#include "network.h"

#include "detection\_layer.h"

#include "cost\_layer.h"

#include "utils.h"

#include "parser.h"

#include "box.h"

#include "demo.h"

char \*voc\_names[] = {"aeroplane", "bicycle", "bird", "boat", "bottle", "bus", "car", "cat", "chair", "cow", "diningtable", "dog", "horse", "motorbike", "person", "pottedplant", "sheep", "sofa", "train", "tvmonitor"};

void train\_yolo(char \*cfgfile, char \*weightfile)

{

char\* train\_images = "data/voc/train.txt";

char\* backup\_directory = "backup/";

srand(time(0));

char \*base = basecfg(cfgfile);

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

float avg\_loss = -1;

network net = parse\_network\_cfg(cfgfile);

if(weightfile){

load\_weights(&net, weightfile);

}

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

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

int i = \*net.seen/imgs;

data train, buffer;

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

int side = l.side;

int classes = l.classes;

float jitter = l.jitter;

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

//int N = plist->size;

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

load\_args args = {0};

args.w = net.w;

args.h = net.h;

args.paths = paths;

args.n = imgs;

args.m = plist->size;

args.classes = classes;

args.jitter = jitter;

args.num\_boxes = side;

args.d = &buffer;

args.type = REGION\_DATA;

args.angle = net.angle;

args.exposure = net.exposure;

args.saturation = net.saturation;

args.hue = net.hue;

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

clock\_t time;

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

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

i += 1;

time=clock();

pthread\_join(load\_thread, 0);

train = buffer;

load\_thread = load\_data\_in\_thread(args);

printf("Loaded: %lf seconds\n", sec(clock()-time));

time=clock();

float loss = train\_network(net, train);

if (avg\_loss < 0) avg\_loss = loss;

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

printf("%d: %f, %f avg, %f rate, %lf seconds, %d images\n", i, loss, avg\_loss, get\_current\_rate(net), sec(clock()-time), i\*imgs);

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

char buff[256];

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

save\_weights(net, buff);

}

free\_data(train);

}

char buff[256];

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

save\_weights(net, buff);

}

void print\_yolo\_detections(FILE \*\*fps, char \*id, box \*boxes, float \*\*probs, int total, int classes, int w, int h)

{

int i, j;

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

float xmin = boxes[i].x - boxes[i].w/2.;

float xmax = boxes[i].x + boxes[i].w/2.;

float ymin = boxes[i].y - boxes[i].h/2.;

float ymax = boxes[i].y + boxes[i].h/2.;

if (xmin < 0) xmin = 0;

if (ymin < 0) ymin = 0;

if (xmax > w) xmax = w;

if (ymax > h) ymax = h;

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

if (probs[i][j]) fprintf(fps[j], "%s %f %f %f %f %f\n", id, probs[i][j],

xmin, ymin, xmax, ymax);

}

}

}

void validate\_yolo(char \*cfgfile, char \*weightfile)

{

network net = parse\_network\_cfg(cfgfile);

if(weightfile){

load\_weights(&net, weightfile);

}

set\_batch\_network(&net, 1);

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

srand(time(0));

char \*base = "results/comp4\_det\_test\_";

//list \*plist = get\_paths("data/voc.2007.test");

list\* plist = get\_paths("data/voc/2007\_test.txt");

//list \*plist = get\_paths("data/voc.2012.test");

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

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

int classes = l.classes;

int j;

FILE\*\* fps = (FILE\*\*)xcalloc(classes, sizeof(FILE\*));

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

char buff[1024];

snprintf(buff, 1024, "%s%s.txt", base, voc\_names[j]);

fps[j] = fopen(buff, "w");

}

box\* boxes = (box\*)xcalloc(l.side \* l.side \* l.n, sizeof(box));

float\*\* probs = (float\*\*)xcalloc(l.side \* l.side \* l.n, sizeof(float\*));

for(j = 0; j < l.side\*l.side\*l.n; ++j) probs[j] = (float\*)xcalloc(classes, sizeof(float));

int m = plist->size;

int i=0;

int t;

float thresh = .001;

int nms = 1;

float iou\_thresh = .5;

int nthreads = 8;

image\* val = (image\*)xcalloc(nthreads, sizeof(image));

image\* val\_resized = (image\*)xcalloc(nthreads, sizeof(image));

image\* buf = (image\*)xcalloc(nthreads, sizeof(image));

image\* buf\_resized = (image\*)xcalloc(nthreads, sizeof(image));

pthread\_t\* thr = (pthread\_t\*)xcalloc(nthreads, sizeof(pthread\_t));

load\_args args = {0};

args.w = net.w;

args.h = net.h;

args.type = IMAGE\_DATA;

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

args.path = paths[i+t];

args.im = &buf[t];

args.resized = &buf\_resized[t];

thr[t] = load\_data\_in\_thread(args);

}

time\_t start = time(0);

for(i = nthreads; i < m+nthreads; i += nthreads){

fprintf(stderr, "%d\n", i);

for(t = 0; t < nthreads && i+t-nthreads < m; ++t){

pthread\_join(thr[t], 0);

val[t] = buf[t];

val\_resized[t] = buf\_resized[t];

}

for(t = 0; t < nthreads && i+t < m; ++t){

args.path = paths[i+t];

args.im = &buf[t];

args.resized = &buf\_resized[t];

thr[t] = load\_data\_in\_thread(args);

}

for(t = 0; t < nthreads && i+t-nthreads < m; ++t){

char \*path = paths[i+t-nthreads];

char \*id = basecfg(path);

float \*X = val\_resized[t].data;

network\_predict(net, X);

int w = val[t].w;

int h = val[t].h;

get\_detection\_boxes(l, w, h, thresh, probs, boxes, 0);

if (nms) do\_nms\_sort\_v2(boxes, probs, l.side\*l.side\*l.n, classes, iou\_thresh);

print\_yolo\_detections(fps, id, boxes, probs, l.side\*l.side\*l.n, classes, w, h);

free(id);

free\_image(val[t]);

free\_image(val\_resized[t]);

}

}

if (fps) free(fps);

if (val) free(val);

if (val\_resized) free(val\_resized);

if (buf) free(buf);

if (buf\_resized) free(buf\_resized);

if (thr) free(thr);

fprintf(stderr, "Total Detection Time: %f Seconds\n", (double)(time(0) - start));

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

fclose(fps[j]);

}

free(fps);

}

void validate\_yolo\_recall(char \*cfgfile, char \*weightfile)

{

network net = parse\_network\_cfg(cfgfile);

if(weightfile){

load\_weights(&net, weightfile);

}

set\_batch\_network(&net, 1);

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

srand(time(0));

list \*plist = get\_paths("data/voc.2007.test");

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

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

int classes = l.classes;

int side = l.side;

int j, k;

box\* boxes = (box\*)xcalloc(side \* side \* l.n, sizeof(box));

float\*\* probs = (float\*\*)xcalloc(side \* side \* l.n, sizeof(float\*));

for(j = 0; j < side\*side\*l.n; ++j) {

probs[j] = (float\*)xcalloc(classes, sizeof(float));

}

int m = plist->size;

int i=0;

float thresh = .001;

float iou\_thresh = .5;

float nms = 0;

int total = 0;

int correct = 0;

int proposals = 0;

float avg\_iou = 0;

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

char \*path = paths[i];

image orig = load\_image\_color(path, 0, 0);

image sized = resize\_image(orig, net.w, net.h);

char \*id = basecfg(path);

network\_predict(net, sized.data);

get\_detection\_boxes(l, orig.w, orig.h, thresh, probs, boxes, 1);

if (nms) do\_nms(boxes, probs, side\*side\*l.n, 1, nms);

char labelpath[4096];

replace\_image\_to\_label(path, labelpath);

int num\_labels = 0;

box\_label \*truth = read\_boxes(labelpath, &num\_labels);

for(k = 0; k < side\*side\*l.n; ++k){

if(probs[k][0] > thresh){

++proposals;

}

}

for (j = 0; j < num\_labels; ++j) {

++total;

box t = {truth[j].x, truth[j].y, truth[j].w, truth[j].h};

float best\_iou = 0;

for(k = 0; k < side\*side\*l.n; ++k){

float iou = box\_iou(boxes[k], t);

if(probs[k][0] > thresh && iou > best\_iou){

best\_iou = iou;

}

}

avg\_iou += best\_iou;

if(best\_iou > iou\_thresh){

++correct;

}

}

fprintf(stderr, "%5d %5d %5d\tRPs/Img: %.2f\tIOU: %.2f%%\tRecall:%.2f%%\n", i, correct, total, (float)proposals/(i+1), avg\_iou\*100/total, 100.\*correct/total);

free(id);

free\_image(orig);

free\_image(sized);

}

}

void test\_yolo(char \*cfgfile, char \*weightfile, char \*filename, float thresh)

{

image \*\*alphabet = load\_alphabet();

network net = parse\_network\_cfg(cfgfile);

if(weightfile){

load\_weights(&net, weightfile);

}

detection\_layer l = net.layers[net.n-1];

set\_batch\_network(&net, 1);

srand(2222222);

char buff[256];

char \*input = buff;

int j;

float nms=.4;

box\* boxes = (box\*)xcalloc(l.side \* l.side \* l.n, sizeof(box));

float\*\* probs = (float\*\*)xcalloc(l.side \* l.side \* l.n, sizeof(float\*));

for(j = 0; j < l.side\*l.side\*l.n; ++j) {

probs[j] = (float\*)xcalloc(l.classes, sizeof(float));

}

while(1){

if(filename){

strncpy(input, filename, 256);

} else {

printf("Enter Image Path: ");

fflush(stdout);

input = fgets(input, 256, stdin);

if(!input) return;

strtok(input, "\n");

}

image im = load\_image\_color(input,0,0);

image sized = resize\_image(im, net.w, net.h);

float \*X = sized.data;

clock\_t time=clock();

network\_predict(net, X);

printf("%s: Predicted in %f seconds.\n", input, sec(clock()-time));

get\_detection\_boxes(l, 1, 1, thresh, probs, boxes, 0);

if (nms) do\_nms\_sort\_v2(boxes, probs, l.side\*l.side\*l.n, l.classes, nms);

//draw\_detections(im, l.side\*l.side\*l.n, thresh, boxes, probs, voc\_names, alphabet, 20);

draw\_detections(im, l.side\*l.side\*l.n, thresh, boxes, probs, voc\_names, alphabet, 20);

save\_image(im, "predictions");

show\_image(im, "predictions");

free\_image(im);

free\_image(sized);

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

destroy\_all\_windows\_cv();

if (filename) break;

}

free(boxes);

for(j = 0; j < l.side\*l.side\*l.n; ++j) {

free(probs[j]);

}

free(probs);

}

void run\_yolo(int argc, char \*\*argv)

{

int dont\_show = find\_arg(argc, argv, "-dont\_show");

int mjpeg\_port = find\_int\_arg(argc, argv, "-mjpeg\_port", -1);

int json\_port = find\_int\_arg(argc, argv, "-json\_port", -1);

char \*out\_filename = find\_char\_arg(argc, argv, "-out\_filename", 0);

char \*prefix = find\_char\_arg(argc, argv, "-prefix", 0);

float thresh = find\_float\_arg(argc, argv, "-thresh", .2);

float hier\_thresh = find\_float\_arg(argc, argv, "-hier", .5);

int cam\_index = find\_int\_arg(argc, argv, "-c", 0);

int frame\_skip = find\_int\_arg(argc, argv, "-s", 0);

int ext\_output = find\_arg(argc, argv, "-ext\_output");

if(argc < 4){

fprintf(stderr, "usage: %s %s [train/test/valid] [cfg] [weights (optional)]\n", argv[0], argv[1]);

return;

}

char \*cfg = argv[3];

char \*weights = (argc > 4) ? argv[4] : 0;

char \*filename = (argc > 5) ? argv[5]: 0;

if(0==strcmp(argv[2], "test")) test\_yolo(cfg, weights, filename, thresh);

else if(0==strcmp(argv[2], "train")) train\_yolo(cfg, weights);

else if(0==strcmp(argv[2], "valid")) validate\_yolo(cfg, weights);

else if(0==strcmp(argv[2], "recall")) validate\_yolo\_recall(cfg, weights);

else if(0==strcmp(argv[2], "demo")) demo(cfg, weights, thresh, hier\_thresh, cam\_index, filename, voc\_names, 20, frame\_skip,

prefix, out\_filename, mjpeg\_port, json\_port, dont\_show, ext\_output, 0, 0, 0, 0, 0);

}