#include <stdio.h>

#include "network.h"

#include "detection\_layer.h"

#include "cost\_layer.h"

#include "utils.h"

#include "parser.h"

#include "box.h"

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

{

srand(time(0));

float avg\_loss = -1;

char \*base = basecfg(cfgfile);

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

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

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 = 1024;

list \*plist = get\_paths("data/compare.train.list");

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

int N = plist->size;

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

clock\_t time;

pthread\_t load\_thread;

data train;

data buffer;

load\_args args = {0};

args.w = net.w;

args.h = net.h;

args.paths = paths;

args.classes = 20;

args.n = imgs;

args.m = N;

args.d = &buffer;

args.type = COMPARE\_DATA;

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

int epoch = \*net.seen/N;

int i = 0;

while(1){

++i;

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 == -1) avg\_loss = loss;

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

printf("%.3f: %f, %f avg, %lf seconds, %ld images\n", (float)\*net.seen/N, loss, avg\_loss, sec(clock()-time), \*net.seen);

free\_data(train);

if(i%100 == 0){

char buff[256];

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

save\_weights(net, buff);

}

if(\*net.seen/N > epoch){

epoch = \*net.seen/N;

i = 0;

char buff[256];

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

save\_weights(net, buff);

if(epoch%22 == 0) net.learning\_rate \*= .1;

}

}

pthread\_join(load\_thread, 0);

free\_data(buffer);

free\_network(net);

free\_ptrs((void\*\*)paths, plist->size);

free\_list(plist);

free(base);

}

void validate\_compare(char \*filename, char \*weightfile)

{

int i = 0;

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

if(weightfile){

load\_weights(&net, weightfile);

}

srand(time(0));

list \*plist = get\_paths("data/compare.val.list");

//list \*plist = get\_paths("data/compare.val.old");

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

int N = plist->size/2;

free\_list(plist);

clock\_t time;

int correct = 0;

int total = 0;

int splits = 10;

int num = (i+1)\*N/splits - i\*N/splits;

data val, buffer;

load\_args args = {0};

args.w = net.w;

args.h = net.h;

args.paths = paths;

args.classes = 20;

args.n = num;

args.m = 0;

args.d = &buffer;

args.type = COMPARE\_DATA;

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

for(i = 1; i <= splits; ++i){

time=clock();

pthread\_join(load\_thread, 0);

val = buffer;

num = (i+1)\*N/splits - i\*N/splits;

char \*\*part = paths+(i\*N/splits);

if(i != splits){

args.paths = part;

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

}

printf("Loaded: %d images in %lf seconds\n", val.X.rows, sec(clock()-time));

time=clock();

matrix pred = network\_predict\_data(net, val);

int j,k;

for(j = 0; j < val.y.rows; ++j){

for(k = 0; k < 20; ++k){

if(val.y.vals[j][k\*2] != val.y.vals[j][k\*2+1]){

++total;

if((val.y.vals[j][k\*2] < val.y.vals[j][k\*2+1]) == (pred.vals[j][k\*2] < pred.vals[j][k\*2+1])){

++correct;

}

}

}

}

free\_matrix(pred);

printf("%d: Acc: %f, %lf seconds, %d images\n", i, (float)correct/total, sec(clock()-time), val.X.rows);

free\_data(val);

}

}

typedef struct {

network net;

char \*filename;

int class\_id;

int classes;

float elo;

float \*elos;

} sortable\_bbox;

int total\_compares = 0;

int current\_class\_id = 0;

int elo\_comparator(const void\*a, const void \*b)

{

sortable\_bbox box1 = \*(sortable\_bbox\*)a;

sortable\_bbox box2 = \*(sortable\_bbox\*)b;

if(box1.elos[current\_class\_id] == box2.elos[current\_class\_id]) return 0;

if(box1.elos[current\_class\_id] > box2.elos[current\_class\_id]) return -1;

return 1;

}

int bbox\_comparator(const void \*a, const void \*b)

{

++total\_compares;

sortable\_bbox box1 = \*(sortable\_bbox\*)a;

sortable\_bbox box2 = \*(sortable\_bbox\*)b;

network net = box1.net;

int class\_id = box1.class\_id;

image im1 = load\_image\_color(box1.filename, net.w, net.h);

image im2 = load\_image\_color(box2.filename, net.w, net.h);

float\* X = (float\*)xcalloc(net.w \* net.h \* net.c, sizeof(float));

memcpy(X, im1.data, im1.w\*im1.h\*im1.c\*sizeof(float));

memcpy(X+im1.w\*im1.h\*im1.c, im2.data, im2.w\*im2.h\*im2.c\*sizeof(float));

float \*predictions = network\_predict(net, X);

free\_image(im1);

free\_image(im2);

free(X);

if (predictions[class\_id\*2] > predictions[class\_id\*2+1]){

return 1;

}

return -1;

}

void bbox\_update(sortable\_bbox \*a, sortable\_bbox \*b, int class\_id, int result)

{

int k = 32;

float EA = 1./(1+pow(10, (b->elos[class\_id] - a->elos[class\_id])/400.));

float EB = 1./(1+pow(10, (a->elos[class\_id] - b->elos[class\_id])/400.));

float SA = result ? 1 : 0;

float SB = result ? 0 : 1;

a->elos[class\_id] += k\*(SA - EA);

b->elos[class\_id] += k\*(SB - EB);

}

void bbox\_fight(network net, sortable\_bbox \*a, sortable\_bbox \*b, int classes, int class\_id)

{

image im1 = load\_image\_color(a->filename, net.w, net.h);

image im2 = load\_image\_color(b->filename, net.w, net.h);

float\* X = (float\*)xcalloc(net.w \* net.h \* net.c, sizeof(float));

memcpy(X, im1.data, im1.w\*im1.h\*im1.c\*sizeof(float));

memcpy(X+im1.w\*im1.h\*im1.c, im2.data, im2.w\*im2.h\*im2.c\*sizeof(float));

float \*predictions = network\_predict(net, X);

++total\_compares;

int i;

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

if(class\_id < 0 || class\_id == i){

int result = predictions[i\*2] > predictions[i\*2+1];

bbox\_update(a, b, i, result);

}

}

free\_image(im1);

free\_image(im2);

free(X);

}

void SortMaster3000(char \*filename, char \*weightfile)

{

int i = 0;

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

if(weightfile){

load\_weights(&net, weightfile);

}

srand(time(0));

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

list \*plist = get\_paths("data/compare.sort.list");

//list \*plist = get\_paths("data/compare.val.old");

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

int N = plist->size;

free\_list(plist);

sortable\_bbox\* boxes = (sortable\_bbox\*)xcalloc(N, sizeof(sortable\_bbox));

printf("Sorting %d boxes...\n", N);

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

boxes[i].filename = paths[i];

boxes[i].net = net;

boxes[i].class\_id = 7;

boxes[i].elo = 1500;

}

clock\_t time=clock();

qsort(boxes, N, sizeof(sortable\_bbox), bbox\_comparator);

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

printf("%s\n", boxes[i].filename);

}

printf("Sorted in %d compares, %f secs\n", total\_compares, sec(clock()-time));

}

void BattleRoyaleWithCheese(char \*filename, char \*weightfile)

{

int classes = 20;

int i,j;

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

if(weightfile){

load\_weights(&net, weightfile);

}

srand(time(0));

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

list \*plist = get\_paths("data/compare.sort.list");

//list \*plist = get\_paths("data/compare.small.list");

//list \*plist = get\_paths("data/compare.cat.list");

//list \*plist = get\_paths("data/compare.val.old");

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

int N = plist->size;

int total = N;

free\_list(plist);

sortable\_bbox\* boxes = (sortable\_bbox\*)xcalloc(N, sizeof(sortable\_bbox));

printf("Battling %d boxes...\n", N);

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

boxes[i].filename = paths[i];

boxes[i].net = net;

boxes[i].classes = classes;

boxes[i].elos = (float\*)xcalloc(classes, sizeof(float));

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

boxes[i].elos[j] = 1500;

}

}

int round;

clock\_t time=clock();

for(round = 1; round <= 4; ++round){

clock\_t round\_time=clock();

printf("Round: %d\n", round);

shuffle(boxes, N, sizeof(sortable\_bbox));

for(i = 0; i < N/2; ++i){

bbox\_fight(net, boxes+i\*2, boxes+i\*2+1, classes, -1);

}

printf("Round: %f secs, %d remaining\n", sec(clock()-round\_time), N);

}

int class\_id;

for (class\_id = 0; class\_id < classes; ++class\_id){

N = total;

current\_class\_id = class\_id;

qsort(boxes, N, sizeof(sortable\_bbox), elo\_comparator);

N /= 2;

for(round = 1; round <= 100; ++round){

clock\_t round\_time=clock();

printf("Round: %d\n", round);

sorta\_shuffle(boxes, N, sizeof(sortable\_bbox), 10);

for(i = 0; i < N/2; ++i){

bbox\_fight(net, boxes+i\*2, boxes+i\*2+1, classes, class\_id);

}

qsort(boxes, N, sizeof(sortable\_bbox), elo\_comparator);

if(round <= 20) N = (N\*9/10)/2\*2;

printf("Round: %f secs, %d remaining\n", sec(clock()-round\_time), N);

}

char buff[256];

sprintf(buff, "results/battle\_%d.log", class\_id);

FILE \*outfp = fopen(buff, "w");

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

fprintf(outfp, "%s %f\n", boxes[i].filename, boxes[i].elos[class\_id]);

}

fclose(outfp);

}

printf("Tournament in %d compares, %f secs\n", total\_compares, sec(clock()-time));

}

void run\_compare(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\_compare(cfg, weights);

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

else if(0==strcmp(argv[2], "sort")) SortMaster3000(cfg, weights);

else if(0==strcmp(argv[2], "battle")) BattleRoyaleWithCheese(cfg, weights);

/\*

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

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

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

\*/

}