#include "network.h"

#include "cost\_layer.h"

#include "utils.h"

#include "parser.h"

#include "blas.h"

#ifdef OPENCV

void reconstruct\_picture(network net, float \*features, image recon, image update, float rate, float momentum, float lambda, int smooth\_size, int iters);

typedef struct {

float \*x;

float \*y;

} float\_pair;

float\_pair get\_rnn\_vid\_data(network net, char \*\*files, int n, int batch, int steps)

{

int b;

assert(net.batch == steps + 1);

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

int output\_size = out\_im.w\*out\_im.h\*out\_im.c;

printf("%d %d %d\n", out\_im.w, out\_im.h, out\_im.c);

float\* feats = (float\*)xcalloc(net.batch \* batch \* output\_size, sizeof(float));

for(b = 0; b < batch; ++b){

int input\_size = net.w\*net.h\*net.c;

float\* input = (float\*)xcalloc(input\_size \* net.batch, sizeof(float));

char \*filename = files[rand()%n];

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

int frames = get\_capture\_frame\_count\_cv(cap);

int index = rand() % (frames - steps - 2);

if (frames < (steps + 4)){

--b;

free(input);

continue;

}

printf("frames: %d, index: %d\n", frames, index);

set\_capture\_position\_frame\_cv(cap, index);

int i;

for(i = 0; i < net.batch; ++i){

mat\_cv \*src = get\_capture\_frame\_cv(cap);

image im = mat\_to\_image\_cv(src);

rgbgr\_image(im);

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

//show\_image(re, "loaded");

//cvWaitKey(10);

memcpy(input + i\*input\_size, re.data, input\_size\*sizeof(float));

free\_image(im);

free\_image(re);

}

float \*output = network\_predict(net, input);

free(input);

for(i = 0; i < net.batch; ++i){

memcpy(feats + (b + i\*batch)\*output\_size, output + i\*output\_size, output\_size\*sizeof(float));

}

release\_capture(cap); //cvReleaseCapture(&cap);

}

//printf("%d %d %d\n", out\_im.w, out\_im.h, out\_im.c);

float\_pair p = {0};

p.x = feats;

p.y = feats + output\_size\*batch; //+ out\_im.w\*out\_im.h\*out\_im.c;

return p;

}

void train\_vid\_rnn(char \*cfgfile, char \*weightfile)

{

char \*train\_videos = "data/vid/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;

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

int N = plist->size;

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

clock\_t time;

int steps = net.time\_steps;

int batch = net.batch / net.time\_steps;

network extractor = parse\_network\_cfg("cfg/extractor.cfg");

load\_weights(&extractor, "trained/yolo-coco.conv");

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

i += 1;

time=clock();

float\_pair p = get\_rnn\_vid\_data(extractor, paths, N, batch, steps);

float loss = train\_network\_datum(net, p.x, p.y) / (net.batch);

free(p.x);

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

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

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

if(i%100==0){

char buff[256];

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

save\_weights(net, buff);

}

if(i%10==0){

char buff[256];

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

save\_weights(net, buff);

}

}

char buff[256];

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

save\_weights(net, buff);

}

image save\_reconstruction(network net, image \*init, float \*feat, char \*name, int i)

{

image recon;

if (init) {

recon = copy\_image(\*init);

} else {

recon = make\_random\_image(net.w, net.h, 3);

}

image update = make\_image(net.w, net.h, 3);

reconstruct\_picture(net, feat, recon, update, .01, .9, .1, 2, 50);

char buff[256];

sprintf(buff, "%s%d", name, i);

save\_image(recon, buff);

free\_image(update);

return recon;

}

void generate\_vid\_rnn(char \*cfgfile, char \*weightfile)

{

network extractor = parse\_network\_cfg("cfg/extractor.recon.cfg");

load\_weights(&extractor, "trained/yolo-coco.conv");

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

if(weightfile){

load\_weights(&net, weightfile);

}

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

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

int i;

cap\_cv \*cap = get\_capture\_video\_stream("extra/vid/ILSVRC2015/Data/VID/snippets/val/ILSVRC2015\_val\_00007030.mp4");

//CvCapture\* cap = cvCaptureFromFile("extra/vid/ILSVRC2015/Data/VID/snippets/val/ILSVRC2015\_val\_00007030.mp4");

float \*feat;

float \*next;

next = NULL;

image last;

for(i = 0; i < 25; ++i){

image im = get\_image\_from\_stream\_cpp(cap);

image re = resize\_image(im, extractor.w, extractor.h);

feat = network\_predict(extractor, re.data);

if(i > 0){

printf("%f %f\n", mean\_array(feat, 14\*14\*512), variance\_array(feat, 14\*14\*512));

printf("%f %f\n", mean\_array(next, 14\*14\*512), variance\_array(next, 14\*14\*512));

printf("%f\n", mse\_array(feat, 14\*14\*512));

axpy\_cpu(14\*14\*512, -1, feat, 1, next, 1);

printf("%f\n", mse\_array(next, 14\*14\*512));

}

next = network\_predict(net, feat);

free\_image(im);

free\_image(save\_reconstruction(extractor, 0, feat, "feat", i));

free\_image(save\_reconstruction(extractor, 0, next, "next", i));

if (i==24) last = copy\_image(re);

free\_image(re);

}

for(i = 0; i < 30; ++i){

next = network\_predict(net, next);

image newimage = save\_reconstruction(extractor, &last, next, "newimage", i);

free\_image(last);

last = newimage;

}

}

void run\_vid\_rnn(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\_vid\_rnn(cfg, weights);

else if(0==strcmp(argv[2], "generate")) generate\_vid\_rnn(cfg, weights);

}

#else

void run\_vid\_rnn(int argc, char \*\*argv){}

#endif