred\_l < truth\_tblr.left ? 1 : 0;

float dCl\_dy = 0;

float dCl\_dw = pred\_l < truth\_tblr.left ? -0.5 : 0;

float dCl\_dh = 0;

float dCr\_dx = pred\_r > truth\_tblr.right ? 1 : 0;

float dCr\_dy = 0;

float dCr\_dw = pred\_r > truth\_tblr.right ? 0.5 : 0;

float dCr\_dh = 0;

float dCw\_dx = dCr\_dx - dCl\_dx;

float dCw\_dy = dCr\_dy - dCl\_dy;

float dCw\_dw = dCr\_dw - dCl\_dw;

float dCw\_dh = dCr\_dh - dCl\_dh;

float dCh\_dx = dCb\_dx - dCt\_dx;

float dCh\_dy = dCb\_dy - dCt\_dy;

float dCh\_dw = dCb\_dw - dCt\_dw;

float dCh\_dh = dCb\_dh - dCt\_dh;

// UNUSED

//// ground truth

//float dI\_wrt\_xhat\_t = pred\_t < truth\_tblr.top ? (-1 \* Iw) : 0;

//float dI\_wrt\_xhat\_b = pred\_b > truth\_tblr.bot ? Iw : 0;

//float dI\_wrt\_xhat\_l = pred\_l < truth\_tblr.left ? (-1 \* Ih) : 0;

//float dI\_wrt\_xhat\_r = pred\_r > truth\_tblr.right ? Ih : 0;

// Final IOU loss (prediction) (negative of IOU gradient, we want the negative loss)

float p\_dx = 0;

float p\_dy = 0;

float p\_dw = 0;

float p\_dh = 0;

p\_dx = p\_dl + p\_dr; //p\_dx, p\_dy, p\_dw and p\_dh are the gradient of IoU or GIoU.

p\_dy = p\_dt + p\_db;

p\_dw = (p\_dr - p\_dl); //For dw and dh, we do not divided by 2.

p\_dh = (p\_db - p\_dt);

// https://github.com/Zzh-tju/DIoU-darknet

// https://arxiv.org/abs/1911.08287

if (iou\_loss == DIOU) {

if (C > 0) {

p\_dx += (2\*(truth.x-pred.x)\*C-(2\*Cw\*dCw\_dx+2\*Ch\*dCh\_dx)\*S) / (C \* C);

p\_dy += (2\*(truth.y-pred.y)\*C-(2\*Cw\*dCw\_dy+2\*Ch\*dCh\_dy)\*S) / (C \* C);

p\_dw += (2\*Cw\*dCw\_dw+2\*Ch\*dCh\_dw)\*S / (C \* C);

p\_dh += (2\*Cw\*dCw\_dh+2\*Ch\*dCh\_dh)\*S / (C \* C);

}

if (Iw<=0||Ih<=0){

p\_dx = (2\*(truth.x-pred.x)\*C-(2\*Cw\*dCw\_dx+2\*Ch\*dCh\_dx)\*S) / (C \* C);

p\_dy = (2\*(truth.y-pred.y)\*C-(2\*Cw\*dCw\_dy+2\*Ch\*dCh\_dy)\*S) / (C \* C);

p\_dw = (2\*Cw\*dCw\_dw+2\*Ch\*dCh\_dw)\*S / (C \* C);

p\_dh = (2\*Cw\*dCw\_dh+2\*Ch\*dCh\_dh)\*S / (C \* C);

}

}

//The following codes are calculating the gradient of ciou.

if (iou\_loss == CIOU) {

float ar\_gt = truth.w / truth.h;

float ar\_pred = pred.w / pred.h;

float ar\_loss = 4 / (M\_PI \* M\_PI) \* (atan(ar\_gt) - atan(ar\_pred)) \* (atan(ar\_gt) - atan(ar\_pred));

float alpha = ar\_loss / (1 - I/U + ar\_loss + 0.000001);

float ar\_dw=8/(M\_PI\*M\_PI)\*(atan(ar\_gt)-atan(ar\_pred))\*pred.h;

float ar\_dh=-8/(M\_PI\*M\_PI)\*(atan(ar\_gt)-atan(ar\_pred))\*pred.w;

if (C > 0) {

// dar\*

p\_dx += (2\*(truth.x-pred.x)\*C-(2\*Cw\*dCw\_dx+2\*Ch\*dCh\_dx)\*S) / (C \* C);

p\_dy += (2\*(truth.y-pred.y)\*C-(2\*Cw\*dCw\_dy+2\*Ch\*dCh\_dy)\*S) / (C \* C);

p\_dw += (2\*Cw\*dCw\_dw+2\*Ch\*dCh\_dw)\*S / (C \* C) + alpha \* ar\_dw;

p\_dh += (2\*Cw\*dCw\_dh+2\*Ch\*dCh\_dh)\*S / (C \* C) + alpha \* ar\_dh;

}

if (Iw<=0||Ih<=0){

p\_dx = (2\*(truth.x-pred.x)\*C-(2\*Cw\*dCw\_dx+2\*Ch\*dCh\_dx)\*S) / (C \* C);

p\_dy = (2\*(truth.y-pred.y)\*C-(2\*Cw\*dCw\_dy+2\*Ch\*dCh\_dy)\*S) / (C \* C);

p\_dw = (2\*Cw\*dCw\_dw+2\*Ch\*dCh\_dw)\*S / (C \* C) + alpha \* ar\_dw;

p\_dh = (2\*Cw\*dCw\_dh+2\*Ch\*dCh\_dh)\*S / (C \* C) + alpha \* ar\_dh;

}

}

ddx.dt = p\_dx; //We follow the original code released from GDarknet. So in yolo\_layer.c, dt, db, dl, dr are already dx, dy, dw, dh.

ddx.db = p\_dy;

ddx.dl = p\_dw;

ddx.dr = p\_dh;

// UNUSED

//// ground truth

//float gt\_dt = ((U \* dI\_wrt\_xhat\_t) - (I \* (dXhat\_wrt\_t - dI\_wrt\_xhat\_t))) / (U \* U);

//float gt\_db = ((U \* dI\_wrt\_xhat\_b) - (I \* (dXhat\_wrt\_b - dI\_wrt\_xhat\_b))) / (U \* U);

//float gt\_dl = ((U \* dI\_wrt\_xhat\_l) - (I \* (dXhat\_wrt\_l - dI\_wrt\_xhat\_l))) / (U \* U);

//float gt\_dr = ((U \* dI\_wrt\_xhat\_r) - (I \* (dXhat\_wrt\_r - dI\_wrt\_xhat\_r))) / (U \* U);

// no min/max grad applied

//dx.dt = dt;

//dx.db = db;

//dx.dl = dl;

//dx.dr = dr;

//// sum in gt -- THIS DOESNT WORK

//dx.dt += gt\_dt;

//dx.db += gt\_db;

//dx.dl += gt\_dl;

//dx.dr += gt\_dr;

//// instead, look at the change between pred and gt, and weight t/b/l/r appropriately...

//// need the real derivative here (I think?)

//float delta\_t = fmax(truth\_tblr.top, pred\_t) - fmin(truth\_tblr.top, pred\_t);

//float delta\_b = fmax(truth\_tblr.bot, pred\_b) - fmin(truth\_tblr.bot, pred\_b);

//float delta\_l = fmax(truth\_tblr.left, pred\_l) - fmin(truth\_tblr.left, pred\_l);

//float delta\_r = fmax(truth\_tblr.right, pred\_r) - fmin(truth\_tblr.right, pred\_r);

//dx.dt \*= delta\_t / (delta\_t + delta\_b);

//dx.db \*= delta\_b / (delta\_t + delta\_b);

//dx.dl \*= delta\_l / (delta\_l + delta\_r);

//dx.dr \*= delta\_r / (delta\_l + delta\_r);

// UNUSED

//// ground truth

//float gt\_dt = ((U \* dI\_wrt\_xhat\_t) - (I \* (dXhat\_wrt\_t - dI\_wrt\_xhat\_t))) / (U \* U);

//float gt\_db = ((U \* dI\_wrt\_xhat\_b) - (I \* (dXhat\_wrt\_b - dI\_wrt\_xhat\_b))) / (U \* U);

//float gt\_dl = ((U \* dI\_wrt\_xhat\_l) - (I \* (dXhat\_wrt\_l - dI\_wrt\_xhat\_l))) / (U \* U);

//float gt\_dr = ((U \* dI\_wrt\_xhat\_r) - (I \* (dXhat\_wrt\_r - dI\_wrt\_xhat\_r))) / (U \* U);

// no min/max grad applied

//dx.dt = dt;

//dx.db = db;

//dx.dl = dl;

//dx.dr = dr;

// apply grad from prediction min/max for correct corner selection

//dx.dt = pred\_tblr.top < pred\_tblr.bot ? p\_dt : p\_db;

//dx.db = pred\_tblr.top < pred\_tblr.bot ? p\_db : p\_dt;

//dx.dl = pred\_tblr.left < pred\_tblr.right ? p\_dl : p\_dr;

//dx.dr = pred\_tblr.left < pred\_tblr.right ? p\_dr : p\_dl;

//// sum in gt -- THIS DOESNT WORK

//dx.dt += gt\_dt;

//dx.db += gt\_db;

//dx.dl += gt\_dl;

//dx.dr += gt\_dr;

//// instead, look at the change between pred and gt, and weight t/b/l/r appropriately...

//// need the real derivative here (I think?)

//float delta\_t = fmax(truth\_tblr.top, pred\_t) - fmin(truth\_tblr.top, pred\_t);

//float delta\_b = fmax(truth\_tblr.bot, pred\_b) - fmin(truth\_tblr.bot, pred\_b);

//float delta\_l = fmax(truth\_tblr.left, pred\_l) - fmin(truth\_tblr.left, pred\_l);

//float delta\_r = fmax(truth\_tblr.right, pred\_r) - fmin(truth\_tblr.right, pred\_r);

//dx.dt \*= delta\_t / (delta\_t + delta\_b);

//dx.db \*= delta\_b / (delta\_t + delta\_b);

//dx.dl \*= delta\_l / (delta\_l + delta\_r);

//dx.dr \*= delta\_r / (delta\_l + delta\_r);

//#ifdef DEBUG\_PRINTS

/\*printf(" directions dt: ");

if ((pred\_tblr.top < truth\_tblr.top && dx.dt > 0) || (pred\_tblr.top > truth\_tblr.top && dx.dt < 0)) {

printf("✓");

} else {

printf("𝒙");

}

printf(", ");

if ((pred\_tblr.bot < truth\_tblr.bot && dx.db > 0) || (pred\_tblr.bot > truth\_tblr.bot && dx.db < 0)) {

printf("✓");

} else {

printf("𝒙");

}

printf(", ");

if ((pred\_tblr.left < truth\_tblr.left && dx.dl > 0) || (pred\_tblr.left > truth\_tblr.left && dx.dl < 0)) {

printf("✓");

} else {

printf("𝒙");

}

printf(", ");

if ((pred\_tblr.right < truth\_tblr.right && dx.dr > 0) || (pred\_tblr.right > truth\_tblr.right && dx.dr < 0)) {

printf("✓");

} else {

printf("𝒙");

}

printf("\n");

printf("dx dt:%f", dx.dt);

printf(", db: %f", dx.db);

printf(", dl: %f", dx.dl);

printf(", dr: %f | ", dx.dr);

#endif

#ifdef DEBUG\_NAN

if (isnan(dx.dt)) { printf("dt isnan\n"); }

if (isnan(dx.db)) { printf("db isnan\n"); }

if (isnan(dx.dl)) { printf("dl isnan\n"); }

if (isnan(dx.dr)) { printf("dr isnan\n"); }

#endif

// // No update if 0 or nan

// if (dx.dt == 0 || isnan(dx.dt)) { dx.dt = 1; }

// if (dx.db == 0 || isnan(dx.db)) { dx.db = 1; }

// if (dx.dl == 0 || isnan(dx.dl)) { dx.dl = 1; }

// if (dx.dr == 0 || isnan(dx.dr)) { dx.dr = 1; }

//

//#ifdef DEBUG\_PRINTS

// printf("dx dt:%f (t: %f, p: %f)", dx.dt, gt\_dt, p\_dt);

// printf(", db: %f (t: %f, p: %f)", dx.db, gt\_db, p\_db);

// printf(", dl: %f (t: %f, p: %f)", dx.dl, gt\_dl, p\_dl);

// printf(", dr: %f (t: %f, p: %f) | ", dx.dr, gt\_dr, p\_dr);

//#endif \*/

return ddx;

}

float box\_rmse(box a, box b)

{

return sqrt(pow(a.x-b.x, 2) +

pow(a.y-b.y, 2) +

pow(a.w-b.w, 2) +

pow(a.h-b.h, 2));

}

dbox dintersect(box a, box b)

{

float w = overlap(a.x, a.w, b.x, b.w);

float h = overlap(a.y, a.h, b.y, b.h);

dbox dover = derivative(a, b);

dbox di;

di.dw = dover.dw\*h;

di.dx = dover.dx\*h;

di.dh = dover.dh\*w;

di.dy = dover.dy\*w;

return di;

}

dbox dunion(box a, box b)

{

dbox du;

dbox di = dintersect(a, b);

du.dw = a.h - di.dw;

du.dh = a.w - di.dh;

du.dx = -di.dx;

du.dy = -di.dy;

return du;

}

void test\_dunion()

{

box a = {0, 0, 1, 1};

box dxa= {0+.0001, 0, 1, 1};

box dya= {0, 0+.0001, 1, 1};

box dwa= {0, 0, 1+.0001, 1};

box dha= {0, 0, 1, 1+.0001};

box b = {.5, .5, .2, .2};

dbox di = dunion(a,b);

printf("Union: %f %f %f %f\n", di.dx, di.dy, di.dw, di.dh);

float inter = box\_union(a, b);

float xinter = box\_union(dxa, b);

float yinter = box\_union(dya, b);

float winter = box\_union(dwa, b);

float hinter = box\_union(dha, b);

xinter = (xinter - inter)/(.0001);

yinter = (yinter - inter)/(.0001);

winter = (winter - inter)/(.0001);

hinter = (hinter - inter)/(.0001);

printf("Union Manual %f %f %f %f\n", xinter, yinter, winter, hinter);

}

void test\_dintersect()

{

box a = {0, 0, 1, 1};

box dxa= {0+.0001, 0, 1, 1};

box dya= {0, 0+.0001, 1, 1};

box dwa= {0, 0, 1+.0001, 1};

box dha= {0, 0, 1, 1+.0001};

box b = {.5, .5, .2, .2};

dbox di = dintersect(a,b);

printf("Inter: %f %f %f %f\n", di.dx, di.dy, di.dw, di.dh);

float inter = box\_intersection(a, b);

float xinter = box\_intersection(dxa, b);

float yinter = box\_intersection(dya, b);

float winter = box\_intersection(dwa, b);

float hinter = box\_intersection(dha, b);

xinter = (xinter - inter)/(.0001);

yinter = (yinter - inter)/(.0001);

winter = (winter - inter)/(.0001);

hinter = (hinter - inter)/(.0001);

printf("Inter Manual %f %f %f %f\n", xinter, yinter, winter, hinter);

}

void test\_box()

{

test\_dintersect();

test\_dunion();

box a = {0, 0, 1, 1};

box dxa= {0+.00001, 0, 1, 1};

box dya= {0, 0+.00001, 1, 1};

box dwa= {0, 0, 1+.00001, 1};

box dha= {0, 0, 1, 1+.00001};

box b = {.5, 0, .2, .2};

float iou = box\_iou(a,b);

iou = (1-iou)\*(1-iou);

printf("%f\n", iou);

dbox d = diou(a, b);

printf("%f %f %f %f\n", d.dx, d.dy, d.dw, d.dh);

float xiou = box\_iou(dxa, b);

float yiou = box\_iou(dya, b);

float wiou = box\_iou(dwa, b);

float hiou = box\_iou(dha, b);

xiou = ((1-xiou)\*(1-xiou) - iou)/(.00001);

yiou = ((1-yiou)\*(1-yiou) - iou)/(.00001);

wiou = ((1-wiou)\*(1-wiou) - iou)/(.00001);

hiou = ((1-hiou)\*(1-hiou) - iou)/(.00001);

printf("manual %f %f %f %f\n", xiou, yiou, wiou, hiou);

}

dbox diou(box a, box b)

{

float u = box\_union(a, b);

float i = box\_intersection(a, b);

dbox di = dintersect(a, b);

dbox du = dunion(a, b);

dbox dd = { 0,0,0,0 };

if (i <= 0 || 1) {

dd.dx = b.x - a.x;

dd.dy = b.y - a.y;

dd.dw = b.w - a.w;

dd.dh = b.h - a.h;

return dd;

}

dd.dx = (di.dx\*u - du.dx\*i) / (u\*u);

dd.dy = (di.dy\*u - du.dy\*i) / (u\*u);

dd.dw = (di.dw\*u - du.dw\*i) / (u\*u);

dd.dh = (di.dh\*u - du.dh\*i) / (u\*u);

return dd;

}

typedef struct{

int index;

int class\_id;

float \*\*probs;

} sortable\_bbox;

int nms\_comparator(const void \*pa, const void \*pb)

{

sortable\_bbox a = \*(sortable\_bbox \*)pa;

sortable\_bbox b = \*(sortable\_bbox \*)pb;

float diff = a.probs[a.index][b.class\_id] - b.probs[b.index][b.class\_id];

if(diff < 0) return 1;

else if(diff > 0) return -1;

return 0;

}

void do\_nms\_sort\_v2(box \*boxes, float \*\*probs, int total, int classes, float thresh)

{

int i, j, k;

sortable\_bbox\* s = (sortable\_bbox\*)xcalloc(total, sizeof(sortable\_bbox));

for(i = 0; i < total; ++i){

s[i].index = i;

s[i].class\_id = 0;

s[i].probs = probs;

}

for(k = 0; k < classes; ++k){

for(i = 0; i < total; ++i){

s[i].class\_id = k;

}

qsort(s, total, sizeof(sortable\_bbox), nms\_comparator);

for(i = 0; i < total; ++i){

if(probs[s[i].index][k] == 0) continue;

box a = boxes[s[i].index];

for(j = i+1; j < total; ++j){

box b = boxes[s[j].index];

if (box\_iou(a, b) > thresh){

probs[s[j].index][k] = 0;

}

}

}

}

free(s);

}

int nms\_comparator\_v3(const void \*pa, const void \*pb)

{

detection a = \*(detection \*)pa;

detection b = \*(detection \*)pb;

float diff = 0;

if (b.sort\_class >= 0) {

diff = a.prob[b.sort\_class] - b.prob[b.sort\_class]; // there is already: prob = objectness\*prob

}

else {

diff = a.objectness - b.objectness;

}

if (diff < 0) return 1;

else if (diff > 0) return -1;

return 0;

}

void do\_nms\_obj(detection \*dets, int total, int classes, float thresh)

{

int i, j, k;

k = total - 1;

for (i = 0; i <= k; ++i) {

if (dets[i].objectness == 0) {

detection swap = dets[i];

dets[i] = dets[k];

dets[k] = swap;

--k;

--i;

}

}

total = k + 1;

for (i = 0; i < total; ++i) {

dets[i].sort\_class = -1;

}

qsort(dets, total, sizeof(detection), nms\_comparator\_v3);

for (i = 0; i < total; ++i) {

if (dets[i].objectness == 0) continue;

box a = dets[i].bbox;

for (j = i + 1; j < total; ++j) {

if (dets[j].objectness == 0) continue;

box b = dets[j].bbox;

if (box\_iou(a, b) > thresh) {

dets[j].objectness = 0;

for (k = 0; k < classes; ++k) {

dets[j].prob[k] = 0;

}

}

}

}

}

void do\_nms\_sort(detection \*dets, int total, int classes, float thresh)

{

int i, j, k;

k = total - 1;

for (i = 0; i <= k; ++i) {

if (dets[i].objectness == 0) {

detection swap = dets[i];

dets[i] = dets[k];

dets[k] = swap;

--k;

--i;

}

}

total = k + 1;

for (k = 0; k < classes; ++k) {

for (i = 0; i < total; ++i) {

dets[i].sort\_class = k;

}

qsort(dets, total, sizeof(detection), nms\_comparator\_v3);

for (i = 0; i < total; ++i) {

//printf(" k = %d, \t i = %d \n", k, i);

if (dets[i].prob[k] == 0) continue;

box a = dets[i].bbox;

for (j = i + 1; j < total; ++j) {

box b = dets[j].bbox;

if (box\_iou(a, b) > thresh) {

dets[j].prob[k] = 0;

}

}

}

}

}

void do\_nms(box \*boxes, float \*\*probs, int total, int classes, float thresh)

{

int i, j, k;

for(i = 0; i < total; ++i){

int any = 0;

for(k = 0; k < classes; ++k) any = any || (probs[i][k] > 0);

if(!any) {

continue;

}

for(j = i+1; j < total; ++j){

if (box\_iou(boxes[i], boxes[j]) > thresh){

for(k = 0; k < classes; ++k){

if (probs[i][k] < probs[j][k]) probs[i][k] = 0;

else probs[j][k] = 0;

}

}

}

}

}

// https://github.com/Zzh-tju/DIoU-darknet

// https://arxiv.org/abs/1911.08287

void diounms\_sort(detection \*dets, int total, int classes, float thresh, NMS\_KIND nms\_kind, float beta1)

{

int i, j, k;

k = total - 1;

for (i = 0; i <= k; ++i) {

if (dets[i].objectness == 0) {

detection swap = dets[i];

dets[i] = dets[k];

dets[k] = swap;

--k;

--i;

}

}

total = k + 1;

for (k = 0; k < classes; ++k) {

for (i = 0; i < total; ++i) {

dets[i].sort\_class = k;

}

qsort(dets, total, sizeof(detection), nms\_comparator\_v3);

for (i = 0; i < total; ++i)

{

if (dets[i].prob[k] == 0) continue;

box a = dets[i].bbox;

for (j = i + 1; j < total; ++j) {

box b = dets[j].bbox;

if (box\_iou(a, b) > thresh && nms\_kind == CORNERS\_NMS)

{

float sum\_prob = pow(dets[i].prob[k], 2) + pow(dets[j].prob[k], 2);

float alpha\_prob = pow(dets[i].prob[k], 2) / sum\_prob;

float beta\_prob = pow(dets[j].prob[k], 2) / sum\_prob;

//dets[i].bbox.x = (dets[i].bbox.x\*alpha\_prob + dets[j].bbox.x\*beta\_prob);

//dets[i].bbox.y = (dets[i].bbox.y\*alpha\_prob + dets[j].bbox.y\*beta\_prob);

//dets[i].bbox.w = (dets[i].bbox.w\*alpha\_prob + dets[j].bbox.w\*beta\_prob);

//dets[i].bbox.h = (dets[i].bbox.h\*alpha\_prob + dets[j].bbox.h\*beta\_prob);

/\*

if (dets[j].points == YOLO\_CENTER && (dets[i].points & dets[j].points) == 0) {

dets[i].bbox.x = (dets[i].bbox.x\*alpha\_prob + dets[j].bbox.x\*beta\_prob);

dets[i].bbox.y = (dets[i].bbox.y\*alpha\_prob + dets[j].bbox.y\*beta\_prob);

}

else if ((dets[i].points & dets[j].points) == 0) {

dets[i].bbox.w = (dets[i].bbox.w\*alpha\_prob + dets[j].bbox.w\*beta\_prob);

dets[i].bbox.h = (dets[i].bbox.h\*alpha\_prob + dets[j].bbox.h\*beta\_prob);

}

dets[i].points |= dets[j].points;

\*/

dets[j].prob[k] = 0;

}

else if (box\_iou(a, b) > thresh && nms\_kind == GREEDY\_NMS) {

dets[j].prob[k] = 0;

}

else {

if (box\_diounms(a, b, beta1) > thresh && nms\_kind == DIOU\_NMS) {

dets[j].prob[k] = 0;

}

}

}

//if ((nms\_kind == CORNERS\_NMS) && (dets[i].points != (YOLO\_CENTER | YOLO\_LEFT\_TOP | YOLO\_RIGHT\_BOTTOM)))

// dets[i].prob[k] = 0;

}

}

}

box encode\_box(box b, box anchor)

{

box encode;

encode.x = (b.x - anchor.x) / anchor.w;

encode.y = (b.y - anchor.y) / anchor.h;

encode.w = log2(b.w / anchor.w);

encode.h = log2(b.h / anchor.h);

return encode;

}

box decode\_box(box b, box anchor)

{

box decode;

decode.x = b.x \* anchor.w + anchor.x;

decode.y = b.y \* anchor.h + anchor.y;

decode.w = pow(2., b.w) \* anchor.w;

decode.h = pow(2., b.h) \* anchor.h;

return decode;

}