Санкт-Петербургский политехнический университет Петра Великого
Институт металлургии, машиностроения и транспорта
Кафедра мехатроники и робототехники

Курсовая работа

Клиент-серверное приложение для поиска объектов на изображениях

Предмет: Программирование на C и C++

Студент гр. 33335/2 Сафронов Н. А.
Преподаватель Ананьевский М. С.
«____»_____2018г.

Санкт-Петербург
2018

Оглавление

1 Введение

Обработка изображений - одна из важнейших областей машинного обучения. В последние годы в этой области были достигнуты значительные успехи благодаря применению сверточных нейросетей.

Цель данного проекта - продемонстрировать возможности по распознаванию объектов в реальном времени.

2 Постановка задачи

Разработать клиент-серверное приложение для поиска объектов на изображении.

Входные данные: путь к распознаваемому изображению.

Выходные данные:

Для каждого из найденных на изображении классов (если таковые были найдены), необходимо:

- вырезать прямоугольник, содержащий данный класс
- сохранить его в рабочую директорию приложения-клиента
- в имени файла изображения записать, к какому классу оно было отнесено

Необходимо распознавать следующие классы:

- человек
- компьютер
- автомобиль
- мотоцикл

Технические требования:

1. Консольный интерфейс
2. ОС - linux
3. Язык программирования - C или C++
4. Клиент и сервер взаимодействуют через интернет

3 Разработка системы

3.1 Нейронная сеть

Поставленные задачи распознавания изображения - классификация и локализация. Существует несколько архитектур нейронных сетей, решающих данные задачи. Среди них: RCNN, FastRCNN, FasterRCNN, YOLO.

Мой выбор остановился на YOLO, поскольку это одноступенчатая сеть, то есть представляет из себя простой граф вычислений с одним входом и одним выходом. Поскольку для обучения нейросети с нуля требуется высокопроизводительное железо, которого у меня не было, было решено использовать готовую обученную нейросеть.

В качестве движка для нейросети был выбран tensorflow, поскольку это наиболее попуярный фреймворк, и у него есть реализации как на C, так и на python. Python потребовался для того, чтобы разобраться с тем, как устроен open-source проект, из которого я взял нейросеть.

Использованная мной реализация yolo - YOLOv3. Она принимает на вход изображение с высотой и шириной, кратными 32. Для удобства я зафиксировал оба параметра на значении 416 пикселей. С учетом 3 каналов изображения, входной тензор представляет собой 4-х мерный массив: $[1, 416, 416, 3]$.

Нейросеть имеет 3 выхода, каждый для 3 масштабов. Мною был использован только первый выход, предназначенный для больших объектов (примерно от 50 пикселей). Выходной тензор имеет размерность $[1, 13, 13, 255]$. Таким образом, каждая ячейка $32x32$ пикселя исходного изображения превращается в вектор из 255 значений. Он состоит из 3 векторов по 85 значений каждый из которых предсказывает класс, находящийся под этой ячейкой. Из этих 85 значений первые 5 - координаты центра, высота, ширина и объектность - уверенность в том, что данная ячейка содержит объект. Остальные 80 - вероятности того, что объект принадлежит к одному из 80 классов, которые умеет различать сеть. Каждая ячейка содержит 3 вектора по 85 значений, поэтому может распознавать сразу 3 объекта разной формы, находящиеся в одном месте. Для этого используются anchor boxes.

3.2 Клиентская часть

Клиентское приложение имеет консольный интерфейс, включающий help, возможность указать ip адрес и порт сервера, а также путь к файлу с изображением.

Для работы с изображениями была использована библиотека Boost GIL, что значительно ускорило разработку.

Приложение читает файл, создает копию изображения размером 416x416, после чего формирует из него массив uint8_t для передачи по сети. Форма массива $[416, 416, 3]$. Физически реализован ка одномерный массив.

Для передачи данных по сокету TCP была использована библиотека Boost Asio. Сначала передается 2 integer значения высоты и ширины изображения, после чего сам массив.

Получив ответ от сервера, программа десериализует его в массив структур Prediction.

Листинг 1: Структура Prediction

```
1  typedef struct{
2      int t;
3      int b;
4      int l;
5      int r;
6  } Box;
7
8  typedef struct{
9      float objectness;
10     Box box;
11     int class_;
12     double class_probability;
13 } Prediction;
```

После этого для каждого распознанного объекта в консоль выводится отформатированная информация об объекте, а на диск сохраняется вырезанное изображение.

3.3 Серверная часть

На стороне сервера при запуске программы поднимается TCP сервер, а также импортируется граф нейросети и инициализируется сессия tensorflow.

При подключении пользователя сервер получает изображение и прогоняет его через нейронную сеть. На выходе нейросети получается массив float вышеописанной формы $[1, 13, 13, 255]$, которую, тем не менее, удобнее представлять как $[13, 13, 3, 85]$.

Данный массив обрабатывается алгоритмом, который для всех векторов (85), у которых вероятность нахождения объекта выше пороговой, создает экземпляр структуры Prediction со всеми необходимыми данными.

Однако полученный массив предсказаний еще не готов к отправке, поскольку в нем встречаются предсказания, сделанные для одного и того же класса соседними ячейками. Для того, чтобы каждому объекту соответствовало только одно, наиболее точное предсказание, полученный массив фильтруется функцией non_max_suppression, реализующей одноименный алгоритм.

После получения окончательного набора предсказаний, он отправляется клиенту.

4 Результаты

4.1 Быстродействие

На моем компьютере распознавание одного изображения занимает около 5 секунд. В случае, если серверное приложение будет запущено на компьютере с GPU с поддержкой CUDA, время выполнения должно сократиться примерно в 50-100 раз. Преимуществом клиент-серверной системы является то, что серверное приложение может быть запущено на удаленном мощном сервере, в то время как доступ к нему будет возможен почти с любого компьютера, для которого была скомпилирована клиентская часть.

4.2 Пример клиентской сессии

4.3 Примеры распознавания изображений

4.3.1 Изображение 1

Рис. 1: Исходное изображение (из сессии на предыдущем изображении)

Рис. 2: Класс: автомобиль

Рис. 3: Класс: автомобиль

Рис. 4: Класс: человек

Рис. 5: Класс: человек

Рис. 6: Класс: человек

4.3.2 Изображение 2

Рис. 7: Исходное изображение 2

Рис. 8: Класс: автомобиль

Рис. 9: Класс: автомобиль

Рис. 10: Класс: автомобиль

Приложение. Исходный код проекта

4.4 Utils

Листинг 2: config.h

```
#ifndef WEB_NN_SERVICE_CONFIG_H
#define WEB_NN_SERVICE_CONFIG_H

const char DEFAULT_TCP_SERVER_ADRESS[] = "127.0.0.1";
const int DEFAULT_TCP_SERVER_PORT = 20180;

const double IOU_THRESHOLD = 0.3;
const double OBJECTNESS_THRESHOLD = 0.8;

#endif //WEB_NN_SERVICE_CONFIG_H
```

Листинг 3: structures.h

```
#ifndef WEB_NN_SERVICE_STRUCTURES_H
#define WEB_NN_SERVICE_STRUCTURES_H

typedef struct{
int w;
int h;
} Anchor;

const Anchor ANCHORS[] = {
{116,90},
{156,198},
{373,326}
};

typedef struct{
int t;
```

```
17 int b;
18 int l;
19 int r;
20 } Box;
21
22 typedef struct{
23 float objectness;
24 Box box;
25 int class_;
26 double class_probability;
27 } Prediction;
28
29 #endif //WEB_NN_SERVICE_STRUCTURES_H
```

4.5 Tensorflow model

Листинг 4: engine.h

```
1 #ifndef WEB_NN_SERVICE_ENGINE_H
2 #define WEB_NN_SERVICE_ENGINE_H
3
4 #include "tensorflow/c/c_api.h"
5
6 #include "../utils/structures.h"
7
8 class Engine{
9 public:
10 bool init();
11 void predict_416_x_416(float * input_array, Prediction ** preds, i
12 void close();
13
14 private:
15 bool run_session_416_x_416(float * input_array, TF_Tensor ** outpu
```

```
16 void output_tensor_to_predictions(float * output_array, Prediction
17
18 TF_Graph* Graph;
19 TF_Status* Status;
20 TF_ImportGraphDefOptions* Graph_opts;
21 TF_SessionOptions* Sess_opts;
22 TF_Session* Session;
23 };
24
25 #endif //WEB_NN_SERVICE_ENGINE_H
```

Листинг 5: engine.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 #include "tensorflow/c/c_api.h"
5
6 #include <stdio.h>
7 #include <stdlib.h>
8 #include <memory.h>
9 #include <string.h>
10 #include <assert.h>
11 #include <vector>
12 #include <algorithm>
13 #include <iterator>
14
15 #include "engine.h"
16 #include "postprocess_utils.h"
17 #include "../utils/structures.h"
18
19 TF_Buffer* read_file(const char* file);
20
21 void free_buffer(void* data, size_t length) {
```

```
22 free(data);
23 }
24
25 static void Deallocator(void* data, size_t length, void* arg) {
26 free(data);
27 // *reinterpret_cast<bool*>(arg) = true;
28 }
29
30 TF_Buffer* read_file(const char* file) {
31 FILE *f = fopen(file, "rb");
32 fseek(f, 0, SEEK_END);
33 long fsize = ftell(f);
34 fseek(f, 0, SEEK_SET);  //same as rewind(f);
35
36 void* data = malloc(fsize);
37 fread(data, fsize, 1, f);
38 fclose(f);
39
40 TF_Buffer* buf = TF_NewBuffer();
41 buf->data = data;
42 buf->length = fsize;
43 buf->data_deallocator = free_buffer;
44 return buf;
45 }
46
47
48
49 bool Engine::init() {
50 // Use read_file to get graph_def as TF_Buffer*
51 TF_Buffer* graph_def = read_file("/home/safic/CPP/web_nn_service/r
52 Graph = TF_NewGraph();
53
54 //create status to get status of operation
```

```
55 Status = TF_NewStatus();
56
57 // Import graph_def into graph
58 Graph_opts = TF_NewImportGraphDefOptions();
59 TF_GraphImportGraphDef(Graph, graph_def, Graph_opts, Status);
60 if (TF_GetCode(Status) != TF_OK) {
61 fprintf(stderr, "ERROR: Unable to import graph %s", TF_Message(Sta
62 return false;
63 }
64 else {
65 fprintf(stdout, "Successfully imported graph\n");
66 }
67
68
69 Sess_opts = TF_NewSessionOptions();
70 Session = TF_NewSession(Graph, Sess_opts, Status);
71
72 if (TF_GetCode(Status) != TF_OK) {
73 fprintf(stderr, "ERROR: Unable to create session %s", TF_Message(S
74 return false;
75 }
76 else {
77 fprintf(stdout, "Successfully created session\n");
78 }
79 }
80
81 void Engine::close() {
82 TF_DeleteSession(Session, Status);
83 TF_DeleteSessionOptions(Sess_opts);
84
85 TF_DeleteImportGraphDefOptions(Graph_opts);
86 TF_DeleteGraph(Graph);
87 TF_DeleteStatus(Status);
```

```
88 }
89
90 bool Engine::run_session_416_x_416(float * input_array, TF_Tensor
91 // ############################
92 // Set up graph input_tensor
93 // ############################
94
95 //length of array of input data
96 const int input_array_len = 416*416*3;
97 const int num_bytes_in = input_array_len * sizeof(float);
98
99 //input array's shape
100 int64_t in_dims[] = {1,416,416,3};
101 int in_dims_len = 4;
102
103 //get input of the graph
104 TF_Operation* input_op = TF_GraphOperationByName(Graph, "input_1")
105 TF_Output input_operations[] = {input_op, 0};
106
107 // Create the input tensor using the dimension (in_dims) and size
108 // variables created earlier
109 TF_Tensor* input_tensor = TF_NewTensor(TF_FLOAT, in_dims, in_dims_
110 TF_Tensor ** input_tensors = {&input_tensor};
111
112 // ############################
113 // Set up graph output_tensor
114 // ############################
115
116 TF_Operation* output_op = TF_GraphOperationByName(Graph, "conv2d_5
117 TF_Output output_operations[] = {output_op, 0};
118
119 TF_Tensor ** output_tensors = {output_tensor};
120
```

```
121 // ##########################
122 // Run graph
123 // ##########################
124
125 TF_SessionRun(Session, nullptr,
126 // Inputs
127 input_operations, input_tensors, 1,
128 // Outputs
129 output_operations, output_tensors, 1,
130 // Target operations
131 nullptr, 0, nullptr,
132 Status);
133
134 fprintf(stdout, "Session Run Status: %d - %s\n", TF_GetCode(Status
135
136 if(TF_GetCode(Status) == TF_OK) {
137 fprintf(stdout, "Successfully run session\n");
138 return true;
139 }else{
140 fprintf(stderr, "Run session with error\n");
141 return false;
142 }
143 }
144
145 void Engine::output_tensor_to_predictions(float *output_array, Pre
146 fprintf(stdout, "Start: output_tensor_to_predictions\n");
147
148 Prediction *preds_raw = nullptr;
149 int preds_raw_len = 0;
150 extract_predictions_by_objectness(output_array, &preds_raw, &preds
151
152 fprintf(stdout, "Raw predictions len: %d\n", preds_raw_len);
153
```

```
154 non_max_supress(preds_raw, preds_raw_len, preds, preds_len);
155 //*preds = preds_raw;
156 //*preds_len = preds_raw_len;
157 if (preds_raw != nullptr)
158 delete[] preds_raw;
159 }
160
161 void Engine::predict_416_x_416(float *input_array, Prediction ** p
162 TF_Tensor * output_tensor = nullptr;
163
164 if (!run_session_416_x_416(input_array, &output_tensor)){
165 *preds = nullptr;
166 *preds_len = 0;
167 return;
168 }
169
170 float *output_array = static_cast<float*>(TF_TensorData(output_ten
171
172 output_tensor_to_predictions(output_array, preds, preds_len);
173
174 TF_DeleteTensor(output_tensor);
175 }
```

Листинг 6: postprocess_utils.h

```
1 #ifndef WEB_NN_SERVICE_POSTPROCESS_UTILS_H
2 #define WEB_NN_SERVICE_POSTPROCESS_UTILS_H
3
4 #include "../utils/config.h"
5 #include "../utils/structures.h"
6
7 double sigmoid(double x);
8
9 void non_max_supress(Prediction * preds, int preds_len, Prediction
```

```
10 void extract_predictions_by_objectness(float *output_array, Predic
11
12 #endif //WEB_NN_SERVICE_POSTPROCESS_UTILS_H
```

Листинг 7: postprocess_utils.cpp

```
1  #include <math.h>
2  #include <algorithm>
3
4  #include "postprocess_utils.h"
5  #include "../utils/config.h"
6
7
8  void sort( int * arr, int size )
9  {
10 int i = 1;
11 int tmp;
12 while(i < size){
13 if(i == 0 || arr[i-1] <= arr[i]){
14 i++;
15 } else{
16 tmp = arr[i-1];
17 arr[i-1] = arr[i];
18 arr[i] = tmp;
19 i--;
20 }
21 }
22 }
23
24 int box_intersection(Box box1, Box box2){
25 if (box1.l >= box2.r || box2.l >= box1.r || box1.t >= box2.b || bo
26 return 0;
27 }
28
```

```
29 int xx[] = {box1.l, box1.r, box2.l, box2.r};
30 sort(xx, 4);
31
32 int yy[] = {box1.t, box1.b, box2.t, box2.b};
33 sort(yy, 4);
34
35 int intersection = (xx[2] - xx[1]) * (yy[2] - yy[1]);
36
37 return intersection;
38 }
39
40 double IOU(Box box1, Box box2){
41 int intersection = box_intersection(box1,box2);
42
43 int box1_area = (box1.b - box1.t) * (box1.r - box1.l);
44 int box2_area = (box2.b - box2.t) * (box2.r - box2.l);
45 int union_ = box1_area + box2_area - intersection;
46
47 double intersection_over_union = intersection / (union_ + 0.00001)
48
49 return intersection_over_union;
50 }
51
52
53 void non_max_supress(Prediction * preds, int preds_len, Prediction
54 bool * is_supressed = new bool[preds_len];
55
56 for (int i = 0; i < preds_len; i++){
57 is_supressed[i] = false;
58 }
59
60 for (int i = 0; i < preds_len; i++){
61 for (int j = i+1; j < preds_len; j++){
```

```
62 if(is_supressed[i] || is_supressed[j])
63 continue;
64
65 if (preds[i].class_ != preds[j].class_)
66 continue;
67
68 double iou = IOU(preds[i].box, preds[j].box);
69 if (iou < IOU_THRESHOLD)
70 continue;
71
72 if (preds[i].class_probability < preds[j].class_probability){
73 is_supressed[i] = true;
74 }else{
75 is_supressed[j] = true;
76 }
77 }
78 }
79
80 int not_supressed_len_tmp = 0;
81 for (int i = 0; i < preds_len; i++){
82 if (!is_supressed[i])
83 not_supressed_len_tmp += 1;
84 }
85
86 Prediction * not_supressed_tmp = new Prediction[not_supressed_len_
87 *not_supressed = not_supressed_tmp;
88 *not_supressed_len = not_supressed_len_tmp;
89
90 int c = 0;
91 for (int i = 0; i < preds_len; i++){
92 if (!is_supressed[i]){
93 not_supressed_tmp[c] = preds[i];
94 c++;
```

```
95 }
96 }
97
98 delete[] is_supressed;
99 }
100
101 double sigmoid(double x){
102 return 1.0 / (1.0 + exp(-x));
103 }
104
105 void extract_predictions_by_objectness(float *output_array, Predic
106 const int array_h = 13; //height
107 const int array_w = 13; //width
108 const int array_c = 3; //channels
109 const int preds_max_len = array_h*array_w*array_c;
110 const int one_pred_len = 85;
111
112 const int image_h = 416;
113 const int image_w = 416;
114
115 bool *is_object = new bool[preds_max_len];
116
117 for (int i = 0; i < array_h; i++){
118 for (int j = 0; j < array_w; j++){
119 for (int k = 0; k < array_c; k++){
120 float *this_pred_raw = output_array + i*array_w*array_c*one_pred_l
121 j*array_c*one_pred_len +
122 k*one_pred_len;
123
124 float objectness_raw = this_pred_raw[4];
125 double objectness = sigmoid(objectness_raw);
126 if (objectness > OBJECTNESS_THRESHOLD){
127 is_object[i*array_w*array_c + j*array_c + k] = true;
```

```
128 }else{
129 is_object[i*array_w*array_c + j*array_c + k] = false;
130 }
131 }
132 }
133 }
134
135 int preds_len_tmp = 0;
136 for (int i = 0; i < preds_max_len; i++){
137 if (is_object[i])
138 preds_len_tmp += 1;
139 }
140
141 Prediction * preds_tmp = new Prediction[preds_len_tmp];
142 *preds = preds_tmp;
143 *preds_len = preds_len_tmp;
144
145 int c = 0;
146 for (int i = 0; i < array_h; i++){
147 for (int j = 0; j < array_w; j++){
148 for (int k = 0; k < array_c; k++){
149 if (!is_object[i*array_w*array_c + j*array_c + k])
150 continue;
151
152 float *this_pred_raw = output_array + i*array_w*array_c*one_pred_l
153 j*array_c*one_pred_len +
154 k*one_pred_len;
155
156 float tx = this_pred_raw[0];
157 float ty = this_pred_raw[1];
158 float tw = this_pred_raw[2];
159 float th = this_pred_raw[3];
160
```

```
161 double bx = j + sigmoid(tx);
162 double by = i + sigmoid(ty);
163 double bw = ANCHORS[k].w * exp(tw);
164 double bh = ANCHORS[k].h * exp(th);
165
166 int t = image_h/array_h * by - bh/2;
167 int b = image_h/array_h * by + bh/2;
168 int l = image_w/array_w * bx - bw/2;
169 int r = image_w/array_w * bx + bw/2;
170
171 preds_tmp[c].box.t = std::max(0, t);
172 preds_tmp[c].box.b = std::min(image_h, b);
173 preds_tmp[c].box.l = std::max(0, l);
174 preds_tmp[c].box.r = std::min(image_w, r);
175
176 preds_tmp[c].objectness = sigmoid(this_pred_raw[4]);
177
178 int max_class = 0;
179 float max_class_probability = -INFINITY;
180 for (int i = 0; i < one_pred_len-5; i++){
181 int this_class_probability = this_pred_raw[5+i];
182 if (this_class_probability > max_class_probability){
183 max_class_probability = this_class_probability;
184 max_class = i;
185 }
186 }
187
188 preds_tmp[c].class_ = max_class;
189 preds_tmp[c].class_probability = sigmoid(this_pred_raw[5+max_class
190
191 c++;
192 }
193 }
```

```
194 }
195
196 delete[] is_object;
197 }
```

4.6 Server

Листинг 8: tcp_server.cpp

```
1  #include <ctime>
2  #include <string>
3  #include <iostream>
4
5  #include <boost/bind.hpp>
6  #include <boost/asio.hpp>
7  #include <boost/thread.hpp>
8  #include <boost/format.hpp>
9  #include <boost/smart_ptr.hpp>
10
11 #include "../utils/config.h"
12 #include "../tf_model/engine.h"
13
14 using boost::asio::ip::tcp;
15 using namespace std;
16
17 Engine engine;
18
19 class client_session
20 {
21 public:
22 typedef boost::shared_ptr<client_session> pointer;
23
24 static pointer create(boost::asio::io_service &io) {
```

```
25 return pointer(new client_session(io));
26 }
27
28 ~client_session() {
29 log("Connection closed");
30 }
31
32 tcp::socket &socket() { return socket_; }
33
34 void start() {
35 log("Connection established");
36
37 int image_shape[2]; //shape is (width, height)
38 boost::system::error_code error;
39 socket_.read_some(boost::asio::buffer(image_shape, 2* sizeof(int))
40
41 long array_size = image_shape[0] * image_shape[1] * 3;
42 uint8_t *array = new uint8_t[array_size];
43
44 //boost::this_thread::sleep(boost::posix_time::milliseconds(1000))
45
46 uint8_t *data_start = array;
47 long already_read = 0;
48 long total_to_read = array_size;
49 long left_to_read = total_to_read;
50 long len = 0;
51 while(already_read < total_to_read){
52 len = socket_.read_some(boost::asio::buffer(data_start, left_to_re
53 //cout << "read " << len << " bytes" << endl;
54 already_read += len;
55 left_to_read -= len;
56 data_start += len;
57 }
```

```
58 //cout << endl;
59
60 fprintf(stdout, "Read bytes: %ld of %ld", array_size, already_read
61
62 if (image_shape[0] != 416 || image_shape[1] != 416){
63 int recog[] = {0};
64 boost::asio::write(socket_, boost::asio::buffer(recog));
65 delete[] array;
66 return;
67 }
68
69 Prediction * preds = nullptr;
70 int preds_len = 0;
71
72 float * array_float = new float[array_size];
73 for (int i = 0; i < array_size; i++){
74 array_float[i] = array[i] / 255.0;
75 }
76
77 engine.predict_416_x_416(array_float, &preds, &preds_len);
78
79 fprintf(stdout, "Prediction results: preds_len = %d\n", preds_len)
80 /*
81 for (size_t i = 0; i < preds_len; i++){
82 fprintf(stdout, "[\n");
83 for (size_t j = 0; j < sizeof(Prediction); j++){
84 char *this_char = (char*)(preds + i*sizeof(Prediction) + j);
85 fprintf(stdout, "%c, ",*this_char);
86 }
87 fprintf(stdout, "]\n");
88 }
89 */
90
```

```
91  boost::asio::write(socket_, boost::asio::buffer({0}, sizeof(char))
92  boost::asio::write(socket_, boost::asio::buffer(preds, preds_len*s
93
94  delete[] array;
95  //no need to "delete[] array_float", because it will be freed insi
96  }
97
98  protected:
99  client_session(boost::asio::io_service &io)
100 : socket_(io) {
101 }
102
103 void log(std::string const &message) {
104 std::clog << boost::format("%|-25| [client address: %|15|]\n")
105 % message % socket_.remote_endpoint().address().to_string();
106 }
107
108 private:
109 tcp::socket socket_;
110 };
111
112 int main()
113 {
114 engine = Engine();
115 engine.init();
116
117 boost::asio::io_service io;
118
119 tcp::acceptor acceptor(io, tcp::endpoint(tcp::v4(), DEFAULT_TCP_SE
120 for (;;) {
121 client_session::pointer new_client = client_session::create(io);
122 acceptor.accept(new_client->socket());
123 boost::thread(boost::bind(&client_session::start, new_client)).det
```

```
124 }
125 }
```

4.7 Client

Листинг 9: tcp_client.cpp

```
1  #include <iostream>
2  #include <fstream>
3  #include <cstring> //atoi
4
5  #include <boost/bind.hpp>
6  #include <boost/asio.hpp>
7  #include <boost/thread.hpp>
8  #include <boost/array.hpp>
9
10 #include <boost/gil.hpp>
11 #include <boost/gil/io/io.hpp>
12 #include <boost/gil/extension/io/jpeg.hpp>
13 #include <boost/gil/extension/numeric/sampler.hpp> //gil::bilinear_
14 #include <boost/gil/extension/numeric/resample.hpp>  //gil::resize_
15
16 #include "../utils/structures.h"
17 #include "../utils/config.h"
18
19 using namespace boost::asio;
20 using namespace boost::system;
21 namespace gil = boost::gil;
22 using namespace std;
23
24 std::string class_idx_to_name(int class_idx){
25 if (class_idx == 0)
26 return std::string("person");
```

```
27 if (class_idx == 2)
28 return std::string("car");
29 if (class_idx == 3)
30 return std::string("motorbike");
31 if (class_idx == 63)
32 return std::string("laptop");
33
34 return std::string("object");
35 }
36
37 bool image_to_array(gil::rgb8_image_t::const_view_t img, uint8_t *
38
39 uint32_t width_tmp = img.width();
40 uint32_t height_tmp = img.height();
41 uint8_t *array_tmp = new uint8_t[width_tmp*height_tmp*3];
42
43 *width = width_tmp;
44 *height = height_tmp;
45 *array = array_tmp;
46
47 uint32_t width_x3 = width_tmp * 3;
48
49 for(int i = 0; i < height_tmp; i++){
50 for(int j = 0; j < width_tmp; j++){
51 gil::rgb8_pixel_t px = *img.at(j, i);
52 for(int k = 0; k < 3; k++){
53 array_tmp[i * width_x3 + j * 3 + k] = (uint8_t)px[k];
54 }
55 }
56 }
57
58 return true;
59 }
```

```
60
61 void resize_box(Box *box, float *from_h_w_to_h_w){
62 float scale_h = from_h_w_to_h_w[2] / from_h_w_to_h_w[0];
63 float scale_w = from_h_w_to_h_w[3] / from_h_w_to_h_w[1];
64
65 //fprintf(stdout, "scale_h %.2f, scale_w %.2f", scale_h, scale_w);
66
67 box->l *= scale_w;
68 box->r *= scale_w;
69 box->t *= scale_h;
70 box->b *= scale_h;
71 }
72
73 void crop_image_and_save(gil::rgb8_image_t img, Prediction pred, i
74 gil::image_write_info<gil::jpeg_tag> write_settings;
75
76 //crop image
77 Box box = pred.box;
78 gil::rgb8_image_t::view_t sv = boost::gil::subimage_view(gil::view
79
80 //generate filename
81 char filename[100];
82 sprintf(filename, "%s_conf_%.2f_%d.jpg", class_idx_to_name(pred.cl
83
84 //write to file
85 gil::write_view(filename, sv, write_settings);
86 }
87
88 void log_class_found(Prediction pred, int image_idx)
89 {
90 fprintf(stdout, "%d. {objectness: %.2f, class_idx: %2d, \n", image
91 fprintf(stdout, "class_name: %s,\n", class_idx_to_name(pred.class_
92 fprintf(stdout, "class_confidence: %.2f,\n", pred.class_probabilit
```

```
93
94 Box box = pred.box;
95 fprintf(stdout, "size: {height:%d, width:%d},\n", box.b-box.t, box
96 fprintf(stdout, "coordinates: {x:[%d, %d], y:[%d, %d]},\n", box.l,
97 fprintf(stdout, "}\n");
98 }
99
100 void print_help()
101 {
102 fprintf(stdout, "Usage:\n");
103 fprintf(stdout, "tcp_CLIENT <path to image> [-p <port number> -a <
104 fprintf(stdout, "Example:\n");
105 fprintf(stdout, "tcp_CLIENT car.jpg -a 192.168.0.1 -p 1024\n");
106 fprintf(stdout, "\n");
107 fprintf(stdout, "-p --port - default is 20180\n");
108 fprintf(stdout, "\n");
109 fprintf(stdout, "-a --address - default is 127.0.0.1\n");
110 fprintf(stdout, "\n");
111 fprintf(stdout, "Supported image formats are: .jpg, .jpeg\n");
112 }
113
114 static inline bool is_file_exist(const char *fileName)
115 {
116 std::ifstream infile(fileName);
117 return infile.good();
118 }
119
120 int main(int argc, char* argv[])
121 {
122 if(argc < 2){
123 print_help();
124 return 1;
125 }
```

```
126
127 //parse "--key" args, set address and port
128 int port = DEFAULT_TCP_SERVER_PORT;
129 const char *address = DEFAULT_TCP_SERVER_ADRESS;
130 for (int i = 0; i < argc; i++){
131 if(std::strncmp(argv[i], "-p", 2) == 0 || std::strncmp(argv[i], "-
132 i++;
133 port = atoi(argv[i]);
134 }
135 if(std::strncmp(argv[i], "-a", 2) == 0 || std::strncmp(argv[i], "-
136 i++;
137 address = argv[i];
138 }
139 if(std::strncmp(argv[i], "-h", 2) == 0 || std::strncmp(argv[i], "-
140 print_help();
141 return 0;
142 }
143 }
144 ip::tcp::endpoint ep( ip::address::from_string(address), port);
145
146 const char *image_path = argv[1];
147 if( !is_file_exist(image_path) ){
148 fprintf(stdout, "No such file or directory: %s\n", argv[1]);
149 return 2;
150 }
151
152 //read image
153 gil::rgb8_image_t img;
154 std::string filename( image_path );
155 gil::image_read_settings<gil::jpeg_tag> read_settings;
156 gil::read_image( filename, img, read_settings);
157 fprintf(stdout, "Image loaded. height = %d, width = %d\n", img.hei
158
```

```
159 //resize image
160 boost::gil::rgb8_image_t img_sized(416, 416);
161 gil::resize_view(gil::const_view(img), gil::view(img_sized), boost
162
163 //serialize image
164 gil::rgb8_image_t::const_view_t cv = gil::const_view(img_sized);
165 uint8_t *array = nullptr;
166 int height = 0;
167 int width = 0;
168 image_to_array(cv, &array, &height, &width);
169 long array_len = height*width*3;
170
171 //establish tcp connection
172 boost::asio::io_service io;
173 ip::tcp::socket sock( io);
174 sock.connect(ep);
175
176 fprintf(stdout, "Connection established with %s:%d\n", address, po
177
178 //write image as bytes
179 boost::asio::write(sock, buffer({height, width}, 2 * sizeof(int)))
180 boost::asio::write(sock, buffer(array, array_len * sizeof(uint8_t)
181
182 fprintf(stdout, "Data sent\n");
183
184 //receive answer from server
185 const int buf_len = 1024;
186 char *buf = new char[buf_len];
187 boost::system::error_code error;
188 size_t response_len_bytes = sock.read_some(boost::asio::buffer(buf
189
190 fprintf(stdout, "Received bytes: %zd\n", response_len_bytes);
191
```

```
192 //check answer for validity
193 if(response_len_bytes < 1){
194 fprintf(stdout, "Error: wrong response from server\n");
195 sock.close();
196 return 1;
197 }
198
199 char error_code = buf[0];
200
201 if (error_code != 0){
202 fprintf(stdout, "Error on server side. Error code: %d\n", error_co
203 sock.close();
204 return 1;
205 }
206
207 //parse predictions and save to files
208 Prediction *preds = (Prediction *)(buf + sizeof(char));
209 int preds_len = (response_len_bytes-1)/sizeof(Prediction);
210
211 for(int i = 0; i < preds_len; i++){
212 Prediction pred = preds[i];
213
214 float from_h_w_to_h_w[] = {416.0, 416.0, (float)img.height(), (flo
215 resize_box(&pred.box, from_h_w_to_h_w);
216
217 crop_image_and_save(img, pred, i);
218 log_class_found(pred, i);
219 }
220
221 //release resources
222 delete [] buf;
223
224 sock.close();
```

```
225 }
```

4.8 CMakeLists

Листинг 10: CMakeLists.txt

```
1  cmake_minimum_required(VERSION 3.12)
2  project(web_nn_service)
3
4  set(CMAKE_CXX_STANDARD 11) #do not change!
5
6  find_package(Boost REQUIRED COMPONENTS system thread)
7
8  add_executable(tcp_SERVER server/tcp_server.cpp
9  tf_model/engine.cpp tf_model/engine.h
10 tf_model/postprocess_utils.cpp tf_model/postprocess_utils.h
11 utils/structures.h utils/config.h)
12 include_directories(tcp_SERVER ${Boost_INCLUDE_DIRS})
13 link_directories(tcp_SERVER ${Boost_LIBRARY_DIRS})
14 target_link_libraries(tcp_SERVER ${Boost_LIBRARIES} -lpthread
15 -ljpeg -ltensorflow)
16
17
18 add_executable(tcp_CLIENT client/tcp_client.cpp
19 utils/structures.h)
20 include_directories(tcp_CLIENT ${Boost_INCLUDE_DIRS})
21 link_directories(tcp_CLIENT ${Boost_LIBRARY_DIRS})
22 target_link_libraries(tcp_CLIENT ${Boost_LIBRARIES} -lpthread
23 -ljpeg)
```

Список литературы

1. Linux kernel coding style https://www.kernel.org/doc/html/v4.10/process/coding-style.html
2. Документация к boost https://www.boost.org/
3. Установка tensorflow для C https://www.tensorflow.org/install/lang_c
4. Код C API библиотеки tensorflow https://github.com/tensorflow/tensorflow/blob/master/tensorflow/c/c_api.h
5. Рабочий код для запуска сессии tensorflow в C

   https://stackoverflow.com/questions/44305647/segmentation-fault-when-using-tf-sessionrun-to-run-tensorflow-graph-in-c-not-c
6. Репозиторий, откуда позаимствована нейронная сеть https://github.com/qqwweee/keras-yolo3
7. Объяснение того, как интерпретировать выходы Yolo v3

   https://blog.paperspace.com/how-to-implement-a-yolo-object-detector-in-pytorch/