cmake\_minimum\_required(VERSION 3.8)

set(Darknet\_MAJOR\_VERSION 0)

set(Darknet\_MINOR\_VERSION 2)

set(Darknet\_PATCH\_VERSION 5)

set(Darknet\_TWEAK\_VERSION 1)

set(Darknet\_VERSION ${Darknet\_MAJOR\_VERSION}.${Darknet\_MINOR\_VERSION}.${Darknet\_PATCH\_VERSION}.${Darknet\_TWEAK\_VERSION})

option(CMAKE\_VERBOSE\_MAKEFILE "Create verbose makefile" OFF)

option(CUDA\_VERBOSE\_BUILD "Create verbose CUDA build" OFF)

option(BUILD\_SHARED\_LIBS "Create dark as a shared library" ON)

option(BUILD\_AS\_CPP "Build Darknet using C++ compiler also for C files" OFF)

option(BUILD\_USELIB\_TRACK "Build uselib\_track" ON)

option(MANUALLY\_EXPORT\_TRACK\_OPTFLOW "Manually export the TRACK\_OPTFLOW=1 define" OFF)

option(ENABLE\_OPENCV "Enable OpenCV integration" ON)

option(ENABLE\_CUDA "Enable CUDA support" ON)

option(ENABLE\_CUDNN "Enable CUDNN" ON)

option(ENABLE\_CUDNN\_HALF "Enable CUDNN Half precision" ON)

option(ENABLE\_ZED\_CAMERA "Enable ZED Camera support" ON)

option(ENABLE\_VCPKG\_INTEGRATION "Enable VCPKG integration" ON)

if(ENABLE\_VCPKG\_INTEGRATION AND DEFINED ENV{VCPKG\_ROOT} AND NOT DEFINED CMAKE\_TOOLCHAIN\_FILE)

set(CMAKE\_TOOLCHAIN\_FILE "$ENV{VCPKG\_ROOT}/scripts/buildsystems/vcpkg.cmake" CACHE STRING "")

message(STATUS "VCPKG found: $ENV{VCPKG\_ROOT}")

message(STATUS "Using VCPKG integration")

endif()

project(Darknet VERSION ${Darknet\_VERSION})

if(WIN32 AND NOT DEFINED CMAKE\_TOOLCHAIN\_FILE)

set(USE\_INTEGRATED\_LIBS "TRUE" CACHE BOOL "Use libs distributed with this repo")

else()

set(USE\_INTEGRATED\_LIBS "FALSE" CACHE BOOL "Use libs distributed with this repo")

endif()

enable\_language(C)

enable\_language(CXX)

set(CMAKE\_CXX\_STANDARD 11)

set(CMAKE\_MODULE\_PATH "${CMAKE\_CURRENT\_LIST\_DIR}/cmake/Modules/" ${CMAKE\_MODULE\_PATH})

if (CMAKE\_INSTALL\_PREFIX\_INITIALIZED\_TO\_DEFAULT)

set(CMAKE\_INSTALL\_PREFIX "${CMAKE\_CURRENT\_LIST\_DIR}" CACHE PATH "Install prefix" FORCE)

endif()

set(INSTALL\_BIN\_DIR "${CMAKE\_CURRENT\_LIST\_DIR}" CACHE PATH "Path where exe and dll will be installed")

set(INSTALL\_LIB\_DIR "${CMAKE\_CURRENT\_LIST\_DIR}" CACHE PATH "Path where lib will be installed")

set(INSTALL\_INCLUDE\_DIR "include/darknet" CACHE PATH "Path where headers will be installed")

set(INSTALL\_CMAKE\_DIR "share/darknet" CACHE PATH "Path where cmake configs will be installed")

if(${CMAKE\_VERSION} VERSION\_LESS "3.9.0")

message(WARNING "To build with CUDA support you need CMake 3.9.0+")

set(ENABLE\_CUDA "FALSE" CACHE BOOL "Enable CUDA support" FORCE)

else()

include(CheckLanguage)

check\_language(CUDA)

if(CMAKE\_CUDA\_COMPILER AND ENABLE\_CUDA)

set(CUDA\_ARCHITECTURES "Auto" CACHE STRING "\"Auto\" detects local machine GPU compute arch at runtime, \"Common\" and \"All\" cover common and entire subsets of architectures, \"Names\" is a list of architectures to enable by name, \"Numbers\" is a list of compute capabilities (version number) to enable")

set\_property(CACHE CUDA\_ARCHITECTURES PROPERTY STRINGS "Auto" "Common" "All" "Kepler Maxwell Kepler+Tegra Maxwell+Tegra Pascal" "3.0 7.5")

enable\_language(CUDA)

find\_package(CUDA REQUIRED)

if(CUDA\_VERSION VERSION\_LESS "9.0")

message(STATUS "Unsupported CUDA version, please upgrade to CUDA 9+. Disabling CUDA support")

set(ENABLE\_CUDA "FALSE" CACHE BOOL "Enable CUDA support" FORCE)

else()

cuda\_select\_nvcc\_arch\_flags(CUDA\_ARCH\_FLAGS ${CUDA\_ARCHITECTURES})

message(STATUS "Building with CUDA flags: " "${CUDA\_ARCH\_FLAGS}")

if (NOT "arch=compute\_70,code=sm\_70" IN\_LIST CUDA\_ARCH\_FLAGS)

set(ENABLE\_CUDNN\_HALF "FALSE" CACHE BOOL "Enable CUDNN Half precision" FORCE)

message(STATUS "Your setup does not supports half precision (it requires CC >= 7.5)")

endif()

endif()

else()

set(ENABLE\_CUDA "FALSE" CACHE BOOL "Enable CUDA support" FORCE)

endif()

endif()

if (WIN32 AND ENABLE\_CUDA AND CMAKE\_MAKE\_PROGRAM MATCHES "ninja")

option(SELECT\_OPENCV\_MODULES "Use only few selected OpenCV modules to circumvent 8192 char limit when using Ninja on Windows" ON)

else()

option(SELECT\_OPENCV\_MODULES "Use only few selected OpenCV modules to circumvent 8192 char limit when using Ninja on Windows" OFF)

endif()

if(USE\_INTEGRATED\_LIBS)

set(PThreads\_windows\_DIR ${CMAKE\_CURRENT\_LIST\_DIR}/3rdparty/pthreads CACHE PATH "Path where pthreads for windows can be located")

endif()

set(Stb\_DIR ${CMAKE\_CURRENT\_LIST\_DIR}/3rdparty/stb CACHE PATH "Path where Stb image library can be located")

set(CMAKE\_DEBUG\_POSTFIX d)

set(CMAKE\_THREAD\_PREFER\_PTHREAD ON)

find\_package(Threads REQUIRED)

if(MSVC)

find\_package(PThreads\_windows REQUIRED)

endif()

if(ENABLE\_OPENCV)

find\_package(OpenCV)

if(OpenCV\_FOUND)

if(SELECT\_OPENCV\_MODULES)

if(TARGET opencv\_world)

list(APPEND OpenCV\_LINKED\_COMPONENTS "opencv\_world")

else()

if(TARGET opencv\_core)

list(APPEND OpenCV\_LINKED\_COMPONENTS "opencv\_core")

endif()

if(TARGET opencv\_highgui)

list(APPEND OpenCV\_LINKED\_COMPONENTS "opencv\_highgui")

endif()

if(TARGET opencv\_imgproc)

list(APPEND OpenCV\_LINKED\_COMPONENTS "opencv\_imgproc")

endif()

if(TARGET opencv\_video)

list(APPEND OpenCV\_LINKED\_COMPONENTS "opencv\_video")

endif()

if(TARGET opencv\_videoio)

list(APPEND OpenCV\_LINKED\_COMPONENTS "opencv\_videoio")

endif()

if(TARGET opencv\_imgcodecs)

list(APPEND OpenCV\_LINKED\_COMPONENTS "opencv\_imgcodecs")

endif()

if(TARGET opencv\_text)

list(APPEND OpenCV\_LINKED\_COMPONENTS "opencv\_text")

endif()

endif()

else()

list(APPEND OpenCV\_LINKED\_COMPONENTS ${OpenCV\_LIBS})

endif()

endif()

endif()

find\_package(Stb REQUIRED)

if(${CMAKE\_VERSION} VERSION\_LESS "3.11.0")

message(WARNING "To build with OpenMP support you need CMake 3.11.0+")

else()

find\_package(OpenMP)

endif()

set(ADDITIONAL\_CXX\_FLAGS "-Wall -Wno-unused-result -Wno-unknown-pragmas -Wfatal-errors -Wno-deprecated-declarations -Wno-write-strings")

set(ADDITIONAL\_C\_FLAGS "-Wall -Wno-unused-result -Wno-unknown-pragmas -Wfatal-errors -Wno-deprecated-declarations -Wno-write-strings")

if(MSVC)

set(ADDITIONAL\_CXX\_FLAGS "/wd4013 /wd4018 /wd4028 /wd4047 /wd4068 /wd4090 /wd4101 /wd4113 /wd4133 /wd4190 /wd4244 /wd4267 /wd4305 /wd4477 /wd4996 /wd4819 /fp:fast")

set(ADDITIONAL\_C\_FLAGS "/wd4013 /wd4018 /wd4028 /wd4047 /wd4068 /wd4090 /wd4101 /wd4113 /wd4133 /wd4190 /wd4244 /wd4267 /wd4305 /wd4477 /wd4996 /wd4819 /fp:fast")

set(CMAKE\_CXX\_FLAGS "${ADDITIONAL\_CXX\_FLAGS} ${CMAKE\_CXX\_FLAGS}")

set(CMAKE\_C\_FLAGS "${ADDITIONAL\_C\_FLAGS} ${CMAKE\_C\_FLAGS}")

string(REGEX REPLACE "/O2" "/Ox" CMAKE\_CXX\_FLAGS\_RELEASE ${CMAKE\_CXX\_FLAGS\_RELEASE})

string(REGEX REPLACE "/O2" "/Ox" CMAKE\_C\_FLAGS\_RELEASE ${CMAKE\_C\_FLAGS\_RELEASE})

endif()

if(CMAKE\_COMPILER\_IS\_GNUCC OR "${CMAKE\_CXX\_COMPILER\_ID}" MATCHES "Clang")

if ("${CMAKE\_CXX\_COMPILER\_ID}" MATCHES "Clang")

if (UNIX AND NOT APPLE)

set(CMAKE\_CXX\_FLAGS "-pthread ${CMAKE\_CXX\_FLAGS}") #force pthread to avoid bugs in some cmake setups

set(CMAKE\_C\_FLAGS "-pthread ${CMAKE\_C\_FLAGS}")

endif()

endif()

set(CMAKE\_CXX\_FLAGS "${ADDITIONAL\_CXX\_FLAGS} ${CMAKE\_CXX\_FLAGS}")

set(CMAKE\_C\_FLAGS "${ADDITIONAL\_C\_FLAGS} ${CMAKE\_C\_FLAGS}")

string(REGEX REPLACE "-O0" "-Og" CMAKE\_CXX\_FLAGS\_DEBUG ${CMAKE\_CXX\_FLAGS\_DEBUG})

string(REGEX REPLACE "-O3" "-Ofast" CMAKE\_CXX\_FLAGS\_RELEASE ${CMAKE\_CXX\_FLAGS\_RELEASE})

string(REGEX REPLACE "-O0" "-Og" CMAKE\_C\_FLAGS\_DEBUG ${CMAKE\_C\_FLAGS\_DEBUG})

string(REGEX REPLACE "-O3" "-Ofast" CMAKE\_C\_FLAGS\_RELEASE ${CMAKE\_C\_FLAGS\_RELEASE})

set(CMAKE\_CXX\_FLAGS\_RELEASE "${CMAKE\_CXX\_FLAGS\_RELEASE} -ffp-contract=fast -mavx -mavx2 -msse3 -msse4.1 -msse4.2 -msse4a")

set(CMAKE\_C\_FLAGS\_RELEASE "${CMAKE\_C\_FLAGS\_RELEASE} -ffp-contract=fast -mavx -mavx2 -msse3 -msse4.1 -msse4.2 -msse4a")

endif()

if(OpenCV\_FOUND)

if(ENABLE\_CUDA AND NOT OpenCV\_CUDA\_VERSION)

set(BUILD\_USELIB\_TRACK "FALSE" CACHE BOOL "Build uselib\_track" FORCE)

message(STATUS " -> darknet is fine for now, but uselib\_track has been disabled!")

message(STATUS " -> Please rebuild OpenCV from sources with CUDA support to enable it")

elseif(ENABLE\_CUDA AND OpenCV\_CUDA\_VERSION)

if(TARGET opencv\_cudaoptflow)

list(APPEND OpenCV\_LINKED\_COMPONENTS "opencv\_cudaoptflow")

endif()

if(TARGET opencv\_cudaimgproc)

list(APPEND OpenCV\_LINKED\_COMPONENTS "opencv\_cudaimgproc")

endif()

endif()

endif()

if(ENABLE\_CUDA)

find\_package(CUDNN)

if(NOT CUDNN\_FOUND)

set(ENABLE\_CUDNN "FALSE" CACHE BOOL "Enable CUDNN" FORCE)

endif()

endif()

if(ENABLE\_CUDA)

if (MSVC)

set(ADDITIONAL\_CXX\_FLAGS "${ADDITIONAL\_CXX\_FLAGS} /DGPU")

if(CUDNN\_FOUND)

set(ADDITIONAL\_CXX\_FLAGS "${ADDITIONAL\_CXX\_FLAGS} /DCUDNN")

endif()

if(OpenCV\_FOUND)

set(ADDITIONAL\_CXX\_FLAGS "${ADDITIONAL\_CXX\_FLAGS} /DOPENCV")

endif()

string(REPLACE " " "," ADDITIONAL\_CXX\_FLAGS\_COMMA\_SEPARATED "${ADDITIONAL\_CXX\_FLAGS}")

set(CUDA\_HOST\_COMPILER\_FLAGS "-Wno-deprecated-declarations -Xcompiler=\"${ADDITIONAL\_CXX\_FLAGS\_COMMA\_SEPARATED}\"")

else()

set(ADDITIONAL\_CXX\_FLAGS "${ADDITIONAL\_CXX\_FLAGS} -DGPU")

if(CUDNN\_FOUND)

set(ADDITIONAL\_CXX\_FLAGS "${ADDITIONAL\_CXX\_FLAGS} -DCUDNN")

endif()

if(OpenCV\_FOUND)

set(ADDITIONAL\_CXX\_FLAGS "${ADDITIONAL\_CXX\_FLAGS} -DOPENCV")

endif()

set(CUDA\_HOST\_COMPILER\_FLAGS "--compiler-options \" ${ADDITIONAL\_CXX\_FLAGS} -fPIC -fopenmp -Ofast \"")

endif()

string (REPLACE ";" " " CUDA\_ARCH\_FLAGS\_SPACE\_SEPARATED "${CUDA\_ARCH\_FLAGS}")

set(CMAKE\_CUDA\_FLAGS "${CUDA\_ARCH\_FLAGS\_SPACE\_SEPARATED} ${CUDA\_HOST\_COMPILER\_FLAGS} ${CMAKE\_CUDA\_FLAGS}")

message(STATUS "CMAKE\_CUDA\_FLAGS: ${CMAKE\_CUDA\_FLAGS}")

endif()

if(ENABLE\_CUDA)

if(ENABLE\_ZED\_CAMERA)

find\_package(ZED 2 QUIET)

if(ZED\_FOUND)

include\_directories(${ZED\_INCLUDE\_DIRS})

link\_directories(${ZED\_LIBRARY\_DIR})

message(STATUS "ZED SDK enabled")

else()

message(STATUS "ZED SDK not found")

set(ENABLE\_ZED\_CAMERA "FALSE" CACHE BOOL "Enable ZED Camera support" FORCE)

endif()

endif()

else()

message(STATUS "ZED SDK not enabled, since it requires CUDA")

set(ENABLE\_ZED\_CAMERA "FALSE" CACHE BOOL "Enable ZED Camera support" FORCE)

endif()

set(DARKNET\_INSTALL\_INCLUDE\_DIR ${INSTALL\_INCLUDE\_DIR})

# Make relative paths absolute (needed later on)

foreach(p LIB BIN INCLUDE CMAKE)

set(var INSTALL\_${p}\_DIR)

if(NOT IS\_ABSOLUTE "${${var}}")

set(${var} "${CMAKE\_INSTALL\_PREFIX}/${${var}}")

endif()

endforeach()

configure\_file(

"${CMAKE\_CURRENT\_LIST\_DIR}/src/version.h.in"

"${CMAKE\_CURRENT\_LIST\_DIR}/src/version.h"

)

#look for all \*.h files in src folder

file(GLOB headers "${CMAKE\_CURRENT\_LIST\_DIR}/src/\*.h")

#add also files in the include folder

list(APPEND headers

${CMAKE\_CURRENT\_LIST\_DIR}/include/darknet.h

)

#remove windows only files

if(NOT WIN32)

list(REMOVE\_ITEM headers

${CMAKE\_CURRENT\_LIST\_DIR}/src/gettimeofday.h

${CMAKE\_CURRENT\_LIST\_DIR}/src/getopt.h

)

endif()

#set(exported\_headers ${headers})

#look for all \*.c files in src folder

file(GLOB sources "${CMAKE\_CURRENT\_LIST\_DIR}/src/\*.c")

#add also .cpp files

list(APPEND sources

${CMAKE\_CURRENT\_LIST\_DIR}/src/http\_stream.cpp

${CMAKE\_CURRENT\_LIST\_DIR}/src/image\_opencv.cpp

)

#remove darknet.c file which is necessary only for the executable, not for the lib

list(REMOVE\_ITEM sources

${CMAKE\_CURRENT\_LIST\_DIR}/src/darknet.c

)

#remove windows only files

if(NOT WIN32)

list(REMOVE\_ITEM sources

${CMAKE\_CURRENT\_LIST\_DIR}/src/gettimeofday.c

${CMAKE\_CURRENT\_LIST\_DIR}/src/getopt.c

)

endif()

if(ENABLE\_CUDA)

file(GLOB cuda\_sources "${CMAKE\_CURRENT\_LIST\_DIR}/src/\*.cu")

endif()

if(BUILD\_AS\_CPP)

set\_source\_files\_properties(${sources} PROPERTIES LANGUAGE CXX)

endif()

add\_library(dark ${CMAKE\_CURRENT\_LIST\_DIR}/include/yolo\_v2\_class.hpp ${CMAKE\_CURRENT\_LIST\_DIR}/src/yolo\_v2\_class.cpp ${sources} ${headers} ${cuda\_sources})

set\_target\_properties(dark PROPERTIES POSITION\_INDEPENDENT\_CODE ON)

if(ENABLE\_CUDA)

set\_target\_properties(dark PROPERTIES CUDA\_SEPARABLE\_COMPILATION ON)

endif()

if(BUILD\_SHARED\_LIBS)

target\_compile\_definitions(dark PRIVATE LIB\_EXPORTS=1)

endif()

if(BUILD\_AS\_CPP)

set\_target\_properties(dark PROPERTIES LINKER\_LANGUAGE CXX)

endif()

if(OpenCV\_FOUND AND OpenCV\_VERSION VERSION\_GREATER "3.0" AND BUILD\_USELIB\_TRACK)

add\_executable(uselib\_track ${CMAKE\_CURRENT\_LIST\_DIR}/src/yolo\_console\_dll.cpp)

endif()

add\_executable(uselib ${CMAKE\_CURRENT\_LIST\_DIR}/src/yolo\_console\_dll.cpp)

if(BUILD\_AS\_CPP)

set\_target\_properties(uselib PROPERTIES LINKER\_LANGUAGE CXX)

endif()

add\_executable(darknet ${CMAKE\_CURRENT\_LIST\_DIR}/src/darknet.c ${sources} ${headers} ${cuda\_sources})

if(BUILD\_AS\_CPP)

set\_source\_files\_properties(${CMAKE\_CURRENT\_LIST\_DIR}/src/darknet.c PROPERTIES LANGUAGE CXX)

set\_target\_properties(darknet PROPERTIES LINKER\_LANGUAGE CXX)

endif()

target\_include\_directories(darknet PUBLIC $<BUILD\_INTERFACE:${CMAKE\_CURRENT\_LIST\_DIR}/include> $<BUILD\_INTERFACE:${CMAKE\_CURRENT\_LIST\_DIR}/src> $<INSTALL\_INTERFACE:${DARKNET\_INSTALL\_INCLUDE\_DIR}> $<BUILD\_INTERFACE:${Stb\_INCLUDE\_DIR}>)

target\_include\_directories(dark PUBLIC $<BUILD\_INTERFACE:${CMAKE\_CURRENT\_LIST\_DIR}/include> $<BUILD\_INTERFACE:${CMAKE\_CURRENT\_LIST\_DIR}/src> $<INSTALL\_INTERFACE:${DARKNET\_INSTALL\_INCLUDE\_DIR}> $<BUILD\_INTERFACE:${Stb\_INCLUDE\_DIR}>)

target\_include\_directories(uselib PUBLIC $<BUILD\_INTERFACE:${CMAKE\_CURRENT\_LIST\_DIR}/include> $<BUILD\_INTERFACE:${CMAKE\_CURRENT\_LIST\_DIR}/src> $<INSTALL\_INTERFACE:${DARKNET\_INSTALL\_INCLUDE\_DIR}> $<BUILD\_INTERFACE:${Stb\_INCLUDE\_DIR}>)

target\_compile\_definitions(darknet PRIVATE -DUSE\_CMAKE\_LIBS)

target\_compile\_definitions(dark PRIVATE -DUSE\_CMAKE\_LIBS)

target\_compile\_definitions(uselib PRIVATE -DUSE\_CMAKE\_LIBS)

if(OpenCV\_FOUND AND OpenCV\_VERSION VERSION\_GREATER "3.0" AND BUILD\_USELIB\_TRACK AND NOT MANUALLY\_EXPORT\_TRACK\_OPTFLOW)

target\_compile\_definitions(dark PUBLIC TRACK\_OPTFLOW=1)

endif()

if(CUDNN\_FOUND)

target\_link\_libraries(darknet PRIVATE CuDNN::CuDNN)

target\_link\_libraries(dark PRIVATE CuDNN::CuDNN)

target\_compile\_definitions(darknet PRIVATE -DCUDNN)

target\_compile\_definitions(dark PUBLIC -DCUDNN)

if(ENABLE\_CUDNN\_HALF)

target\_compile\_definitions(darknet PRIVATE -DCUDNN\_HALF)

target\_compile\_definitions(dark PUBLIC -DCUDNN\_HALF)

endif()

endif()

if(OpenCV\_FOUND)

target\_link\_libraries(darknet PRIVATE ${OpenCV\_LINKED\_COMPONENTS})

target\_link\_libraries(uselib PRIVATE ${OpenCV\_LINKED\_COMPONENTS})

target\_link\_libraries(dark PUBLIC ${OpenCV\_LINKED\_COMPONENTS})

target\_include\_directories(dark PUBLIC ${OpenCV\_INCLUDE\_DIRS})

target\_compile\_definitions(darknet PRIVATE -DOPENCV)

target\_compile\_definitions(dark PUBLIC -DOPENCV)

endif()

if(OPENMP\_FOUND)

target\_link\_libraries(darknet PRIVATE OpenMP::OpenMP\_CXX)

target\_link\_libraries(darknet PRIVATE OpenMP::OpenMP\_C)

target\_link\_libraries(dark PUBLIC OpenMP::OpenMP\_CXX)

target\_link\_libraries(dark PUBLIC OpenMP::OpenMP\_C)

endif()

if(CMAKE\_COMPILER\_IS\_GNUCC)

target\_link\_libraries(darknet PRIVATE m)

target\_link\_libraries(dark PUBLIC m)

endif()

if(MSVC)

target\_link\_libraries(darknet PRIVATE PThreads\_windows::PThreads\_windows)

target\_link\_libraries(darknet PRIVATE wsock32 ws2\_32)

target\_link\_libraries(dark PUBLIC PThreads\_windows::PThreads\_windows)

target\_link\_libraries(dark PUBLIC wsock32 ws2\_32)

target\_link\_libraries(uselib PRIVATE PThreads\_windows::PThreads\_windows)

target\_compile\_definitions(darknet PRIVATE -D\_CRT\_RAND\_S -DNOMINMAX -D\_USE\_MATH\_DEFINES)

target\_compile\_definitions(dark PRIVATE -D\_CRT\_RAND\_S -DNOMINMAX -D\_USE\_MATH\_DEFINES)

target\_compile\_definitions(dark PUBLIC -D\_CRT\_SECURE\_NO\_WARNINGS)

target\_compile\_definitions(uselib PRIVATE -D\_CRT\_RAND\_S -DNOMINMAX -D\_USE\_MATH\_DEFINES)

endif()

target\_link\_libraries(darknet PRIVATE Threads::Threads)

target\_link\_libraries(dark PUBLIC Threads::Threads)

target\_link\_libraries(uselib PRIVATE Threads::Threads)

if(ENABLE\_ZED\_CAMERA)

target\_link\_libraries(darknet PRIVATE ${ZED\_LIBRARIES})

target\_link\_libraries(dark PUBLIC ${ZED\_LIBRARIES})

target\_link\_libraries(uselib PRIVATE ${ZED\_LIBRARIES})

target\_compile\_definitions(darknet PRIVATE -DZED\_STEREO)

target\_compile\_definitions(uselib PRIVATE -DZED\_STEREO)

target\_compile\_definitions(dark PUBLIC -DZED\_STEREO)

endif()

if(ENABLE\_CUDA)

target\_include\_directories(darknet PRIVATE ${CMAKE\_CUDA\_TOOLKIT\_INCLUDE\_DIRECTORIES})

target\_include\_directories(dark PUBLIC ${CMAKE\_CUDA\_TOOLKIT\_INCLUDE\_DIRECTORIES})

target\_link\_libraries(darknet PRIVATE curand cublas cuda)

target\_link\_libraries(dark PRIVATE curand cublas cuda)

set\_target\_properties(dark PROPERTIES CUDA\_RESOLVE\_DEVICE\_SYMBOLS ON)

target\_compile\_definitions(darknet PRIVATE -DGPU)

target\_compile\_definitions(dark PUBLIC -DGPU)

endif()

if(USE\_INTEGRATED\_LIBS)

target\_compile\_definitions(darknet PRIVATE -D\_TIMESPEC\_DEFINED)

target\_compile\_definitions(dark PRIVATE -D\_TIMESPEC\_DEFINED)

endif()

target\_link\_libraries(uselib PRIVATE dark)

if(OpenCV\_FOUND AND OpenCV\_VERSION VERSION\_GREATER "3.0" AND BUILD\_USELIB\_TRACK)

target\_link\_libraries(uselib\_track PRIVATE dark)

target\_compile\_definitions(uselib\_track PRIVATE TRACK\_OPTFLOW=1)

target\_compile\_definitions(uselib\_track PRIVATE -DUSE\_CMAKE\_LIBS)

if(BUILD\_AS\_CPP)

set\_target\_properties(uselib\_track PROPERTIES LINKER\_LANGUAGE CXX)

endif()

target\_include\_directories(uselib\_track PRIVATE ${CMAKE\_CURRENT\_LIST\_DIR}/include)

target\_link\_libraries(uselib\_track PRIVATE ${OpenCV\_LINKED\_COMPONENTS})

if(ENABLE\_ZED\_CAMERA)

target\_link\_libraries(uselib\_track PRIVATE ${ZED\_LIBRARIES})

target\_compile\_definitions(uselib\_track PRIVATE -DZED\_STEREO)

endif()

if(MSVC)

target\_link\_libraries(uselib\_track PRIVATE PThreads\_windows::PThreads\_windows)

target\_compile\_definitions(uselib\_track PRIVATE -D\_CRT\_RAND\_S -DNOMINMAX -D\_USE\_MATH\_DEFINES)

endif()

target\_link\_libraries(uselib\_track PRIVATE Threads::Threads)

endif()

#set\_target\_properties(dark PROPERTIES PUBLIC\_HEADER "${exported\_headers};${CMAKE\_CURRENT\_LIST\_DIR}/include/yolo\_v2\_class.hpp")

set\_target\_properties(dark PROPERTIES PUBLIC\_HEADER "${CMAKE\_CURRENT\_LIST\_DIR}/include/darknet.h;${CMAKE\_CURRENT\_LIST\_DIR}/include/yolo\_v2\_class.hpp")

set\_target\_properties(dark PROPERTIES CXX\_VISIBILITY\_PRESET hidden)

install(TARGETS dark EXPORT DarknetTargets

RUNTIME DESTINATION "${INSTALL\_BIN\_DIR}"

LIBRARY DESTINATION "${INSTALL\_LIB\_DIR}"

ARCHIVE DESTINATION "${INSTALL\_LIB\_DIR}"

PUBLIC\_HEADER DESTINATION "${INSTALL\_INCLUDE\_DIR}"

COMPONENT dev

)

install(TARGETS uselib darknet

DESTINATION "${INSTALL\_BIN\_DIR}"

)

if(OpenCV\_FOUND AND OpenCV\_VERSION VERSION\_GREATER "3.0" AND BUILD\_USELIB\_TRACK)

install(TARGETS uselib\_track

DESTINATION "${INSTALL\_BIN\_DIR}"

)

endif()

install(EXPORT DarknetTargets

FILE DarknetTargets.cmake

NAMESPACE Darknet::

DESTINATION "${INSTALL\_CMAKE\_DIR}"

)

# Export the package for use from the build-tree (this registers the build-tree with a global CMake-registry)

export(PACKAGE Darknet)

# Create the DarknetConfig.cmake

# First of all we compute the relative path between the cmake config file and the include path

file(RELATIVE\_PATH REL\_INCLUDE\_DIR "${INSTALL\_CMAKE\_DIR}" "${INSTALL\_INCLUDE\_DIR}")

set(CONF\_INCLUDE\_DIRS "${PROJECT\_SOURCE\_DIR}" "${PROJECT\_BINARY\_DIR}")

configure\_file(DarknetConfig.cmake.in "${PROJECT\_BINARY\_DIR}/DarknetConfig.cmake" @ONLY)

set(CONF\_INCLUDE\_DIRS "\${Darknet\_CMAKE\_DIR}/${REL\_INCLUDE\_DIR}")

configure\_file(DarknetConfig.cmake.in "${PROJECT\_BINARY\_DIR}${CMAKE\_FILES\_DIRECTORY}/DarknetConfig.cmake" @ONLY)

# Create the DarknetConfigVersion.cmake

include(CMakePackageConfigHelpers)

write\_basic\_package\_version\_file("${PROJECT\_BINARY\_DIR}/DarknetConfigVersion.cmake"

COMPATIBILITY SameMajorVersion

)

install(FILES

"${PROJECT\_BINARY\_DIR}${CMAKE\_FILES\_DIRECTORY}/DarknetConfig.cmake"

"${PROJECT\_BINARY\_DIR}/DarknetConfigVersion.cmake"

DESTINATION "${INSTALL\_CMAKE\_DIR}"

)