#include "network.h"

#include "detection\_layer.h"

#include "region\_layer.h"

#include "cost\_layer.h"

#include "utils.h"

#include "parser.h"

#include "box.h"

#include "image.h"

#include "demo.h"

#include "darknet.h"

#ifdef WIN32

#include <time.h>

#include "gettimeofday.h"

#else

#include <sys/time.h>

#endif

#ifdef OPENCV

#include "http\_stream.h"

static char \*\*demo\_names;

static image \*\*demo\_alphabet;

static int demo\_classes;

static int nboxes = 0;

static detection \*dets = NULL;

static network net;

static image in\_s ;

static image det\_s;

static cap\_cv \*cap;

static float fps = 0;

static float demo\_thresh = 0;

static int demo\_ext\_output = 0;

static long long int frame\_id = 0;

static int demo\_json\_port = -1;

#define NFRAMES 3

static float\* predictions[NFRAMES];

static int demo\_index = 0;

static mat\_cv\* cv\_images[NFRAMES];

static float \*avg;

mat\_cv\* in\_img;

mat\_cv\* det\_img;

mat\_cv\* show\_img;

static volatile int flag\_exit;

static int letter\_box = 0;

static const int thread\_wait\_ms = 1;

static volatile int run\_fetch\_in\_thread = 0;

static volatile int run\_detect\_in\_thread = 0;

void \*fetch\_in\_thread(void \*ptr)

{

while (!custom\_atomic\_load\_int(&flag\_exit)) {

while (!custom\_atomic\_load\_int(&run\_fetch\_in\_thread)) {

if (custom\_atomic\_load\_int(&flag\_exit)) return 0;

this\_thread\_yield();

}

int dont\_close\_stream = 0; // set 1 if your IP-camera periodically turns off and turns on video-stream

if (letter\_box)

in\_s = get\_image\_from\_stream\_letterbox(cap, net.w, net.h, net.c, &in\_img, dont\_close\_stream);

else

in\_s = get\_image\_from\_stream\_resize(cap, net.w, net.h, net.c, &in\_img, dont\_close\_stream);

if (!in\_s.data) {

printf("Stream closed.\n");

custom\_atomic\_store\_int(&flag\_exit, 1);

custom\_atomic\_store\_int(&run\_fetch\_in\_thread, 0);

//exit(EXIT\_FAILURE);

return 0;

}

//in\_s = resize\_image(in, net.w, net.h);

custom\_atomic\_store\_int(&run\_fetch\_in\_thread, 0);

}

return 0;

}

void \*fetch\_in\_thread\_sync(void \*ptr)

{

custom\_atomic\_store\_int(&run\_fetch\_in\_thread, 1);

while (custom\_atomic\_load\_int(&run\_fetch\_in\_thread)) this\_thread\_sleep\_for(thread\_wait\_ms);

return 0;

}

void \*detect\_in\_thread(void \*ptr)

{

while (!custom\_atomic\_load\_int(&flag\_exit)) {

while (!custom\_atomic\_load\_int(&run\_detect\_in\_thread)) {

if (custom\_atomic\_load\_int(&flag\_exit)) return 0;

this\_thread\_yield();

}

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

float \*X = det\_s.data;

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

memcpy(predictions[demo\_index], prediction, l.outputs \* sizeof(float));

mean\_arrays(predictions, NFRAMES, l.outputs, avg);

l.output = avg;

cv\_images[demo\_index] = det\_img;

det\_img = cv\_images[(demo\_index + NFRAMES / 2 + 1) % NFRAMES];

demo\_index = (demo\_index + 1) % NFRAMES;

if (letter\_box)

dets = get\_network\_boxes(&net, get\_width\_mat(in\_img), get\_height\_mat(in\_img), demo\_thresh, demo\_thresh, 0, 1, &nboxes, 1); // letter box

else

dets = get\_network\_boxes(&net, net.w, net.h, demo\_thresh, demo\_thresh, 0, 1, &nboxes, 0); // resized

custom\_atomic\_store\_int(&run\_detect\_in\_thread, 0);

}

return 0;

}

void \*detect\_in\_thread\_sync(void \*ptr)

{

custom\_atomic\_store\_int(&run\_detect\_in\_thread, 1);

while (custom\_atomic\_load\_int(&run\_detect\_in\_thread)) this\_thread\_sleep\_for(thread\_wait\_ms);

return 0;

}

double get\_wall\_time()

{

struct timeval walltime;

if (gettimeofday(&walltime, NULL)) {

return 0;

}

return (double)walltime.tv\_sec + (double)walltime.tv\_usec \* .000001;

}

void demo(char \*cfgfile, char \*weightfile, float thresh, float hier\_thresh, int cam\_index, const char \*filename, char \*\*names, int classes,

int frame\_skip, char \*prefix, char \*out\_filename, int mjpeg\_port, int json\_port, int dont\_show, int ext\_output, int letter\_box\_in, int time\_limit\_sec, char \*http\_post\_host,

int benchmark, int benchmark\_layers)

{

letter\_box = letter\_box\_in;

in\_img = det\_img = show\_img = NULL;

//skip = frame\_skip;

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

int delay = frame\_skip;

demo\_names = names;

demo\_alphabet = alphabet;

demo\_classes = classes;

demo\_thresh = thresh;

demo\_ext\_output = ext\_output;

demo\_json\_port = json\_port;

printf("Demo\n");

net = parse\_network\_cfg\_custom(cfgfile, 1, 1); // set batch=1

if(weightfile){

load\_weights(&net, weightfile);

}

net.benchmark\_layers = benchmark\_layers;

fuse\_conv\_batchnorm(net);

calculate\_binary\_weights(net);

srand(2222222);

if(filename){

printf("video file: %s\n", filename);

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

}else{

printf("Webcam index: %d\n", cam\_index);

cap = get\_capture\_webcam(cam\_index);

}

if (!cap) {

#ifdef WIN32

printf("Check that you have copied file opencv\_ffmpeg340\_64.dll to the same directory where is darknet.exe \n");

#endif

error("Couldn't connect to webcam.\n");

}

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

int j;

avg = (float \*) calloc(l.outputs, sizeof(float));

for(j = 0; j < NFRAMES; ++j) predictions[j] = (float \*) calloc(l.outputs, sizeof(float));

if (l.classes != demo\_classes) {

printf("\n Parameters don't match: in cfg-file classes=%d, in data-file classes=%d \n", l.classes, demo\_classes);

getchar();

exit(0);

}

flag\_exit = 0;

custom\_thread\_t fetch\_thread = NULL;

custom\_thread\_t detect\_thread = NULL;

if (custom\_create\_thread(&fetch\_thread, 0, fetch\_in\_thread, 0)) error("Thread creation failed");

if (custom\_create\_thread(&detect\_thread, 0, detect\_in\_thread, 0)) error("Thread creation failed");

fetch\_in\_thread\_sync(0); //fetch\_in\_thread(0);

det\_img = in\_img;

det\_s = in\_s;

fetch\_in\_thread\_sync(0); //fetch\_in\_thread(0);

detect\_in\_thread\_sync(0); //fetch\_in\_thread(0);

det\_img = in\_img;

det\_s = in\_s;

for (j = 0; j < NFRAMES / 2; ++j) {

free\_detections(dets, nboxes);

fetch\_in\_thread\_sync(0); //fetch\_in\_thread(0);

detect\_in\_thread\_sync(0); //fetch\_in\_thread(0);

det\_img = in\_img;

det\_s = in\_s;

}

int count = 0;

if(!prefix && !dont\_show){

int full\_screen = 0;

create\_window\_cv("Demo", full\_screen, 1352, 1013);

}

write\_cv\* output\_video\_writer = NULL;

if (out\_filename && !flag\_exit)

{

int src\_fps = 25;

src\_fps = get\_stream\_fps\_cpp\_cv(cap);

output\_video\_writer =

create\_video\_writer(out\_filename, 'D', 'I', 'V', 'X', src\_fps, get\_width\_mat(det\_img), get\_height\_mat(det\_img), 1);

//'H', '2', '6', '4'

//'D', 'I', 'V', 'X'

//'M', 'J', 'P', 'G'

//'M', 'P', '4', 'V'

//'M', 'P', '4', '2'

//'X', 'V', 'I', 'D'

//'W', 'M', 'V', '2'

}

int send\_http\_post\_once = 0;

const double start\_time\_lim = get\_time\_point();

double before = get\_time\_point();

double start\_time = get\_time\_point();

float avg\_fps = 0;

int frame\_counter = 0;

while(1){

++count;

{

const float nms = .45; // 0.4F

int local\_nboxes = nboxes;

detection \*local\_dets = dets;

this\_thread\_yield();

if (!benchmark) custom\_atomic\_store\_int(&run\_fetch\_in\_thread, 1); // if (custom\_create\_thread(&fetch\_thread, 0, fetch\_in\_thread, 0)) error("Thread creation failed");

custom\_atomic\_store\_int(&run\_detect\_in\_thread, 1); // if (custom\_create\_thread(&detect\_thread, 0, detect\_in\_thread, 0)) error("Thread creation failed");

//if (nms) do\_nms\_obj(local\_dets, local\_nboxes, l.classes, nms); // bad results

if (nms) {

if (l.nms\_kind == DEFAULT\_NMS) do\_nms\_sort(local\_dets, local\_nboxes, l.classes, nms);

else diounms\_sort(local\_dets, local\_nboxes, l.classes, nms, l.nms\_kind, l.beta\_nms);

}

//printf("\033[2J");

//printf("\033[1;1H");

//printf("\nFPS:%.1f\n", fps);

printf("Objects:\n\n");

++frame\_id;

if (demo\_json\_port > 0) {

int timeout = 400000;

send\_json(local\_dets, local\_nboxes, l.classes, demo\_names, frame\_id, demo\_json\_port, timeout);

}

//char \*http\_post\_server = "webhook.site/898bbd9b-0ddd-49cf-b81d-1f56be98d870";

if (http\_post\_host && !send\_http\_post\_once) {

int timeout = 3; // 3 seconds

int http\_post\_port = 80; // 443 https, 80 http

if (send\_http\_post\_request(http\_post\_host, http\_post\_port, filename,

local\_dets, nboxes, classes, names, frame\_id, ext\_output, timeout))

{

if (time\_limit\_sec > 0) send\_http\_post\_once = 1;

}

}

if (!benchmark) draw\_detections\_cv\_v3(show\_img, local\_dets, local\_nboxes, demo\_thresh, demo\_names, demo\_alphabet, demo\_classes, demo\_ext\_output);

free\_detections(local\_dets, local\_nboxes);

printf("\nFPS:%.1f \t AVG\_FPS:%.1f\n", fps, avg\_fps);

if(!prefix){

if (!dont\_show) {

show\_image\_mat(show\_img, "Demo");

int c = wait\_key\_cv(1);

if (c == 10) {

if (frame\_skip == 0) frame\_skip = 60;

else if (frame\_skip == 4) frame\_skip = 0;

else if (frame\_skip == 60) frame\_skip = 4;

else frame\_skip = 0;

}

else if (c == 27 || c == 1048603) // ESC - exit (OpenCV 2.x / 3.x)

{

flag\_exit = 1;

}

}

}else{

char buff[256];

sprintf(buff, "%s\_%08d.jpg", prefix, count);

if(show\_img) save\_cv\_jpg(show\_img, buff);

}

// if you run it with param -mjpeg\_port 8090 then open URL in your web-browser: http://localhost:8090

if (mjpeg\_port > 0 && show\_img) {

int port = mjpeg\_port;

int timeout = 400000;

int jpeg\_quality = 40; // 1 - 100

send\_mjpeg(show\_img, port, timeout, jpeg\_quality);

}

// save video file

if (output\_video\_writer && show\_img) {

write\_frame\_cv(output\_video\_writer, show\_img);

printf("\n cvWriteFrame \n");

}

while (custom\_atomic\_load\_int(&run\_detect\_in\_thread)) {

if(avg\_fps > 180) this\_thread\_yield();

else this\_thread\_sleep\_for(thread\_wait\_ms); // custom\_join(detect\_thread, 0);

}

if (!benchmark) {

while (custom\_atomic\_load\_int(&run\_fetch\_in\_thread)) {

if (avg\_fps > 180) this\_thread\_yield();

else this\_thread\_sleep\_for(thread\_wait\_ms); // custom\_join(fetch\_thread, 0);

}

free\_image(det\_s);

}

if (time\_limit\_sec > 0 && (get\_time\_point() - start\_time\_lim)/1000000 > time\_limit\_sec) {

printf(" start\_time\_lim = %f, get\_time\_point() = %f, time spent = %f \n", start\_time\_lim, get\_time\_point(), get\_time\_point() - start\_time\_lim);

break;

}

if (flag\_exit == 1) break;

if(delay == 0){

if(!benchmark) release\_mat(&show\_img);

show\_img = det\_img;

}

det\_img = in\_img;

det\_s = in\_s;

}

--delay;

if(delay < 0){

delay = frame\_skip;

//double after = get\_wall\_time();

//float curr = 1./(after - before);

double after = get\_time\_point(); // more accurate time measurements

float curr = 1000000. / (after - before);

fps = fps\*0.9 + curr\*0.1;

before = after;

float spent\_time = (get\_time\_point() - start\_time) / 1000000;

frame\_counter++;

if (spent\_time >= 3.0f) {

//printf(" spent\_time = %f \n", spent\_time);

avg\_fps = frame\_counter / spent\_time;

frame\_counter = 0;

start\_time = get\_time\_point();

}

}

}

printf("input video stream closed. \n");

if (output\_video\_writer) {

release\_video\_writer(&output\_video\_writer);

printf("output\_video\_writer closed. \n");

}

this\_thread\_sleep\_for(thread\_wait\_ms);

custom\_join(detect\_thread, 0);

custom\_join(fetch\_thread, 0);

// free memory

free\_image(in\_s);

free\_detections(dets, nboxes);

free(avg);

for (j = 0; j < NFRAMES; ++j) free(predictions[j]);

demo\_index = (NFRAMES + demo\_index - 1) % NFRAMES;

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

release\_mat(&cv\_images[j]);

}

free\_ptrs((void \*\*)names, net.layers[net.n - 1].classes);

int i;

const int nsize = 8;

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

for (i = 32; i < 127; ++i) {

free\_image(alphabet[j][i]);

}

free(alphabet[j]);

}

free(alphabet);

free\_network(net);

//cudaProfilerStop();

}

#else

void demo(char \*cfgfile, char \*weightfile, float thresh, float hier\_thresh, int cam\_index, const char \*filename, char \*\*names, int classes,

int frame\_skip, char \*prefix, char \*out\_filename, int mjpeg\_port, int json\_port, int dont\_show, int ext\_output, int letter\_box\_in, int time\_limit\_sec, char \*http\_post\_host,

int benchmark, int benchmark\_layers)

{

fprintf(stderr, "Demo needs OpenCV for webcam images.\n");

}

#endif