#include "network.h"

#include "cost\_layer.h"

#include "utils.h"

#include "parser.h"

void extract\_voxel(char \*lfile, char \*rfile, char \*prefix)

{

#ifdef OPENCV

int w = 1920;

int h = 1080;

int shift = 0;

int count = 0;

cap\_cv \*lcap = get\_capture\_video\_stream(lfile);

cap\_cv \*rcap = get\_capture\_video\_stream(rfile);

while(1){

image l = get\_image\_from\_stream\_cpp(lcap);

image r = get\_image\_from\_stream\_cpp(rcap);

if(!l.w || !r.w) break;

if(count%100 == 0) {

shift = best\_3d\_shift\_r(l, r, -l.h/100, l.h/100);

printf("%d\n", shift);

}

image ls = crop\_image(l, (l.w - w)/2, (l.h - h)/2, w, h);

image rs = crop\_image(r, 105 + (r.w - w)/2, (r.h - h)/2 + shift, w, h);

char buff[256];

sprintf(buff, "%s\_%05d\_l", prefix, count);

save\_image(ls, buff);

sprintf(buff, "%s\_%05d\_r", prefix, count);

save\_image(rs, buff);

free\_image(l);

free\_image(r);

free\_image(ls);

free\_image(rs);

++count;

}

#else

printf("need OpenCV for extraction\n");

#endif

}

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

{

char\* train\_images = "data/imagenet/imagenet1k.train.list";

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;

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.scale = 4;

args.paths = paths;

args.n = imgs;

args.m = plist->size;

args.d = &buffer;

args.type = SUPER\_DATA;

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){

char buff[256];

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

save\_weights(net, buff);

}

if(i%100==0){

char buff[256];

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

save\_weights(net, buff);

}

free\_data(train);

}

char buff[256];

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

save\_weights(net, buff);

}

void test\_voxel(char \*cfgfile, char \*weightfile, char \*filename)

{

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

if(weightfile){

load\_weights(&net, weightfile);

}

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

srand(2222222);

clock\_t time;

char buff[256];

char \*input = buff;

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);

resize\_network(&net, im.w, im.h);

printf("%d %d\n", im.w, im.h);

float \*X = im.data;

time=clock();

network\_predict(net, X);

image out = get\_network\_image(net);

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

save\_image(out, "out");

free\_image(im);

if (filename) break;

}

}

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

{

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], "train")) train\_voxel(cfg, weights);

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

else if(0==strcmp(argv[2], "extract")) extract\_voxel(argv[3], argv[4], argv[5]);

/\*

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

\*/

}