## ПРИЛОЖЕНИЕ А

(справочное)

**Результаты расчёта работы алгоритмов сегментации**

Таблица А.1 – Время исполнения в секундах алгоритмов компьютерного зрения при разных размерностях обрабатываемого изображения

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Алгоритм / Размер изображения, пиксели | 1000 | 510 | 360 | 310 | 220 |
| *ResNet*50 | 31.7255 | 8.7604 | 4.5548 | 3.3633 | 1.7663 |
| *MobileNet* | 10.2384 | 2.6914 | 1.4496 | 1.0121 | 0.65571 |
| Без нейронной сети | 0.0929 | 0.0259 | 0.0169 | 0.0139 | 0.00994 |

Таблица А.2 – Качество сегментации алгоритмов компьютерного зрения при разных размерностях обрабатываемого изображения

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Метрики / Размер изображения, пиксели | | 1000 | | 510 | | 360 | 310 | 220 |
| *ResNet*50 | | | | | | | | |
| mloU | | 0,8849 | | 0,8046 | | 0,8042 | 0,7718 | 0,7142 |
| Pixel Accuracy | | 0,9177 | | 0,8414 | | 0,8442 | 0,8265 | 0,7796 |
| Общий балл | | 0,9013 | | 0,823 | | 0,8242 | 0,7992 | 0,7469 |
| MobileNet | | | | | | | | |
| mloU | 0,8345 | | 0,8054 | | 0,7693 | | 0,7429 | 0,6645 |
| Pixel Accuracy | 0,8916 | | 0,94 | | 0,9913 | | 0,8873 | 0,7847 |
| Общий балл | 0,8631 | | 0,8727 | | 0,8803 | | 0,8151 | 0,7246 |
| Без нейронной сети | | | | | | | | |
| mloU | 0,3435 | | 0,3533 | | 0,3335 | | 0,3352 | 0,3115 |
| Pixel Accuracy | 0,3854 | | 0,4348 | | 0,4378 | | 0,4641 | 0,4884 |
| Общий балл | 0,3645 | | 0,3941 | | 0,3857 | | 0,3997 | 0,3999 |

**ПРИЛОЖЕНИЕ Б**

(справочное)

**Результаты расчета экономического обоснования**

Таблица Б.1 – Расчет коэффициента эквивалентности

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Наименование параметра | Вес параметра, β | Значение параметра | | |  |  |  |  |
|  |  |  |
| Точность результирующих данных | 0.4 | 45 | 54 | 70 | 0.64 | 0.77 | 0.26 | 0.31 |
| Время обработки данных | 0.4 | 1.44 | 1.01 | 0.08 | 0.06 | 0.08 | 0.02 | 0.03 |
| Функциональные возможности | 0.2 | 1 | 2 | 4 | 0.25 | 0.5 | 0.05 | 0.1 |
| Коэффициент эквивалентности | | | | | | | 0.44/0.33 = 1.33 | |

Таблица Б.2 – Расчет коэффициента изменения функциональных возможностей

|  |  |  |
| --- | --- | --- |
| Наименование показателя | Оценка базового продукта | Оценка нового продукта |
| Функциональные возможности | 2 | 4 |
| Быстродействие | 3 | 4 |
| Удобство пользовательского интерфейса | 4 | 4 |
| Объём памяти | 3 | 5 |
| Коэффициент функциональных возможностей | 17/12 = 1.42 | |

Таблица Б.3 – Расчет уровня конкурентоспособности нового ПП

|  |  |
| --- | --- |
| Коэффициенты | Значение |
| Коэффициент эквивалентности () | 1.33 |
| Коэффициент изменения функциональных возможностей () | 1.42 |
| Коэффициент соответствия нормативам (Кн) | 1 |
| Коэффициент цены потребления () | 1 |
| Интегральный коэффициент конкурентоспособности | (1.33\*1.42\*1)/1=1.89 |

Таблица Б.4 – Исходный и уточнённый объём строк исходного кода

|  |  |  |  |
| --- | --- | --- | --- |
| Код функций | Наименование (содержание) функций | Объем функции строк исходного кода (*LOC*) | |
| по каталогу (*V*о) | уточненный (*V*y) |
| 102 | Контроль, предварительная обработка и ввод информации | 490 | 186 |
| 108 | Организация ввода-вывода информации с сети терминалов | 280 | 143 |
| 304 | Управление файлами | 5240 | 44 |
| 502 | Монитор системы (управление работой комплекса ПО) | 5760 | 623 |
| 506 | Обработка ошибочных сбойных ситуаций | 1540 | 96 |
| 701 | Математическая статистика и прогнозирование | 3780 | 750 |
| 707 | Графический вывод результатов | 420 | 173 |
| Итого: | | 17510 | 2015 |

Таблица Б.5 – Значения коэффициентов удельных весов трудоемкости стадий разработки ПО в общей трудоемкости

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Категория новизны ПО | Без применения CASE-технологий | | | | |
| Стадии разработки ПО | | | | |
| ТЗ | ЭП | ТП | РП | ВН |
| Значения коэффициентов | | | | |
|  |  |  |  |  |
| В | 0,08 | 0,19 | 0,28 | 0,34 | 0,11 |
|  | Скорректированные значения коэффициентов | | | | |
| В | 4,54 | 10,78 | 15,88 | 17,36 | 6,24 |

Таблица Б.6 – Расчет общей трудоемкости разработки ПО

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Показатели | Стадии разработки | | | | | Итого |
| ТЗ | ЭП | ТП | РП | ВН |
| Общий объем ПО , кол-во строк (*LOC*) | – | – | – | – | – | 17510 |
| Общий уточненный объем ПО , кол-во строк (*LOC*) | – | – | – | – | – | 2015 |

Продолжение таблицы Б.6

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Категория сложности разрабатываемого ПО | – | – | – | – | – | 1 |
| Нормативная трудоемкость разработки ПО  , чел.-дн. | – | – | – | – | – | 134 |
| Коэффициент повышения сложности ПО | 1,12 | 1,12 | 1,12 | 1,12 | 1,12 | – |
| Коэффициент, учитывающий новизну ПО | 0,63 | 0, 63 | 0, 63 | 0, 63 | 0, 63 | – |
| Коэффициент, учитывающий степень использования стандартных модулей () | – | – | – | 0,9 | – | – |
| Коэффициент, учитывающий средства разработки ПО | 0,6 | 0,6 | 0,6 | 0,6 | 0,6 | – |
| Коэффициенты удельных весов трудоемкости стадий разработки ПО | 0,08 | 0,19 | 0,28 | 0,34 | 0,11 | 1 |
| Распределение скорректированной (с учетом , , ) трудоемкости ПО по стадиям, чел.-дн. | 4,54 | 10,78 | 15,88 | 17.36 | 6,24 | – |
| Общая трудоемкость разработки ПО , чел.-дн. | – | – | – | – | – | 54,8 |

Таблица Б.7 – Параметры для расчета производственных затрат на разработку

|  |  |  |
| --- | --- | --- |
| Параметр | Единица измерения | Значение |
| Базовая ставка специалиста | бел. руб. | 195 |
| Разряд разработчика | – | 11 |
| Тарифный коэффициент | – | 1,21 |

Продолжение таблицы Б.7

|  |  |  |
| --- | --- | --- |
| Коэффициент *К*ув | – | 1,6 |
| Норматив отчислений на доп. Зарплату разработчиков (Ндоп) | % | 20 |
| Численность обслуживающего персонала | чел. | 1 |
| Разряд обслуживающего персонала | – | 8 |
| Тарифный коэффициент | – | 2,17 |
| Стоимость одного кВт-часа электроэнергии (СЭЛ) | бел. руб. | 0,390852 |
| Коэффициент потерь рабочего времени (Кпот) | – | 0,2 |
| Премия | % | 5 |
| Доплата за стаж | бел. руб. | 19,5 |

Таблица Б.8 – Расчет суммарных затрат на разработку ПО, руб

|  |  |
| --- | --- |
| Статья затрат | Итого |
| Затраты на оплату труда разработчиков (Зтр) | 225,588 |
| Основная заработная плата разработчиков (ЗПосн) | 140,288 |
| Дополнительная заработная плата разработчиков (ЗПдоп) | 28,06 |
| Отчисления от основной и дополнительной заработной платы (ОТЧс.н.) | 57,24 |
| Затраты машинного времени (Змв) | 256,768 |
| Стоимость машино-часа, руб./час (Сч) | 1,36 |
| Стоимость электроэнергии, потребляемой за год (Зэ.п.) | 254,25 |
| Затраты на текущий и профилактический ремонт | 113 |
| Прочие затраты, связанные с эксплуатацией ЭВМ | 113 |
| Машинное время ЭВМ, час | 188,8 |
| Затраты на изготовление эталонного экземпляра (Зэт) | 24,12 |
| Затраты на технологию (Зтех) | 0 |
| Затраты на материалы (Змат) | 22,6 |
| Общепроизводственные затраты (Зобщ.пр) | 14,03 |
| Непроизводственные (коммерческие) затраты (Знепр) | 7,01 |
| Суммарные затраты на разработку ПО (Зр) | 550,12 |

Таблица Б.9 – Расчет торговой цены ПО, руб

|  |  |
| --- | --- |
| Статья затрат | Итого |
| Прибыль от реализации ПП | 165,036 |
| Отпускная цена ПО без НДС | 715,156 |
| Налог на добавленную стоимость | 143,03 |
| Отпускная цена ПО с НДС | 858,187 |
| Торговая наценка | 20 |
| Розничная цена ПО | 1002,648 |

Таблица Б.10 – Технико-экономические показатели проекта

|  |  |  |
| --- | --- | --- |
| Наименование показателя | Единица измерения | Проектный вариант |
| Общая трудоемкость разработки ПО | чел.-дн. | 54,8 |
| Капитальные вложения в проект | руб. | 2000,00 |
| Затраты на разработку программы | руб. | 550,12 |
| Затраты на оплату труда разработчиков | руб. | 225,588 |
| Затраты машинного времени | руб. | 256,768 |
| Затраты на технологию | руб. | 0 |
| Затраты на материалы | руб. | 22,6 |
| Общепроизводственные затраты | руб. | 14,03 |
| Непроизводственные (коммерческие) затраты | руб. | 7,01 |
| Цена без НДС | руб. | 715,156 |
| НДС | руб. | 143,03 |
| Цена с НДС | руб. | 858,187 |
| Простой срок окупаемости проекта | лет | 0,352 |
| Прибыль от реализации ПП | руб. | 165,036 |
| Прибыль от реализации ПП компании-владельцу | руб./год | 12605,352 |

## ПРИЛОЖЕНИЕ В

(обязательное)

**Листинг программы**

**package** org.pytorch.helloworld;  
  
**import** android.content.Context;  
**import** android.graphics.Bitmap;  
**import** android.graphics.BitmapFactory;  
**import** android.os.AsyncTask;  
**import** android.os.Bundle;  
**import** android.util.Log;  
**import** android.view.View;  
**import** android.widget.Button;  
**import** android.widget.ImageView;  
**import** android.widget.ProgressBar;  
**import** android.widget.Toast;  
  
**import** org.pytorch.IValue;  
**import** org.pytorch.Module;  
**import** org.pytorch.Tensor;  
**import** org.pytorch.torchvision.TensorImageUtils;  
  
**import** java.io.File;  
**import** java.io.FileOutputStream;  
**import** java.io.IOException;  
**import** java.io.InputStream;  
**import** java.io.OutputStream;  
**import** java.util.Map;  
  
**import** androidx.appcompat.app.AppCompatActivity;  
  
**public class** MainActivity **extends** AppCompatActivity **implements** Runnable {  
  
 **private static final int *CLASSNUM*** = 21;  
 **private static final int *DOG*** = 12;  
 **private static final int *PERSON*** = 15;  
 **private static final int *SHEEP*** = 17;  
 **private int color** = 0xFFFF0000;  
  
 **private static** Module *module*;  
 **private** Bitmap **bitmap**;  
 **private** Bitmap **transferredBitmap**;  
  
 **private** Button **bt\_c\_red**;  
 **private** Button **bt\_c\_blue**;  
 **private** Button **bt\_c\_black**;  
 **private** Button **bt\_uploat\_pic**;  
 **private** Button **bt\_reset**;  
  
 **private** ImageView **imageView**;  
 **private** ProgressBar **progressBar**;  
  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_main***);  
  
 **bt\_c\_red** = findViewById(R.id.***button\_red***);  
 **bt\_c\_blue** = findViewById(R.id.***button\_blue***);  
 **bt\_c\_black** = findViewById(R.id.***button\_black***);  
 **bt\_uploat\_pic** = findViewById(R.id.***button\_upload\_photo***);  
 **bt\_reset** = findViewById(R.id.***button\_reset***);  
 **imageView** = findViewById(R.id.***imv\_main***);  
 **progressBar** = findViewById(R.id.***progressBar***);  
  
 **progressBar**.setVisibility(ProgressBar.***INVISIBLE***);  
  
 **bt\_c\_red**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View view) {  
 **color** = 0xFFFF0000;  
 }  
 });  
  
 **bt\_c\_blue**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View view) {  
 **color** = 0x380ead;  
 }  
 });  
  
 **bt\_c\_black**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View view) {  
 **color** = 0x1c1b1f;  
 }  
 });  
  
 **bt\_reset**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View view) {  
 **imageView**.setImageBitmap(**bitmap**);  
 }  
 });  
  
 **bt\_uploat\_pic**.setOnClickListener(**new** View.OnClickListener() {  
 **public void** onClick(View v) {  
 **bt\_uploat\_pic**.setEnabled(**false**);  
 **progressBar**.setVisibility(ProgressBar.***VISIBLE***);  
 **bt\_uploat\_pic**.setText(**"Running model.."**);  
  
 Thread thread = **new** Thread(MainActivity.**this**);  
 thread.start();  
 }  
 });  
  
 **new** LoadResursesTask().execute(**"deeplabv3\_scripted.pt"**);  
 }  
  
 *// запускает предикт модели и возвращает обработанный результат в юай* @Override  
 **public void** run() {  
 *// preparing input tensor* **final** Tensor inputTensor = TensorImageUtils.*bitmapToFloat32Tensor*(  
 **bitmap**,  
 TensorImageUtils.*TORCHVISION\_NORM\_MEAN\_RGB*,  
 TensorImageUtils.*TORCHVISION\_NORM\_STD\_RGB*);  
 *//final float[] inputs = inputTensor.getDataAsFloatArray(); // ?  
  
 // running the model* Map<String, IValue> outTensors = *module*.forward(IValue.*from*(inputTensor)).toDictStringKey();  
  
 *// обработка результата* **final** Tensor outputTensor = outTensors.get(**"out"**).toTensor();  
 **final float**[] scores = outputTensor.getDataAsFloatArray();  
 **int** width = **bitmap**.getWidth();  
 **int** height = **bitmap**.getHeight();  
 **int**[] intValues = **new int**[width \* height];  
 **for** (**int** j = 0; j < width; j++) {  
 **for** (**int** k = 0; k < height; k++) {  
 **int** maxi = 0, maxj = 0, maxk = 0;  
 **double** maxnum = -Double.***MAX\_VALUE***;  
 **for** (**int** i = 0; i < ***CLASSNUM***; i++) {  
 **if** (scores[i \* (width \* height) + j \* width + k] > maxnum) {  
 maxnum = scores[i \* (width \* height) + j \* width + k];  
 maxi = i; maxj = j; maxk = k;  
 }  
 }  
 **if** (maxi == ***PERSON***)  
 intValues[maxj \* width + maxk] = **color**;  
 **else if** (maxi == ***DOG***)  
 intValues[maxj \* width + maxk] = **color**;  
 **else if** (maxi == ***SHEEP***)  
 intValues[maxj \* width + maxk] = **color**;  
 **else** intValues[maxj \* width + maxk] = 0xFF000000;  
 }  
 }  
  
 Bitmap bmpSegmentation = Bitmap.*createScaledBitmap*(**bitmap**, width, height, **true**);  
 Bitmap outputBitmap = bmpSegmentation.copy(bmpSegmentation.getConfig(), **true**);  
 *// вносим цвета пикселей* outputBitmap.setPixels(intValues, 0, outputBitmap.getWidth(), 0, 0,  
 outputBitmap.getWidth(), outputBitmap.getHeight());  
 *// преобразуем в изначальный масштаб* **transferredBitmap** = Bitmap.*createScaledBitmap*(outputBitmap, **bitmap**.getWidth(),  
 **bitmap**.getHeight(), **true**);  
  
 runOnUiThread(**new** Runnable() {  
 @Override  
 **public void** run() {  
 **imageView**.setImageBitmap(**transferredBitmap**);  
 **bt\_uploat\_pic**.setEnabled(**true**);  
 **bt\_uploat\_pic**.setText(**"Upload photo"**);  
 **progressBar**.setVisibility(ProgressBar.***INVISIBLE***);  
 }  
 });  
 }  
  
 */\*\*  
 \* Copies specified asset to the file in /files app directory and returns this file absolute path.  
 \** ***@return*** *absolute file path  
 \*/* **public static** String assetFilePath(Context context, String assetName) **throws** IOException {  
 File file = **new** File(context.getFilesDir(), assetName);  
 **if** (file.exists() && file.length() > 0) {  
 **return** file.getAbsolutePath();  
 }  
  
 **try** (InputStream is = context.getAssets().open(assetName)) {  
 **try** (OutputStream os = **new** FileOutputStream(file)) {  
 **byte**[] buffer = **new byte**[4 \* 1024];  
 **int** read;  
 **while** ((read = is.read(buffer)) != -1) {  
 os.write(buffer, 0, read);  
 }  
 os.flush();  
 }  
 **return** file.getAbsolutePath();  
 }  
 }  
  
 **class** LoadResursesTask **extends** AsyncTask<String, Void, Void> {  
  
 @Override  
 **protected** Void doInBackground(String ... params) {  
 **try** {  
 *// creating bitmap from packaged into app android asset 'image.jpg',* **bitmap** = BitmapFactory.*decodeStream*(getAssets().open(**"image\_1.jpg"**));  
 *// loading serialized torchscript module from packaged into app android asset model.pt,  
 module* = Module.*load*(*assetFilePath*(getApplicationContext(), params[0]));  
 } **catch** (IOException e) {  
 Log.*e*(**"error"**, **"Error reading assets"**, e);  
 Toast.*makeText*(getApplicationContext(),**"can't load photo or nn model"** ,Toast.***LENGTH\_LONG***).show();  
 }  
 **return null**;  
 }  
  
 @Override  
 **protected void** onPostExecute(Void aVoid) {  
 **super**.onPostExecute(aVoid);  
 *// showing image on UI* **if** (**bitmap** != **null**) {  
 **imageView**.setImageBitmap(**bitmap**);  
 }  
 }  
 }  
  
}

Листинг скрипта с нейронной сетью

*#!/usr/bin/env python  
# coding: utf-8  
  
# In[1]:***import** torch  
**import** torch.nn **as** nn  
**import** torch.optim **as** optim  
  
**from** mit\_semseg.config **import** cfg  
**from** mit\_semseg.dataset **import** TestDataset  
**from** mit\_semseg.models **import** ModelBuilder, SegmentationModule  
**from** mit\_semseg.utils **import** colorEncode  
  
**from** torch.utils.mobile\_optimizer **import** optimize\_for\_mobile  
  
*# System libs***import** os, csv, torch, numpy, scipy.io, PIL.Image, torchvision.transforms  
  
  
*# ### ручной вызов енкодера и декодера  
  
# In[2]:*colors = scipy.io.loadmat(**'E:/LAB\_DISK/diploma/color150.mat'**)[**'colors'**]  
names = {}  
**with** open(**'E:/LAB\_DISK/diploma/object150\_info.csv'**) **as** f:  
 reader = csv.reader(f)  
 next(reader)  
 **for** row **in** reader:  
 names[int(row[0])] = row[5].split(**";"**)[0]  
  
  
*# In[58]:***def** visualize\_result(img, pred, index=None):  
 *# filter prediction class if requested* **if** index **is not** None:  
 pred = pred.copy()  
 pred[pred != index] = -1  
 **print**(**f'{**names[index+1]**}:'**)  
   
 *# colorize prediction* pred\_color = colorEncode(pred, colors).astype(numpy.uint8)  
  
 *# aggregate images and save* im\_vis = numpy.concatenate((img, pred\_color), axis=1)  
 display(PIL.Image.fromarray(im\_vis))  
  
  
*# In[4]:*name\_model = **"ade20k-resnet50dilated-ppm\_deepsup"***#name\_model = "ade20k-mobilenetv2dilated-c1\_deepsup"  
  
# Network Builders net\_encoder net\_decoder  
  
# net\_encoder = ModelBuilder.build\_encoder(  
# arch="mobilenetv2dilated",  
# fc\_dim=320,  
# weights=name\_model+'/encoder\_epoch\_20.pth')  
  
# net\_decoder = ModelBuilder.build\_decoder(  
# arch="c1\_deepsup",  
# fc\_dim=320,  
# num\_class=150,  
# weights=name\_model+'/decoder\_epoch\_20.pth',  
# use\_softmax=True)*net\_encoder = ModelBuilder.build\_encoder(  
 arch=**'resnet50dilated'**,  
 fc\_dim=2048,  
 weights=name\_model+**'/encoder\_epoch\_20.pth'**)  
  
net\_decoder = ModelBuilder.build\_decoder(  
 arch=**'ppm\_deepsup'**,  
 fc\_dim=2048,  
 num\_class=150,  
 weights=name\_model+**'/decoder\_epoch\_20.pth'**,  
 use\_softmax=True)  
  
  
*# In[5]:*crit = torch.nn.NLLLoss(ignore\_index=-1)  
segmentation\_module = SegmentationModule(net\_encoder, net\_decoder, crit)  
segmentation\_module.eval()  
*#segmentation\_module.cuda()  
  
  
# In[62]:*pil\_image = PIL.Image.open(**'E:/LAB\_DISK/diploma/img/24.jpg'**)  
size = 310, 310  
pil\_image.thumbnail(size, PIL.Image.ANTIALIAS)  
pil\_image.convert(**"RGB"**)  
  
  
*# In[63]:  
  
  
# Load and normalize one image as a singleton tensor batch*pil\_to\_tensor = torchvision.transforms.Compose([  
 torchvision.transforms.ToTensor(),  
 torchvision.transforms.Normalize(  
 mean=[0.485, 0.456, 0.406], *# These are RGB mean+std values* std=[0.229, 0.224, 0.225]) *# across a large photo dataset.*])  
img\_original = numpy.array(pil\_image)  
img\_data = pil\_to\_tensor(pil\_image)  
  
singleton\_batch = {**'img\_data'**: img\_data[None]}  
output\_size = img\_data.shape[1:]  
  
img\_data[None].shape  
  
  
*# In[64]:  
  
  
# Run the segmentation at the highest resolution.***with** torch.no\_grad():  
 scores = segmentation\_module(singleton\_batch, segSize=output\_size)  
  
*# Get the predicted scores for each pixel*\_, pred = torch.max(scores, dim=1)  
pred = pred.cpu()[0].numpy()  
visualize\_result(img\_original, pred)  
  
  
*# In[71]:  
  
  
# Top classes in answer*predicted\_classes = numpy.bincount(pred.flatten()).argsort()[::-1]  
visualize\_result(img\_original, pred, 0)  
 *# In[20]:*encoder\_input=torch.tensor(1,3,224,224)  
encoder\_hidden=torch.zeros(1, 3, 224,224)  
  
decoder\_input1=torch.tensor([[0]])  
decoder\_input2=torch.zeros(1, 1, hidden\_size)  
decoder\_input3=torch.zeros(MAX\_LENGTH, hidden\_size)  
  
traced\_encoder = torch.jit.trace(encoder, (encoder\_input, encoder\_hidden))  
traced\_decoder = torch.jit.trace(decoder, (decoder\_input1, decoder\_input2, decoder\_input3))  
  
  
*# net\_encoder\_s = torch.jit.script(net\_encoder)   
# net\_decoder\_s = torch.jit.script(net\_decoder)  
  
  
# In[59]:  
  
  
# serialize model   
  
# ошибка при попытке сериализации собственной модели  
  
# model = torch.hub.load('pytorch/vision:v0.7.0', 'deeplabv3\_resnet50', pretrained=True)  
# model.eval()  
  
# scriptedm = torch.jit.script(model)  
# torch.jit.save(scriptedm, "deeplabv3\_scripted.pt")  
  
# model = segmentation\_module*model = torchvision.models.segmentation.deeplabv3\_resnet101(pretrained = True)  
model.eval()  
scriptedm = torch.jit.script(model)  
torch.jit.save(scriptedm, **"E:/LAB\_DISK/diploma/android\_models/"**+**"deeplabv3\_resnet101"**+**"\_model.pt"**)

Листинг скрипта с OpenCv

import numpy as np

import cv2 as cv

import pandas as pd

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

from matplotlib import cm

from matplotlib import colors

from matplotlib.colors import hsv\_to\_rgb

import math

from scipy.interpolate import splprep, splev

def GetCounters(im\_thresh, approx\_coef):

contours, hierarchy = cv.findContours(im\_thresh.copy(), cv.RETR\_CCOMP, cv.CHAIN\_APPROX\_NONE)

# вычисляем площади замкнутых контуров,

# для отсеивания мусора

areas = []

for i in contours:

areas.append(cv.contourArea (i))

areas\_average = sum(areas) / len(areas)

# убираем небольшие контура

contours\_new = []

for i in contours:

if cv.contourArea(i) > areas\_average :

contours\_new.append(i)

#уменьшение количества точек и аппроксимация контуров

contours\_appr = contours\_new.copy()

for i in range(len(contours\_new)):

epsilon = approx\_coef\*cv.arcLength(contours\_new[i], True)

approx = cv.approxPolyDP(contours\_new[i], epsilon, True)

contours\_appr[i] = approx

return contours\_appr

def drowDots(dots, img, color2):

if dots is not None:

for i in range(0, len(dots)):

cv.circle(img, (dots[i][0][0],dots[i][0][1]), radius=2, color=color2, thickness=-1)

return img

def drowLine(lines, img, color):

if lines is not None:

for i in range(0, len(lines)):

rho = lines[i][0][0]

theta = lines[i][0][1]

a = math.cos(theta)

b = math.sin(theta)

x0 = a \* rho

y0 = b \* rho

pt1 = (int(x0 + 1000\*(-b)), int(y0 + 1000\*(a)))

pt2 = (int(x0 - 1000\*(-b)), int(y0 - 1000\*(a)))

cv.line(img, pt1, pt2, color, 1, cv.LINE\_4)

return img

from collections import defaultdict

def segment\_by\_angle\_kmeans(lines, k=2, \*\*kwargs):

# Define criteria = (type, max\_iter, epsilon)

default\_criteria\_type = cv.TERM\_CRITERIA\_EPS + cv.TERM\_CRITERIA\_MAX\_ITER

criteria = kwargs.get('criteria', (default\_criteria\_type, 10, 1.0))

flags = kwargs.get('flags', cv.KMEANS\_RANDOM\_CENTERS)

attempts = kwargs.get('attempts', 10)

# returns angles in [0, pi] in radians

angles = np.array([line[0][1] for line in lines])

# multiply the angles by two and find coordinates of that angle

pts = np.array([[np.cos(2\*angle), np.sin(2\*angle)]

for angle in angles], dtype=np.float32)

# run kmeans on the coords

labels, centers = cv.kmeans(pts, k, None, criteria, attempts, flags)[1:]

labels = labels.reshape(-1) # transpose to row vec

# segment lines based on their kmeans label

segmented = defaultdict(list)

for i, line in zip(range(len(lines)), lines):

segmented[labels[i]].append(line)

segmented = list(segmented.values())

return segmented

def intersection(line1, line2):

rho1, theta1 = line1[0]

rho2, theta2 = line2[0]

A = np.array([

[np.cos(theta1), np.sin(theta1)],

[np.cos(theta2), np.sin(theta2)]

])

b = np.array([[rho1], [rho2]])

x0, y0 = np.linalg.solve(A, b)

x0, y0 = int(np.round(x0)), int(np.round(y0))

return [[x0, y0]]

def segmented\_intersections(lines):

intersections = []

for i, group in enumerate(lines[:-1]):

for next\_group in lines[i+1:]:

for line1 in group:

for line2 in next\_group:

intersections.append(intersection(line1, line2))

return intersections

def wallsFinder(img\_path\_in, img\_path\_out):

src2 = cv.imread(img\_path\_in, 1)

dst = cv.Canny(src2.copy(), 0, 255, None, 3)

h, w, c = src2.shape

blank\_image = np.zeros((h,w,1), np.uint8)

counters = GetCounters(dst, 0.001)

dst = cv.drawContours(blank\_image.copy(), counters, -1, (255, 255, 255), 3, cv.LINE\_4, None, 1)

dst = cv.copyMakeBorder(dst, top=2, bottom=2, left=2, right=2, borderType= cv.BORDER\_CONSTANT, value=[255,255,255] )

dst = cv.bitwise\_not(dst)

dst = cv.Canny(dst, 0, 255, None, 3)

cdst = cv.cvtColor(dst, cv.COLOR\_GRAY2BGR)

cdstP = np.copy(cdst)

lines = cv.HoughLines(dst, 1, np.pi / 180, 150, None, 0, 0)

cdst = drowLine(lines, cdst, (0,0,255))

# linesP = cv.HoughLinesP(dst, 1, np.pi / 180, 50, None, 50, 10)

# if linesP is not None:

# for i in range(0, len(linesP)):

# l = linesP[i][0]

# cv.line(cdstP, (l[0], l[1]), (l[2], l[3]), (0,0,255), 1, cv.LINE\_4)

segmented = segment\_by\_angle\_kmeans(lines)

intersections = segmented\_intersections(segmented)

img = src2.copy()

drowLine(segmented[0], img, (0,0,255))

drowLine(segmented[1], img, (255,0,0))

drowDots(intersections, img, (0,255,255))

cv.imwrite(img\_path\_out, img)

def wallsFinderSimple(img\_path\_in, img\_path\_out):

src2 = cv.imread(img\_path\_in, 1)

dst = cv.Canny(src2.copy(), 0, 255, None, 3)

h, w, c = src2.shape

blank\_image = np.zeros((h,w,1), np.uint8)

counters = GetCounters(dst, 0.002)

dst = cv.drawContours(src2.copy(), counters, -1, (255, 0, 0), 1, cv.LINE\_4, None, 1)

cv.imwrite(img\_path\_out, dst)

import sys

if \_\_name\_\_ == "\_\_main\_\_":

wallsFinder(sys.argv[1], sys.argv[2])

# wallsFinderSimple(sys.argv[1], sys.argv[2])

**import** javax.imageio.ImageIO;  
**import** java.awt.image.BufferedImage;  
**import** java.io.\*;  
**import** java.net.\*;  
**import** java.util.concurrent.ExecutorService;  
**import** java.util.concurrent.Executors;  
  
**public class** Server {  
  
 **public static final int *SERVER\_PORT*** = 8086;  
 **private static final int *BUFER\_SIZE*** = 1024\*2;  
  
 **private** ServerSocket **server**;  
 **private** ExecutorService **executor** = Executors.*newCachedThreadPool*();  
  
 **public void** Start() **throws** IOException {  
  
 **server** = **new** ServerSocket(***SERVER\_PORT***);  
  
 InetAddress ip = InetAddress.*getLocalHost*();  
 System.***out***.println(**"Current IP address (look for ipconfig).. : "** + ip.getHostAddress());  
 System.***out***.println(**"Current port : "** + ***SERVER\_PORT***);  
  
 **while**(**true**){  
 **final** Socket socket = **server**.accept();  
  
 executor.execute(**new** Runnable() {  
  
 **public void** run() {  
 **try** {  
 handleSocket(socket);  
 } **catch** (IOException e) {  
 System.out.println(**"handle exeption "** + e.getMessage());  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 **private void** handleSocket(**final** Socket socket) **throws** IOException, InterruptedException {  
 **byte**[] buffer = **new byte**[BUFER\_SIZE];  
  
 DataOutputStream dataOutputStream = **new** DataOutputStream(**new** BufferedOutputStream(socket.getOutputStream()));  
 dataOutputStream.writeUTF(**"Success connected!"**);  
 dataOutputStream.flush();  
  
 DataInputStream dataInputStream = **new** DataInputStream(**new** BufferedInputStream(socket.getInputStream()));  
 **int** client\_type = dataInputStream.readInt();  
  
 System.out.println(**"client start work. "** + client\_type);  
  
 **if**(client\_type == 1){  
 WorkWithFirstClientType(dataOutputStream, dataInputStream, buffer);  
 }  
 **else if**(client\_type == 2){  
 WorkWithSecondClientType(dataOutputStream, dataInputStream, buffer);  
 }  
 **else**{  
 System.out.println(**"client type wasn't recognized: "** + client\_type);  
 }  
  
 System.out.println(**"client ended work. "** + client\_type);  
 dataOutputStream.close();  
 dataInputStream.close();  
 socket.close();  
 }  
 });  
 }  
 }  
  
 **private void** WorkWithFirstClientType(DataOutputStream dataOutputStream,  
 DataInputStream dataInputStream, **byte**[] buffer)  
 **throws** IOException, InterruptedException {  
  
 **while**(**true**){  
 **int** length = 0;  
 **int** current\_size = 0;  
 **int** size = 0;  
 ByteArrayOutputStream byteArrayOutputStream = **new** ByteArrayOutputStream();  
  
 *// получение размера принимаемого масиива* size = dataInputStream.readInt();  
 **if**(size < 1) {  
 **break**;  
 }  
 System.out.println(**"future img size: "** + size);  
  
 *// приём самого массива частями* System.out.println(**"got\t | all\_got\t | need\_all\_got"**);  
 **do**{  
 length = dataInputStream.read(buffer);  
 byteArrayOutputStream.write(buffer, 0, length);  
 current\_size += length;  
 System.out.print(**"\r"** + length + **"\t | "** + current\_size + **"\t | "** + size);  
  
 } **while**(current\_size < size);  
  
 **if**(length < 2) {  
 **break**;  
 }  
  
 **byte**[] b\_img = byteArrayOutputStream.toByteArray();  
 System.out.println(**"\nimg size: "** + b\_img.length);  
 ByteArrayInputStream bais = **new** ByteArrayInputStream(b\_img);  
 BufferedImage img = ImageIO.read(bais);  
 bais.close();  
  
 *// обработка картинки* System.out.println(**"\nlooking for wallses borders.."**);  
 BufferedImage new\_img = ImgHelper.FindWalls(img);  
  
 *// отправка обработанного изображения* **try**{  
 ByteArrayOutputStream baos = **new** ByteArrayOutputStream();  
 ImageIO.write( new\_img, **"jpg"**, baos );  
 baos.flush();  
 **byte**[] new\_b\_img = baos.toByteArray();  
 baos.close();  
  
 System.out.println(**"new img size: "** + new\_b\_img.length);  
 dataOutputStream.writeInt(new\_b\_img.length);  
  
 dataOutputStream.write(new\_b\_img, 0, new\_b\_img.length);  
 dataOutputStream.flush();  
  
 }**catch** (Exception e) {  
 System.out.println(**"error: "** + e.getMessage());  
 }  
  
 byteArrayOutputStream.close();  
 }  
  
 dataOutputStream.close();  
 dataInputStream.close();  
 }  
  
 **private void** WorkWithSecondClientType(DataOutputStream dataOutputStream,  
 DataInputStream dataInputStream, **byte**[] buffer){  
 System.out.println(**"work with second client doesnt implement"**);  
 }  
  
}

*//import org.opencv.core.Mat;  
//import org.opencv.core.MatOfByte;  
//import org.opencv.imgcodecs.Imgcodecs;***import** javax.imageio.ImageIO;  
**import** javax.swing.\*;  
**import** java.awt.image.BufferedImage;  
**import** java.io.File;  
**import** java.io.IOException;  
  
**public class** ImgHelper **extends** JComponent {  
  
 **public static void** SaveImg(BufferedImage img, String path) **throws** IOException {  
 **if**(img != **null**){  
 File outputfile = **new** File(path);  
 ImageIO.*write*(img, **"jpg"**, outputfile);  
 }  
 }  
  
 **public static** BufferedImage ReadImg(String path) **throws** IOException {  
 File outputfile = **new** File(path);  
 **return** ImageIO.*read*(outputfile);  
 }  
  
 **public static** Mat BufferedImage2Mat(BufferedImage image) **throws** IOException {  
 ByteArrayOutputStream byteArrayOutputStream = **new** ByteArrayOutputStream();  
 ImageIO.write(image, **"jpg"**, byteArrayOutputStream);  
 byteArrayOutputStream.flush();  
 **return** Imgcodecs.imdecode(**new** MatOfByte(byteArrayOutputStream.toByteArray()), Imgcodecs.CV\_LOAD\_IMAGE\_UNCHANGED);  
 }  
  
 **public static** BufferedImage Mat2BufferedImage(Mat matrix)**throws** IOException {  
 MatOfByte mob=**new** MatOfByte();  
 Imgcodecs.imencode(**".jpg"**, matrix, mob);  
 **return** ImageIO.read(**new** ByteArrayInputStream(mob.toArray()));  
 }  
  
 **public static** BufferedImage FindWalls(BufferedImage source) **throws** IOException, InterruptedException {  
  
 Mat image = *BufferedImage2Mat*(source);  
 *//image = Imgcodecs.imread("./original.jpg");* Imgproc.cvtColor(image, image, Imgproc.COLOR\_RGB2GRAY);  
 *//Imgcodecs.imwrite("./edited.jpg", image);  
 Mat2BufferedImage*(image);  
  
 *// python делает работу за java* ImgHelper.*SaveImg*(source, **"./original.jpg"**);  
 PyHelper.*CallPy*(**"python ./wd.py \"original.jpg\" \"edited.jpg\""**);  
 **return** ImgHelper.*ReadImg*(**"./edited.jpg"**);  
 }  
}

**import** java.io.BufferedReader;  
**import** java.io.IOException;  
**import** java.io.InputStream;  
**import** java.io.InputStreamReader;  
  
**public class** PyHelper {  
  
 **public static void** CallPy(String command) **throws** IOException, InterruptedException {  
 Process p = Runtime.*getRuntime*().exec(command);  
  
 InputStream stdout = p.getInputStream();  
 InputStream stderr = p.getErrorStream();  
 InputStreamReader isr = **new** InputStreamReader(stdout);  
 InputStreamReader isrerr = **new** InputStreamReader(stderr);  
 BufferedReader br = **new** BufferedReader(isr);  
 BufferedReader brerr = **new** BufferedReader(isrerr);  
  
 String line = **null**;  
  
 System.***out***.println(**"OUTPUT:"**);  
 **while** ((line = br.readLine()) != **null**)  
 System.***out***.println(line);  
 System.***out***.println();  
  
 System.***out***.println(**"ERROR:"**);  
 **while** ((line = brerr.readLine()) != **null**)  
 System.***out***.println(line);  
 System.***out***.println();  
  
 p.waitFor();  
 }  
  
}

*<?***xml version="1.0" encoding="utf-8"***?>*<**manifest xmlns:android="http://schemas.android.com/apk/res/android"  
 package="com.example.wallsdetector"**>  
  
 <**uses-permission android:name="android.permission.INTERNET"**/>  
 <**uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE"**/>  
 <**uses-permission android:name="android.permission.READ\_EXTERNAL\_STORAGE"**/>  
  
 <**application  
 android:allowBackup="true"  
 android:icon="@mipmap/ic\_launcher"  
 android:label="@string/app\_name"  
 android:roundIcon="@mipmap/ic\_launcher\_round"  
 android:supportsRtl="true"  
 android:theme="@style/AppTheme"**>  
 <**activity android:name=".WorkActivity"**></**activity**>  
 <**activity android:name=".MainActivity"**>  
 <**intent-filter**>  
 <**action android:name="android.intent.action.MAIN"** />  
  
 <**category android:name="android.intent.category.LAUNCHER"** />  
 </**intent-filter**>  
 </**activity**>  
 </**application**>  
  
</**manifest**>

**package** com.example.wallsdetector.ClientSocket;  
  
**import** android.util.Log;  
  
**import** java.io.BufferedInputStream;  
**import** java.io.BufferedOutputStream;  
**import** java.io.ByteArrayOutputStream;  
**import** java.io.DataInputStream;  
**import** java.io.DataOutputStream;  
  
**import** java.io.IOException;  
**import** java.net.InetSocketAddress;  
**import** java.net.Socket;  
**import** java.net.UnknownHostException;  
**import** java.nio.charset.Charset;  
**import** java.nio.charset.StandardCharsets;  
  
**public class** MobSocket {  
  
 **private** String **TAG\_LOG** = **"info"**;  
  
 **private int BUFER\_SIZE** = 1024\*2;  
 **private** Socket **socket**;  
 **private** DataOutputStream **dataOutputStream** = **null**;  
 **private** DataInputStream **dataInputStream** = **null**;  
 **private byte**[] **buffer** = **new byte**[**BUFER\_SIZE**];  
  
 **public boolean** isConnected(){  
 **return socket**.isConnected();  
 }  
  
 **private** MobSocket(){  
 **socket** = **new** Socket();  
 }  
  
 **private static** MobSocket *instance*;  
  
 **public static** MobSocket getInstance(){  
 **return** (*instance* == **null**) ? *instance* = **new** MobSocket() : *instance*;  
 }  
  
 **public boolean** unconnectToDevice() {  
 **try** {  
 **if**(**dataInputStream** != **null**) **dataInputStream**.close();  
 Log.*d*(**TAG\_LOG**, **"dataInputStream close"**);  
 **if**(**dataOutputStream** != **null**) **dataOutputStream**.close();  
 Log.*d*(**TAG\_LOG**, **"dataOutputStream close"**);  
 **if**(**socket** != **null**) **socket**.close();  
 Log.*d*(**TAG\_LOG**, **"Soket close"**);  
  
 *instance* = **null**;  
 **return true**;  
  
 }  
 **catch** (UnknownHostException e) {  
 e.printStackTrace();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
  
 **return false**;  
  
 }  
  
 **public void** connectToDevice(String ip, **int** port) {  
 **try** {  
 Log.*d*(**TAG\_LOG**, **"start created new soket object"**);  
 **socket**.connect(**new** InetSocketAddress(ip,port));  
 Log.*d*(**TAG\_LOG**, **"new Soket object"**);  
  
 }  
 **catch** (UnknownHostException e) {  
 e.printStackTrace();  
 *instance* = **null**;  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 *instance* = **null**;  
 }  
  
 }  
  
 **public boolean** sendMessage(String msg) {  
 **try** {  
 **if**(**dataOutputStream** == **null**){  
 **dataOutputStream** = **new** DataOutputStream(**new** BufferedOutputStream(**socket**.getOutputStream()));  
 Log.*d*(**TAG\_LOG**, **"output stream created"**);  
 }  
  
 **dataOutputStream**.writeUTF(msg);  
 **dataOutputStream**.flush();  
  
 Log.*d*(**TAG\_LOG**, **"output stream worked"**);  
 **return true**;  
  
 }  
 **catch** (IOException e)  
 { e.printStackTrace(); }  
  
 **return false**;  
 }  
  
 **public boolean** sendMessage(**int** msg) {  
 **try** {  
 **if**(**dataOutputStream** == **null**){  
 **dataOutputStream** = **new** DataOutputStream(**new** BufferedOutputStream(**socket**.getOutputStream()));  
 Log.*d*(**TAG\_LOG**, **"output stream created"**);  
 }  
  
 **dataOutputStream**.writeInt(msg);  
 **dataOutputStream**.flush();  
  
 Log.*d*(**TAG\_LOG**, **"output stream worked"**);  
 **return true**;  
  
 }  
 **catch** (IOException e)  
 { e.printStackTrace(); }  
  
 **return false**;  
 }  
  
 **public** String getMessage() {  
 String s = **""**;  
 **try** {  
 **if**(**dataInputStream** == **null**){  
 **dataInputStream** = **new** DataInputStream(**new** BufferedInputStream(**socket**.getInputStream()));  
 Log.*d*(**TAG\_LOG**, **"input stream created"**);  
 }  
  
 s = **dataInputStream**.readUTF();  
 }  
 **catch** (NullPointerException e){  
 e.printStackTrace();  
 }  
 **catch** (IOException e)  
 { e.printStackTrace(); }  
  
 Log.*d*(**TAG\_LOG**, **"input stream worked"**);  
 **return** s;  
 }  
  
 **public void** sendFile(**byte**[] bytes) {  
 **try** {  
 **if**(**dataOutputStream** == **null**){  
 **dataOutputStream** = **new** DataOutputStream(**new** BufferedOutputStream(**socket**.getOutputStream()));  
 Log.*d*(**TAG\_LOG**, **"output stream created"**);  
 }  
  
 **dataOutputStream**.writeInt(bytes.**length**);  
 **dataOutputStream**.write(bytes,0,bytes.**length**);  
 **dataOutputStream**.flush();  
 }  
 **catch** (IOException e)  
 { e.printStackTrace(); }  
  
 Log.*d*(**TAG\_LOG**, **"output stream worked"**);  
 }  
  
 **public byte**[] getFile() {  
 **byte**[] result = **new byte**[1];  
 **try** {  
 **if**(**dataInputStream** == **null**){  
 **dataInputStream** = **new** DataInputStream(**new** BufferedInputStream(**socket**.getInputStream()));  
 Log.*d*(**TAG\_LOG**, **"input stream created"**);  
 }  
  
 ByteArrayOutputStream byteArrayOutputStream = **new** ByteArrayOutputStream();  
 **int** current\_size = 0;  
 **int** size = 0;  
 **int** length = 0;  
  
 *// получение размера принимаемого масиива* size= dataInputStream.readInt();  
 Log.d(TAG\_LOG, **"future size: "** + size);  
  
 *// приём самого массива частями* **do**{  
 length = dataInputStream.read(buffer);  
 byteArrayOutputStream.write(buffer, 0, length);  
 current\_size += length;  
 Log.d(TAG\_LOG, length + **"\t | "** + current\_size + **"\t | "** + size);  
 } **while**(current\_size < size);  
  
 result = byteArrayOutputStream.toByteArray();  
  
 Log.d(TAG\_LOG, **"current size: "** + result.length);  
 byteArrayOutputStream.close();  
 }  
 **catch** (NullPointerException e){  
 e.printStackTrace();  
 }  
 **catch** (IOException e)  
 { e.printStackTrace(); }  
  
 Log.d(TAG\_LOG, **"input stream worked"**);  
 **return** result;  
 }  
}

**package** com.example.wallsdetector.ClientSocket;  
  
**import** android.os.AsyncTask;  
**import** android.util.Log;  
  
**import** java.nio.charset.StandardCharsets;  
**import** java.sql.Array;  
**import** java.util.Arrays;  
  
**public class** MobTask **extends** AsyncTask<String, Void, Boolean> {  
  
 @Override  
 **protected** Boolean doInBackground(String ... params) {  
 **try** {  
 MobSocket mobSocket = MobSocket.*getInstance*();  
 mobSocket.connectToDevice(params[0], Integer.*parseInt*(params[1]));  
  
 **if**(mobSocket.isConnected()){  
 String messageFromServer = mobSocket.getMessage();  
  
 Log.*d*(**"info"**, **"server msg: "** + messageFromServer);  
 Log.*d*(**"info"**, **"connect successful"**);  
 **return true**;  
  
 }**else** {  
 Log.*d*(**"info"**, **"Can't connected"**);  
 }  
  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return false**;  
 }  
  
}

**package** com.example.wallsdetector.fragments;  
  
**import** android.app.Activity;  
**import** android.app.AlertDialog;  
**import** android.content.Context;  
**import** android.content.DialogInterface;  
**import** android.content.Intent;  
**import** android.graphics.Bitmap;  
**import** android.graphics.BitmapFactory;  
**import** android.graphics.drawable.BitmapDrawable;  
**import** android.net.Uri;  
**import** android.os.AsyncTask;  
**import** android.os.Bundle;  
**import** android.provider.MediaStore;  
**import** android.util.Log;  
**import** android.view.LayoutInflater;  
**import** android.view.View;  
**import** android.view.ViewGroup;  
**import** android.widget.Button;  
**import** android.widget.ImageView;  
**import** android.widget.Toast;  
  
**import** androidx.fragment.app.Fragment;  
  
**import** com.example.wallsdetector.ClientSocket.MobSocket;  
**import** com.example.wallsdetector.R;  
  
**import** java.io.ByteArrayOutputStream;  
**import** java.io.IOException;  
**import** java.util.Arrays;  
  
**public class** FragmentPicture **extends** Fragment {  
  
 **private** ImageView **iv\_image**;  
 **private** Button **b\_send\_photo**;  
 **private** Button **b\_upload\_photo**;  
  
 **private** MobSocket **mobSocket**;  
 **private** ControllerTask **controllerTask**;  
  
 **private boolean sended** = **true**;  
  
 @Override  
 **public** View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {  
  
 View v = inflater.inflate(R.layout.***fragment\_picture***, container, **false**);  
 **mobSocket** = MobSocket.*getInstance*();  
 **controllerTask** = **new** ControllerTask();  
  
 **b\_upload\_photo** = v.findViewById(R.id.***b\_picture\_upload\_photo***);  
 **b\_send\_photo** = v.findViewById(R.id.***b\_picture\_send\_photo***);  
 **iv\_image** = v.findViewById(R.id.***iv\_picture***);  
  
 **b\_send\_photo**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 **if**(**sended**){  
 **sended** = **false**;  
 **if**(**iv\_image**.getImageMatrix() != **null**){  
 Bitmap bitmap = ((BitmapDrawable) **iv\_image**.getDrawable()).getBitmap();  
  
 ByteArrayOutputStream baos = **new** ByteArrayOutputStream();  
 bitmap.compress(Bitmap.CompressFormat.***JPEG***, 100, baos);  
 **byte**[] imageInByte = baos.toByteArray();  
 **try** {  
 baos.close();  
 }  
 **catch** (IOException e) {  
 e.printStackTrace();  
 }  
  
 Toast.*makeText*(v.getContext(),**"sending picture.."** ,Toast.***LENGTH\_SHORT***).show();  
  
 **new** ControllerTask().execute(imageInByte);  
 }  
 **else** {  
 Toast.*makeText*(v.getContext(),**"no one picture chosen!"** ,Toast.***LENGTH\_SHORT***).show();  
 **sended** = **true**;  
 }  
 }  
 }  
 });  
  
 **b\_upload\_photo**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 selectImage(getContext());  
 }  
 });  
  
 **return** v;  
 }  
  
 @Override  
 **public void** onActivityResult(**int** requestCode, **int** resultCode, Intent data) {  
 getActivity();  
 **if**(resultCode != Activity.***RESULT\_CANCELED***) {  
 **switch** (requestCode) {  
 **case** 0:  
 **if** (resultCode == Activity.***RESULT\_OK*** && data != **null**) {  
 Bitmap selectedImage = (Bitmap) data.getExtras().get(**"data"**);  
 **iv\_image**.setImageBitmap(selectedImage);  
 }  
  
 **break**;  
 **case** 1:  
 **if** (resultCode == Activity.***RESULT\_OK*** && data != **null**) {  
 Uri returnUri = data.getData();  
 Bitmap bitmapImage = **null**;  
 **try** {  
 bitmapImage = MediaStore.Images.Media.*getBitmap*(getActivity().getContentResolver(), returnUri);  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 **iv\_image**.setImageBitmap(bitmapImage);  
 }  
 **break**;  
 }  
 }  
 }  
  
 **private void** selectImage(Context context) {  
 **final** CharSequence[] options = { **"Take Photo"**, **"Choose from Gallery"**,**"Cancel"** };  
  
 AlertDialog.Builder builder = **new** AlertDialog.Builder(context);  
 builder.setTitle(**"Choose your profile picture"**);  
  
 builder.setItems(options, **new** DialogInterface.OnClickListener() {  
  
 @Override  
 **public void** onClick(DialogInterface dialog, **int** item) {  
  
 **if** (options[item].equals(**"Take Photo"**)) {  
 Intent takePicture = **new** Intent(android.provider.MediaStore.***ACTION\_IMAGE\_CAPTURE***);  
 startActivityForResult(takePicture, 0);  
  
 } **else if** (options[item].equals(**"Choose from Gallery"**)) {  
 Intent pickPhoto = **new** Intent(Intent.***ACTION\_PICK***,  
 android.provider.MediaStore.Images.Media.***EXTERNAL\_CONTENT\_URI***);  
 startActivityForResult(pickPhoto , 1);  
  
 } **else if** (options[item].equals(**"Cancel"**)) {  
 dialog.dismiss();  
 }  
 }  
 });  
 builder.show();  
 }  
  
 **class** ControllerTask **extends** AsyncTask<**byte**[], Void, Void> {  
 **byte**[] **im\_bytes**;  
  
 @Override  
 **protected** Void doInBackground(**byte**[] ... params) {  
 **try** {  
 **mobSocket**.sendFile(params[0]);  
 Log.*d*(**"info"**, **"send message: "** + params[0].**length**);  
 *// принимаем обработанное фото* **im\_bytes** = **mobSocket**.getFile();  
  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
  
 **return null**;  
 }  
  
 @Override  
 **protected void** onPostExecute(Void aVoid) {  
 **super**.onPostExecute(aVoid);  
 Log.*d*(**"info"**, **"task ended"**);  
 *// отображаем обработанное изображение* **if**(**im\_bytes** != **null**) {  
 Bitmap bitmap = BitmapFactory.*decodeByteArray*(**im\_bytes**, 0, **im\_bytes**.**length**);  
 **iv\_image**.setImageBitmap(bitmap);  
 Toast.*makeText*(getContext(), **"the resulting new image!"**,Toast.***LENGTH\_SHORT***).show();  
 }  
 **else**{  
 **iv\_image**.setImageBitmap(**null**);  
 Toast.*makeText*(getContext(), **"img was't correct send from server -\_-"**,Toast.***LENGTH\_SHORT***).show();  
 }  
  
 **sended** = **true**;  
 }  
 }  
}

**package** com.example.wallsdetector;  
  
**import** androidx.appcompat.app.AppCompatActivity;  
  
**import** android.content.Intent;  
**import** android.os.Bundle;  
**import** android.view.View;  
**import** android.widget.Button;  
**import** android.widget.EditText;  
**import** android.widget.Toast;  
  
**import** com.example.wallsdetector.ClientSocket.MobTask;  
  
**import** java.util.concurrent.ExecutionException;  
  
**public class** MainActivity **extends** AppCompatActivity {  
 **private** Button b\_enter;  
 **private** EditText port;  
 **private** EditText ip;  
  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.activity\_main);  
  
 b\_enter = findViewById(R.id.b\_connect);  
 port = findViewById(R.id.car\_port);  
 ip = findViewById(R.id.car\_ip);  
  
  
 b\_enter.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
  
 String p = port.getText().toString();  
 String ip\_str = ip.getText().toString();  
  
 MobTask mobTask = **new** MobTask();  
 mobTask.execute(ip\_str, p);  
  
 **boolean** connected = **false**;  
 **try** {  
 *//изза ожидания skipped >100 frames* connected = mobTask.get();  
  
 } **catch** (ExecutionException e) {  
 e.printStackTrace();  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }  
  
 Toast.makeText(getApplicationContext(),  
 String.valueOf(connected) ,Toast.LENGTH\_SHORT).show();  
  
 **if**(connected){  
 Intent intent2 = **new** Intent(getApplicationContext(), WorkActivity.**class**);  
 startActivity(intent2);  
 }  
 }  
 });  
 }  
}

**package** com.example.wallsdetector;  
  
**import** android.content.Intent;  
**import** android.os.AsyncTask;  
**import** android.os.Bundle;  
**import** android.util.Log;  
**import** android.view.View;  
**import** android.widget.Button;  
**import** android.widget.TextView;  
**import** android.widget.Toast;  
  
**import** androidx.appcompat.app.AppCompatActivity;  
**import** androidx.fragment.app.Fragment;  
**import** androidx.fragment.app.FragmentManager;  
  
**import** com.example.wallsdetector.fragments.FragmentPicture;  
**import** com.example.wallsdetector.fragments.FragmentVideo;  
**import** com.example.wallsdetector.ClientSocket.MobSocket;  
  
**import** java.nio.ByteBuffer;  
**import** java.util.concurrent.ExecutionException;  
  
**public class** WorkActivity **extends** AppCompatActivity {  
  
 **private** Button **b\_picture**;  
 **private** Button **b\_video**;  
 **private** Button **b\_unconnected**;  
 **protected** TextView **tv\_connected**;  
  
 **private** MobSocket **mobSocket**;  
  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_work***);  
  
 **b\_unconnected** = findViewById(R.id.***b\_unconnected***);  
 **tv\_connected** = findViewById(R.id.***tv\_connected***);  
 **b\_picture** = findViewById(R.id.***b\_picture***);  
 **b\_video** = findViewById(R.id.***b\_video***);  
  
 **mobSocket** = MobSocket.*getInstance*();  
*// clientTypeTask = new ClientTypeTask();* **b\_unconnected**.setOnClickListener(**new** View.OnClickListener()  
 {  
 @Override  
 **public void** onClick(View v) {  
 ClientTypeTask c = **new** ClientTypeTask();  
 c.execute(0);  
  
 **boolean** b = **false**;  
 **try** {  
 b = c.get();  
  
 } **catch** (ExecutionException e) {  
 e.printStackTrace();  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }  
  
 **if**(b){  
 **if**(**mobSocket**.unconnectToDevice()){  
 **tv\_connected**.setText(**"unconnected"**);  
 Intent intent = **new** Intent(getApplicationContext(), MainActivity.**class**);  
 startActivity(intent);  
 Toast.*makeText*(getApplicationContext(),**"unconnected"** ,Toast.***LENGTH\_SHORT***).show();  
  
 }**else**{  
 Toast.*makeText*(getApplicationContext(),**"can't unconnected"** ,Toast.***LENGTH\_SHORT***).show();  
 }  
 }  
  
 }  
 });  
  
  
 *//прежде чем переходить на другой тип клиента необходимо убедиться- закончил ли работу другой тип  
 //после того как клиент выполняет свою задачу- он разрывает соединение* **b\_picture**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 *//отправляю серверу необходимое сообщение, указывая на тип подключаемого клиента* **if**(**tv\_connected**.getText().equals(**"unconnected"**)){  
 Log.*d*(**"info"**, **"click picture"**);  
 **tv\_connected**.setText(**"connected"**);  
 **new** ClientTypeTask().execute(1);  
 }  
 }  
 });  
  
 **b\_video**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 **if**(**tv\_connected**.getText().equals(**"unconnected"**)){  
 Log.*d*(**"info"**, **"click video"**);  
 **tv\_connected**.setText(**"connected"**);  
 **new** ClientTypeTask().execute(2);  
 }  
 }  
 });  
 }  
 *//единственная задача-отправить серверу сообщение о том, какой тип контроллера выбран  
 //и открывать соответствующий фрагмент* **class** ClientTypeTask **extends** AsyncTask<Integer, Void, Boolean> {  
  
 **private int id**;  
  
 *//маленький фабричный метод* **private** Fragment getFragment(**int** id){  
 **switch** (id){  
 **case** 0: **return null**;  
 **case** 1: **return new** FragmentPicture();  
 **case** 2: **return new** FragmentVideo();  
 }  
 **return null**;  
 }  
 @Override  
 **protected** Boolean doInBackground(Integer ... params) {  
 Log.*d*(**"info"**, **"start asynck task with 1"**);  
 **try** {  
 **if**(**mobSocket**.sendMessage(params[0])) {  
 **id** = params[0];  
 **return true**;  
 }  
  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
  
 **return false**;  
 }  
 @Override  
 **protected void** onPostExecute(Boolean result) {  
 **super**.onPostExecute(result);  
  
 **if**(result){  
 *//подключаю нужный фрагмент* Log.*d*(**"info"**, **"start go to fragment"**);  
 Log.*d*(**"info"**, **"id = "**.concat(String.*valueOf*(**id**)));  
 Fragment dv = getFragment(**id**);  
 **if**(dv !=**null**){  
 Log.*d*(**"info"**, **"fragment != null"**);  
 FragmentManager fragmentManager = getSupportFragmentManager();  
 fragmentManager.beginTransaction()  
 .replace(R.id.***fragment\_data\_list***, dv).commit();  
 }**else** {  
 Log.*d*(**"info"**, **"fragment == null"**);  
 }  
  
 }**else** {  
 Toast.*makeText*(getApplicationContext(),**"can't send message to server"** ,Toast.***LENGTH\_LONG***).show();  
 }  
 }  
 }  
}

## ПРИЛОЖЕНИЕ Л

(обязательное)

**Руководство системного-программиста**

1. Общие сведения о программном комплексе.

Программный комплекс разработан для распознавания контуров стен на двумерных изображениях. Сервер контролируется мобильным приложением.

Для корректной работы сервера на стороне запускающей вычислительной машины необходимо наличие *python* версии 2.x.

Минимальные технические требования к компьютеру:

– возможность подключаться к беспроводной сети *WiFi*;

– оперативная память 1 ГБ;

– не менее 50 МБ свободной памяти на жестком диске;

1. Структура программного обеспечения

Сервер представлен в виде классов, отвечающих за передачу данных по протоколу *TCP/IP* и обработку изображений.

Мобильное приложение состоит из двух Activity – регистрационного и главного. На главном может располагаться один из двух фрагментов, в зависимости от выбранного типа клиента.

1. Настройка программного обеспечения

Со стороны сервера есть возможность изменять опции обработки изображения, без перезапуска и перекомпиляции. Также можно настроить область прослушивания клиентских подключений.

Со стороны мобильных приложений никаких дополнительных настроек нет.

1. Проверка программного обеспечения

Программный комплекс не вызовет затруднений в работе, если подключение клиента к серверу длилось менее двух секунд. В противном случае связь может быть нестабильна.

1. Дополнительные возможности.

Возможность загружать изображения из галереи мобильного устройства.

1. Сообщения системному программисту.

Некоторые ситуации, например, отправка пустых данных на сервер могут вызвать сбой в работе клиента. В такое случае необходимо его перезапустить.

## ПРИЛОЖЕНИЕ М

(обязательное)

**Руководство программиста**

1. Назначение и условия применения

Система программного обеспечения предназначена для распознавания контуров стен на двумерных изображениях.

Программное клиент-серверное обеспечение выполняет задачу связи с сервером используя протокол *TCP/IP.*

Для корректной работы мобильных приложений рекомендуется использовать устройства с операционной системой *Android* версии 4.1 и выше.

Для корректной работы сервера и его успешного запуска необходимо наличие на персональном компьютере *python* версии 2.х и сети *WiFi* на обоих устройствах.

Минимальные технические требования к компьютеру:

* оперативная память 2 ГБ;
* не менее 50 МБ свободной памяти на жестком диске;

Минимальные технические требования к мобильному устройству:

* диагональ экрана от 4.3 дюйма;
* не менее 10 МБ свободной внутренней памяти;

2. Характеристика программного обеспечения

Сервер работает автоматически и не требует стороннего вмешательства. Мобильные приложения нуждаются в действиях и настройке пользователя.

3. Обращение к программному комплексу

Для запуска сервера необходимо открыть консоль, перейти в директорию *scc/*, расположенную в основном рабочем пространстве, и запустить исполняемый файл *wallsdetector.server.exe*

Для запуска мобильного приложения – установить его на мобильное устройство и ввести требуемые для подключения IP адрес и порт.

4. Входные и выходные данные

Данные поступающие на вход серверу – это изображения от мобильных приложений. Выходные данные – обработанные изображения.

5. Сообщения

Код мобильных приложений оснащён логированием, которое можно просмотреть в режиме отладки.

## ПРИЛОЖЕНИЕ Н

(обязательное)

**Руководство пользователя**

1. Введение

Руководство пользователя обеспечивает получение пользователем базовых навыков по эксплуатации приложения «*WallsDetector*».

Разработанное программное обеспечение предназначено для распознавания границ стен на изображениях.

Приложение обладает следующим функционалом:

* передача изображений на сервер;
* загрузка изображений с галереи мобильного устройства;
* загрузка изображений посредством снимка фотокамеры.

Для использования программного обеспечения пользователь должен быть ознакомлен со следующими руководствами:

* настоящим руководством пользователя;
* правилами использования ЭВМ.

2. Назначение и условия применения

Приложение предназначено для распознавания границ стен на изображениях, посредством отправки изображения с мобильного телефона на сервер для дальнейшего анализа.

Для корректной работы мобильного приложения пользователь должен обладать мобильным устройством с операционной системой Android версии 4.1 и выше.

Минимальные технические требования к мобильному устройству:

– доступ к сети *WiFi*;

– диагональ экрана от 4.3 дюйма;

– не менее 5 МБ свободной внутренней памяти;

3. Подготовка к работе

Для установки приложения необходимо загрузить на мобильное устройство файл с расширением *apk* и запустить процесс установки, нажав на него. При необходимости установить в настройках телефона разрешение на установку приложений из неизвестных источников. Узнайте IP адрес и порт сервера.

4. Аварийные ситуации

Чтобы избежать ошибок при использовании программы, необходимо соблюдать порядок действий и условия пользования. Прежде чем завершить работу с приложением разорвите соединение с сервером, путём нажатия клавиши «*Unconnect*».

5. Правила работы с приложением

При запуске мобильного приложения введите *IP* адрес сервера и порт, нажмите клавишу «*Connect*». После успешного подключения откроется рабочее окно. Выберете тип передаваемых данных– изображение или видео ряд, нажав соответствующую названиям кнопку.

Для изображения станет доступно окно управления с возможностью импорта изображения и его последующей отправки на сервер. Создайте или загрузите изображение и нажмите кнопку «*Send*».

По завершении работы с сервером нажмите кнопку «*Unconnect*» и выйдете из приложения.

6. Рекомендации по освоению.

Изучить базовые навыки работы с мобильными устройствами, при необходимости.