#include "image\_opencv.h"

#include <iostream>

#ifdef OPENCV

#include "utils.h"

#include <cstdio>

#include <cstdlib>

#include <cmath>

#include <string>

#include <vector>

#include <fstream>

#include <algorithm>

#include <atomic>

#include <opencv2/core/version.hpp>

#include <opencv2/imgproc/imgproc.hpp>

#include <opencv2/opencv.hpp>

#include <opencv2/opencv\_modules.hpp>

#include <opencv2/highgui/highgui.hpp>

#include <opencv2/video/video.hpp>

// includes for OpenCV >= 3.x

#ifndef CV\_VERSION\_EPOCH

#include <opencv2/core/types.hpp>

#include <opencv2/videoio/videoio.hpp>

#include <opencv2/imgcodecs/imgcodecs.hpp>

#endif

// OpenCV includes for OpenCV 2.x

#ifdef CV\_VERSION\_EPOCH

#include <opencv2/highgui/highgui\_c.h>

#include <opencv2/imgproc/imgproc\_c.h>

#include <opencv2/core/types\_c.h>

#include <opencv2/core/version.hpp>

#endif

//using namespace cv;

using std::cerr;

using std::endl;

#ifdef DEBUG

#define OCV\_D "d"

#else

#define OCV\_D

#endif//DEBUG

// OpenCV libraries

#ifndef CV\_VERSION\_EPOCH

#define OPENCV\_VERSION CVAUX\_STR(CV\_VERSION\_MAJOR)"" CVAUX\_STR(CV\_VERSION\_MINOR)"" CVAUX\_STR(CV\_VERSION\_REVISION) OCV\_D

#ifndef USE\_CMAKE\_LIBS

#pragma comment(lib, "opencv\_world" OPENCV\_VERSION ".lib")

#endif // USE\_CMAKE\_LIBS

#else // CV\_VERSION\_EPOCH

#define OPENCV\_VERSION CVAUX\_STR(CV\_VERSION\_EPOCH)"" CVAUX\_STR(CV\_VERSION\_MAJOR)"" CVAUX\_STR(CV\_VERSION\_MINOR) OCV\_D

#ifndef USE\_CMAKE\_LIBS

#pragma comment(lib, "opencv\_core" OPENCV\_VERSION ".lib")

#pragma comment(lib, "opencv\_imgproc" OPENCV\_VERSION ".lib")

#pragma comment(lib, "opencv\_highgui" OPENCV\_VERSION ".lib")

#endif // USE\_CMAKE\_LIBS

#endif // CV\_VERSION\_EPOCH

#include "http\_stream.h"

#ifndef CV\_RGB

#define CV\_RGB(r, g, b) cvScalar( (b), (g), (r), 0 )

#endif

#ifndef CV\_FILLED

#define CV\_FILLED cv::FILLED

#endif

#ifndef CV\_AA

#define CV\_AA cv::LINE\_AA

#endif

extern "C" {

//struct mat\_cv : cv::Mat { };

//struct cap\_cv : cv::VideoCapture { };

//struct write\_cv : cv::VideoWriter { };

//struct mat\_cv : cv::Mat { int a[0]; };

//struct cap\_cv : cv::VideoCapture { int a[0]; };

//struct write\_cv : cv::VideoWriter { int a[0]; };

// ====================================================================

// cv::Mat

// ====================================================================

image mat\_to\_image(cv::Mat mat);

cv::Mat image\_to\_mat(image img);

// image ipl\_to\_image(mat\_cv\* src);

// mat\_cv \*image\_to\_ipl(image img);

// cv::Mat ipl\_to\_mat(IplImage \*ipl);

// IplImage \*mat\_to\_ipl(cv::Mat mat);

extern "C" mat\_cv \*load\_image\_mat\_cv(const char \*filename, int flag)

{

cv::Mat \*mat\_ptr = NULL;

try {

cv::Mat mat = cv::imread(filename, flag);

if (mat.empty())

{

std::string shrinked\_filename = filename;

if (shrinked\_filename.length() > 1024) {

shrinked\_filename.resize(1024);

shrinked\_filename = std::string("name is too long: ") + shrinked\_filename;

}

cerr << "Cannot load image " << shrinked\_filename << std::endl;

std::ofstream bad\_list("bad.list", std::ios::out | std::ios::app);

bad\_list << shrinked\_filename << std::endl;

//if (check\_mistakes) getchar();

return NULL;

}

cv::Mat dst;

if (mat.channels() == 3) cv::cvtColor(mat, dst, cv::COLOR\_RGB2BGR);

else if (mat.channels() == 4) cv::cvtColor(mat, dst, cv::COLOR\_RGBA2BGRA);

else dst = mat;

mat\_ptr = new cv::Mat(dst);

return (mat\_cv \*)mat\_ptr;

}

catch (...) {

cerr << "OpenCV exception: load\_image\_mat\_cv \n";

}

if (mat\_ptr) delete mat\_ptr;

return NULL;

}

// ----------------------------------------

cv::Mat load\_image\_mat(char \*filename, int channels)

{

int flag = cv::IMREAD\_UNCHANGED;

if (channels == 0) flag = cv::IMREAD\_COLOR;

else if (channels == 1) flag = cv::IMREAD\_GRAYSCALE;

else if (channels == 3) flag = cv::IMREAD\_COLOR;

else {

fprintf(stderr, "OpenCV can't force load with %d channels\n", channels);

}

//flag |= IMREAD\_IGNORE\_ORIENTATION; // un-comment it if you want

cv::Mat \*mat\_ptr = (cv::Mat \*)load\_image\_mat\_cv(filename, flag);

if (mat\_ptr == NULL) {

return cv::Mat();

}

cv::Mat mat = \*mat\_ptr;

delete mat\_ptr;

return mat;

}

// ----------------------------------------

extern "C" image load\_image\_cv(char \*filename, int channels)

{

cv::Mat mat = load\_image\_mat(filename, channels);

if (mat.empty()) {

return make\_image(10, 10, channels);

}

return mat\_to\_image(mat);

}

// ----------------------------------------

extern "C" image load\_image\_resize(char \*filename, int w, int h, int c, image \*im)

{

image out;

try {

cv::Mat loaded\_image = load\_image\_mat(filename, c);

\*im = mat\_to\_image(loaded\_image);

cv::Mat resized(h, w, CV\_8UC3);

cv::resize(loaded\_image, resized, cv::Size(w, h), 0, 0, cv::INTER\_LINEAR);

out = mat\_to\_image(resized);

}

catch (...) {

cerr << " OpenCV exception: load\_image\_resize() can't load image %s " << filename << " \n";

out = make\_image(w, h, c);

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

}

return out;

}

// ----------------------------------------

extern "C" int get\_width\_mat(mat\_cv \*mat)

{

if (mat == NULL) {

cerr << " Pointer is NULL in get\_width\_mat() \n";

return 0;

}

return ((cv::Mat \*)mat)->cols;

}

// ----------------------------------------

extern "C" int get\_height\_mat(mat\_cv \*mat)

{

if (mat == NULL) {

cerr << " Pointer is NULL in get\_height\_mat() \n";

return 0;

}

return ((cv::Mat \*)mat)->rows;

}

// ----------------------------------------

extern "C" void release\_mat(mat\_cv \*\*mat)

{

try {

cv::Mat \*\*mat\_ptr = (cv::Mat \*\*)mat;

if (\*mat\_ptr) delete \*mat\_ptr;

\*mat\_ptr = NULL;

}

catch (...) {

cerr << "OpenCV exception: release\_mat \n";

}

}

// ====================================================================

// IplImage

// ====================================================================

/\*

extern "C" int get\_width\_cv(mat\_cv \*ipl\_src)

{

IplImage \*ipl = (IplImage \*)ipl\_src;

return ipl->width;

}

// ----------------------------------------

extern "C" int get\_height\_cv(mat\_cv \*ipl\_src)

{

IplImage \*ipl = (IplImage \*)ipl\_src;

return ipl->height;

}

// ----------------------------------------

extern "C" void release\_ipl(mat\_cv \*\*ipl)

{

IplImage \*\*ipl\_img = (IplImage \*\*)ipl;

if (\*ipl\_img) cvReleaseImage(ipl\_img);

\*ipl\_img = NULL;

}

// ----------------------------------------

// ====================================================================

// image-to-ipl, ipl-to-image, image\_to\_mat, mat\_to\_image

// ====================================================================

extern "C" mat\_cv \*image\_to\_ipl(image im)

{

int x, y, c;

IplImage \*disp = cvCreateImage(cvSize(im.w, im.h), IPL\_DEPTH\_8U, im.c);

int step = disp->widthStep;

for (y = 0; y < im.h; ++y) {

for (x = 0; x < im.w; ++x) {

for (c = 0; c < im.c; ++c) {

float val = im.data[c\*im.h\*im.w + y\*im.w + x];

disp->imageData[y\*step + x\*im.c + c] = (unsigned char)(val \* 255);

}

}

}

return (mat\_cv \*)disp;

}

// ----------------------------------------

extern "C" image ipl\_to\_image(mat\_cv\* src\_ptr)

{

IplImage\* src = (IplImage\*)src\_ptr;

int h = src->height;

int w = src->width;

int c = src->nChannels;

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

unsigned char \*data = (unsigned char \*)src->imageData;

int step = src->widthStep;

int i, j, k;

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

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

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

im.data[k\*w\*h + i\*w + j] = data[i\*step + j\*c + k] / 255.;

}

}

}

return im;

}

// ----------------------------------------

cv::Mat ipl\_to\_mat(IplImage \*ipl)

{

Mat m = cvarrToMat(ipl, true);

return m;

}

// ----------------------------------------

IplImage \*mat\_to\_ipl(cv::Mat mat)

{

IplImage \*ipl = new IplImage;

\*ipl = mat;

return ipl;

}

// ----------------------------------------

\*/

extern "C" cv::Mat image\_to\_mat(image img)

{

int channels = img.c;

int width = img.w;

int height = img.h;

cv::Mat mat = cv::Mat(height, width, CV\_8UC(channels));

int step = mat.step;

for (int y = 0; y < img.h; ++y) {

for (int x = 0; x < img.w; ++x) {

for (int c = 0; c < img.c; ++c) {

float val = img.data[c\*img.h\*img.w + y\*img.w + x];

mat.data[y\*step + x\*img.c + c] = (unsigned char)(val \* 255);

}

}

}

return mat;

}

// ----------------------------------------

extern "C" image mat\_to\_image(cv::Mat mat)

{

int w = mat.cols;

int h = mat.rows;

int c = mat.channels();

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

unsigned char \*data = (unsigned char \*)mat.data;

int step = mat.step;

for (int y = 0; y < h; ++y) {

for (int k = 0; k < c; ++k) {

for (int x = 0; x < w; ++x) {

//uint8\_t val = mat.ptr<uint8\_t>(y)[c \* x + k];

//uint8\_t val = mat.at<Vec3b>(y, x).val[k];

//im.data[k\*w\*h + y\*w + x] = val / 255.0f;

im.data[k\*w\*h + y\*w + x] = data[y\*step + x\*c + k] / 255.0f;

}

}

}

return im;

}

image mat\_to\_image\_cv(mat\_cv \*mat)

{

return mat\_to\_image(\*(cv::Mat\*)mat);

}

// ====================================================================

// Window

// ====================================================================

extern "C" void create\_window\_cv(char const\* window\_name, int full\_screen, int width, int height)

{

try {

int window\_type = cv::WINDOW\_NORMAL;

#ifdef CV\_VERSION\_EPOCH // OpenCV 2.x

if (full\_screen) window\_type = CV\_WINDOW\_FULLSCREEN;

#else

if (full\_screen) window\_type = cv::WINDOW\_FULLSCREEN;

#endif

cv::namedWindow(window\_name, window\_type);

cv::moveWindow(window\_name, 0, 0);

cv::resizeWindow(window\_name, width, height);

}

catch (...) {

cerr << "OpenCV exception: create\_window\_cv \n";

}

}

// ----------------------------------------

extern "C" void destroy\_all\_windows\_cv()

{

try {

cv::destroyAllWindows();

}

catch (...) {

cerr << "OpenCV exception: destroy\_all\_windows\_cv \n";

}

}

// ----------------------------------------

extern "C" int wait\_key\_cv(int delay)

{

try {

return cv::waitKey(delay);

}

catch (...) {

cerr << "OpenCV exception: wait\_key\_cv \n";

}

return -1;

}

// ----------------------------------------

extern "C" int wait\_until\_press\_key\_cv()

{

return wait\_key\_cv(0);

}

// ----------------------------------------

extern "C" void make\_window(char \*name, int w, int h, int fullscreen)

{

try {

cv::namedWindow(name, cv::WINDOW\_NORMAL);

if (fullscreen) {

#ifdef CV\_VERSION\_EPOCH // OpenCV 2.x

cv::setWindowProperty(name, cv::WND\_PROP\_FULLSCREEN, CV\_WINDOW\_FULLSCREEN);

#else

cv::setWindowProperty(name, cv::WND\_PROP\_FULLSCREEN, cv::WINDOW\_FULLSCREEN);

#endif

}

else {

cv::resizeWindow(name, w, h);

if (strcmp(name, "Demo") == 0) cv::moveWindow(name, 0, 0);

}

}

catch (...) {

cerr << "OpenCV exception: make\_window \n";

}

}

// ----------------------------------------

static float get\_pixel(image m, int x, int y, int c)

{

assert(x < m.w && y < m.h && c < m.c);

return m.data[c\*m.h\*m.w + y\*m.w + x];

}

// ----------------------------------------

extern "C" void show\_image\_cv(image p, const char \*name)

{

try {

image copy = copy\_image(p);

constrain\_image(copy);

cv::Mat mat = image\_to\_mat(copy);

if (mat.channels() == 3) cv::cvtColor(mat, mat, cv::COLOR\_RGB2BGR);

else if (mat.channels() == 4) cv::cvtColor(mat, mat, cv::COLOR\_RGBA2BGR);

cv::namedWindow(name, cv::WINDOW\_NORMAL);

cv::imshow(name, mat);

free\_image(copy);

}

catch (...) {

cerr << "OpenCV exception: show\_image\_cv \n";

}

}

// ----------------------------------------

/\*

extern "C" void show\_image\_cv\_ipl(mat\_cv \*disp, const char \*name)

{

if (disp == NULL) return;

char buff[256];

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

cv::namedWindow(buff, WINDOW\_NORMAL);

cvShowImage(buff, disp);

}

// ----------------------------------------

\*/

extern "C" void show\_image\_mat(mat\_cv \*mat\_ptr, const char \*name)

{

try {

if (mat\_ptr == NULL) return;

cv::Mat &mat = \*(cv::Mat \*)mat\_ptr;

cv::namedWindow(name, cv::WINDOW\_NORMAL);

cv::imshow(name, mat);

}

catch (...) {

cerr << "OpenCV exception: show\_image\_mat \n";

}

}

// ====================================================================

// Video Writer

// ====================================================================

extern "C" write\_cv \*create\_video\_writer(char \*out\_filename, char c1, char c2, char c3, char c4, int fps, int width, int height, int is\_color)

{

try {

cv::VideoWriter \* output\_video\_writer =

#ifdef CV\_VERSION\_EPOCH

new cv::VideoWriter(out\_filename, CV\_FOURCC(c1, c2, c3, c4), fps, cv::Size(width, height), is\_color);

#else

new cv::VideoWriter(out\_filename, cv::VideoWriter::fourcc(c1, c2, c3, c4), fps, cv::Size(width, height), is\_color);

#endif

return (write\_cv \*)output\_video\_writer;

}

catch (...) {

cerr << "OpenCV exception: create\_video\_writer \n";

}

return NULL;

}

extern "C" void write\_frame\_cv(write\_cv \*output\_video\_writer, mat\_cv \*mat)

{

try {

cv::VideoWriter \*out = (cv::VideoWriter \*)output\_video\_writer;

out->write(\*(cv::Mat\*)mat);

}

catch (...) {

cerr << "OpenCV exception: write\_frame\_cv \n";

}

}

extern "C" void release\_video\_writer(write\_cv \*\*output\_video\_writer)

{

try {

if (output\_video\_writer) {

std::cout << " closing...";

cv::VideoWriter \*out = \*(cv::VideoWriter \*\*)output\_video\_writer;

out->release();

delete out;

output\_video\_writer = NULL;

std::cout << " closed!";

}

else {

cerr << "OpenCV exception: output\_video\_writer isn't created \n";

}

}

catch (...) {

cerr << "OpenCV exception: release\_video\_writer \n";

}

}

/\*

extern "C" void \*open\_video\_stream(const char \*f, int c, int w, int h, int fps)

{

VideoCapture \*cap;

if(f) cap = new VideoCapture(f);

else cap = new VideoCapture(c);

if(!cap->isOpened()) return 0;

if(w) cap->set(CV\_CAP\_PROP\_FRAME\_WIDTH, w);

if(h) cap->set(CV\_CAP\_PROP\_FRAME\_HEIGHT, w);

if(fps) cap->set(CV\_CAP\_PROP\_FPS, w);

return (void \*) cap;

}

extern "C" image get\_image\_from\_stream(void \*p)

{

VideoCapture \*cap = (VideoCapture \*)p;

Mat m;

\*cap >> m;

if(m.empty()) return make\_empty\_image(0,0,0);

return mat\_to\_image(m);

}

extern "C" int show\_image\_cv(image im, const char\* name, int ms)

{

Mat m = image\_to\_mat(im);

imshow(name, m);

int c = waitKey(ms);

if (c != -1) c = c%256;

return c;

}

\*/

// ====================================================================

// Video Capture

// ====================================================================

extern "C" cap\_cv\* get\_capture\_video\_stream(const char \*path) {

cv::VideoCapture\* cap = NULL;

try {

cap = new cv::VideoCapture(path);

}

catch (...) {

cerr << " OpenCV exception: video-stream " << path << " can't be opened! \n";

}

return (cap\_cv\*)cap;

}

// ----------------------------------------

extern "C" cap\_cv\* get\_capture\_webcam(int index)

{

cv::VideoCapture\* cap = NULL;

try {

cap = new cv::VideoCapture(index);

//cap->set(CV\_CAP\_PROP\_FRAME\_WIDTH, 1280);

//cap->set(CV\_CAP\_PROP\_FRAME\_HEIGHT, 960);

}

catch (...) {

cerr << " OpenCV exception: Web-camera " << index << " can't be opened! \n";

}

return (cap\_cv\*)cap;

}

// ----------------------------------------

extern "C" void release\_capture(cap\_cv\* cap)

{

try {

cv::VideoCapture \*cpp\_cap = (cv::VideoCapture \*)cap;

delete cpp\_cap;

}

catch (...) {

cerr << " OpenCV exception: cv::VideoCapture " << cap << " can't be released! \n";

}

}

// ----------------------------------------

extern "C" mat\_cv\* get\_capture\_frame\_cv(cap\_cv \*cap) {

cv::Mat \*mat = NULL;

try {

mat = new cv::Mat();

if (cap) {

cv::VideoCapture &cpp\_cap = \*(cv::VideoCapture \*)cap;

if (cpp\_cap.isOpened())

{

cpp\_cap >> \*mat;

}

else std::cout << " Video-stream stopped! \n";

}

else cerr << " cv::VideoCapture isn't created \n";

}

catch (...) {

std::cout << " OpenCV exception: Video-stream stoped! \n";

}

return (mat\_cv \*)mat;

}

// ----------------------------------------

extern "C" int get\_stream\_fps\_cpp\_cv(cap\_cv \*cap)

{

int fps = 25;

try {

cv::VideoCapture &cpp\_cap = \*(cv::VideoCapture \*)cap;

#ifndef CV\_VERSION\_EPOCH // OpenCV 3.x

fps = cpp\_cap.get(cv::CAP\_PROP\_FPS);

#else // OpenCV 2.x

fps = cpp\_cap.get(CV\_CAP\_PROP\_FPS);

#endif

}

catch (...) {

cerr << " Can't get FPS of source videofile. For output video FPS = 25 by default. \n";

}

return fps;

}

// ----------------------------------------

extern "C" double get\_capture\_property\_cv(cap\_cv \*cap, int property\_id)

{

try {

cv::VideoCapture &cpp\_cap = \*(cv::VideoCapture \*)cap;

return cpp\_cap.get(property\_id);

}

catch (...) {

cerr << " OpenCV exception: Can't get property of source video-stream. \n";

}

return 0;

}

// ----------------------------------------

extern "C" double get\_capture\_frame\_count\_cv(cap\_cv \*cap)

{

try {

cv::VideoCapture &cpp\_cap = \*(cv::VideoCapture \*)cap;

#ifndef CV\_VERSION\_EPOCH // OpenCV 3.x

return cpp\_cap.get(cv::CAP\_PROP\_FRAME\_COUNT);

#else // OpenCV 2.x

return cpp\_cap.get(CV\_CAP\_PROP\_FRAME\_COUNT);

#endif

}

catch (...) {

cerr << " OpenCV exception: Can't get CAP\_PROP\_FRAME\_COUNT of source videofile. \n";

}

return 0;

}

// ----------------------------------------

extern "C" int set\_capture\_property\_cv(cap\_cv \*cap, int property\_id, double value)

{

try {

cv::VideoCapture &cpp\_cap = \*(cv::VideoCapture \*)cap;

return cpp\_cap.set(property\_id, value);

}

catch (...) {

cerr << " Can't set property of source video-stream. \n";

}

return false;

}

// ----------------------------------------

extern "C" int set\_capture\_position\_frame\_cv(cap\_cv \*cap, int index)

{

try {

cv::VideoCapture &cpp\_cap = \*(cv::VideoCapture \*)cap;

#ifndef CV\_VERSION\_EPOCH // OpenCV 3.x

return cpp\_cap.set(cv::CAP\_PROP\_POS\_FRAMES, index);

#else // OpenCV 2.x

return cpp\_cap.set(CV\_CAP\_PROP\_POS\_FRAMES, index);

#endif

}

catch (...) {

cerr << " Can't set CAP\_PROP\_POS\_FRAMES of source videofile. \n";

}

return false;

}

// ----------------------------------------

// ====================================================================

// ... Video Capture

// ====================================================================

extern "C" image get\_image\_from\_stream\_cpp(cap\_cv \*cap)

{

cv::Mat \*src = NULL;

static int once = 1;

if (once) {

once = 0;

do {

if (src) delete src;

src = (cv::Mat\*)get\_capture\_frame\_cv(cap);

if (!src) return make\_empty\_image(0, 0, 0);

} while (src->cols < 1 || src->rows < 1 || src->channels() < 1);

printf("Video stream: %d x %d \n", src->cols, src->rows);

}

else

src = (cv::Mat\*)get\_capture\_frame\_cv(cap);

if (!src) return make\_empty\_image(0, 0, 0);

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

rgbgr\_image(im);

if (src) delete src;

return im;

}

// ----------------------------------------

extern "C" int wait\_for\_stream(cap\_cv \*cap, cv::Mat\* src, int dont\_close)

{

if (!src) {

if (dont\_close) src = new cv::Mat(416, 416, CV\_8UC(3)); // cvCreateImage(cvSize(416, 416), IPL\_DEPTH\_8U, 3);

else return 0;

}

if (src->cols < 1 || src->rows < 1 || src->channels() < 1) {

if (dont\_close) {

delete src;// cvReleaseImage(&src);

int z = 0;

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

src = (cv::Mat\*)get\_capture\_frame\_cv(cap);

delete src;// cvReleaseImage(&src);

}

src = new cv::Mat(416, 416, CV\_8UC(3)); // cvCreateImage(cvSize(416, 416), IPL\_DEPTH\_8U, 3);

}

else return 0;

}

return 1;

}

// ----------------------------------------

extern "C" image get\_image\_from\_stream\_resize(cap\_cv \*cap, int w, int h, int c, mat\_cv\*\* in\_img, int dont\_close)

{

c = c ? c : 3;

cv::Mat \*src = NULL;

static int once = 1;

if (once) {

once = 0;

do {

if (src) delete src;

src = (cv::Mat\*)get\_capture\_frame\_cv(cap);

if (!src) return make\_empty\_image(0, 0, 0);

} while (src->cols < 1 || src->rows < 1 || src->channels() < 1);

printf("Video stream: %d x %d \n", src->cols, src->rows);

}

else

src = (cv::Mat\*)get\_capture\_frame\_cv(cap);

if (!wait\_for\_stream(cap, src, dont\_close)) return make\_empty\_image(0, 0, 0);

\*(cv::Mat \*\*)in\_img = src;

cv::Mat new\_img = cv::Mat(h, w, CV\_8UC(c));

cv::resize(\*src, new\_img, new\_img.size(), 0, 0, cv::INTER\_LINEAR);

if (c>1) cv::cvtColor(new\_img, new\_img, cv::COLOR\_RGB2BGR);

image im = mat\_to\_image(new\_img);

//show\_image\_cv(im, "im");

//show\_image\_mat(\*in\_img, "in\_img");

return im;

}

// ----------------------------------------

extern "C" image get\_image\_from\_stream\_letterbox(cap\_cv \*cap, int w, int h, int c, mat\_cv\*\* in\_img, int dont\_close)

{

c = c ? c : 3;

cv::Mat \*src = NULL;

static int once = 1;

if (once) {

once = 0;

do {

if (src) delete src;

src = (cv::Mat\*)get\_capture\_frame\_cv(cap);

if (!src) return make\_empty\_image(0, 0, 0);

} while (src->cols < 1 || src->rows < 1 || src->channels() < 1);

printf("Video stream: %d x %d \n", src->cols, src->rows);

}

else

src = (cv::Mat\*)get\_capture\_frame\_cv(cap);

if (!wait\_for\_stream(cap, src, dont\_close)) return make\_empty\_image(0, 0, 0); // passes (cv::Mat \*)src while should be (cv::Mat \*\*)src

\*in\_img = (mat\_cv \*)new cv::Mat(src->rows, src->cols, CV\_8UC(c));

cv::resize(\*src, \*\*(cv::Mat\*\*)in\_img, (\*(cv::Mat\*\*)in\_img)->size(), 0, 0, cv::INTER\_LINEAR);

if (c>1) cv::cvtColor(\*src, \*src, cv::COLOR\_RGB2BGR);

image tmp = mat\_to\_image(\*src);

image im = letterbox\_image(tmp, w, h);

free\_image(tmp);

release\_mat((mat\_cv \*\*)&src);

//show\_image\_cv(im, "im");

//show\_image\_mat(\*in\_img, "in\_img");

return im;

}

// ----------------------------------------

// ====================================================================

// Image Saving

// ====================================================================

extern int stbi\_write\_png(char const \*filename, int w, int h, int comp, const void \*data, int stride\_in\_bytes);

extern int stbi\_write\_jpg(char const \*filename, int x, int y, int comp, const void \*data, int quality);

extern "C" void save\_mat\_png(cv::Mat img\_src, const char \*name)

{

cv::Mat img\_rgb;

if (img\_src.channels() >= 3) cv::cvtColor(img\_src, img\_rgb, cv::COLOR\_RGB2BGR);

stbi\_write\_png(name, img\_rgb.cols, img\_rgb.rows, 3, (char \*)img\_rgb.data, 0);

}

// ----------------------------------------

extern "C" void save\_mat\_jpg(cv::Mat img\_src, const char \*name)

{

cv::Mat img\_rgb;

if (img\_src.channels() >= 3) cv::cvtColor(img\_src, img\_rgb, cv::COLOR\_RGB2BGR);

stbi\_write\_jpg(name, img\_rgb.cols, img\_rgb.rows, 3, (char \*)img\_rgb.data, 80);

}

// ----------------------------------------

extern "C" void save\_cv\_png(mat\_cv \*img\_src, const char \*name)

{

cv::Mat\* img = (cv::Mat\* )img\_src;

save\_mat\_png(\*img, name);

}

// ----------------------------------------

extern "C" void save\_cv\_jpg(mat\_cv \*img\_src, const char \*name)

{

cv::Mat\* img = (cv::Mat\*)img\_src;

save\_mat\_jpg(\*img, name);

}

// ----------------------------------------

// ====================================================================

// Draw Detection

// ====================================================================

extern "C" void draw\_detections\_cv\_v3(mat\_cv\* mat, detection \*dets, int num, float thresh, char \*\*names, image \*\*alphabet, int classes, int ext\_output)

{

try {

cv::Mat \*show\_img = (cv::Mat\*)mat;

int i, j;

if (!show\_img) return;

static int frame\_id = 0;

frame\_id++;

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

char labelstr[4096] = { 0 };

int class\_id = -1;

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

int show = strncmp(names[j], "dont\_show", 9);

if (dets[i].prob[j] > thresh && show) {

if (class\_id < 0) {

strcat(labelstr, names[j]);

class\_id = j;

char buff[10];

sprintf(buff, " (%2.0f%%)", dets[i].prob[j] \* 100);

strcat(labelstr, buff);

printf("%s: %.0f%% ", names[j], dets[i].prob[j] \* 100);

}

else {

strcat(labelstr, ", ");

strcat(labelstr, names[j]);

printf(", %s: %.0f%% ", names[j], dets[i].prob[j] \* 100);

}

}

}

if (class\_id >= 0) {

int width = std::max(1.0f, show\_img->rows \* .002f);

//if(0){

//width = pow(prob, 1./2.)\*10+1;

//alphabet = 0;

//}

//printf("%d %s: %.0f%%\n", i, names[class\_id], prob\*100);

int offset = class\_id \* 123457 % classes;

float red = get\_color(2, offset, classes);

float green = get\_color(1, offset, classes);

float blue = get\_color(0, offset, classes);

float rgb[3];

//width = prob\*20+2;

rgb[0] = red;

rgb[1] = green;

rgb[2] = blue;

box b = dets[i].bbox;

if (std::isnan(b.w) || std::isinf(b.w)) b.w = 0.5;

if (std::isnan(b.h) || std::isinf(b.h)) b.h = 0.5;

if (std::isnan(b.x) || std::isinf(b.x)) b.x = 0.5;

if (std::isnan(b.y) || std::isinf(b.y)) b.y = 0.5;

b.w = (b.w < 1) ? b.w : 1;

b.h = (b.h < 1) ? b.h : 1;

b.x = (b.x < 1) ? b.x : 1;

b.y = (b.y < 1) ? b.y : 1;

//printf("%f %f %f %f\n", b.x, b.y, b.w, b.h);

int left = (b.x - b.w / 2.)\*show\_img->cols;

int right = (b.x + b.w / 2.)\*show\_img->cols;

int top = (b.y - b.h / 2.)\*show\_img->rows;

int bot = (b.y + b.h / 2.)\*show\_img->rows;

if (left < 0) left = 0;

if (right > show\_img->cols - 1) right = show\_img->cols - 1;

if (top < 0) top = 0;

if (bot > show\_img->rows - 1) bot = show\_img->rows - 1;

//int b\_x\_center = (left + right) / 2;

//int b\_y\_center = (top + bot) / 2;

//int b\_width = right - left;

//int b\_height = bot - top;

//sprintf(labelstr, "%d x %d - w: %d, h: %d", b\_x\_center, b\_y\_center, b\_width, b\_height);

float const font\_size = show\_img->rows / 1000.F;

cv::Size const text\_size = cv::getTextSize(labelstr, cv::FONT\_HERSHEY\_COMPLEX\_SMALL, font\_size, 1, 0);

cv::Point pt1, pt2, pt\_text, pt\_text\_bg1, pt\_text\_bg2;

pt1.x = left;

pt1.y = top;

pt2.x = right;

pt2.y = bot;

pt\_text.x = left;

pt\_text.y = top - 4;// 12;

pt\_text\_bg1.x = left;

pt\_text\_bg1.y = top - (3 + 18 \* font\_size);

pt\_text\_bg2.x = right;

if ((right - left) < text\_size.width) pt\_text\_bg2.x = left + text\_size.width;

pt\_text\_bg2.y = top;

cv::Scalar color;

color.val[0] = red \* 256;

color.val[1] = green \* 256;

color.val[2] = blue \* 256;

// you should create directory: result\_img

//static int copied\_frame\_id = -1;

//static IplImage\* copy\_img = NULL;

//if (copied\_frame\_id != frame\_id) {

// copied\_frame\_id = frame\_id;

// if(copy\_img == NULL) copy\_img = cvCreateImage(cvSize(show\_img->width, show\_img->height), show\_img->depth, show\_img->nChannels);

// cvCopy(show\_img, copy\_img, 0);

//}

//static int img\_id = 0;

//img\_id++;

//char image\_name[1024];

//sprintf(image\_name, "result\_img/img\_%d\_%d\_%d\_%s.jpg", frame\_id, img\_id, class\_id, names[class\_id]);

//CvRect rect = cvRect(pt1.x, pt1.y, pt2.x - pt1.x, pt2.y - pt1.y);

//cvSetImageROI(copy\_img, rect);

//cvSaveImage(image\_name, copy\_img, 0);

//cvResetImageROI(copy\_img);

cv::rectangle(\*show\_img, pt1, pt2, color, width, 8, 0);

if (ext\_output)

printf("\t(left\_x: %4.0f top\_y: %4.0f width: %4.0f height: %4.0f)\n",

(float)left, (float)top, b.w\*show\_img->cols, b.h\*show\_img->rows);

else

printf("\n");

cv::rectangle(\*show\_img, pt\_text\_bg1, pt\_text\_bg2, color, width, 8, 0);

cv::rectangle(\*show\_img, pt\_text\_bg1, pt\_text\_bg2, color, CV\_FILLED, 8, 0); // filled

cv::Scalar black\_color = CV\_RGB(0, 0, 0);

cv::putText(\*show\_img, labelstr, pt\_text, cv::FONT\_HERSHEY\_COMPLEX\_SMALL, font\_size, black\_color, 2 \* font\_size, CV\_AA);

// cv::FONT\_HERSHEY\_COMPLEX\_SMALL, cv::FONT\_HERSHEY\_SIMPLEX

}

}

if (ext\_output) {

fflush(stdout);

}

}

catch (...) {

cerr << "OpenCV exception: draw\_detections\_cv\_v3() \n";

}

}

// ----------------------------------------

// ====================================================================

// Draw Loss & Accuracy chart

// ====================================================================

extern "C" mat\_cv\* draw\_train\_chart(char \*windows\_name, float max\_img\_loss, int max\_batches, int number\_of\_lines, int img\_size, int dont\_show, char\* chart\_path)

{

int img\_offset = 60;

int draw\_size = img\_size - img\_offset;

cv::Mat \*img\_ptr = new cv::Mat(img\_size, img\_size, CV\_8UC3, CV\_RGB(255, 255, 255));

cv::Mat &img = \*img\_ptr;

cv::Point pt1, pt2, pt\_text;

try {

// load chart from file

if (chart\_path != NULL && chart\_path[0] != '\0') {

\*img\_ptr = cv::imread(chart\_path);

}

else {

// draw new chart

char char\_buff[100];

int i;

// vertical lines

pt1.x = img\_offset; pt2.x = img\_size, pt\_text.x = 30;

for (i = 1; i <= number\_of\_lines; ++i) {

pt1.y = pt2.y = (float)i \* draw\_size / number\_of\_lines;

cv::line(img, pt1, pt2, CV\_RGB(224, 224, 224), 1, 8, 0);

if (i % 10 == 0) {

sprintf(char\_buff, "%2.1f", max\_img\_loss\*(number\_of\_lines - i) / number\_of\_lines);

pt\_text.y = pt1.y + 3;

cv::putText(img, char\_buff, pt\_text, cv::FONT\_HERSHEY\_COMPLEX\_SMALL, 0.7, CV\_RGB(0, 0, 0), 1, CV\_AA);

cv::line(img, pt1, pt2, CV\_RGB(128, 128, 128), 1, 8, 0);

}

}

// horizontal lines

pt1.y = draw\_size; pt2.y = 0, pt\_text.y = draw\_size + 15;

for (i = 0; i <= number\_of\_lines; ++i) {

pt1.x = pt2.x = img\_offset + (float)i \* draw\_size / number\_of\_lines;

cv::line(img, pt1, pt2, CV\_RGB(224, 224, 224), 1, 8, 0);

if (i % 10 == 0) {

sprintf(char\_buff, "%d", max\_batches \* i / number\_of\_lines);

pt\_text.x = pt1.x - 20;

cv::putText(img, char\_buff, pt\_text, cv::FONT\_HERSHEY\_COMPLEX\_SMALL, 0.7, CV\_RGB(0, 0, 0), 1, CV\_AA);

cv::line(img, pt1, pt2, CV\_RGB(128, 128, 128), 1, 8, 0);

}

}

cv::putText(img, "Loss", cv::Point(10, 55), cv::FONT\_HERSHEY\_COMPLEX\_SMALL, 0.7, CV\_RGB(0, 0, 255), 1, CV\_AA);

cv::putText(img, "Iteration number", cv::Point(draw\_size / 2, img\_size - 10), cv::FONT\_HERSHEY\_COMPLEX\_SMALL, 0.7, CV\_RGB(0, 0, 0), 1, CV\_AA);

char max\_batches\_buff[100];

sprintf(max\_batches\_buff, "in cfg max\_batches=%d", max\_batches);

cv::putText(img, max\_batches\_buff, cv::Point(draw\_size - 195, img\_size - 10), cv::FONT\_HERSHEY\_COMPLEX\_SMALL, 0.7, CV\_RGB(0, 0, 0), 1, CV\_AA);

cv::putText(img, "Press 's' to save : chart.png", cv::Point(5, img\_size - 10), cv::FONT\_HERSHEY\_COMPLEX\_SMALL, 0.7, CV\_RGB(0, 0, 0), 1, CV\_AA);

}

if (!dont\_show) {

printf(" If error occurs - run training with flag: -dont\_show \n");

cv::namedWindow(windows\_name, cv::WINDOW\_NORMAL);

cv::moveWindow(windows\_name, 0, 0);

cv::resizeWindow(windows\_name, img\_size, img\_size);

cv::imshow(windows\_name, img);

cv::waitKey(20);

}

}

catch (...) {

cerr << "OpenCV exception: draw\_train\_chart() \n";

}

return (mat\_cv\*)img\_ptr;

}

// ----------------------------------------

extern "C" void draw\_train\_loss(char \*windows\_name, mat\_cv\* img\_src, int img\_size, float avg\_loss, float max\_img\_loss, int current\_batch, int max\_batches,

float precision, int draw\_precision, char \*accuracy\_name, int dont\_show, int mjpeg\_port, double time\_remaining)

{

try {

cv::Mat &img = \*(cv::Mat\*)img\_src;

int img\_offset = 60;

int draw\_size = img\_size - img\_offset;

char char\_buff[100];

cv::Point pt1, pt2;

pt1.x = img\_offset + draw\_size \* (float)current\_batch / max\_batches;

pt1.y = draw\_size \* (1 - avg\_loss / max\_img\_loss);

if (pt1.y < 0) pt1.y = 1;

cv::circle(img, pt1, 1, CV\_RGB(0, 0, 255), CV\_FILLED, 8, 0);

// precision

if (draw\_precision) {

static float old\_precision = 0;

static float max\_precision = 0;

static int iteration\_old = 0;

static int text\_iteration\_old = 0;

if (iteration\_old == 0)

cv::putText(img, accuracy\_name, cv::Point(10, 12), cv::FONT\_HERSHEY\_COMPLEX\_SMALL, 0.7, CV\_RGB(255, 0, 0), 1, CV\_AA);

if (iteration\_old != 0){

cv::line(img,

cv::Point(img\_offset + draw\_size \* (float)iteration\_old / max\_batches, draw\_size \* (1 - old\_precision)),

cv::Point(img\_offset + draw\_size \* (float)current\_batch / max\_batches, draw\_size \* (1 - precision)),

CV\_RGB(255, 0, 0), 1, 8, 0);

}

sprintf(char\_buff, "%2.1f%% ", precision \* 100);

cv::putText(img, char\_buff, cv::Point(10, 28), cv::FONT\_HERSHEY\_COMPLEX\_SMALL, 0.7, CV\_RGB(255, 255, 255), 5, CV\_AA);

cv::putText(img, char\_buff, cv::Point(10, 28), cv::FONT\_HERSHEY\_COMPLEX\_SMALL, 0.7, CV\_RGB(200, 0, 0), 1, CV\_AA);

if ((std::fabs(old\_precision - precision) > 0.1) || (max\_precision < precision) || (current\_batch - text\_iteration\_old) >= max\_batches / 10) {

text\_iteration\_old = current\_batch;

max\_precision = std::max(max\_precision, precision);

sprintf(char\_buff, "%2.0f%% ", precision \* 100);

cv::putText(img, char\_buff, cv::Point(pt1.x - 30, draw\_size \* (1 - precision) + 15), cv::FONT\_HERSHEY\_COMPLEX\_SMALL, 0.7, CV\_RGB(255, 255, 255), 5, CV\_AA);

cv::putText(img, char\_buff, cv::Point(pt1.x - 30, draw\_size \* (1 - precision) + 15), cv::FONT\_HERSHEY\_COMPLEX\_SMALL, 0.7, CV\_RGB(200, 0, 0), 1, CV\_AA);

}

old\_precision = precision;

iteration\_old = current\_batch;

}

sprintf(char\_buff, "current avg loss = %2.4f iteration = %d approx. time left = %2.2f hours", avg\_loss, current\_batch, time\_remaining);

pt1.x = 15, pt1.y = draw\_size + 18;

pt2.x = pt1.x + 800, pt2.y = pt1.y + 20;

cv::rectangle(img, pt1, pt2, CV\_RGB(255, 255, 255), CV\_FILLED, 8, 0);

pt1.y += 15;

cv::putText(img, char\_buff, pt1, cv::FONT\_HERSHEY\_COMPLEX\_SMALL, 0.7, CV\_RGB(0, 0, 100), 1, CV\_AA);

int k = 0;

if (!dont\_show) {

cv::imshow(windows\_name, img);

k = cv::waitKey(20);

}

static int old\_batch = 0;

if (k == 's' || current\_batch == (max\_batches - 1) || (current\_batch / 100 > old\_batch / 100)) {

old\_batch = current\_batch;

save\_mat\_png(img, "chart.png");

save\_mat\_png(img, windows\_name);

cv::putText(img, "- Saved", cv::Point(260, img\_size - 10), cv::FONT\_HERSHEY\_COMPLEX\_SMALL, 0.7, CV\_RGB(255, 0, 0), 1, CV\_AA);

}

else

cv::putText(img, "- Saved", cv::Point(260, img\_size - 10), cv::FONT\_HERSHEY\_COMPLEX\_SMALL, 0.7, CV\_RGB(255, 255, 255), 1, CV\_AA);

if (mjpeg\_port > 0) send\_mjpeg((mat\_cv \*)&img, mjpeg\_port, 500000, 70);

}

catch (...) {

cerr << "OpenCV exception: draw\_train\_loss() \n";

}

}

// ----------------------------------------

// ====================================================================

// Data augmentation

// ====================================================================

extern "C" image image\_data\_augmentation(mat\_cv\* mat, int w, int h,

int pleft, int ptop, int swidth, int sheight, int flip,

float dhue, float dsat, float dexp,

int gaussian\_noise, int blur, int num\_boxes, float \*truth)

{

image out;

try {

cv::Mat img = \*(cv::Mat \*)mat;

// crop

cv::Rect src\_rect(pleft, ptop, swidth, sheight);

cv::Rect img\_rect(cv::Point2i(0, 0), img.size());

cv::Rect new\_src\_rect = src\_rect & img\_rect;

cv::Rect dst\_rect(cv::Point2i(std::max<int>(0, -pleft), std::max<int>(0, -ptop)), new\_src\_rect.size());

cv::Mat sized;

if (src\_rect.x == 0 && src\_rect.y == 0 && src\_rect.size() == img.size()) {

cv::resize(img, sized, cv::Size(w, h), 0, 0, cv::INTER\_LINEAR);

}

else {

cv::Mat cropped(src\_rect.size(), img.type());

//cropped.setTo(cv::Scalar::all(0));

cropped.setTo(cv::mean(img));

img(new\_src\_rect).copyTo(cropped(dst\_rect));

// resize

cv::resize(cropped, sized, cv::Size(w, h), 0, 0, cv::INTER\_LINEAR);

}

// flip

if (flip) {

cv::Mat cropped;

cv::flip(sized, cropped, 1); // 0 - x-axis, 1 - y-axis, -1 - both axes (x & y)

sized = cropped.clone();

}

// HSV augmentation

// cv::COLOR\_BGR2HSV, cv::COLOR\_RGB2HSV, cv::COLOR\_HSV2BGR, cv::COLOR\_HSV2RGB

if (dsat != 1 || dexp != 1 || dhue != 0) {

if (img.channels() >= 3)

{

cv::Mat hsv\_src;

cvtColor(sized, hsv\_src, cv::COLOR\_RGB2HSV); // RGB to HSV

std::vector<cv::Mat> hsv;

cv::split(hsv\_src, hsv);

hsv[1] \*= dsat;

hsv[2] \*= dexp;

hsv[0] += 179 \* dhue;

cv::merge(hsv, hsv\_src);

cvtColor(hsv\_src, sized, cv::COLOR\_HSV2RGB); // HSV to RGB (the same as previous)

}

else

{

sized \*= dexp;

}

}

//std::stringstream window\_name;

//window\_name << "augmentation - " << ipl;

//cv::imshow(window\_name.str(), sized);

//cv::waitKey(0);

if (blur) {

cv::Mat dst(sized.size(), sized.type());

if (blur == 1) {

cv::GaussianBlur(sized, dst, cv::Size(17, 17), 0);

//cv::bilateralFilter(sized, dst, 17, 75, 75);

}

else {

int ksize = (blur / 2) \* 2 + 1;

cv::Size kernel\_size = cv::Size(ksize, ksize);

cv::GaussianBlur(sized, dst, kernel\_size, 0);

//cv::medianBlur(sized, dst, ksize);

//cv::bilateralFilter(sized, dst, ksize, 75, 75);

// sharpen

//cv::Mat img\_tmp;

//cv::GaussianBlur(dst, img\_tmp, cv::Size(), 3);

//cv::addWeighted(dst, 1.5, img\_tmp, -0.5, 0, img\_tmp);

//dst = img\_tmp;

}

//std::cout << " blur num\_boxes = " << num\_boxes << std::endl;

if (blur == 1) {

cv::Rect img\_rect(0, 0, sized.cols, sized.rows);

int t;

for (t = 0; t < num\_boxes; ++t) {

box b = float\_to\_box\_stride(truth + t\*(4 + 1), 1);

if (!b.x) break;

int left = (b.x - b.w / 2.)\*sized.cols;

int width = b.w\*sized.cols;

int top = (b.y - b.h / 2.)\*sized.rows;

int height = b.h\*sized.rows;

cv::Rect roi(left, top, width, height);

roi = roi & img\_rect;

sized(roi).copyTo(dst(roi));

}

}

dst.copyTo(sized);

}

if (gaussian\_noise) {

cv::Mat noise = cv::Mat(sized.size(), sized.type());

gaussian\_noise = std::min(gaussian\_noise, 127);

gaussian\_noise = std::max(gaussian\_noise, 0);

cv::randn(noise, 0, gaussian\_noise); //mean and variance

cv::Mat sized\_norm = sized + noise;

//cv::normalize(sized\_norm, sized\_norm, 0.0, 255.0, cv::NORM\_MINMAX, sized.type());

//cv::imshow("source", sized);

//cv::imshow("gaussian noise", sized\_norm);

//cv::waitKey(0);

sized = sized\_norm;

}

//char txt[100];

//sprintf(txt, "blur = %d", blur);

//cv::putText(sized, txt, cv::Point(100, 100), cv::FONT\_HERSHEY\_COMPLEX\_SMALL, 1.7, CV\_RGB(255, 0, 0), 1, CV\_AA);

// Mat -> image

out = mat\_to\_image(sized);

}

catch (...) {

cerr << "OpenCV can't augment image: " << w << " x " << h << " \n";

out = mat\_to\_image(\*(cv::Mat\*)mat);

}

return out;

}

// blend two images with (alpha and beta)

extern "C" void blend\_images\_cv(image new\_img, float alpha, image old\_img, float beta)

{

cv::Mat new\_mat(cv::Size(new\_img.w, new\_img.h), CV\_32FC(new\_img.c), new\_img.data);// , size\_t step = AUTO\_STEP)

cv::Mat old\_mat(cv::Size(old\_img.w, old\_img.h), CV\_32FC(old\_img.c), old\_img.data);

cv::addWeighted(new\_mat, alpha, old\_mat, beta, 0.0, new\_mat);

}

// bilateralFilter bluring

extern "C" image blur\_image(image src\_img, int ksize)

{

cv::Mat src = image\_to\_mat(src\_img);

cv::Mat dst;

cv::Size kernel\_size = cv::Size(ksize, ksize);

cv::GaussianBlur(src, dst, kernel\_size, 0);

//cv::bilateralFilter(src, dst, ksize, 75, 75);

image dst\_img = mat\_to\_image(dst);

return dst\_img;

}

// ====================================================================

// Draw object - adversarial attack dnn

// ====================================================================

std::atomic<int> x\_start, y\_start;

std::atomic<int> x\_end, y\_end;

std::atomic<int> x\_size, y\_size;

std::atomic<bool> draw\_select, selected;

void callback\_mouse\_click(int event, int x, int y, int flags, void\* user\_data)

{

if (event == cv::EVENT\_LBUTTONDOWN)

{

draw\_select = true;

selected = false;

x\_start = x;

y\_start = y;

//if (prev\_img\_rect.contains(Point2i(x, y))) add\_id\_img = -1;

//else if (next\_img\_rect.contains(Point2i(x, y))) add\_id\_img = 1;

//else add\_id\_img = 0;

//std::cout << "cv::EVENT\_LBUTTONDOWN \n";

}

else if (event == cv::EVENT\_LBUTTONUP)

{

x\_size = abs(x - x\_start);

y\_size = abs(y - y\_start);

x\_end = std::max(x, 0);

y\_end = std::max(y, 0);

draw\_select = false;

selected = true;

//std::cout << "cv::EVENT\_LBUTTONUP \n";

}

else if (event == cv::EVENT\_MOUSEMOVE)

{

x\_size = abs(x - x\_start);

y\_size = abs(y - y\_start);

x\_end = std::max(x, 0);

y\_end = std::max(y, 0);

}

}

extern "C" void cv\_draw\_object(image sized, float \*truth\_cpu, int max\_boxes, int num\_truth, int \*it\_num\_set, float \*lr\_set, int \*boxonly, int classes, char \*\*names)

{

cv::Mat frame = image\_to\_mat(sized);

if(frame.channels() == 3) cv::cvtColor(frame, frame, cv::COLOR\_RGB2BGR);

cv::Mat frame\_clone = frame.clone();

std::string const window\_name = "Marking image";

cv::namedWindow(window\_name, cv::WINDOW\_NORMAL);

cv::resizeWindow(window\_name, 1280, 720);

cv::imshow(window\_name, frame);

cv::moveWindow(window\_name, 0, 0);

cv::setMouseCallback(window\_name, callback\_mouse\_click);

int it\_trackbar\_value = 200;

std::string const it\_trackbar\_name = "iterations";

int it\_tb\_res = cv::createTrackbar(it\_trackbar\_name, window\_name, &it\_trackbar\_value, 1000);

int lr\_trackbar\_value = 10;

std::string const lr\_trackbar\_name = "learning\_rate exp";

int lr\_tb\_res = cv::createTrackbar(lr\_trackbar\_name, window\_name, &lr\_trackbar\_value, 20);

int cl\_trackbar\_value = 0;

std::string const cl\_trackbar\_name = "class\_id";

int cl\_tb\_res = cv::createTrackbar(cl\_trackbar\_name, window\_name, &cl\_trackbar\_value, classes-1);

std::string const bo\_trackbar\_name = "box-only";

int bo\_tb\_res = cv::createTrackbar(bo\_trackbar\_name, window\_name, boxonly, 1);

int i = 0;

while (!selected) {

#ifndef CV\_VERSION\_EPOCH

int pressed\_key = cv::waitKeyEx(20); // OpenCV 3.x

#else

int pressed\_key = cv::waitKey(20); // OpenCV 2.x

#endif

if (pressed\_key == 27 || pressed\_key == 1048603) break;// break; // ESC - save & exit

frame\_clone = frame.clone();

char buff[100];

std::string lr\_value = "learning\_rate = " + std::to\_string(1.0 / pow(2, lr\_trackbar\_value));

cv::putText(frame\_clone, lr\_value, cv::Point2i(10, 20), cv::FONT\_HERSHEY\_SIMPLEX, 0.5, cv::Scalar(10, 50, 10), 3);

cv::putText(frame\_clone, lr\_value, cv::Point2i(10, 20), cv::FONT\_HERSHEY\_SIMPLEX, 0.5, cv::Scalar(20, 120, 60), 2);

cv::putText(frame\_clone, lr\_value, cv::Point2i(10, 20), cv::FONT\_HERSHEY\_SIMPLEX, 0.5, cv::Scalar(50, 200, 100), 1);

if (names) {

std::string obj\_name = names[cl\_trackbar\_value];

cv::putText(frame\_clone, obj\_name, cv::Point2i(10, 40), cv::FONT\_HERSHEY\_SIMPLEX, 0.5, cv::Scalar(10, 50, 10), 3);

cv::putText(frame\_clone, obj\_name, cv::Point2i(10, 40), cv::FONT\_HERSHEY\_SIMPLEX, 0.5, cv::Scalar(20, 120, 60), 2);

cv::putText(frame\_clone, obj\_name, cv::Point2i(10, 40), cv::FONT\_HERSHEY\_SIMPLEX, 0.5, cv::Scalar(50, 200, 100), 1);

}

if (draw\_select) {

cv::Rect selected\_rect(

cv::Point2i((int)min(x\_start, x\_end), (int)min(y\_start, y\_end)),

cv::Size(x\_size, y\_size));

rectangle(frame\_clone, selected\_rect, cv::Scalar(150, 200, 150));

}

cv::imshow(window\_name, frame\_clone);

}

if (selected) {

cv::Rect selected\_rect(

cv::Point2i((int)min(x\_start, x\_end), (int)min(y\_start, y\_end)),

cv::Size(x\_size, y\_size));

printf(" x\_start = %d, y\_start = %d, x\_size = %d, y\_size = %d \n",

x\_start.load(), y\_start.load(), x\_size.load(), y\_size.load());

rectangle(frame, selected\_rect, cv::Scalar(150, 200, 150));

cv::imshow(window\_name, frame);

cv::waitKey(100);

float width = x\_end - x\_start;

float height = y\_end - y\_start;

float const relative\_center\_x = (float)(x\_start + width / 2) / frame.cols;

float const relative\_center\_y = (float)(y\_start + height / 2) / frame.rows;

float const relative\_width = (float)width / frame.cols;

float const relative\_height = (float)height / frame.rows;

truth\_cpu[i \* 5 + 0] = relative\_center\_x;

truth\_cpu[i \* 5 + 1] = relative\_center\_y;

truth\_cpu[i \* 5 + 2] = relative\_width;

truth\_cpu[i \* 5 + 3] = relative\_height;

truth\_cpu[i \* 5 + 4] = cl\_trackbar\_value;

}

\*it\_num\_set = it\_trackbar\_value;

\*lr\_set = 1.0 / pow(2, lr\_trackbar\_value);

}

// ====================================================================

// Show Anchors

// ====================================================================

extern "C" void show\_acnhors(int number\_of\_boxes, int num\_of\_clusters, float \*rel\_width\_height\_array, model anchors\_data, int width, int height)

{

cv::Mat labels = cv::Mat(number\_of\_boxes, 1, CV\_32SC1);

cv::Mat points = cv::Mat(number\_of\_boxes, 2, CV\_32FC1);

cv::Mat centers = cv::Mat(num\_of\_clusters, 2, CV\_32FC1);

for (int i = 0; i < number\_of\_boxes; ++i) {

points.at<float>(i, 0) = rel\_width\_height\_array[i \* 2];

points.at<float>(i, 1) = rel\_width\_height\_array[i \* 2 + 1];

}

for (int i = 0; i < num\_of\_clusters; ++i) {

centers.at<float>(i, 0) = anchors\_data.centers.vals[i][0];

centers.at<float>(i, 1) = anchors\_data.centers.vals[i][1];

}

for (int i = 0; i < number\_of\_boxes; ++i) {

labels.at<int>(i, 0) = anchors\_data.assignments[i];

}

size\_t img\_size = 700;

cv::Mat img = cv::Mat(img\_size, img\_size, CV\_8UC3);

for (int i = 0; i < number\_of\_boxes; ++i) {

cv::Point pt;

pt.x = points.at<float>(i, 0) \* img\_size / width;

pt.y = points.at<float>(i, 1) \* img\_size / height;

int cluster\_idx = labels.at<int>(i, 0);

int red\_id = (cluster\_idx \* (uint64\_t)123 + 55) % 255;

int green\_id = (cluster\_idx \* (uint64\_t)321 + 33) % 255;

int blue\_id = (cluster\_idx \* (uint64\_t)11 + 99) % 255;

cv::circle(img, pt, 1, CV\_RGB(red\_id, green\_id, blue\_id), CV\_FILLED, 8, 0);

//if(pt.x > img\_size || pt.y > img\_size) printf("\n pt.x = %d, pt.y = %d \n", pt.x, pt.y);

}

for (int j = 0; j < num\_of\_clusters; ++j) {

cv::Point pt1, pt2;

pt1.x = pt1.y = 0;

pt2.x = centers.at<float>(j, 0) \* img\_size / width;

pt2.y = centers.at<float>(j, 1) \* img\_size / height;

cv::rectangle(img, pt1, pt2, CV\_RGB(255, 255, 255), 1, 8, 0);

}

save\_mat\_png(img, "cloud.png");

cv::imshow("clusters", img);

cv::waitKey(0);

cv::destroyAllWindows();

}

void show\_opencv\_info()

{

std::cerr << " OpenCV version: " << CV\_VERSION\_MAJOR << "." << CV\_VERSION\_MINOR << "." << CVAUX\_STR(CV\_VERSION\_REVISION) OCV\_D

<< std::endl;

}

} // extern "C"

#else // OPENCV

extern "C" void show\_opencv\_info()

{

std::cerr << " OpenCV isn't used - data augmentation will be slow \n";

}

extern "C" int wait\_key\_cv(int delay) { return 0; }

extern "C" int wait\_until\_press\_key\_cv() { return 0; }

extern "C" void destroy\_all\_windows\_cv() {}

#endif // OPENCV