#include "network.h"

#include "parser.h"

#include "blas.h"

#include "utils.h"

// ./darknet nightmare cfg/extractor.recon.cfg ~/trained/yolo-coco.conv frame6.png -reconstruct -iters 500 -i 3 -lambda .1 -rate .01 -smooth 2

float abs\_mean(float \*x, int n)

{

int i;

float sum = 0;

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

sum += fabs(x[i]);

}

return sum/n;

}

void calculate\_loss(float \*output, float \*delta, int n, float thresh)

{

int i;

float mean = mean\_array(output, n);

float var = variance\_array(output, n);

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

if(delta[i] > mean + thresh\*sqrt(var)) delta[i] = output[i];

else delta[i] = 0;

}

}

void optimize\_picture(network \*net, image orig, int max\_layer, float scale, float rate, float thresh, int norm)

{

//scale\_image(orig, 2);

//translate\_image(orig, -1);

net->n = max\_layer + 1;

int dx = rand()%16 - 8;

int dy = rand()%16 - 8;

int flip = rand()%2;

image crop = crop\_image(orig, dx, dy, orig.w, orig.h);

image im = resize\_image(crop, (int)(orig.w \* scale), (int)(orig.h \* scale));

if(flip) flip\_image(im);

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

layer last = net->layers[net->n-1];

//net->layers[net->n - 1].activation = LINEAR;

image delta = make\_image(im.w, im.h, im.c);

network\_state state = {0};

#ifdef GPU

state.input = cuda\_make\_array(im.data, im.w\*im.h\*im.c);

state.delta = cuda\_make\_array(im.data, im.w\*im.h\*im.c);

forward\_network\_gpu(\*net, state);

copy\_ongpu(last.outputs, last.output\_gpu, 1, last.delta\_gpu, 1);

cuda\_pull\_array(last.delta\_gpu, last.delta, last.outputs);

calculate\_loss(last.delta, last.delta, last.outputs, thresh);

cuda\_push\_array(last.delta\_gpu, last.delta, last.outputs);

backward\_network\_gpu(\*net, state);

cuda\_pull\_array(state.delta, delta.data, im.w\*im.h\*im.c);

cuda\_free(state.input);

cuda\_free(state.delta);

#else

state.input = im.data;

state.delta = delta.data;

forward\_network(\*net, state);

copy\_cpu(last.outputs, last.output, 1, last.delta, 1);

calculate\_loss(last.output, last.delta, last.outputs, thresh);

backward\_network(\*net, state);

#endif

if(flip) flip\_image(delta);

//normalize\_array(delta.data, delta.w\*delta.h\*delta.c);

image resized = resize\_image(delta, orig.w, orig.h);

image out = crop\_image(resized, -dx, -dy, orig.w, orig.h);

/\*

image g = grayscale\_image(out);

free\_image(out);

out = g;

\*/

//rate = rate / abs\_mean(out.data, out.w\*out.h\*out.c);

if(norm) normalize\_array(out.data, out.w\*out.h\*out.c);

axpy\_cpu(orig.w\*orig.h\*orig.c, rate, out.data, 1, orig.data, 1);

/\*

normalize\_array(orig.data, orig.w\*orig.h\*orig.c);

scale\_image(orig, sqrt(var));

translate\_image(orig, mean);

\*/

//translate\_image(orig, 1);

//scale\_image(orig, .5);

//normalize\_image(orig);

constrain\_image(orig);

free\_image(crop);

free\_image(im);

free\_image(delta);

free\_image(resized);

free\_image(out);

}

void smooth(image recon, image update, float lambda, int num)

{

int i, j, k;

int ii, jj;

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

for(j = 0; j < recon.h; ++j){

for(i = 0; i < recon.w; ++i){

int out\_index = i + recon.w\*(j + recon.h\*k);

for(jj = j-num; jj <= j + num && jj < recon.h; ++jj){

if (jj < 0) continue;

for(ii = i-num; ii <= i + num && ii < recon.w; ++ii){

if (ii < 0) continue;

int in\_index = ii + recon.w\*(jj + recon.h\*k);

update.data[out\_index] += lambda \* (recon.data[in\_index] - recon.data[out\_index]);

}

}

}

}

}

}

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

{

int iter = 0;

for (iter = 0; iter < iters; ++iter) {

image delta = make\_image(recon.w, recon.h, recon.c);

network\_state state = {0};

#ifdef GPU

state.input = cuda\_make\_array(recon.data, recon.w\*recon.h\*recon.c);

state.delta = cuda\_make\_array(delta.data, delta.w\*delta.h\*delta.c);

state.truth = cuda\_make\_array(features, get\_network\_output\_size(net));

forward\_network\_gpu(net, state);

backward\_network\_gpu(net, state);

cuda\_pull\_array(state.delta, delta.data, delta.w\*delta.h\*delta.c);

cuda\_free(state.input);

cuda\_free(state.delta);

cuda\_free(state.truth);

#else

state.input = recon.data;

state.delta = delta.data;

state.truth = features;

forward\_network(net, state);

backward\_network(net, state);

#endif

axpy\_cpu(recon.w\*recon.h\*recon.c, 1, delta.data, 1, update.data, 1);

smooth(recon, update, lambda, smooth\_size);

axpy\_cpu(recon.w\*recon.h\*recon.c, rate, update.data, 1, recon.data, 1);

scal\_cpu(recon.w\*recon.h\*recon.c, momentum, update.data, 1);

//float mag = mag\_array(recon.data, recon.w\*recon.h\*recon.c);

//scal\_cpu(recon.w\*recon.h\*recon.c, 600/mag, recon.data, 1);

constrain\_image(recon);

free\_image(delta);

}

}

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

{

srand(time(0));

if(argc < 4){

fprintf(stderr, "usage: %s %s [cfg] [weights] [image] [layer] [options! (optional)]\n", argv[0], argv[1]);

return;

}

char \*cfg = argv[2];

char \*weights = argv[3];

char \*input = argv[4];

int max\_layer = atoi(argv[5]);

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

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

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

int iters = find\_int\_arg(argc, argv, "-iters", 10);

int octaves = find\_int\_arg(argc, argv, "-octaves", 4);

float zoom = find\_float\_arg(argc, argv, "-zoom", 1.);

float rate = find\_float\_arg(argc, argv, "-rate", .04);

float thresh = find\_float\_arg(argc, argv, "-thresh", 1.);

float rotate = find\_float\_arg(argc, argv, "-rotate", 0);

float momentum = find\_float\_arg(argc, argv, "-momentum", .9);

float lambda = find\_float\_arg(argc, argv, "-lambda", .01);

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

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

int smooth\_size = find\_int\_arg(argc, argv, "-smooth", 1);

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

load\_weights(&net, weights);

char \*cfgbase = basecfg(cfg);

char \*imbase = basecfg(input);

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

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

if(0){

float scale = 1;

if(im.w > 512 || im.h > 512){

if(im.w > im.h) scale = 512.0/im.w;

else scale = 512.0/im.h;

}

image resized = resize\_image(im, scale\*im.w, scale\*im.h);

free\_image(im);

im = resized;

}

float \*features = 0;

image update;

if (reconstruct){

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

int zz = 0;

network\_predict(net, im.data);

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

image crop = crop\_image(out\_im, zz, zz, out\_im.w-2\*zz, out\_im.h-2\*zz);

//flip\_image(crop);

image f\_im = resize\_image(crop, out\_im.w, out\_im.h);

free\_image(crop);

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

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

f\_im = resize\_image(f\_im, f\_im.w, f\_im.h);

features = f\_im.data;

int i;

for(i = 0; i < 14\*14\*512; ++i){

features[i] += rand\_uniform(-.19, .19);

}

free\_image(im);

im = make\_random\_image(im.w, im.h, im.c);

update = make\_image(im.w, im.h, im.c);

}

int e;

int n;

for(e = 0; e < rounds; ++e){

fprintf(stderr, "Iteration: ");

fflush(stderr);

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

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

fflush(stderr);

if(reconstruct){

reconstruct\_picture(net, features, im, update, rate, momentum, lambda, smooth\_size, 1);

//if ((n+1)%30 == 0) rate \*= .5;

show\_image(im, "reconstruction");

#ifdef OPENCV

wait\_key\_cv(10);

#endif

}else{

int layer = max\_layer + rand()%range - range/2;

int octave = rand()%octaves;

optimize\_picture(&net, im, layer, 1/pow(1.33333333, octave), rate, thresh, norm);

}

}

fprintf(stderr, "done\n");

if(0){

image g = grayscale\_image(im);

free\_image(im);

im = g;

}

char buff[256];

if (prefix){

sprintf(buff, "%s/%s\_%s\_%d\_%06d",prefix, imbase, cfgbase, max\_layer, e);

}else{

sprintf(buff, "%s\_%s\_%d\_%06d",imbase, cfgbase, max\_layer, e);

}

printf("%d %s\n", e, buff);

save\_image(im, buff);

//show\_image(im, buff);

//wait\_key\_cv(0);

if(rotate){

image rot = rotate\_image(im, rotate);

free\_image(im);

im = rot;

}

image crop = crop\_image(im, im.w \* (1. - zoom)/2., im.h \* (1.-zoom)/2., im.w\*zoom, im.h\*zoom);

image resized = resize\_image(crop, im.w, im.h);

free\_image(im);

free\_image(crop);

im = resized;

}

}