## Data.c

#include<math.h>

**//Ryolo 242行**

Rbox\_label\* read\_Rboxes(char\* filename, int\* n)

{

Rbox\_label\* Rboxes = (Rbox\_label\*)xcalloc(1, sizeof(Rbox\_label));

*FILE*\* file = *fopen*(filename, "r");

if (!file) {

*printf*("Can't open label file. (This can be normal only if you use MSCOCO): %s \n", filename);

//file\_error(filename);

*FILE*\* fw = *fopen*("bad.list", "a");

*fwrite*(filename, sizeof(char), *strlen*(filename), fw);

char\* new\_line = "\n";

*fwrite*(new\_line, sizeof(char), *strlen*(new\_line), fw);

*fclose*(fw);

if (check\_mistakes) {

*printf*("\n Error in read\_boxes() \n");

*getchar*();

}

\*n = 0;

return Rboxes;

}

const int max\_obj\_img = 4000;// 30000;

const int img\_hash = (custom\_hash(filename) % max\_obj\_img) \* max\_obj\_img;

//printf(" img\_hash = %d, filename = %s; ", img\_hash, filename);

float x, y, h, w, r1, r2, theta, ra, rb;//Ryolo

float pi = 3.1415926;//Ryolo

int jud;//Ryolo

int adv;//Ryolo

int id;

int count = 0;

while (*fscanf*(file, "%d %f %f %f %f %f", &id, &x, &y, &theta, &ra, &rb) == 6) {//Ryolo

//&id, & x, & y, & w, & h, & r1, & r2, & jud, & adv

//判断jud和adv

theta = theta + pi;

if (0 < theta < pi ) { jud = 1; }

if (pi < theta < 2\*pi) { jud = 0; }

if (0 < theta < 0.5\*pi || 1.5\*pi < theta < 2\*pi) { adv = 1; }

if(0.5\*pi < theta < 1.5\*pi){ adv = 0; }

//对于特殊情况

if (theta == 0 || theta==2\*pi) { jud = 1; adv = 1; }

if (theta == 0.5\*pi) { jud = 1; adv = 0; }

if (theta == pi) { jud = 0; adv = 0; }

if (theta == 1.5\*pi) { jud = 0; adv = 1; }

if (theta == 0. || theta == 2\*pi || theta == 0.5\*pi) { w = ra; h = rb; r1 = 1; r2 = 0;}

if (theta == 0.5\*pi || theta == 1.5\*pi) { w = rb; h = ra; r1 = 0; r2 = 1; }

//判断四个顶点位置

if (theta > 2\*pi) { theta = theta - 2\*pi; }

if (0 < theta < 0.5\*pi) {

w = ra \* *cos*(theta) + rb \* *sin*(theta);

h = ra \* *sin*(theta) + rb \* *cos*(theta);

r1 = (ra \* *cos*(theta))/w;

r2 = (ra \* *sin*(theta))/h;

}

if (0.5\*pi < theta < pi) {

theta = pi - theta;

w = ra \* *cos*(theta) + rb \* *sin*(theta);

h = ra \* *sin*(theta) + rb \* *cos*(theta);

r1 = (rb \* *sin*(theta))/w;

r2 = (rb \* *cos*(theta))/h;

}

Rboxes = (Rbox\_label\*)xrealloc(Rboxes, (count + 1) \* sizeof(Rbox\_label));

Rboxes[count].track\_id = count + img\_hash;

//printf(" Rboxes[count].track\_id = %d, count = %d \n", Rboxes[count].track\_id, count);

Rboxes[count].id = id;

Rboxes[count].x = x;

Rboxes[count].y = y;

Rboxes[count].h = h;

Rboxes[count].w = w;

Rboxes[count].r1 = r1;//Ryolo

Rboxes[count].r2 = r2;//Ryolo

Rboxes[count].jud = jud;//Ryolo

Rboxes[count].adv = adv;//Ryolo

Rboxes[count].left = x - w / 2;

Rboxes[count].right = x + w / 2;

Rboxes[count].top = y - h / 2;

Rboxes[count].bottom = y + h / 2;

++count;

}

*fclose*(file);

\*n = count;

return Rboxes;

}//Ryolo 303

void randomize\_Rboxes(Rbox\_label\* b, int n)

{

int i;

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

Rbox\_label swap = b[i];

int index = random\_gen() % n;

b[i] = b[index];

b[index] = swap;

}

}

//Ryolo

void correct\_Rboxes(Rbox\_label\* Rboxes, int n, float dx, float dy, float sx, float sy, int flip)

{

int i;

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

if (Rboxes[i].x == 0 && Rboxes[i].y == 0) {

Rboxes[i].x = 999999;

Rboxes[i].y = 999999;

Rboxes[i].w = 999999;

Rboxes[i].h = 999999;

continue;

}

if ((Rboxes[i].x + Rboxes[i].w / 2) < 0 || (Rboxes[i].y + Rboxes[i].h / 2) < 0 ||

(Rboxes[i].x - Rboxes[i].w / 2) > 1 || (Rboxes[i].y - Rboxes[i].h / 2) > 1)

{

Rboxes[i].x = 999999;

Rboxes[i].y = 999999;

Rboxes[i].w = 999999;

Rboxes[i].h = 999999;

continue;

}

Rboxes[i].*left* = Rboxes[i].*left* \* sx - dx;

Rboxes[i].*right* = Rboxes[i].*right* \* sx - dx;

Rboxes[i].top = Rboxes[i].top \* sy - dy;

Rboxes[i].bottom = Rboxes[i].bottom \* sy - dy;

if (flip) {

float swap = Rboxes[i].*left*;

Rboxes[i].*left* = 1. - Rboxes[i].*right*;

Rboxes[i].*right* = 1. - swap;

}

Rboxes[i].*left* = constrain(0, 1, Rboxes[i].*left*);

Rboxes[i].*right* = constrain(0, 1, Rboxes[i].*right*);

Rboxes[i].top = constrain(0, 1, Rboxes[i].top);

Rboxes[i].bottom = constrain(0, 1, Rboxes[i].bottom);

Rboxes[i].x = (Rboxes[i].*left* + Rboxes[i].*right*) / 2;

Rboxes[i].y = (Rboxes[i].top + Rboxes[i].bottom) / 2;

Rboxes[i].w = (Rboxes[i].*right* - Rboxes[i].*left*);

Rboxes[i].h = (Rboxes[i].bottom - Rboxes[i].top);

Rboxes[i].w = constrain(0, 1, Rboxes[i].w);

Rboxes[i].h = constrain(0, 1, Rboxes[i].h);

}

}

**427行**

int fill\_truth\_Rdetection(const char\* path, int num\_boxes, int truth\_size, float\* truth, int classes, int flip, float dx, float dy, float sx, float sy,

int net\_w, int net\_h)

{

box\_label\* boxes = read\_Rboxes(labelpath, &count);

float x, y, w, h, r1, r2;//Ryolo

float jud;//Ryolo

int id;

int sub = 0;

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

x = boxes[i].x;

y = boxes[i].y;

w = boxes[i].w;

h = boxes[i].h;

r1 = boxes[i].r1;//Ryolo

r2 = boxes[i].r2;//Ryolo

jud = boxes[i].jud; //Ryolo

id = boxes[i].id;

int track\_id = boxes[i].track\_id;

truth[(i - sub) \* truth\_size + 0] = x;

truth[(i - sub) \* truth\_size + 1] = y;

truth[(i - sub) \* truth\_size + 2] = w;

truth[(i - sub) \* truth\_size + 3] = h;

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

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

truth[(i - sub) \* truth\_size + 6] = jud;//Ryolo

truth[(i - sub) \* truth\_size + 7] = id;

truth[(i - sub) \* truth\_size + 8] = track\_id;

}

//Ryolo 1517行

data load\_data\_Rdetection(int n, char\*\* paths, int m, int w, int h, int c, int boxes, int truth\_size, int classes, int use\_flip, int use\_gaussian\_noise, int use\_blur, int use\_mixup,

float jitter, float resize, float hue, float saturation, float exposure, int mini\_batch, int track, int augment\_speed, int letter\_box, int mosaic\_bound, int contrastive, int contrastive\_jit\_flip, int show\_imgs)

{

const int random\_index = random\_gen();

c = c ? c : 3;

if (use\_mixup == 2 || use\_mixup == 4) {

printf("\n cutmix=1 - isn't supported for Detector (use cutmix=1 only for Classifier) \n");

if (check\_mistakes) getchar();

if (use\_mixup == 2) use\_mixup = 0;

else use\_mixup = 3;

}

if (use\_mixup == 3 && letter\_box) {

//printf("\n Combination: letter\_box=1 & mosaic=1 - isn't supported, use only 1 of these parameters \n");

//if (check\_mistakes) getchar();

//exit(0);

}

if (random\_gen() % 2 == 0) use\_mixup = 0;

int i;

int\* cut\_x = NULL, \* cut\_y = NULL;

if (use\_mixup == 3) {

cut\_x = (int\*)calloc(n, sizeof(int));

cut\_y = (int\*)calloc(n, sizeof(int));

const float min\_offset = 0.2; // 20%

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

cut\_x[i] = rand\_int(w \* min\_offset, w \* (1 - min\_offset));

cut\_y[i] = rand\_int(h \* min\_offset, h \* (1 - min\_offset));

}

}

data d = { 0 };

d.shallow = 0;

d.X.rows = n;

d.X.vals = (float\*\*)xcalloc(d.X.rows, sizeof(float\*));

d.X.cols = h \* w \* c;

float r1 = 0, r2 = 0, r3 = 0, r4 = 0, r\_scale = 0;

float resize\_r1 = 0, resize\_r2 = 0;

float dhue = 0, dsat = 0, dexp = 0, flip = 0, blur = 0;

int augmentation\_calculated = 0, gaussian\_noise = 0;

d.y = make\_matrix(n, truth\_size \* boxes);

int i\_mixup = 0;

for (i\_mixup = 0; i\_mixup <= use\_mixup; i\_mixup++) {

if (i\_mixup) augmentation\_calculated = 0; // recalculate augmentation for the 2nd sequence if(track==1)

char\*\* random\_paths;

if (track) random\_paths = get\_sequential\_paths(paths, n, m, mini\_batch, augment\_speed, contrastive);

else random\_paths = get\_random\_paths\_custom(paths, n, m, contrastive);

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

float\* truth = (float\*)xcalloc(truth\_size \* boxes, sizeof(float));

const char\* filename = random\_paths[i];

int flag = (c >= 3);

mat\_cv\* src;

src = load\_image\_mat\_cv(filename, flag);

if (src == NULL) {

printf("\n Error in load\_data\_detection() - OpenCV \n");

fflush(stdout);

if (check\_mistakes) {

getchar();

}

continue;

}

int oh = get\_height\_mat(src);

int ow = get\_width\_mat(src);

int dw = (ow \* jitter);

int dh = (oh \* jitter);

float resize\_down = resize, resize\_up = resize;

if (resize\_down > 1.0) resize\_down = 1 / resize\_down;

int min\_rdw = ow \* (1 - (1 / resize\_down)) / 2; // < 0

int min\_rdh = oh \* (1 - (1 / resize\_down)) / 2; // < 0

if (resize\_up < 1.0) resize\_up = 1 / resize\_up;

int max\_rdw = ow \* (1 - (1 / resize\_up)) / 2; // > 0

int max\_rdh = oh \* (1 - (1 / resize\_up)) / 2; // > 0

//printf(" down = %f, up = %f \n", (1 - (1 / resize\_down)) / 2, (1 - (1 / resize\_up)) / 2);

if (!augmentation\_calculated || !track)

{

augmentation\_calculated = 1;

resize\_r1 = random\_float();

resize\_r2 = random\_float();

if (!contrastive || contrastive\_jit\_flip || i % 2 == 0)

{

r1 = random\_float();

r2 = random\_float();

r3 = random\_float();

r4 = random\_float();

flip = use\_flip ? random\_gen() % 2 : 0;

}

r\_scale = random\_float();

dhue = rand\_uniform\_strong(-hue, hue);

dsat = rand\_scale(saturation);

dexp = rand\_scale(exposure);

if (use\_blur) {

int tmp\_blur = rand\_int(0, 2); // 0 - disable, 1 - blur background, 2 - blur the whole image

if (tmp\_blur == 0) blur = 0;

else if (tmp\_blur == 1) blur = 1;

else blur = use\_blur;

}

if (use\_gaussian\_noise && rand\_int(0, 1) == 1) gaussian\_noise = use\_gaussian\_noise;

else gaussian\_noise = 0;

}

int pleft = rand\_precalc\_random(-dw, dw, r1);

int pright = rand\_precalc\_random(-dw, dw, r2);

int ptop = rand\_precalc\_random(-dh, dh, r3);

int pbot = rand\_precalc\_random(-dh, dh, r4);

if (resize < 1) {

// downsize only

pleft += rand\_precalc\_random(min\_rdw, 0, resize\_r1);

pright += rand\_precalc\_random(min\_rdw, 0, resize\_r2);

ptop += rand\_precalc\_random(min\_rdh, 0, resize\_r1);

pbot += rand\_precalc\_random(min\_rdh, 0, resize\_r2);

}

else {

pleft += rand\_precalc\_random(min\_rdw, max\_rdw, resize\_r1);

pright += rand\_precalc\_random(min\_rdw, max\_rdw, resize\_r2);

ptop += rand\_precalc\_random(min\_rdh, max\_rdh, resize\_r1);

pbot += rand\_precalc\_random(min\_rdh, max\_rdh, resize\_r2);

}

//printf("\n pleft = %d, pright = %d, ptop = %d, pbot = %d, ow = %d, oh = %d \n", pleft, pright, ptop, pbot, ow, oh);

//float scale = rand\_precalc\_random(.25, 2, r\_scale); // unused currently

//printf(" letter\_box = %d \n", letter\_box);

if (letter\_box)

{

float img\_ar = (float)ow / (float)oh;

float net\_ar = (float)w / (float)h;

float result\_ar = img\_ar / net\_ar;

//printf(" ow = %d, oh = %d, w = %d, h = %d, img\_ar = %f, net\_ar = %f, result\_ar = %f \n", ow, oh, w, h, img\_ar, net\_ar, result\_ar);

if (result\_ar > 1) // sheight - should be increased

{

float oh\_tmp = ow / net\_ar;

float delta\_h = (oh\_tmp - oh) / 2;

ptop = ptop - delta\_h;

pbot = pbot - delta\_h;

//printf(" result\_ar = %f, oh\_tmp = %f, delta\_h = %d, ptop = %f, pbot = %f \n", result\_ar, oh\_tmp, delta\_h, ptop, pbot);

}

else // swidth - should be increased

{

float ow\_tmp = oh \* net\_ar;

float delta\_w = (ow\_tmp - ow) / 2;

pleft = pleft - delta\_w;

pright = pright - delta\_w;

//printf(" result\_ar = %f, ow\_tmp = %f, delta\_w = %d, pleft = %f, pright = %f \n", result\_ar, ow\_tmp, delta\_w, pleft, pright);

}

//printf("\n pleft = %d, pright = %d, ptop = %d, pbot = %d, ow = %d, oh = %d \n", pleft, pright, ptop, pbot, ow, oh);

}

// move each 2nd image to the corner - so that most of it was visible

if (use\_mixup == 3 && random\_gen() % 2 == 0) {

if (flip) {

if (i\_mixup == 0) pleft += pright, pright = 0, pbot += ptop, ptop = 0;

if (i\_mixup == 1) pright += pleft, pleft = 0, pbot += ptop, ptop = 0;

if (i\_mixup == 2) pleft += pright, pright = 0, ptop += pbot, pbot = 0;

if (i\_mixup == 3) pright += pleft, pleft = 0, ptop += pbot, pbot = 0;

}

else {

if (i\_mixup == 0) pright += pleft, pleft = 0, pbot += ptop, ptop = 0;

if (i\_mixup == 1) pleft += pright, pright = 0, pbot += ptop, ptop = 0;

if (i\_mixup == 2) pright += pleft, pleft = 0, ptop += pbot, pbot = 0;

if (i\_mixup == 3) pleft += pright, pright = 0, ptop += pbot, pbot = 0;

}

}

int swidth = ow - pleft - pright;

int sheight = oh - ptop - pbot;

float sx = (float)swidth / ow;

float sy = (float)sheight / oh;

float dx = ((float)pleft / ow) / sx;

float dy = ((float)ptop / oh) / sy;

int min\_w\_h = fill\_truth\_Rdetection(filename, boxes, truth\_size, truth, classes, flip, dx, dy, 1. / sx, 1. / sy, w, h);

//for (int z = 0; z < boxes; ++z) if(truth[z\*truth\_size] > 0) printf(" track\_id = %f \n", truth[z\*truth\_size + 5]);

//printf(" truth\_size = %d \n", truth\_size);

if ((min\_w\_h / 8) < blur && blur > 1) blur = min\_w\_h / 8; // disable blur if one of the objects is too small

image ai = image\_data\_augmentation(src, w, h, pleft, ptop, swidth, sheight, flip, dhue, dsat, dexp,

gaussian\_noise, blur, boxes, truth\_size, truth);

if (use\_mixup == 0) {

d.X.vals[i] = ai.data;

memcpy(d.y.vals[i], truth, truth\_size \* boxes \* sizeof(float));

}

else if (use\_mixup == 1) {

if (i\_mixup == 0) {

d.X.vals[i] = ai.data;

memcpy(d.y.vals[i], truth, truth\_size \* boxes \* sizeof(float));

}

else if (i\_mixup == 1) {

image old\_img = make\_empty\_image(w, h, c);

old\_img.data = d.X.vals[i];

//show\_image(ai, "new");

//show\_image(old\_img, "old");

//wait\_until\_press\_key\_cv();

blend\_images\_cv(ai, 0.5, old\_img, 0.5);

blend\_truth(d.y.vals[i], boxes, truth\_size, truth);

free\_image(old\_img);

d.X.vals[i] = ai.data;

}

}

else if (use\_mixup == 3) {

if (i\_mixup == 0) {

image tmp\_img = make\_image(w, h, c);

d.X.vals[i] = tmp\_img.data;

}

if (flip) {

int tmp = pleft;

pleft = pright;

pright = tmp;

}

const int left\_shift = min\_val\_cmp(cut\_x[i], max\_val\_cmp(0, (-pleft \* w / ow)));

const int top\_shift = min\_val\_cmp(cut\_y[i], max\_val\_cmp(0, (-ptop \* h / oh)));

const int right\_shift = min\_val\_cmp((w - cut\_x[i]), max\_val\_cmp(0, (-pright \* w / ow)));

const int bot\_shift = min\_val\_cmp(h - cut\_y[i], max\_val\_cmp(0, (-pbot \* h / oh)));

int k, x, y;

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

for (y = 0; y < h; ++y) {

int j = y \* w + k \* w \* h;

if (i\_mixup == 0 && y < cut\_y[i]) {

int j\_src = (w - cut\_x[i] - right\_shift) + (y + h - cut\_y[i] - bot\_shift) \* w + k \* w \* h;

memcpy(&d.X.vals[i][j + 0], &ai.data[j\_src], cut\_x[i] \* sizeof(float));

}

if (i\_mixup == 1 && y < cut\_y[i]) {

int j\_src = left\_shift + (y + h - cut\_y[i] - bot\_shift) \* w + k \* w \* h;

memcpy(&d.X.vals[i][j + cut\_x[i]], &ai.data[j\_src], (w - cut\_x[i]) \* sizeof(float));

}

if (i\_mixup == 2 && y >= cut\_y[i]) {

int j\_src = (w - cut\_x[i] - right\_shift) + (top\_shift + y - cut\_y[i]) \* w + k \* w \* h;

memcpy(&d.X.vals[i][j + 0], &ai.data[j\_src], cut\_x[i] \* sizeof(float));

}

if (i\_mixup == 3 && y >= cut\_y[i]) {

int j\_src = left\_shift + (top\_shift + y - cut\_y[i]) \* w + k \* w \* h;

memcpy(&d.X.vals[i][j + cut\_x[i]], &ai.data[j\_src], (w - cut\_x[i]) \* sizeof(float));

}

}

}

blend\_truth\_mosaic(d.y.vals[i], boxes, truth\_size, truth, w, h, cut\_x[i], cut\_y[i], i\_mixup, left\_shift, right\_shift, top\_shift, bot\_shift, w, h, mosaic\_bound);

free\_image(ai);

ai.data = d.X.vals[i];

}

if (show\_imgs && i\_mixup == use\_mixup) // delete i\_mixup

{

image tmp\_ai = copy\_image(ai);

char buff[1000];

//sprintf(buff, "aug\_%d\_%d\_%s\_%d", random\_index, i, basecfg((char\*)filename), random\_gen());

sprintf(buff, "aug\_%d\_%d\_%d", random\_index, i, random\_gen());

int t;

for (t = 0; t < boxes; ++t) {

box b = float\_to\_Rbox\_stride(d.y.vals[i] + t \* truth\_size, 1);

if (!b.x) break;

int left = (b.x - b.w / 2.) \* ai.w;

int right = (b.x + b.w / 2.) \* ai.w;

int top = (b.y - b.h / 2.) \* ai.h;

int bot = (b.y + b.h / 2.) \* ai.h;

draw\_box\_width(tmp\_ai, left, top, right, bot, 1, 150, 100, 50); // 3 channels RGB

}

save\_image(tmp\_ai, buff);

if (show\_imgs == 1) {

//char buff\_src[1000];

//sprintf(buff\_src, "src\_%d\_%d\_%s\_%d", random\_index, i, basecfg((char\*)filename), random\_gen());

//show\_image\_mat(src, buff\_src);

show\_image(tmp\_ai, buff);

wait\_until\_press\_key\_cv();

}

printf("\nYou use flag -show\_imgs, so will be saved aug\_...jpg images. Click on window and press ESC button \n");

free\_image(tmp\_ai);

}

release\_mat(&src);

free(truth);

}

if (random\_paths) free(random\_paths);

}

return d;

}

//Ryolo 2034行

data load\_data\_Rdetection(int n, char\*\* paths, int m, int w, int h, int c, int boxes, int truth\_size, int classes, int use\_flip, int gaussian\_noise, int use\_blur, int use\_mixup,

float jitter, float resize, float hue, float saturation, float exposure, int mini\_batch, int track, int augment\_speed, int letter\_box, int mosaic\_bound, int contrastive, int contrastive\_jit\_flip, int show\_imgs)

{

const int random\_index = random\_gen();

c = c ? c : 3;

char\*\* random\_paths;

char\*\* mixup\_random\_paths = *NULL*;

if (track) random\_paths = get\_sequential\_paths(paths, n, m, mini\_batch, augment\_speed, contrastive);

else random\_paths = get\_random\_paths\_custom(paths, n, m, contrastive);

//assert(use\_mixup < 2);

if (use\_mixup == 2) {

*printf*("\n cutmix=1 - isn't supported for Detector \n");

*exit*(0);

}

if (use\_mixup == 3 || use\_mixup == 4) {

*printf*("\n mosaic=1 - compile Darknet with OpenCV for using mosaic=1 \n");

*exit*(0);

}

int mixup = use\_mixup ? random\_gen() % 2 : 0;

//printf("\n mixup = %d \n", mixup);

if (mixup) {

if (track) mixup\_random\_paths = get\_sequential\_paths(paths, n, m, mini\_batch, augment\_speed, contrastive);

else mixup\_random\_paths = get\_random\_paths(paths, n, m);

}

int i;

data d = { 0 };

d.shallow = 0;

d.X.rows = n;

d.X.vals = (float\*\*)xcalloc(d.X.rows, sizeof(float\*));

d.X.cols = h \* w \* c;

float r1 = 0, r2 = 0, r3 = 0, r4 = 0, r\_scale;

float resize\_r1 = 0, resize\_r2 = 0;

float dhue = 0, dsat = 0, dexp = 0, flip = 0;

int augmentation\_calculated = 0;

d.y = make\_matrix(n, truth\_size \* boxes);

int i\_mixup = 0;

for (i\_mixup = 0; i\_mixup <= mixup; i\_mixup++) {

if (i\_mixup) augmentation\_calculated = 0;

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

float\* truth = (float\*)xcalloc(truth\_size \* boxes, sizeof(float));

char\* filename = (i\_mixup) ? mixup\_random\_paths[i] : random\_paths[i];

image orig = load\_image(filename, 0, 0, c);

int oh = orig.h;

int ow = orig.w;

int dw = (ow \* jitter);

int dh = (oh \* jitter);

float resize\_down = resize, resize\_up = resize;

if (resize\_down > 1.0) resize\_down = 1 / resize\_down;

int min\_rdw = ow \* (1 - (1 / resize\_down)) / 2;

int min\_rdh = oh \* (1 - (1 / resize\_down)) / 2;

if (resize\_up < 1.0) resize\_up = 1 / resize\_up;

int max\_rdw = ow \* (1 - (1 / resize\_up)) / 2;

int max\_rdh = oh \* (1 - (1 / resize\_up)) / 2;

if (!augmentation\_calculated || !track)

{

augmentation\_calculated = 1;

resize\_r1 = random\_float();

resize\_r2 = random\_float();

if (!contrastive || contrastive\_jit\_flip || i % 2 == 0)

{

r1 = random\_float();

r2 = random\_float();

r3 = random\_float();

r4 = random\_float();

flip = use\_flip ? random\_gen() % 2 : 0;

}

r\_scale = random\_float();

dhue = rand\_uniform\_strong(-hue, hue);

dsat = rand\_scale(saturation);

dexp = rand\_scale(exposure);

}

int pleft = rand\_precalc\_random(-dw, dw, r1);

int pright = rand\_precalc\_random(-dw, dw, r2);

int ptop = rand\_precalc\_random(-dh, dh, r3);

int pbot = rand\_precalc\_random(-dh, dh, r4);

if (resize < 1) {

// downsize only

pleft += rand\_precalc\_random(min\_rdw, 0, resize\_r1);

pright += rand\_precalc\_random(min\_rdw, 0, resize\_r2);

ptop += rand\_precalc\_random(min\_rdh, 0, resize\_r1);

pbot += rand\_precalc\_random(min\_rdh, 0, resize\_r2);

}

else {

pleft += rand\_precalc\_random(min\_rdw, max\_rdw, resize\_r1);

pright += rand\_precalc\_random(min\_rdw, max\_rdw, resize\_r2);

ptop += rand\_precalc\_random(min\_rdh, max\_rdh, resize\_r1);

pbot += rand\_precalc\_random(min\_rdh, max\_rdh, resize\_r2);

}

if (letter\_box)

{

float img\_ar = (float)ow / (float)oh;

float net\_ar = (float)w / (float)h;

float result\_ar = img\_ar / net\_ar;

//printf(" ow = %d, oh = %d, w = %d, h = %d, img\_ar = %f, net\_ar = %f, result\_ar = %f \n", ow, oh, w, h, img\_ar, net\_ar, result\_ar);

if (result\_ar > 1) // sheight - should be increased

{

float oh\_tmp = ow / net\_ar;

float delta\_h = (oh\_tmp - oh) / 2;

ptop = ptop - delta\_h;

pbot = pbot - delta\_h;

//printf(" result\_ar = %f, oh\_tmp = %f, delta\_h = %d, ptop = %f, pbot = %f \n", result\_ar, oh\_tmp, delta\_h, ptop, pbot);

}

else // swidth - should be increased

{

float ow\_tmp = oh \* net\_ar;

float delta\_w = (ow\_tmp - ow) / 2;

pleft = pleft - delta\_w;

pright = pright - delta\_w;

//printf(" result\_ar = %f, ow\_tmp = %f, delta\_w = %d, pleft = %f, pright = %f \n", result\_ar, ow\_tmp, delta\_w, pleft, pright);

}

}

int swidth = ow - pleft - pright;

int sheight = oh - ptop - pbot;

float sx = (float)swidth / ow;

float sy = (float)sheight / oh;

image cropped = crop\_image(orig, pleft, ptop, swidth, sheight);

float dx = ((float)pleft / ow) / sx;

float dy = ((float)ptop / oh) / sy;

image sized = resize\_image(cropped, w, h);

if (flip) flip\_image(sized);

distort\_image(sized, dhue, dsat, dexp);

//random\_distort\_image(sized, hue, saturation, exposure);

fill\_truth\_Rdetection(filename, boxes, truth\_size, truth, classes, flip, dx, dy, 1. / sx, 1. / sy, w, h);

if (i\_mixup) {

image old\_img = sized;

old\_img.*data* = d.X.vals[i];

//show\_image(sized, "new");

//show\_image(old\_img, "old");

//wait\_until\_press\_key\_cv();

blend\_images(sized, 0.5, old\_img, 0.5);

blend\_truth(truth, boxes, truth\_size, d.y.vals[i]);

free\_image(old\_img);

}

d.X.vals[i] = sized.*data*;

*memcpy*(d.y.vals[i], truth, truth\_size \* boxes \* sizeof(float));

if (show\_imgs)// && i\_mixup)

{

char buff[1000];

*sprintf*(buff, "aug\_%d\_%d\_%s\_%d", random\_index, i, basecfg(filename), random\_gen());

int t;

for (t = 0; t < boxes; ++t) {

box b = float\_to\_Rbox\_stride(d.y.vals[i] + t \* truth\_size, 1);

if (!b.x) break;

int left = (b.x - b.w / 2.) \* sized.w;

int right = (b.x + b.w / 2.) \* sized.w;

int top = (b.y - b.h / 2.) \* sized.h;

int bot = (b.y + b.h / 2.) \* sized.h;

draw\_box\_width(sized, left, top, right, bot, 1, 150, 100, 50); // 3 channels RGB

}

save\_image(sized, buff);

if (show\_imgs == 1) {

show\_image(sized, buff);

wait\_until\_press\_key\_cv();

}

*printf*("\nYou use flag -show\_imgs, so will be saved aug\_...jpg images. Press Enter: \n");

//getchar();

}

free\_image(orig);

free\_image(cropped);

*free*(truth);

}

}

*free*(random\_paths);

if (mixup\_random\_paths) *free*(mixup\_random\_paths);

return d;

}

else if (a.type == RDETECTION\_DATA) {//Ryolo 2310

\*a.d = load\_data\_Rdetection(a.n, a.paths, a.m, a.w, a.h, a.c, a.num\_boxes, a.truth\_size, a.classes, a.*flip*, a.gaussian\_noise, a.*blur*, a.mixup, a.jitter, a.*resize*,

a.hue, a.saturation, a.exposure, a.mini\_batch, a.track, a.augment\_speed, a.letter\_box, a.mosaic\_bound, a.contrastive, a.contrastive\_jit\_flip, a.show\_imgs);

}

## Data.h

**99行**

box\_label\* Rread\_boxes(char\* filename, int\* n);//Ryolo

## box.c

31行

//Ryolo

box float\_to\_Rbox\_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];//Ryolo

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

b.jud = f[6 \* stride];//Ryolo

return b;

}

//Ryolo

boxabs to\_Rtblr(Rbox 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;

}

//Ryolo

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

{

//IOU, GIOU, MSE, DIOU, CIOU

switch (iou\_kind) {

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

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

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

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

}

return Rbox\_iou(a, b);

}

//Ryolo

dxrep dx\_Rbox\_iou(Rbox pred, Rbox truth, IOU\_LOSS iou\_loss) {

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

#ifdef DEBUG\_PRINTS

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

}

## Box.h

35行

box float\_to\_Rbox\_stride(float\* f, int stride);//Ryolo

dxrep dx\_Rbox\_iou(box a, box b, IOU\_LOSS iou\_loss);//Ryolo

boxabs to\_Rtblr(box a);//Ryolo

## Darknet.h

976行

typedef struct Rbox {

float x, y, w, h, r1, r2;//Ryolo

float jud;

} Rbox;

typedef struct Rbox\_label {

int id;

int track\_id;

int jud;//Ryolo

float x, y, w, h, r1, r2;//Ryolo

float left, right, top, bottom;

} Rbox\_label;

914行

typedef enum {//Ryolo

CLASSIFICATION\_DATA, DETECTION\_DATA, **RDETECTION\_DATA**, CAPTCHA\_DATA, REGION\_DATA, IMAGE\_DATA, COMPARE\_DATA, WRITING\_DATA, SWAG\_DATA, TAG\_DATA, OLD\_CLASSIFICATION\_DATA, STUDY\_DATA, DET\_DATA, SUPER\_DATA, LETTERBOX\_DATA, REGRESSION\_DATA, SEGMENTATION\_DATA, INSTANCE\_DATA, ISEG\_DATA

} data\_type;

186行

typedef enum {

RYOLO,//Ryolo

} LAYER\_TYPE;

877行

typedef struct detection{

Rbox Rbbox;//Ryolo

} detection;

## Parser.c

//Ryolo

layer parse\_Ryolo(list\* options, size\_params params)

{

int classes = option\_find\_int(options, "classes", 20);

int total = option\_find\_int(options, "num", 1);

int num = total;

char\* a = option\_find\_str(options, "mask", 0);

int\* mask = parse\_yolo\_mask(a, &num);

int max\_boxes = option\_find\_int\_quiet(options, "max", 200);

layer l = make\_yolo\_layer(params.batch, params.w, params.h, num, total, mask, classes, max\_boxes);

if (l.outputs != params.inputs) {

*printf*("Error: l.outputs == params.inputs \n");

*printf*("filters= in the [convolutional]-layer doesn't correspond to classes= or mask= in [Ryolo]-layer \n");

*exit*(*EXIT\_FAILURE*);

}

//assert(l.outputs == params.inputs);

l.max\_delta = option\_find\_float\_quiet(options, "max\_delta", *FLT\_MAX*); // set 10

char\* cpc = option\_find\_str(options, "counters\_per\_class", 0);

l.classes\_multipliers = get\_classes\_multipliers(cpc, classes, l.max\_delta);

l.label\_smooth\_eps = option\_find\_float\_quiet(options, "label\_smooth\_eps", 0.0f);

l.scale\_x\_y = option\_find\_float\_quiet(options, "scale\_x\_y", 1);

l.objectness\_smooth = option\_find\_int\_quiet(options, "objectness\_smooth", 0);

l.iou\_normalizer = option\_find\_float\_quiet(options, "iou\_normalizer", 0.75);

l.obj\_normalizer = option\_find\_float\_quiet(options, "obj\_normalizer", 1);

l.cls\_normalizer = option\_find\_float\_quiet(options, "cls\_normalizer", 1);

l.delta\_normalizer = option\_find\_float\_quiet(options, "delta\_normalizer", 1);

char\* iou\_loss = option\_find\_str\_quiet(options, "iou\_loss", "mse"); // "iou");

if (*strcmp*(iou\_loss, "mse") == 0) l.iou\_loss = MSE;

else if (*strcmp*(iou\_loss, "giou") == 0) l.iou\_loss = GIOU;

else if (*strcmp*(iou\_loss, "diou") == 0) l.iou\_loss = DIOU;

else if (*strcmp*(iou\_loss, "ciou") == 0) l.iou\_loss = CIOU;

else l.iou\_loss = IOU;

*fprintf*(*stderr*, "[Ryolo] params: iou loss: %s (%d), iou\_norm: %2.2f, obj\_norm: %2.2f, cls\_norm: %2.2f, delta\_norm: %2.2f, scale\_x\_y: %2.2f\n",

iou\_loss, l.iou\_loss, l.iou\_normalizer, l.obj\_normalizer, l.cls\_normalizer, l.delta\_normalizer, l.scale\_x\_y);

char\* iou\_thresh\_kind\_str = option\_find\_str\_quiet(options, "iou\_thresh\_kind", "iou");

if (*strcmp*(iou\_thresh\_kind\_str, "iou") == 0) l.iou\_thresh\_kind = IOU;

else if (*strcmp*(iou\_thresh\_kind\_str, "giou") == 0) l.iou\_thresh\_kind = GIOU;

else if (*strcmp*(iou\_thresh\_kind\_str, "diou") == 0) l.iou\_thresh\_kind = DIOU;

else if (*strcmp*(iou\_thresh\_kind\_str, "ciou") == 0) l.iou\_thresh\_kind = CIOU;

else {

*fprintf*(*stderr*, " Wrong iou\_thresh\_kind = %s \n", iou\_thresh\_kind\_str);

l.iou\_thresh\_kind = IOU;

}

l.beta\_nms = option\_find\_float\_quiet(options, "beta\_nms", 0.6);

char\* nms\_kind = option\_find\_str\_quiet(options, "nms\_kind", "default");

if (*strcmp*(nms\_kind, "default") == 0) l.nms\_kind = DEFAULT\_NMS;

else {

if (*strcmp*(nms\_kind, "greedynms") == 0) l.nms\_kind = GREEDY\_NMS;

else if (*strcmp*(nms\_kind, "diounms") == 0) l.nms\_kind = DIOU\_NMS;

else l.nms\_kind = DEFAULT\_NMS;

*printf*("nms\_kind: %s (%d), beta = %f \n", nms\_kind, l.nms\_kind, l.beta\_nms);

}

l.jitter = option\_find\_float(options, "jitter", .2);

l.*resize* = option\_find\_float\_quiet(options, "resize", 1.0);

l.focal\_loss = option\_find\_int\_quiet(options, "focal\_loss", 0);

l.ignore\_thresh = option\_find\_float(options, "ignore\_thresh", .5);

l.truth\_thresh = option\_find\_float(options, "truth\_thresh", 1);

l.iou\_thresh = option\_find\_float\_quiet(options, "iou\_thresh", 1); // recommended to use iou\_thresh=0.213 in [yolo]

l.random = option\_find\_float\_quiet(options, "random", 0);

l.track\_history\_size = option\_find\_int\_quiet(options, "track\_history\_size", 5);

l.sim\_thresh = option\_find\_int\_quiet(options, "sim\_thresh", 0.8);

l.dets\_for\_track = option\_find\_int\_quiet(options, "dets\_for\_track", 1);

l.dets\_for\_show = option\_find\_int\_quiet(options, "dets\_for\_show", 1);

l.track\_ciou\_norm = option\_find\_float\_quiet(options, "track\_ciou\_norm", 0.01);

int embedding\_layer\_id = option\_find\_int\_quiet(options, "embedding\_layer", 999999);

if (embedding\_layer\_id < 0) embedding\_layer\_id = params.index + embedding\_layer\_id;

if (embedding\_layer\_id != 999999) {

*printf*(" embedding\_layer\_id = %d, ", embedding\_layer\_id);

layer le = params.net.layers[embedding\_layer\_id];

l.embedding\_layer\_id = embedding\_layer\_id;

l.embedding\_output = (float\*)xcalloc(le.batch \* le.outputs, sizeof(float));

l.embedding\_size = le.n / l.n;

*printf*(" embedding\_size = %d \n", l.embedding\_size);

if (le.n % l.n != 0) {

*printf*(" Warning: filters=%d number in embedding\_layer=%d isn't divisable by number of anchors %d \n", le.n, embedding\_layer\_id, l.n);

*getchar*();

}

}

char\* map\_file = option\_find\_str(options, "map", 0);

if (map\_file) l.*map* = read\_map(map\_file);

a = option\_find\_str(options, "anchors", 0);

if (a) {

int len = *strlen*(a);

int n = 1;

int i;

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

if (a[i] == '#') break;

if (a[i] == ',') ++n;

}

for (i = 0; i < n && i < total \* 2; ++i) {

float bias = *atof*(a);

l.biases[i] = bias;

a = *strchr*(a, ',') + 1;

}

}

return l;

}

//Ryolo

LAYER\_TYPE string\_to\_layer\_type

if (*strcmp*(type, "[ryolo]") == 0) return RYOLO;//Ryolo

## network.c

#include "Ryolo\_layer.h"//Ryolo

case RYOLO://Ryolo

return "ryolo";

int resize\_network

else if (l.*type* == RYOLO) {//Ryolo

resize\_Ryolo\_layer(&l, w, h);

}

//Ryolo

if (l.type == RYOLO) {

s += Ryolo\_num\_detections(l, thresh);

}

//Ryolo

if (l.type == RYOLO) {

int count = get\_yolo\_detections(l, w, h, net->w, net->h, thresh, map, relative, dets, letter);

dets += count;

if (prev\_classes < 0) prev\_classes = l.classes;

else if (prev\_classes != l.classes) {

*printf*(" Error: Different [yolo] layers have different number of classes = %d and %d - check your cfg-file! \n",

prev\_classes, l.classes);

}

}

## Detector.c

//Ryolo

char\* GetFilename(char\* p)

{

static char name[20] = { "" };

char\* q = *strrchr*(p, '\\') + 1;

*strncpy*(name, q, 23);

return name;

}

//Ryolo

else if (0 == *strcmp*(argv[2], "train\_Ryolo")) train\_Rdetector(datacfg, cfg, weights, gpus, ngpus, clear, dont\_show, calc\_map, mjpeg\_port, show\_imgs, benchmark\_layers, chart\_path);

//Ryolo

else if (0 == *strcmp*(argv[2], "test\_Ryolo")) test\_Rdetector(datacfg, cfg, weights, filename, thresh, hier\_thresh, dont\_show, ext\_output, save\_labels, outfile, letter\_box, benchmark\_layers);

//Ryolo

void train\_Rdetector(char\* datacfg, char\* cfgfile, char\* weightfile, int\* gpus, int ngpus, int clear, int dont\_show, int calc\_map, int mjpeg\_port, int show\_imgs, int benchmark\_layers, char\* chart\_path)

{

list\* options = read\_data\_cfg(datacfg);

char\* train\_images = option\_find\_str(options, "train", "data/train.txt");

char\* valid\_images = option\_find\_str(options, "valid", train\_images);

char\* backup\_directory = option\_find\_str(options, "backup", "/backup/");

network net\_map;

if (calc\_map) {

FILE\* valid\_file = fopen(valid\_images, "r");

if (!valid\_file) {

printf("\n Error: There is no %s file for mAP calculation!\n Don't use -map flag.\n Or set valid=%s in your %s file. \n", valid\_images, train\_images, datacfg);

getchar();

exit(-1);

}

else fclose(valid\_file);

cuda\_set\_device(gpus[0]);

printf(" Prepare additional network for mAP calculation...\n");

net\_map = parse\_network\_cfg\_custom(cfgfile, 1, 1);

net\_map.benchmark\_layers = benchmark\_layers;

const int net\_classes = net\_map.layers[net\_map.n - 1].classes;

int k; // free memory unnecessary arrays

for (k = 0; k < net\_map.n - 1; ++k) free\_layer\_custom(net\_map.layers[k], 1);

char\* name\_list = option\_find\_str(options, "names", "data/names.list");

int names\_size = 0;

char\*\* names = get\_labels\_custom(name\_list, &names\_size);

if (net\_classes != names\_size) {

printf("\n Error: in the file %s number of names %d that isn't equal to classes=%d in the file %s \n",

name\_list, names\_size, net\_classes, cfgfile);

if (net\_classes > names\_size) getchar();

}

free\_ptrs((void\*\*)names, net\_map.layers[net\_map.n - 1].classes);

}

srand(time(0));

char\* base = basecfg(cfgfile);

printf("%s\n", base);

float avg\_loss = -1;

float avg\_contrastive\_acc = 0;

network\* nets = (network\*)xcalloc(ngpus, sizeof(network));

srand(time(0));

int seed = rand();

int k;

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

srand(seed);

#ifdef GPU

cuda\_set\_device(gpus[k]);

#endif

nets[k] = parse\_network\_cfg(cfgfile);

nets[k].benchmark\_layers = benchmark\_layers;

if (weightfile) {

load\_weights(&nets[k], weightfile);

}

if (clear) {

\*nets[k].seen = 0;

\*nets[k].cur\_iteration = 0;

}

nets[k].learning\_rate \*= ngpus;

}

srand(time(0));

network net = nets[0];

const int actual\_batch\_size = net.batch \* net.subdivisions;

if (actual\_batch\_size == 1) {

printf("\n Error: You set incorrect value batch=1 for Training! You should set batch=64 subdivision=64 \n");

getchar();

}

else if (actual\_batch\_size < 8) {

printf("\n Warning: You set batch=%d lower than 64! It is recommended to set batch=64 subdivision=64 \n", actual\_batch\_size);

}

int imgs = net.batch \* net.subdivisions \* ngpus;

printf("Learning Rate: %g, Momentum: %g, Decay: %g\n", net.learning\_rate, net.momentum, net.decay);

data train, buffer;

layer l = net.layers[net.n - 1];

for (k = 0; k < net.n; ++k) {

layer lk = net.layers[k];

if (lk.type == YOLO || lk.type == GAUSSIAN\_YOLO || lk.type == REGION) {

l = lk;

printf(" Detection layer: %d - type = %d \n", k, l.type);

}

}

int classes = l.classes;

list\* plist = get\_paths(train\_images);

int train\_images\_num = plist->size;

char\*\* paths = (char\*\*)list\_to\_array(plist);

const int init\_w = net.w;

const int init\_h = net.h;

const int init\_b = net.batch;

int iter\_save, iter\_save\_last, iter\_map;

iter\_save = get\_current\_iteration(net);

iter\_save\_last = get\_current\_iteration(net);

iter\_map = get\_current\_iteration(net);

float mean\_average\_precision = -1;

float best\_map = mean\_average\_precision;

load\_args args = { 0 };

args.w = net.w;

args.h = net.h;

args.c = net.c;

args.paths = paths;

args.n = imgs;

args.m = plist->size;

args.classes = classes;

args.flip = net.flip;

args.jitter = l.jitter;

args.resize = l.resize;

args.num\_boxes = l.max\_boxes;

args.truth\_size = l.truth\_size;

net.num\_boxes = args.num\_boxes;

net.train\_images\_num = train\_images\_num;

args.d = &buffer;

args.type = DETECTION\_DATA;

args.threads = 64; // 16 or 64

args.angle = net.angle;

args.gaussian\_noise = net.gaussian\_noise;

args.blur = net.blur;

args.mixup = net.mixup;

args.exposure = net.exposure;

args.saturation = net.saturation;

args.hue = net.hue;

args.letter\_box = net.letter\_box;

args.mosaic\_bound = net.mosaic\_bound;

args.contrastive = net.contrastive;

args.contrastive\_jit\_flip = net.contrastive\_jit\_flip;

if (dont\_show && show\_imgs) show\_imgs = 2;

args.show\_imgs = show\_imgs;

#ifdef OPENCV

//int num\_threads = get\_num\_threads();

//if(num\_threads > 2) args.threads = get\_num\_threads() - 2;

args.threads = 6 \* ngpus; // 3 for - Amazon EC2 Tesla V100: p3.2xlarge (8 logical cores) - p3.16xlarge

//args.threads = 12 \* ngpus; // Ryzen 7 2700X (16 logical cores)

mat\_cv\* img = NULL;

float max\_img\_loss = net.max\_chart\_loss;

int number\_of\_lines = 100;

int img\_size = 1000;

char windows\_name[100];

sprintf(windows\_name, "chart\_%s.png", base);

img = draw\_train\_chart(windows\_name, max\_img\_loss, net.max\_batches, number\_of\_lines, img\_size, dont\_show, chart\_path);

#endif //OPENCV

if (net.contrastive && args.threads > net.batch / 2) args.threads = net.batch / 2;

if (net.track) {

args.track = net.track;

args.augment\_speed = net.augment\_speed;

if (net.sequential\_subdivisions) args.threads = net.sequential\_subdivisions \* ngpus;

else args.threads = net.subdivisions \* ngpus;

args.mini\_batch = net.batch / net.time\_steps;

printf("\n Tracking! batch = %d, subdiv = %d, time\_steps = %d, mini\_batch = %d \n", net.batch, net.subdivisions, net.time\_steps, args.mini\_batch);

}

//printf(" imgs = %d \n", imgs);

pthread\_t load\_thread = load\_data(args);

int count = 0;

double time\_remaining, avg\_time = -1, alpha\_time = 0.01;

//while(i\*imgs < N\*120){

while (get\_current\_iteration(net) < net.max\_batches) {

if (l.random && count++ % 10 == 0) {

float rand\_coef = 1.4;

if (l.random != 1.0) rand\_coef = l.random;

printf("Resizing, random\_coef = %.2f \n", rand\_coef);

float random\_val = rand\_scale(rand\_coef); // \*x or /x

int dim\_w = roundl(random\_val \* init\_w / net.resize\_step + 1) \* net.resize\_step;

int dim\_h = roundl(random\_val \* init\_h / net.resize\_step + 1) \* net.resize\_step;

if (random\_val < 1 && (dim\_w > init\_w || dim\_h > init\_h)) dim\_w = init\_w, dim\_h = init\_h;

int max\_dim\_w = roundl(rand\_coef \* init\_w / net.resize\_step + 1) \* net.resize\_step;

int max\_dim\_h = roundl(rand\_coef \* init\_h / net.resize\_step + 1) \* net.resize\_step;

// at the beginning (check if enough memory) and at the end (calc rolling mean/variance)

if (avg\_loss < 0 || get\_current\_iteration(net) > net.max\_batches - 100) {

dim\_w = max\_dim\_w;

dim\_h = max\_dim\_h;

}

if (dim\_w < net.resize\_step) dim\_w = net.resize\_step;

if (dim\_h < net.resize\_step) dim\_h = net.resize\_step;

int dim\_b = (init\_b \* max\_dim\_w \* max\_dim\_h) / (dim\_w \* dim\_h);

int new\_dim\_b = (int)(dim\_b \* 0.8);

if (new\_dim\_b > init\_b) dim\_b = new\_dim\_b;

args.w = dim\_w;

args.h = dim\_h;

int k;

if (net.dynamic\_minibatch) {

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

(\*nets[k].seen) = init\_b \* net.subdivisions \* get\_current\_iteration(net); // remove this line, when you will save to weights-file both: seen & cur\_iteration

nets[k].batch = dim\_b;

int j;

for (j = 0; j < nets[k].n; ++j)

nets[k].layers[j].batch = dim\_b;

}

net.batch = dim\_b;

imgs = net.batch \* net.subdivisions \* ngpus;

args.n = imgs;

printf("\n %d x %d (batch = %d) \n", dim\_w, dim\_h, net.batch);

}

else

printf("\n %d x %d \n", dim\_w, dim\_h);

pthread\_join(load\_thread, 0);

train = buffer;

free\_data(train);

load\_thread = load\_data(args);

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

resize\_network(nets + k, dim\_w, dim\_h);

}

net = nets[0];

}

double time = what\_time\_is\_it\_now();

pthread\_join(load\_thread, 0);

train = buffer;

if (net.track) {

net.sequential\_subdivisions = get\_current\_seq\_subdivisions(net);

args.threads = net.sequential\_subdivisions \* ngpus;

printf(" sequential\_subdivisions = %d, sequence = %d \n", net.sequential\_subdivisions, get\_sequence\_value(net));

}

load\_thread = load\_data(args);

//wait\_key\_cv(500);

/\*

int k;

for(k = 0; k < l.max\_boxes; ++k){

box b = float\_to\_box(train.y.vals[10] + 1 + k\*5);

if(!b.x) break;

printf("loaded: %f %f %f %f\n", b.x, b.y, b.w, b.h);

}

image im = float\_to\_image(448, 448, 3, train.X.vals[10]);

int k;

for(k = 0; k < l.max\_boxes; ++k){

box b = float\_to\_box(train.y.vals[10] + 1 + k\*5);

printf("%d %d %d %d\n", truth.x, truth.y, truth.w, truth.h);

draw\_bbox(im, b, 8, 1,0,0);

}

save\_image(im, "truth11");

\*/

const double load\_time = (what\_time\_is\_it\_now() - time);

printf("Loaded: %lf seconds", load\_time);

if (load\_time > 0.1 && avg\_loss > 0) printf(" - performance bottleneck on CPU or Disk HDD/SSD");

printf("\n");

time = what\_time\_is\_it\_now();

float loss = 0;

#ifdef GPU

if (ngpus == 1) {

int wait\_key = (dont\_show) ? 0 : 1;

loss = train\_network\_waitkey(net, train, wait\_key);

}

else {

loss = train\_networks(nets, ngpus, train, 4);

}

#else

loss = train\_network(net, train);

#endif

if (avg\_loss < 0 || avg\_loss != avg\_loss) avg\_loss = loss; // if(-inf or nan)

avg\_loss = avg\_loss \* .9 + loss \* .1;

const int iteration = get\_current\_iteration(net);

//i = get\_current\_batch(net);

int calc\_map\_for\_each = 4 \* train\_images\_num / (net.batch \* net.subdivisions); // calculate mAP for each 4 Epochs

calc\_map\_for\_each = fmax(calc\_map\_for\_each, 100);

int next\_map\_calc = iter\_map + calc\_map\_for\_each;

next\_map\_calc = fmax(next\_map\_calc, net.burn\_in);

//next\_map\_calc = fmax(next\_map\_calc, 400);

if (calc\_map) {

printf("\n (next mAP calculation at %d iterations) ", next\_map\_calc);

if (mean\_average\_precision > 0) printf("\n Last accuracy mAP@0.5 = %2.2f %%, best = %2.2f %% ", mean\_average\_precision \* 100, best\_map \* 100);

}

if (net.cudnn\_half) {

if (iteration < net.burn\_in \* 3) fprintf(stderr, "\n Tensor Cores are disabled until the first %d iterations are reached.\n", 3 \* net.burn\_in);

else fprintf(stderr, "\n Tensor Cores are used.\n");

fflush(stderr);

}

printf("\n %d: %f, %f avg loss, %f rate, %lf seconds, %d images, %f hours left\n", iteration, loss, avg\_loss, get\_current\_rate(net), (what\_time\_is\_it\_now() - time), iteration \* imgs, avg\_time);

fflush(stdout);

int draw\_precision = 0;

if (calc\_map && (iteration >= next\_map\_calc || iteration == net.max\_batches)) {

if (l.random) {

printf("Resizing to initial size: %d x %d ", init\_w, init\_h);

args.w = init\_w;

args.h = init\_h;

int k;

if (net.dynamic\_minibatch) {

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

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

nets[k].batch = init\_b;

int j;

for (j = 0; j < nets[k].n; ++j)

nets[k].layers[j].batch = init\_b;

}

}

net.batch = init\_b;

imgs = init\_b \* net.subdivisions \* ngpus;

args.n = imgs;

printf("\n %d x %d (batch = %d) \n", init\_w, init\_h, init\_b);

}

pthread\_join(load\_thread, 0);

free\_data(train);

train = buffer;

load\_thread = load\_data(args);

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

resize\_network(nets + k, init\_w, init\_h);

}

net = nets[0];

}

copy\_weights\_net(net, &net\_map);

// combine Training and Validation networks

//network net\_combined = combine\_train\_valid\_networks(net, net\_map);

iter\_map = iteration;

mean\_average\_precision = validate\_detector\_map(datacfg, cfgfile, weightfile, 0.25, 0.5, 0, net.letter\_box, &net\_map);// &net\_combined);

printf("\n mean\_average\_precision (mAP@0.5) = %f \n", mean\_average\_precision);

if (mean\_average\_precision > best\_map) {

best\_map = mean\_average\_precision;

printf("New best mAP!\n");

char buff[256];

sprintf(buff, "%s/%s\_best.weights", backup\_directory, base);

save\_weights(net, buff);

}

draw\_precision = 1;

}

time\_remaining = ((net.max\_batches - iteration) / ngpus) \* (what\_time\_is\_it\_now() - time + load\_time) / 60 / 60;

// set initial value, even if resume training from 10000 iteration

if (avg\_time < 0) avg\_time = time\_remaining;

else avg\_time = alpha\_time \* time\_remaining + (1 - alpha\_time) \* avg\_time;

#ifdef OPENCV

if (net.contrastive) {

float cur\_con\_acc = -1;

for (k = 0; k < net.n; ++k)

if (net.layers[k].type == CONTRASTIVE) cur\_con\_acc = \*net.layers[k].loss;

if (cur\_con\_acc >= 0) avg\_contrastive\_acc = avg\_contrastive\_acc \* 0.99 + cur\_con\_acc \* 0.01;

printf(" avg\_contrastive\_acc = %f \n", avg\_contrastive\_acc);

}

draw\_train\_loss(windows\_name, img, img\_size, avg\_loss, max\_img\_loss, iteration, net.max\_batches, mean\_average\_precision, draw\_precision, "mAP%", avg\_contrastive\_acc / 100, dont\_show, mjpeg\_port, avg\_time);

#endif // OPENCV

//if (i % 1000 == 0 || (i < 1000 && i % 100 == 0)) {

//if (i % 100 == 0) {

if (iteration >= (iter\_save + 1000) || iteration % 1000 == 0) {

iter\_save = iteration;

#ifdef GPU

if (ngpus != 1) sync\_nets(nets, ngpus, 0);

#endif

char buff[256];

sprintf(buff, "%s/%s\_%d.weights", backup\_directory, base, iteration);

save\_weights(net, buff);

}

if (iteration >= (iter\_save\_last + 100) || (iteration % 100 == 0 && iteration > 1)) {

iter\_save\_last = iteration;

#ifdef GPU

if (ngpus != 1) sync\_nets(nets, ngpus, 0);

#endif

char buff[256];

sprintf(buff, "%s/%s\_last.weights", backup\_directory, base);

save\_weights(net, buff);

}

free\_data(train);

}

#ifdef GPU

if (ngpus != 1) sync\_nets(nets, ngpus, 0);

#endif

char buff[256];

sprintf(buff, "%s/%s\_final.weights", backup\_directory, base);

save\_weights(net, buff);

printf("If you want to train from the beginning, then use flag in the end of training command: -clear \n");

#ifdef OPENCV

release\_mat(&img);

destroy\_all\_windows\_cv();

#endif

// free memory

pthread\_join(load\_thread, 0);

free\_data(buffer);

free\_load\_threads(&args);

free(base);

free(paths);

free\_list\_contents(plist);

free\_list(plist);

free\_list\_contents\_kvp(options);

free\_list(options);

for (k = 0; k < ngpus; ++k) free\_network(nets[k]);

free(nets);

//free\_network(net);

if (calc\_map) {

net\_map.n = 0;

free\_network(net\_map);

}

}

## Image.c

//Ryolo

//Ryolo

image border\_gray\_image(image a, int border)

{

image b = make\_image(a.w + 2 \* border, a.h + 2 \* border, 1);

int x, y, k;

for (k = 0; k < b.c; ++k) {

for (y = 0; y < b.h; ++y) {

for (x = 0; x < b.w; ++x) {

float val = get\_pixel\_extend(a, x - border, y - border, k);

if (x - border < 0 || x - border >= a.w || y - border < 0 || y - border >= a.h) val = 1;

set\_pixel(b, x, y, k, val);

}

}

}

return b;

}