```
#ifndef TF_DETECTOR_EXAMPLE_UTILS_H
#define TF_DETECTOR_EXAMPLE_UTILS_H

#endif //TF_DETECTOR_EXAMPLE_UTILS_H

#include <vector>
#include "tensorflow/core/framework/tensor.h"
#include "tensorflow/core/public/session.h"
#include <opencv2/core/mat.hpp>


using tensorflow::Tensor;
using tensorflow::Status;
using tensorflow::string;


Status readLabelsMapFile(const string &fileName, std::map<int, string>
&labelsMap);

Status loadGraph(const string &graph_file_name,
                 std::unique_ptr<tensorflow::Session> *session);

Status readTensorFromMat(const cv::Mat &mat, Tensor &outTensor);

void drawBoundingBoxOnImage(cv::Mat &image, double xMin, double yMin, double
xMax, double yMax, double score, std::string label, bool scaled);

void drawBoundingBoxesOnImage(cv::Mat &image,
                              tensorflow::TTypes<float>::Flat &scores,
                              tensorflow::TTypes<float>::Flat &classes,
                              tensorflow::TTypes<float,3>::Tensor &boxes,
                              std::map<int, string> &labelsMap,
                              std::vector<size_t> &idxs);

double IOU(cv::Rect box1, cv::Rect box2);

std::vector<size_t> filterBoxes(tensorflow::TTypes<float>::Flat &scores,
                                tensorflow::TTypes<float, 3>::Tensor &boxes,
                                double thresholdIOU, double thresholdScore);

#include "utils.h"

#include <math.h>
#include <fstream>
#include <utility>
#include <vector>
#include <iostream>
#include <regex>

#include "tensorflow/cc/ops/const_op.h"
#include "tensorflow/cc/ops/image_ops.h"
```

```
#include "tensorflow/cc/ops/standard_ops.h"
#include "tensorflow/core/framework/graph.pb.h"
#include "tensorflow/core/framework/tensor.h"
#include "tensorflow/core/graph/default_device.h"
#include "tensorflow/core/graph/graph_def_builder.h"
#include "tensorflow/core/lib/core/errors.h"
#include "tensorflow/core/lib/core/stringpiece.h"
#include "tensorflow/core/lib/core/threadpool.h"
#include "tensorflow/core/lib/io/path.h"
#include "tensorflow/core/lib/strings/stringprintf.h"
#include "tensorflow/core/platform/env.h"
#include "tensorflow/core/platform/init_main.h"
#include "tensorflow/core/platform/logging.h"
#include "tensorflow/core/platform/types.h"
#include "tensorflow/core/public/session.h"
#include "tensorflow/core/util/command_line_flags.h"

#include <cv.hpp>
#include <opencv2/core/mat.hpp>
#include <opencv2/imgproc/imgproc.hpp>

using namespace std;
using namespace cv;

using tensorflow::Flag;
using tensorflow::Tensor;
using tensorflow::Status;
using tensorflow::string;
using tensorflow::int32;

/** Read a model graph definition (xxx.pb) from disk, and creates a session
object you can use to run it.
 */
Status loadGraph(const string &graph_file_name,
                 unique_ptr<tensorflow::Session> *session) {
    tensorflow::GraphDef graph_def;
    Status load_graph_status =
            ReadBinaryProto(tensorflow::Env::Default(), graph_file_name,
&graph_def);
    if (!load_graph_status.ok()) {
        return tensorflow::errors::NotFound("Failed to load compute graph at
'",
                                            graph_file_name, "'");
    }
    session->reset(tensorflow::NewSession(tensorflow::SessionOptions()));
    Status session_create_status = (*session)->Create(graph_def);
    if (!session_create_status.ok()) {
        return session_create_status;
    }
    return Status::OK();
}
```

```
/** Read a labels map file (xxx.pbtxt) from disk to translate class numbers
into human-readable labels.
 */
Status readLabelsMapFile(const string &fileName, map<int, string> &labelsMap)
{

    // Read file into a string
    ifstream t(fileName);
    if (t.bad())
        return tensorflow::errors::NotFound("Failed to load labels map at '",
fileName, "'");
    stringstream buffer;
    buffer << t.rdbuf();
    string fileString = buffer.str();

    // Search entry patterns of type 'item { ... }' and parse each of them
    smatch matcherEntry;
    smatch matcherId;
    smatch matcherName;
    const regex reEntry("item \\{([\\S\\s]*?)\\}");
    const regex reId("[0-9]+");
    const regex reName("\'.+\'");
    string entry;

    auto stringBegin = sregex_iterator(fileString.begin(), fileString.end(),
reEntry);
    auto stringEnd = sregex_iterator();

    int id;
    string name;
    for (sregex_iterator i = stringBegin; i != stringEnd; i++) {
        matcherEntry = *i;
        entry = matcherEntry.str();
        regex_search(entry, matcherId, reId);
        if (!matcherId.empty())
            id = stoi(matcherId[0].str());
        else
            continue;
        regex_search(entry, matcherName, reName);
        if (!matcherName.empty())
            name = matcherName[0].str().substr(1,
matcherName[0].str().length() - 2);
        else
            continue;
        labelsMap.insert(pair<int, string>(id, name));
    }
    return Status::OK();
}

/** Convert Mat image into tensor of shape (1, height, width, d) where last
```

```
three dims are equal to the original dims.
 */
Status readTensorFromMat(const Mat &mat, Tensor &outTensor) {

    auto root = tensorflow::Scope::NewRootScope();
    using namespace ::tensorflow::ops;

    // Trick from https://github.com/tensorflow/tensorflow/issues/8033
    float *p = outTensor.flat<float>().data();
    Mat fakeMat(mat.rows, mat.cols, CV_32FC3, p);
    mat.convertTo(fakeMat, CV_32FC3);

    auto input_tensor = Placeholder(root.WithOpName("input"),
tensorflow::DT_FLOAT);
    vector<pair<string, tensorflow::Tensor>> inputs = {{"input", outTensor}};
    auto uint8Caster = Cast(root.WithOpName("uint8_Cast"), outTensor,
tensorflow::DT_UINT8);

    // This runs the GraphDef network definition that we've just constructed,
and
    // returns the results in the output outTensor.
    tensorflow::GraphDef graph;
    TF_RETURN_IF_ERROR(root.ToGraphDef(&graph));

    vector<Tensor> outTensors;
    unique_ptr<tensorflow::Session>
session(tensorflow::NewSession(tensorflow::SessionOptions()));

    TF_RETURN_IF_ERROR(session->Create(graph));
    TF_RETURN_IF_ERROR(session->Run({inputs}, {"uint8_Cast"}, {},
&outTensors));

    outTensor = outTensors.at(0);
    return Status::OK();
}

/** Draw bounding box and add caption to the image.
 *  Boolean flag _scaled_ shows if the passed coordinates are in relative
units (true by default in tensorflow detection)
 */
void drawBoundingBoxOnImage(Mat &image, double yMin, double xMin, double
yMax, double xMax, double score, string label, bool scaled=true) {
    cv::Point tl, br;
    if (scaled) {
        tl = cv::Point((int) (xMin * image.cols), (int) (yMin * image.rows));
        br = cv::Point((int) (xMax * image.cols), (int) (yMax * image.rows));
    } else {
        tl = cv::Point((int) xMin, (int) yMin);
        br = cv::Point((int) xMax, (int) yMax);
    }
    cv::rectangle(image, tl, br, cv::Scalar(0, 255, 255), 1);
```

```
    // Ceiling the score down to 3 decimals (weird!)
    float scoreRounded = floorf(score * 1000) / 1000;
    string scoreString = to_string(scoreRounded).substr(0, 5);
    string caption = label + " (" + scoreString + ")";

    // Adding caption of type "LABEL (X.XXX)" to the top-left corner of the
bounding box
    int fontCoeff = 12;
    cv::Point brRect = cv::Point(tl.x + caption.length() * fontCoeff / 1.6,
tl.y + fontCoeff);
    cv::rectangle(image, tl, brRect, cv::Scalar(0, 255, 255), -1);
    cv::Point textCorner = cv::Point(tl.x, tl.y + fontCoeff * 0.9);
    cv::putText(image, caption, textCorner, FONT_HERSHEY_SIMPLEX, 0.4,
cv::Scalar(255, 0, 0));
}

/** Draw bounding boxes and add captions to the image.
 *  Box is drawn only if corresponding score is higher than the _threshold_.
 */
void drawBoundingBoxesOnImage(Mat &image,
                              tensorflow::TTypes<float>::Flat &scores,
                              tensorflow::TTypes<float>::Flat &classes,
                              tensorflow::TTypes<float,3>::Tensor &boxes,
                              map<int, string> &labelsMap,
                              vector<size_t> &idxs) {
    for (int j = 0; j < idxs.size(); j++)
        drawBoundingBoxOnImage(image,
                               boxes(0,idxs.at(j),0), boxes(0,idxs.at(j),1),
                               boxes(0,idxs.at(j),2), boxes(0,idxs.at(j),3),
                               scores(idxs.at(j)),
labelsMap[classes(idxs.at(j))]);
}

/** Calculate intersection-over-union (IOU) for two given bbox Rects.
 */
double IOU(Rect2f box1, Rect2f box2) {

    float xA = max(box1.tl().x, box2.tl().x);
    float yA = max(box1.tl().y, box2.tl().y);
    float xB = min(box1.br().x, box2.br().x);
    float yB = min(box1.br().y, box2.br().y);

    float intersectArea = abs((xB - xA) * (yB - yA));
    float unionArea = abs(box1.area()) + abs(box2.area()) - intersectArea;

    return 1. * intersectArea / unionArea;
}

/** Return idxs of good boxes (ones with highest confidence score (>=
thresholdScore)
```

```
 *  and IOU <= thresholdIOU with others).
 */
vector<size_t> filterBoxes(tensorflow::TTypes<float>::Flat &scores,
                           tensorflow::TTypes<float, 3>::Tensor &boxes,
                           double thresholdIOU, double thresholdScore) {

    vector<size_t> sortIdxs(scores.size());
    iota(sortIdxs.begin(), sortIdxs.end(), 0);

    // Create set of "bad" idxs
    set<size_t> badIdxs = set<size_t>();
    size_t i = 0;
    while (i < sortIdxs.size()) {
        if (scores(sortIdxs.at(i)) < thresholdScore)
            badIdxs.insert(sortIdxs[i]);
        if (badIdxs.find(sortIdxs.at(i)) != badIdxs.end()) {
            i++;
            continue;
        }

        Rect2f box1 = Rect2f(Point2f(boxes(0, sortIdxs.at(i), 1), boxes(0,
sortIdxs.at(i), 0)),
                             Point2f(boxes(0, sortIdxs.at(i), 3), boxes(0,
sortIdxs.at(i), 2)));
        for (size_t j = i + 1; j < sortIdxs.size(); j++) {
            if (scores(sortIdxs.at(j)) < thresholdScore) {
                badIdxs.insert(sortIdxs[j]);
                continue;
            }
            Rect2f box2 = Rect2f(Point2f(boxes(0, sortIdxs.at(j), 1),
boxes(0, sortIdxs.at(j), 0)),
                                 Point2f(boxes(0, sortIdxs.at(j), 3),
boxes(0, sortIdxs.at(j), 2)));
            if (IOU(box1, box2) > thresholdIOU)
                badIdxs.insert(sortIdxs[j]);
        }
        i++;
    }

    // Prepare "good" idxs for return
    vector<size_t> goodIdxs = vector<size_t>();
    for (auto it = sortIdxs.begin(); it != sortIdxs.end(); it++)
        if (badIdxs.find(sortIdxs.at(*it)) == badIdxs.end())
            goodIdxs.push_back(*it);

    return goodIdxs;
}
#include <fstream>
#include <utility>
#include <vector>
#include <iostream>
```

```
#include "tensorflow/cc/ops/const_op.h"
#include "tensorflow/cc/ops/image_ops.h"
#include "tensorflow/cc/ops/standard_ops.h"
#include "tensorflow/core/framework/graph.pb.h"
#include "tensorflow/core/graph/default_device.h"
#include "tensorflow/core/graph/graph_def_builder.h"
#include "tensorflow/core/lib/core/threadpool.h"
#include "tensorflow/core/lib/io/path.h"
#include "tensorflow/core/lib/strings/stringprintf.h"
#include "tensorflow/core/platform/init_main.h"
#include "tensorflow/core/public/session.h"
#include "tensorflow/core/util/command_line_flags.h"

#include <opencv2/core/mat.hpp>
#include <opencv2/videoio.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <cv.hpp>

#include <time.h>

#include "utils.h"

using tensorflow::Flag;
using tensorflow::Tensor;
using tensorflow::Status;
using tensorflow::string;
using tensorflow::int32;

using namespace std;
using namespace cv;

int main(int argc, char* argv[]) {

    // Set dirs variables
    string ROOTDIR = "../";
    string LABELS = "demo/ssd_mobilenet_v1_egohands/labels_map.pbtxt";
    string GRAPH =
"demo/ssd_mobilenet_v1_egohands/frozen_inference_graph.pb";

    // Set input & output nodes names
    string inputLayer = "image_tensor:0";
    vector<string> outputLayer = {"detection_boxes:0", "detection_scores:0",
"detection_classes:0", "num_detections:0"};

    // Load and initialize the model from .pb file
    std::unique_ptr<tensorflow::Session> session;
    string graphPath = tensorflow::io::JoinPath(ROOTDIR, GRAPH);
    LOG(INFO) << "graphPath:" << graphPath;
    Status loadGraphStatus = loadGraph(graphPath, &session);
```

```
    if (!loadGraphStatus.ok()) {
        LOG(ERROR) << "loadGraph(): ERROR" << loadGraphStatus;
        return -1;
    } else
        LOG(INFO) << "loadGraph(): frozen graph loaded" << endl;


    // Load labels map from .pbtxt file
    std::map<int, std::string> labelsMap = std::map<int,std::string>();
    Status readLabelsMapStatus =
readLabelsMapFile(tensorflow::io::JoinPath(ROOTDIR, LABELS), labelsMap);
    if (!readLabelsMapStatus.ok()) {
        LOG(ERROR) << "readLabelsMapFile(): ERROR" << loadGraphStatus;
        return -1;
    } else
        LOG(INFO) << "readLabelsMapFile(): labels map loaded with " <<
labelsMap.size() << " label(s)" << endl;

    Mat frame;
    Tensor tensor;
    std::vector<Tensor> outputs;
    double thresholdScore = 0.5;
    double thresholdIOU = 0.8;

    // FPS count
    int nFrames = 25;
    int iFrame = 0;
    double fps = 0.;
    time_t start, end;
    time(&start);

    // Start streaming frames from camera
    VideoCapture cap(1);

    tensorflow::TensorShape shape = tensorflow::TensorShape();
    shape.AddDim(1);
    shape.AddDim((int64)cap.get(CAP_PROP_FRAME_HEIGHT));
    shape.AddDim((int64)cap.get(CAP_PROP_FRAME_WIDTH));
    shape.AddDim(3);

    while (cap.isOpened()) {
        cap >> frame;
        cvtColor(frame, frame, COLOR_BGR2RGB);
        cout << "Frame # " << iFrame << endl;

        if (nFrames % (iFrame + 1) == 0) {
            time(&end);
            fps = 1. * nFrames / difftime(end, start);
            time(&start);
        }
        iFrame++;
```

```
        // Convert mat to tensor
        tensor = Tensor(tensorflow::DT_FLOAT, shape);
        Status readTensorStatus = readTensorFromMat(frame, tensor);
        if (!readTensorStatus.ok()) {
            LOG(ERROR) << "Mat->Tensor conversion failed: " <<
readTensorStatus;
            return -1;
        }

        // Run the graph on tensor
        outputs.clear();
        Status runStatus = session->Run({{inputLayer, tensor}}, outputLayer,
{}, &outputs);
        if (!runStatus.ok()) {
            LOG(ERROR) << "Running model failed: " << runStatus;
            return -1;
        }

        // Extract results from the outputs vector
        tensorflow::TTypes<float>::Flat scores = outputs[1].flat<float>();
        tensorflow::TTypes<float>::Flat classes = outputs[2].flat<float>();
        tensorflow::TTypes<float>::Flat numDetections =
outputs[3].flat<float>();
        tensorflow::TTypes<float, 3>::Tensor boxes =
outputs[0].flat_outer_dims<float,3>();

        vector<size_t> goodIdxs = filterBoxes(scores, boxes, thresholdIOU,
thresholdScore);
        for (size_t i = 0; i < goodIdxs.size(); i++)
            LOG(INFO) << "score:" << scores(goodIdxs.at(i)) << ",class:" <<
labelsMap[classes(goodIdxs.at(i))]
                      << " (" << classes(goodIdxs.at(i)) << "), box:" << ","
<< boxes(0, goodIdxs.at(i), 0) << ","
                      << boxes(0, goodIdxs.at(i), 1) << "," << boxes(0,
goodIdxs.at(i), 2) << ","
                      << boxes(0, goodIdxs.at(i), 3);

        // Draw bboxes and captions
        cvtColor(frame, frame, COLOR_BGR2RGB);
        drawBoundingBoxesOnImage(frame, scores, classes, boxes, labelsMap,
goodIdxs);

        putText(frame, to_string(fps).substr(0, 5), Point(0, frame.rows),
FONT_HERSHEY_SIMPLEX, 0.7, Scalar(255, 255, 255));
        imshow("stream", frame);
        waitKey(5);
    }
    destroyAllWindows();
```

```
    return 0;
}
```