Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатики и систем управления

КАФЕДРА Теоретической информатики и компьютерных технологий

ОТЧЕТ ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ

Студент_____
фамилия, имя, отчество

Группа_____

Тип практики _____

Название предприятия _____

Студент _____ _____
подпись, дата *фамилия, и.о.*

Рекомендуемая оценка: _____
Руководитель практики
от предприятия: _____ _____
подпись, дата *фамилия, и.о.*

Руководитель практики _____ _____
подпись, дата *фамилия, и.о.*

Оценка _____

2023 г.

СОДЕРЖАНИЕ

ПОСТАНОВКА ЗАДАЧИ

Целью задания по практике является реализация распознавания объектов с web-камеры при помощи OpenCV, нейросети yolov3_tiny и комплекта программных средств для разработки и реализации глубоких нейронных сетей NeuroMatrix DeepLearling.

Для достижения поставленной цели необходимо решить следующие задачи:

1) изучить нейросеть yolov3_tiny;

2) изучить комплект программных средств для разработки и реализации глубоких нейронных сетей NeuroMatrix DeepLearling;

3) реализовать распознавание объектов с web-камеры при помощи OpenCV, нейросети yolov3_tiny и NeuroMatrix DeepLearling;

4) провести тестирование программы.

1. Изучение нейросети YOLO

Под архитектурой нейронной сети понимается её устройство — последовательность нейронов и связей между ними.

YOLO (You Only Look Once) — архитектура нейронных сетей, предназначенная для детекции объектов на изображении. Отличительной особенностью YOLO является подход к решению задачи детекции.

Один из способов решения задачи детекции заключается в разбиении изображения на квадратные области, затем классификация этих областей на наличие объекта и классификация самого объекта. Таким образом, изображение просматривается дважды (один раз для определения областей, где есть объект, второй — для классификации этого объекта.) Этот способ работает долго и требует больших затрат вычислительных мощностей.

YOLO же использует другой принцип. Исходное изображение сжимается таким образом, чтобы получить квадратную матрицу размером 13 на 13, в каждой клетке которой записана информация о наличии объекта и классе этого объекта на соответствующей части картинки. Таким образом, YOLO просматривает картинку один раз, что существенно увеличивает скорость обработки.

Для того, чтобы научиться работать с нейросетью yolov3_tiny, была изучена информация с сайта [2].

2. Изучение NeuroMatrix DeepLearning

NMDL (NeuroMatrix DeepLearning) – комплект программных средств для разработки и реализации глубоких нейронных сетей, разработанный в компании АО НТЦ «Модуль».

Для того, чтобы научиться работать с NMDL, было изучено руководство [4].

3. Реализация распознавания объектов с web-камеры при помощи OpenCV, нейросети yolov3_tiny и NeuroMatrix DeepLearling

Сначала были подключены заголовочные файлы. "nmdl.h" - описание библиотеки нейросетевой обработки, "nmdl_compiler.h" - описание библиотеки для компиляции моделей, "nmdl_image_converter.h" - описание библиотеки для подготовки изображений. На рисунке 3.1 представлено подключение заголовочных файлов.

```
#include <array>
#include <fstream>
#include <iostream>
#include <string>
#include <vector>
#include "nmdl.h"
#include "nmdl_compiler.h"
#include "nmdl_image_converter.h"
#include "cv_yolo_webcam.hpp"
#include <opencv2/opencv.hpp>
```

Рисунок 3.1 – Подключение заголовочных файлов

Затем была реализована функция-обёртка для вызовов функций библиотеки компиляции моделей. На рисунке 3.2 представлен код данной функции.

```
NMDL_RESULT Call(NMDL_COMPILER_RESULT result, const std::string &function_name) {
    switch(result) {
    case NMDL_COMPILER_RESULT_OK:
        return NMDL_RESULT_OK;
    case NMDL_COMPILER_RESULT_MEMORY_ALLOCATION_ERROR:
        throw std::runtime_error(function_name + ": MEMORY_ALLOCATION_ERROR");
    case NMDL_COMPILER_RESULT_MODEL_LOADING_ERROR:
        throw std::runtime_error(function_name + ": MODEL_LOADING_ERROR");
    case NMDL_COMPILER_RESULT_INVALID_PARAMETER:
        throw std::runtime_error(function_name + ": INVALID_PARAMETER");
    case NMDL_COMPILER_RESULT_INVALID_MODEL:
        throw std::runtime_error(function_name + ": INVALID_MODEL");
    case NMDL_COMPILER_RESULT_UNSUPPORTED_OPERATION:
        throw std::runtime_error(function_name + ": UNSUPPORTED_OPERATION");
    default:
        throw std::runtime_error(function_name + ": UNKNOWN ERROR");
    }
}
```

Рисунок 3.2 – Функция-обёртка для вызовов функций библиотеки компиляции моделей

Также была реализована функция-обёртка для вызовов функций библиотеки нейросетевой обработки. На рисунках 3.3 и 3.4 представлен код данной функции.

```
NMDL_RESULT Call(NMDL_RESULT result, const std::string &function_name) {
    switch(result) {
    case NMDL_RESULT_OK:
        return NMDL_RESULT_OK;
    case NMDL_RESULT_INVALID_FUNC_PARAMETER:
        throw std::runtime_error( arg: function_name + ": INVALID_FUNC_PARAMETER");
    case NMDL_RESULT_NO_LOAD_LIBRARY:
        throw std::runtime_error( arg: function_name + ": NO_LOAD_LIBRARY");
    case NMDL_RESULT_NO_BOARD:
        throw std::runtime_error( arg: function_name + ": NO_BOARD");
    case NMDL_RESULT_BOARD_RESET_ERROR:
        throw std::runtime_error( arg: function_name + ": BOARD_RESET_ERROR");
    case NMDL_RESULT_INIT_CODE_LOADING_ERROR:
        throw std::runtime_error( arg: function_name + ": INIT_CODE_LOADING_ERROR");
    case NMDL_RESULT_CORE_HANDLE_RETRIEVAL_ERROR:
        throw std::runtime_error( arg: function_name + ": CORE_HANDLE_RETRIEVAL_ERROR");
    case NMDL_RESULT_FILE_LOADING_ERROR:
        throw std::runtime_error( arg: function_name + ": FILE_LOADING_ERROR");
    case NMDL_RESULT_MEMORY_WRITE_ERROR:
        throw std::runtime_error( arg: function_name + ": MEMORY_WRITE_ERROR");
    case NMDL_RESULT_MEMORY_READ_ERROR:
        throw std::runtime_error( arg: function_name + ": MEMORY_READ_ERROR");
    case NMDL_RESULT_MEMORY_ALLOCATION_ERROR:
        throw std::runtime_error( arg: function_name + ": MEMORY_ALLOCATION_ERROR");
    case NMDL_RESULT_MODEL_LOADING_ERROR:
        throw std::runtime_error( arg: function_name + ": MODEL_LOADING_ERROR");
```

Рисунок 3.3 – Функция-обёртка для вызовов функций библиотеки нейросетевой обработки

```
    case NMDL_RESULT_INVALID_MODEL:
        throw std::runtime_error( arg: function_name + ": INVALID_MODEL");
    case NMDL_RESULT_BOARD_SYNC_ERROR:
        throw std::runtime_error( arg: function_name + ": BOARD_SYNC_ERROR");
    case NMDL_RESULT_BOARD_MEMORY_ALLOCATION_ERROR:
        throw std::runtime_error( arg: function_name + ": BOARD_MEMORY_ALLOCATION_ERROR");
    case NMDL_RESULT_NN_CREATION_ERROR:
        throw std::runtime_error( arg: function_name + ": NN_CREATION_ERROR");
    case NMDL_RESULT_NN_LOADING_ERROR:
        throw std::runtime_error( arg: function_name + ": NN_LOADING_ERROR");
    case NMDL_RESULT_NN_INFO_RETRIEVAL_ERROR:
        throw std::runtime_error( arg: function_name + ": NN_INFO_RETRIEVAL_ERROR");
    case NMDL_RESULT_MODEL_IS_TOO_BIG:
        throw std::runtime_error( arg: function_name + ": MODEL_IS_TOO_BIG");
    case NMDL_RESULT_NOT_INITIALIZED:
        throw std::runtime_error( arg: function_name + ": NOT_INITIALIZED");
    case NMDL_RESULT_INCOMPLETE:
        throw std::runtime_error( arg: function_name + ": INCOMPLETE");
    case NMDL_RESULT_UNKNOWN_ERROR:
        throw std::runtime_error( arg: function_name + ": UNKNOWN_ERROR");
    default:
        throw std::runtime_error( arg: function_name + ": UNKNOWN ERROR");
    };
}
```

Рисунок 3.4 – Функция-обёртка для вызовов функций библиотеки нейросетевой обработки

Затем была реализована универсальная функция для чтения данных из файла в вектор. На рисунке 3.5 представлен код данной функции.

```
template <typename T>
std::vector<T> ReadFile(const std::string &filename) {
    std::ifstream ifs(filename, std::ios::binary | std::ios::ate);
    if(!ifs.is_open()) {
        throw std::runtime_error("Unable to open input file: " + filename);
    }
    auto fsize = static_cast<std::size_t>(ifs.tellg());
    ifs.seekg(0);
    std::vector<T> data(fsize / sizeof(T));
    ifs.read(reinterpret_cast<char*>(data.data()), data.size() * sizeof(T));
    return data;
}
```

Рисунок 3.5 – Универсальная функция для чтения данных из файла в вектор

Также была реализована функция преобразования матрицы OpenCV в вектор. На рисунке 3.6 представлен код данной функции.

```
std::vector<char> ReadMat(cv::Mat image) {
    std::vector<uchar> buffer;
    cv::imencode(".jpg", image, buffer);
    std::vector<char> buffer1(buffer.begin(), buffer.end());
    return buffer1;
}
```

Рисунок 3.6 – Функция преобразования матрицы OpenCV в вектор

Потом была реализована функция вывода версии NMDL. На рисунке 3.7 представлен код данной функции.

```
void ShowNMDLVersion() {
    std::uint32_t major = 0;
    std::uint32_t minor = 0;
    std::uint32_t patch = 0;
    Call(NMDL_GetLibVersion(&major, &minor, &patch), "GetLibVersion");
    std::cout << "Lib version: " << major << "." << minor
              << "." << patch << std::endl;
}
```

Рисунок 3.7 – Функция вывода версии NMDL

Была реализована функция проверки наличия модуля заданного типа. На

рисунке 3.8 представлен код данной функции.

```
void CheckBoard(std::uint32_t required_board_type) {
    std::uint32_t boards = 0;
    std::uint32_t board_number = -1;
    Call(NMDL_GetBoardCount(required_board_type, &boards), "GetBoardCount");
    std::cout << "Detected boards: " << boards << std::endl;
    if(!boards) {
        throw std::runtime_error("Board not found");
    }
}
```

Рисунок 3.8 – Функция проверки наличия модуля заданного типа

Была реализована функция компиляции исходной модели. На рисунке 3.9 представлен код данной функции.

```
std::vector<float> CompileModel(const std::string &config_filename,
        const std::string &weights_filename,
        std::uint32_t board_type,
        bool is_multi_unit) {
    float *nm_model = nullptr;
    std::uint32_t nm_model_floats = 0u;
    auto config = ReadFile<char>(config_filename);
    auto weights = ReadFile<char>(weights_filename);
    Call(NMDL_COMPILER_CompileDarkNet(is_multi_unit, board_type,
            config.data(), config.size(), weights.data(), weights.size(),
            &nm_model, &nm_model_floats), function_name: "CompileONNX");
    std::vector<float> result(nm_model, last: nm_model + nm_model_floats);
    NMDL_COMPILER_FreeModel(board_type, nm_model);
    return result;
}
```

Рисунок 3.9 – Функция компиляции исходной модели

Была реализована функция получения и вывода информации о параметрах входных и выходных тензоров. На рисунке 3.10 представлен код данной функции.

```cpp
NMDL_ModelInfo GetModelInformation(NMDL_HANDLE nmdl, std::uint32_t unit_num) {
    NMDL_ModelInfo model_info;
    Call(NMDL_GetModelInfo(nmdl, unit_num, &model_info), function_name: "GetModelInfo");
    std::cout << "Input tensor number: " << model_info.input_tensor_num << std::endl;
    for(std::size_t i = 0; i < model_info.input_tensor_num; ++i) {
        std::cout << "Input tensor " << i << ": " <<
            model_info.input_tensors[i].width << ", " <<
            model_info.input_tensors[i].height << ", " <<
            model_info.input_tensors[i].depth <<
            std::endl;
    }
    std::cout << "Output tensor number: " << model_info.output_tensor_num << std::endl;
    for(std::size_t i = 0; i < model_info.output_tensor_num; ++i) {
        std::cout << "Output tensor " << i << ": " <<
            model_info.output_tensors[i].width << ", " <<
            model_info.output_tensors[i].height << ", " <<
            model_info.output_tensors[i].depth <<
            std::endl;
    }
    return model_info;
}
```

Рисунок 3.10 – Функция получения и вывода информации о параметрах входных и выходных тензоров

Затем была реализована функция подготовки кадра. В ней выполняется чтение изображения, его декодирование и предобработка. На рисунке 3.11 представлен код данной функции.

```cpp
std::vector<float> PrepareInput(cv::Mat mat, std::uint32_t width,
        std::uint32_t height, std::uint32_t board_type,
        std::uint32_t color_format, const float rgb_divider[3],
        const float rgb_adder[3]) {
    auto bmp_frame = ReadMat( image: mat);
    std::vector<float> input(NMDL_IMAGE_CONVERTER_RequiredSize(
            width, height, color_format, board_type));
    if(NMDL_IMAGE_CONVERTER_Convert(bmp_frame.data(), input.data(),
            static_cast<std::uint32_t>(bmp_frame.size()), width, height,
            color_format, rgb_divider, rgb_adder, board_type)) {
        throw std::runtime_error("Image conversion error");
    }
    return input;
}
```

Рисунок 3.11 – Функция подготовки кадра

Также была реализована функция ожидания обработки кадра. На рисунке 3.12 представлен код данной функции.

```
void WaitForOutput(NMDL_HANDLE nmdl, std::uint32_t unit_num, float *outputs[]) {
    std::uint32_t status = NMDL_PROCESS_FRAME_STATUS_INCOMPLETE;
    while(status == NMDL_PROCESS_FRAME_STATUS_INCOMPLETE) {
        NMDL_GetStatus(nmdl, unit_num, &status);
    };
    double fps;

    Call(NMDL_GetOutput(nmdl, unit_num, outputs, &fps), function_name: "GetOutput");
    std::cout << "First four result values:" << std::endl;
    for(std::size_t i = 0; i < 4; ++i) {
        std::cout << outputs[0][i] << std::endl;
    }
    std::cout << "FPS:" << fps << std::endl;
}
```

Рисунок 3.12 – Функция ожидания обработки кадра

Затем была реализована главная функция main. В ней запускается бесконечный цикл, в котором происходит получение изображения с web-камеры, преобразование этого изображения, его обработка и получение результата. На рисунках 3.13 – 3.18 представлен код данной функции.

```
int main() {

    const std::uint32_t BOARD_TYPE = NMDL_BOARD_TYPE_SIMULATOR;
    //const uint32_t BOARD_TYPE = NMDL_BOARD_TYPE_MC12705;
    //const uint32_t BOARD_TYPE = NMDL_BOARD_TYPE_NMCARD;
    //const uint32_t BOARD_TYPE = NMDL_BOARD_TYPE_NMMEZZO;
    //const uint32_t BOARD_TYPE = NMDL_BOARD_TYPE_NMQUAD;
    //const uint32_t BOARD_TYPE = NMDL_BOARD_TYPE_MC12101;
    //const uint32_t BOARD_TYPE = NMDL_BOARD_TYPE_NMSTICK;
    //const uint32_t BOARD_TYPE = NMDL_BOARD_TYPE_NMEMBED;
    const std::uint32_t IMAGE_CONVERTER_BOARD_TYPE =
            (BOARD_TYPE == NMDL_BOARD_TYPE_MC12101 ||
             BOARD_TYPE == NMDL_BOARD_TYPE_NMSTICK) ?
            NMDL_IMAGE_CONVERTER_BOARD_TYPE_MC12101 :
            NMDL_IMAGE_CONVERTER_BOARD_TYPE_MC12705;
    //const std::string MODEL_NAME = "yolo_v2_tiny_pascal_voc";
    const std::string MODEL_NAME = "yolo_v3_tiny_coco";
    //const std::string MODEL_NAME = "yolo_v3_coco";
    //const std::string MODEL_NAME = "yolo_v5s_coco";
    const std::string MODEL_FILENAME = "../../../nmdl_ref_data/" + MODEL_NAME + "/model_mu.nm8";

    const NMDL_IMAGE_CONVERTER_COLOR_FORMAT IMAGE_CONVERTER_COLOR_FORMAT =
        NMDL_IMAGE_CONVERTER_COLOR_FORMAT_RGB;
    const float NM_FRAME_RGB_DIVIDER[3] = {255.0f, 255.0f, 255.0f};
    const float NM_FRAME_RGB_ADDER[3] = {0.0f, 0.0f, 0.0f};
```

Рисунок 3.13 – Функция main

```
cv::Mat image;
cv::VideoCapture capture;
capture.open( index: 0);

capture.set( propId: cv::CAP_PROP_FRAME_WIDTH,  value: 640);
capture.set( propId: cv::CAP_PROP_FRAME_HEIGHT,  value: 480);

std::string arr[80] = { "person", "bicycle", "car", "motorbike", "aeroplane", "bus", "train",

for(;;){
    capture>>image;

    if(image.empty())
        break;

    NMDL_HANDLE nmdl = 0;

    try {
        std::cout << "Query library version..." << std::endl;
        ShowNMDLVersion();

        std::cout << "Board detection... " << std::endl;
        CheckBoard(BOARD_TYPE);

        std::cout << "NMDL initialization... " << std::endl;
        Call(NMDL_Create(&nmdl),  function_name: "Create");
```

Рисунок 3.14 – Функция main

```
        std::cout << "Use multi unit... " << std::endl;

        auto model = ReadFile<float>(MODEL_FILENAME);
        std::array<const float*, NMDL_MAX_UNITS> models = {model.data()};
        std::array<std::uint32_t, NMDL_MAX_UNITS> model_floats =
                {static_cast<std::uint32_t>(model.size())};
        Call( result: NMDL_Initialize(nmdl, BOARD_TYPE,  board_number: 0,  proc_number: 0, models.data(),
                              model_floats.data()),  function_name: "Initialize");

        std::cout << "Get model information... " << std::endl;
        auto model_info = GetModelInformation(nmdl,  unit_num: 0);

        std::cout << "Prepare inputs... " << std::endl;
        auto input = PrepareInput( mat: image, model_info.input_tensors[0].width,
                                   model_info.input_tensors[0].height, IMAGE_CONVERTER_BOARD_TYPE,
                                   IMAGE_CONVERTER_COLOR_FORMAT, NM_FRAME_RGB_DIVIDER, NM_FRAME_RGB_ADDER);
        std::array<const float*, 1> inputs = {input.data()};

        std::cout << "Reserve outputs... " << std::endl;
        std::vector<std::vector<float>> output_tensors(model_info.output_tensor_num);
        std::vector<float*> outputs(model_info.output_tensor_num);
        for(std::size_t i = 0; i < model_info.output_tensor_num; ++i) {
            output_tensors[i].resize( new_size: static_cast<std::size_t>(
                                                model_info.output_tensors[i].width) *
                                       model_info.output_tensors[i].height *
                                       model_info.output_tensors[i].depth);
            outputs[i] = output_tensors[i].data();
        }
```

Рисунок 3.15 – Функция main

```
std::cout << "Process input... " << std::endl;
Call( result: NMDL_Process(nmdl, unit_num: 0, inputs.data()), function_name: "Process");
WaitForOutput(nmdl, unit_num: 0, outputs.data());


YoloPostprocessing::Parameters parameters;
parameters.input_width = model_info.input_tensors[0].width;
parameters.input_height = model_info.input_tensors[0].height;
parameters.output_tensors.resize(model_info.output_tensor_num);
for(std::size_t t = 0; t < model_info.output_tensor_num; ++t) {
    parameters.output_tensors[t].width = model_info.output_tensors[t].width;
    parameters.output_tensors[t].height = model_info.output_tensors[t].height;
    parameters.output_tensors[t].depth = model_info.output_tensors[t].depth;
}
if(!MODEL_NAME.compare( s: "yolo_v2_tiny_pascal_voc")) {
    parameters.yolo_version = YoloPostprocessing::Parameters::YOLO_VERSION::YOLO2;
    parameters.classes = 20;
    parameters.anchors = {1.3221, 1.73145, 3.19275, 4.00944, 5.05587, 8.09892, 9.47112, 4.84053, 11.2364, 10.0071};
    parameters.confidence_threshold = 0.675;
    parameters.iou_threshold = 0.45;
}
else if(!MODEL_NAME.compare( s: "yolo_v3_tiny_coco")) {
    parameters.yolo_version = YoloPostprocessing::Parameters::YOLO_VERSION::YOLO3;
    parameters.classes = 80;
    parameters.anchors = {10.0, 14.0, 23.0, 27.0, 37.0, 58.0, 81.0, 82.0, 135.0, 169.0, 344.0, 319.0};
    parameters.confidence_threshold = 0.5;
    parameters.iou_threshold = 0.45;
}
```

Рисунок 3.16 – Функция main

```
else if(!MODEL_NAME.compare( s: "yolo_v3_coco")) {
    parameters.yolo_version = YoloPostprocessing::Parameters::YOLO_VERSION::YOLO3;
    parameters.classes = 80;
    parameters.anchors = {10.0, 13.0, 16.0, 30.0, 33.0, 23.0, 30.0, 61.0, 62.0, 45.0, 59.0, 119.0, 116.0, 90.0, 156.0,
    parameters.confidence_threshold = 0.5;
    parameters.iou_threshold = 0.45;
}
else {
    parameters.yolo_version = YoloPostprocessing::Parameters::YOLO_VERSION::YOLO5;
    parameters.classes = 80;
    parameters.anchors = {1.25, 1.625, 2.0, 3.75, 4.125, 2.875, 1.875, 3.8125, 3.8750, 2.8125, 3.6875, 7.4375, 3.6250,
    parameters.confidence_threshold = 0.25;
    parameters.iou_threshold = 0.45;
}
auto boxes = YoloPostprocessing::GetBoxes(output_tensors, parameters);
std::cout << "Total boxes:" << boxes.size() << std::endl;
for(std::size_t b = 0; b < boxes.size(); ++b) {
    std::cout << "Box " << b << ": " << std::endl;
    std::cout << "\tx = " << boxes[b].x << std::endl;
    std::cout << "\ty = " << boxes[b].y << std::endl;
    std::cout << "\twidth = " << boxes[b].width << std::endl;
    std::cout << "\theight = " << boxes[b].height << std::endl;
    std::cout << "\tconfidence = " << boxes[b].confidence << std::endl;
    std::cout << "\tclass_index = " << boxes[b].class_index << std::endl;
    cv::rectangle( img: image, pt1: cv::Point(boxes[b].x, boxes[b].y), pt2: cv::Point( _x: boxes[b].x + boxes[b].width,
    std::string who = arr[boxes[b].class_index];
    cv::rectangle( img: image, pt1: cv::Point(boxes[b].x, boxes[b].y), pt2: cv::Point( _x: boxes[b].x + boxes[b].width,
    cv::putText( img: image, who, org: cv::Point(boxes[b].x, boxes[b].y), fontFace: cv::FONT_HERSHEY_SIMPLEX, fontScale: 1,
}
```

Рисунок 3.17 – Функция main

Рисунок 3.18 – Функция main

4. Результаты работы

На рисунках 4.1 – 4.4 представлены результаты работы программы.

Рисунок 4.1 – Результаты работы программы

Рисунок 4.2 – Результаты работы программы

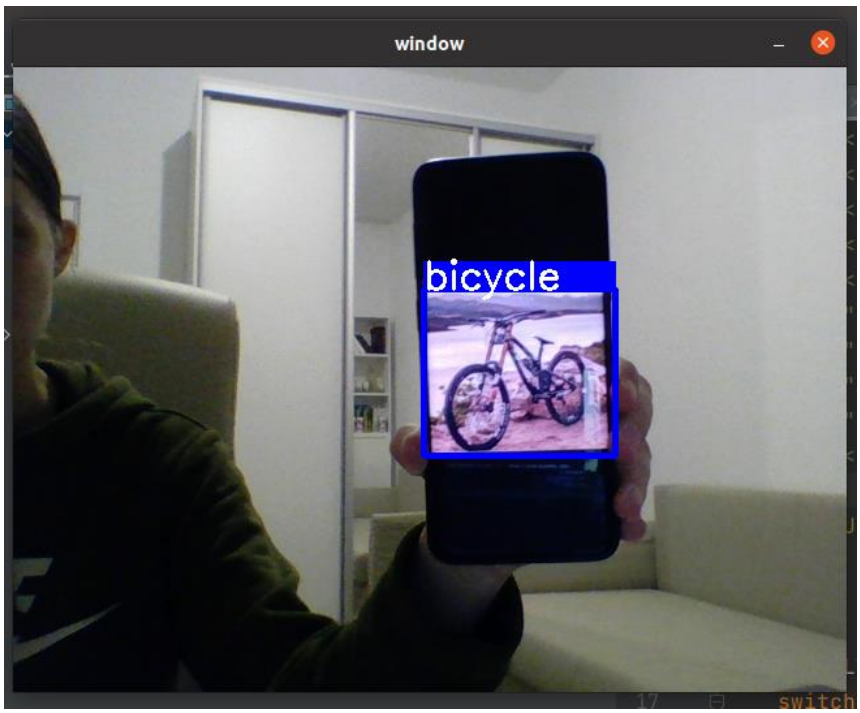

Рисунок 4.3 – Результаты работы программы

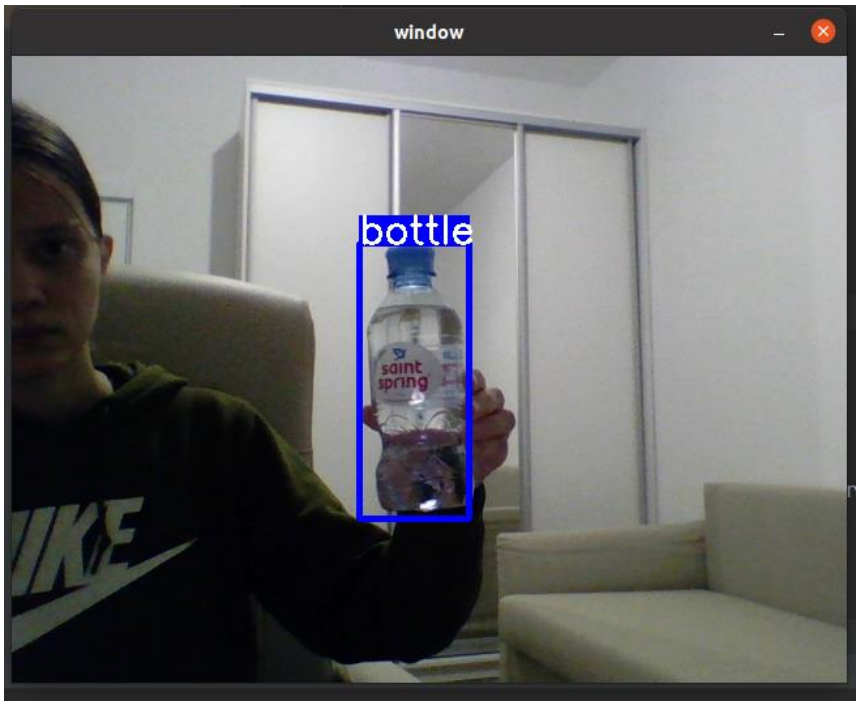

Рисунок 4.4 – Результаты работы программы

ЗАКЛЮЧЕНИЕ

В результате выполнения задания по практике:

1) была изучена нейросеть yolov3_tiny;

2) был изучен комплект программных средств для разработки и реализации глубоких нейронных сетей NeuroMatrix DeepLearling;

3) было реализовано распознавание объектов с web-камеры при помощи OpenCV, нейросети yolov3_tiny и NeuroMatrix DeepLearling;

4) было проведено тестирование программы.

СПИСОК ЛИТЕРАТУРЫ

1. Описание нейросети YOLO: сайт // https://vc.ru/newtechaudit/326571-ispolzovanie-yolov5-dlya-zadachi-detekcii (дата обращения 14.06.23).
2. Описание работы с нейросетью YOLO: сайт // https://pjreddie.com/darknet/yolo/ (дата обращения 14.06.23).
3. Описание NeuroMatrix DeepLearning: сайт // https://pjreddie.com/darknet/yolo/ (дата обращения 17.06.23).
4. Руководство по NeuroMatrix DeepLearning: электронный ресурс // https://pjreddie.com/darknet/yolo/ (дата обращения 17.06.23).