##### ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

***МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ***

***Факультет «Информационных технологий»***

***Кафедра «СМАРТ-технологии»***

***Курсовая работа***

***по дисциплине «Инженерный проект»***

***на тему: «Сегментация на основе градаций серого»***

*Вариант 2.2.*

*Работу выполнил\_и студент\_ка 3 курса очного отделения:  
<ФИО>*

*Учебная группа: <ГРУППА>*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Работу проверил\_и:*

*<ФИО препод.>*

*Москва 2021*

*Содержание*

[*Техническое задание на разработку проекта* 3](#_Toc62078774)

[*1.* *Реализация алгоритма Split & Merge.* 4](#_Toc62078775)

[*1.1* *Разделение изображения (Этап Split)* 4](#_Toc62078776)

[*1.2* *Соединение сегментов (Этап Merge)* 5](#_Toc62078777)

[*2.* *Реализация алгоритма поиска символов (OCR).* 7](#_Toc62078778)

[*2.1* *Поиск контуров (описанных прямоугольников)* 7](#_Toc62078779)

[*2.2* *Распознавание символов (MLP)* 8](#_Toc62078780)

[*2.3* *Построение слов и строк из классифицированных символов* 10](#_Toc62078781)

[*3.* *Генератор обучающей выборки для нейросети.* 12](#_Toc62078782)

[*3.1* *Разделение изображения* 12](#_Toc62078783)

[*3.2. Создание обучающей выборки* 13](#_Toc62078784)

[*4.* *Вывод.* 13](#_Toc62078785)

[*Приложение A. Скриншот основной формы приложения EngineeringProject* 14](file:///C:\Users\gyas\Downloads\181_311_Pavel_Neshumov_Kursovaya_Rabota_Variant_2_2%20(1).docx#_Toc62078786)

[*Приложение B. Скриншот формы приложения Split & Merge* 15](file:///C:\Users\gyas\Downloads\181_311_Pavel_Neshumov_Kursovaya_Rabota_Variant_2_2%20(1).docx#_Toc62078790)

[*Приложение C. Исходный код EngineeringProject.py* 16](file:///C:\Users\gyas\Downloads\181_311_Pavel_Neshumov_Kursovaya_Rabota_Variant_2_2%20(1).docx#_Toc62078792)

[*Приложение D. Исходный код ImageSplitter.py* 19](file:///C:\Users\gyas\Downloads\181_311_Pavel_Neshumov_Kursovaya_Rabota_Variant_2_2%20(1).docx#_Toc62078797)

[*Приложение E. Исходный код Network\_DIY.py* 21](file:///C:\Users\gyas\Downloads\181_311_Pavel_Neshumov_Kursovaya_Rabota_Variant_2_2%20(1).docx#_Toc62078800)

[*Приложение F. Исходный код gui\_ep\_ocr.py* 23](file:///C:\Users\gyas\Downloads\181_311_Pavel_Neshumov_Kursovaya_Rabota_Variant_2_2%20(1).docx#_Toc62078804)

[*Приложение F. Исходный код gui\_ep\_ocr.py* 24](file:///C:\Users\gyas\Downloads\181_311_Pavel_Neshumov_Kursovaya_Rabota_Variant_2_2%20(1).docx#_Toc62078806)

[*Приложение G. Исходный код SplitAndMerge.py* 27](file:///C:\Users\gyas\Downloads\181_311_Pavel_Neshumov_Kursovaya_Rabota_Variant_2_2%20(1).docx#_Toc62078811)

[*Приложение H. Исходный код SAM\_worker.py* 28](file:///C:\Users\gyas\Downloads\181_311_Pavel_Neshumov_Kursovaya_Rabota_Variant_2_2%20(1).docx#_Toc62078813)

[*Приложение I. Исходный код gui\_ep.py* 31](file:///C:\Users\gyas\Downloads\181_311_Pavel_Neshumov_Kursovaya_Rabota_Variant_2_2%20(1).docx#_Toc62078819)

# *Техническое задание на разработку проекта*

***Общее направление работы:***

*2. Сегментация на основе градаций серого*

*2.2. Метод разделения и объединения для символов текста Применить алгоритм Split&Merge для сегментации и применить классификатор MLP для определения надписей на изображениях. Задано: источник данных – наборы изображений Порядок обработки:*

*• выполнить покадровое считывание методами OpenCV или AForgeNET и обесцвечивание для видеофайла 640х480*

*• применить операцию расщепления изображения за счет сужения дифференциации по яркости и контролем плотности и размера объектов (областей).*

*• применить операцию слияния для смежных областей для определения кандидатов на распознавание.*

*• обучить классификатор на набор печатных символов*

*• реализовать вырезку кандидатов на распознавание (строк) и разбиения на фрагменты по пропорции символа (высота к ширине)*

*• реализовать распознавание выдачу списка видов и положений знаков (с объединением в текст при построчном сканировании изображения)*

***Ожидаемое практическое применение результатов работы:***

*Студент должен научиться применять практические навыки по работе с нейросетями.*

# ***Реализация алгоритма Split & Merge.***

*Алгоритм Split & Merge состоит из двух этапов:*

1. *Большие неоднородные области разбиваются на более мелкие области, которые могут быть однородными.*
2. *Соседние области сравниваются и объединяются, если они достаточно близки по некоторому свойству.*

*Область изображения делится на 4 области. Затем каждая делится еще на 4. Этот процесс, продолжается до определенного предела. Как правило, этот предел налагается из-за соображений хранения или ограничения времени обработки или из-за разрешения устройства вывода. Пиксель — это наименьшая область деления.*

## ***Разделение изображения (Этап Split)***

*Реализованный алгоритм разделения на регионы является рекурсивным. На выходе алгоритма – линейный массив контуров (прямоугольников с соотношением сторон исходного изображения). Отрисованные контуры изображены на рис. 1.*



*Рисунок 1 – Результат разделения изображения (Split)*

*Метод разделения работает следующим образом:*

1. *На вход метода подаются 4 координаты (верх-лево, верх-право, низ-право, низ-лево). В самом начале – это координаты исходного изображения (640x480)*
2. *Вычисляется максимальное и минимально значение яркости пикселей в регионе (нужный сегмент вырезается по координатам из исходного изображения)*
3. *Проверяется условие, если максимум – минимум <= порог или размер сегмента слишком маленький, его разделение прекращается (контур сегмента добавляется в массив).*
4. *Если условие не соблюдается (максимум – минимум > порог), происходит разделение сегмента на 4 подсегмента, для этого вычисляются координаты середин отрезков (по 4 исходным точкам).*
5. *Для каждого из 4 новых подсегментов снова вызывается данный метод, но, уже с новыми координатами.*
6. *Рекурсия продолжается до тех пора, пока условие разделения не прекратит выполняться.*

## ***Соединение сегментов (Этап Merge)***

*Реализованный алгоритм объединения сегментов работает «графически».*

*Для хранения регионов используется не линейный массив координат точек контуров (как при операции Split), а графическая ч/б карта регионов. См рис. 2.*



*Рисунок 2 – Карта регионов во время операции Merge*

*Работа метода Merge выглядит следующим образом:*

1. *Случайным образом выбирается пиксель, лежащий в регионе, который не обрабатывался (черный). Для проверки существует карта «проработанных» областей.*
2. *Методом cv2.floodFill выполняется заливка региона.*
3. *Залитый регион переносится на новое изображение-маску*
4. *При помощи cv2.dilate, залитая область увеличивается на 2 пикселя во всех направлениях*
5. *С помощью бинарных операций, по маске ищутся точки пересечения с картой регионов (см рис. 2)*
6. *Между каждой полученной точкой, вычисляется средняя (начало новой заливки)*
7. *По полученным точкам начинается новая процедура заливки и выделения региона-маски.*
8. *По двум маскам (текущая и новая заливка) вырезаются 2 фрагмента исходного изображения*
9. *По фрагментам вычисляются минимальные и максимальные значения яркости пикселей*
10. *Если удовлетворятся условие (new\_region\_max - region\_min <= self.threshold and region\_max - new\_region\_min <= self.threshold), происходит объединение масок*
11. *Полученная «общая» маска вырезается (закрашивает белые линии чёрным) на карте регионов*

*Результатом работы данного алгоритма является новая карта регионов, прошедших операцию слияния. Пример на рис. 3.*



*Рисунок 3 – Результат объединения сегментов (Merge)*

*Исходный код алгоритма см. в приложении H (лист 2-3).*

# ***Реализация алгоритма поиска символов (OCR).***

*Для извлечения текста из изображения, описанный выше алгоритм Split & Merge не подходит по множеству причин. Поэтому, для выполнения задачи необходимо найти контур каждого символа, а именно, прямоугольник, в который данный контур вписан. Далее, вырезать этот прямоугольник из исходного изображения и изменить размер для того, чтобы все полученные изображения были одного размера.*

*Полученный массив изображений необходимо подать на вход MLP – нейросети для распознавания символов.*

*Далее, учитывая расстояния между символами и строками, составить предложения и строки.*

## ***Поиск контуров (описанных прямоугольников)***

*Для реализации всех методов по поиску символов, использовались стандартные функции библиотеки OpenCV.*

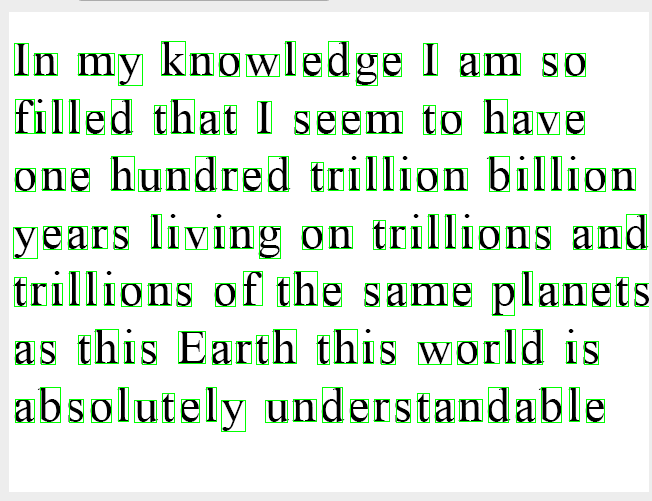
1. *На первом этапе происходит бинаризация исходной картинки методом cv2.threshold.*
2. *Далее, выполняется поиск контуров cv2.findContours методами «внешней обводки» - cv2.RETR\_EXTERNAL и аппроксимация cv2.CHAIN\_APPROX\_SIMPLE*
3. *После, выполняется обязательный этап проверки полученных контуров на соответствие размеру. Порог задаётся вручную, с помощью слайдера на форме.*
4. *Для дальнейшего составления слов и предложений, контуры проходят сортировку по положению относительно изображения*

self.contours.sort(key=**lambda** sort\_x: self.get\_contour\_precedence(sort\_x, self.image.shape[1]))

**def** get\_contour\_precedence(self, contour, cols):  
 origin = cv2.boundingRect(contour)  
 **return** ((origin[1] // self.tolerance\_factor) \* self.tolerance\_factor) \* cols + origin[0]

1. *Затем, по полученным контурам, из исходного Ч/Б изображения вырезаются участки, приводятся к размеру 20x30 пикселей (интерполяция cv2.INTER\_NEAREST) и заполняются в массив.*

*Результат работы поиска символов см. на рис. 4. Зелёным цветом обведены найденные символы.*



*Рисунок 4 – Результат алгоритма распознавания отдельных символов*

## ***Распознавание символов (MLP)***

*Для распознавания символов (из полученного массива изображений) применяется нейросеть - многослойный перцептрон.*

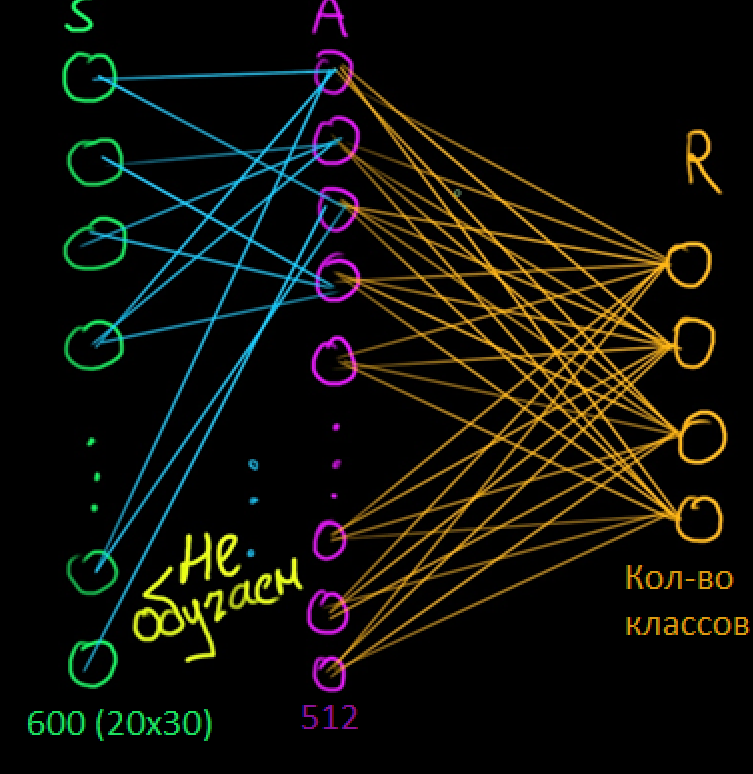
*Для соответствия требования заданию, применяется нейросеть, написанная вручную. Хотя, для улучшения работы алгоритма, имеется возможность использовать готовую библиотеку TensorFlow.*

*«Самодельный» перцептрон работает, аналогично второй лабораторной работы из курса «Нейронные сети в задачах технического зрения и управления», а именно:*

*1. Полученные изображения (20x30) в одномерный массив 0-1, перемешиваются и сопоставляясь с «метками» (порядковый номер символа для классификации)*

*2. Далее производится генерация весов случайным образом. Первые веса содержат значения {-1; 0; 1}, причем, не более трёх значимых весов. Второй слой имеет диапазон [0; 1]*

*3. Структура MLP-сети показана на рис. 5.*



*Рисунок 5 – Структура многослойного перцептрона*

*Обучение производиться только оранжевых весов и выглядит следующим образом:*

*• За первый слой принимается массив входных данных (изображения)*

*• Далее, вычисляется скалярное произведение первого слоя на первые веса (значения которых {-1; 0; 1}). Результат проходит через функцию активации:*

**def** layer\_0\_activator(weights\_sum):  
 *# return max(0, weights\_sum)* result = [[0] \* weights\_sum[0]] \* weights\_sum  
 **for** i **in** range(len(weights\_sum)):  
 sample\_result = [0] \* weights\_sum[0]  
 **for** k **in** range(len(weights\_sum[i])):  
  
 threshold = 1.8 *# 1.79* **if** weights\_sum[i][k] >= threshold:  
 sample\_result[k] = weights\_sum[i][k] - 2.2 *# 0.79* **else**:  
 sample\_result[k] = 0  
  
 result[i] = sample\_result  
 result = np.array(result)  
 *# print(result)  
 # exit(0)* **return** np.array(result)

*• Результат считается вторым слоем. Далее производится скалярное произведение этого [второго] слоя на вторые веса (значения которых [0; 1]). Результат пропускается через сигмоидную функцию активации:*

**def** sigmoid(x):  
 **return** 1 / (1 + np.exp(-x))

*• В итоге, последний (третий) слой содержит 4 элемента.*

*• Вычисляется ошибка (тот выход, который должен соответствовать метке принимается за 1, остальные три – за 0.).*

*• На основе ошибки вычисляются корректирующие значения для весов*

*adjustments\_l2 = output\_l1.T.dot(error\_l2 \* (output\_l2 \* (1 - output\_l2)))*

*• Эти корректировки прибавляются к весам второго слоя.*

*В среднем, необходимо 20-30 итераций обучения, чтобы достичь точности в 70-80%*

## ***Построение слов и строк из классифицированных символов***

*Для составления текста, используются значения расстояний между соседними символами (контурами) по горизонтали и вертикали.*

*Метод составления текста перебирает все классифицированные символы, определяет расстояние по горизонтали, и, если оно меньше порога, считается, что данные символы принадлежат одному слову.*

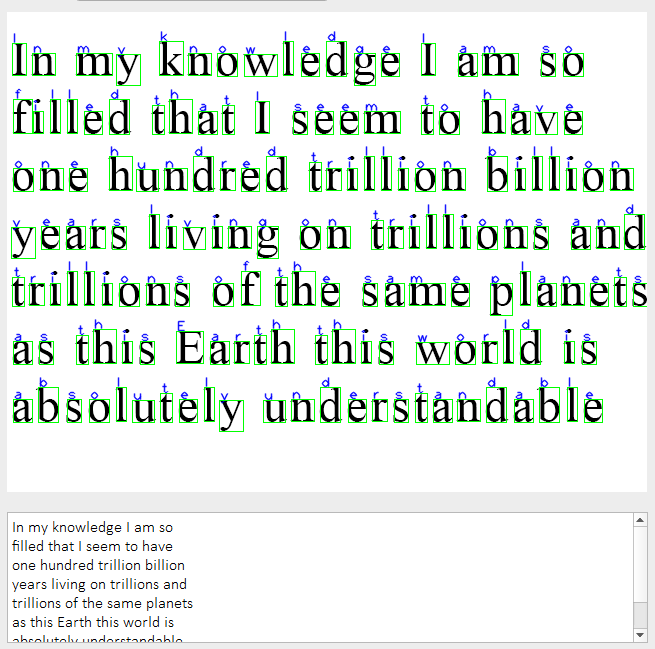
*Также, проверяется расстояние между символами по вертикали, если меньше порога, считается, что слова принадлежат одной строке.*

*Исходный код метода выглядит следующим образом:*

last\_contour\_right\_center = [0, 0]  
word = []  
words = []  
lines = []  
**for** i **in** range(len(self.contours)):  
 **if** i == 0:  
 word = [i]  
 last\_contour\_right\_center = self.contour\_right\_center(i)  
 **else**:  
 contour\_left\_center = self.contour\_left\_center(i)  
 diff\_x = abs(last\_contour\_right\_center[0] - contour\_left\_center[0])  
 diff\_y = abs(last\_contour\_right\_center[1] - contour\_left\_center[1])  
 **if** diff\_y < height\_threshold: *# In one line* **if** diff\_x < width\_threshold: *# In one word* word.append(i)  
 **else**:  
 words.append(word)  
 word = [i]  
 **else**:  
 words.append(word)  
 lines.append(words)  
 words = []  
 word = [i]  
 last\_contour\_right\_center = self.contour\_right\_center(i)  
  
words.append(word)  
lines.append(words)

*Полученный текст выводится в текстовое поле на форме.*

*Пример работы алгоритма на рис. 6.*



*Рисунок 6 – Пример распознавания текста*

*Распознанный текст:*

In my knowledge I am so

filled that I seem to have

one hundred trillion billion

years living on trillions and

trillions of the same planets

as this Earth this world is

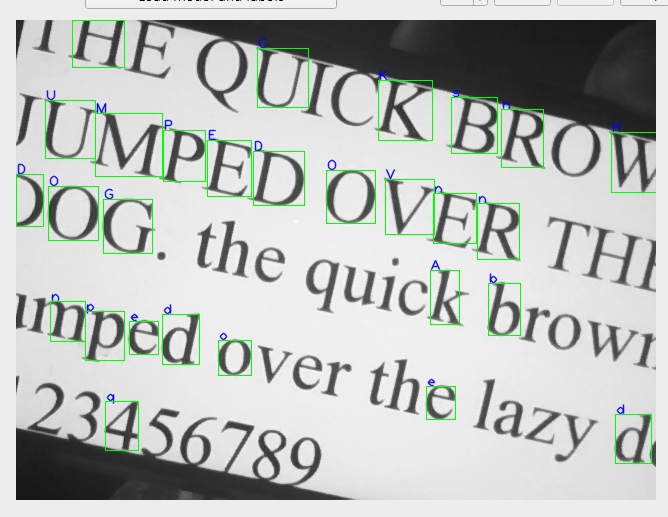
absolutely understandable

*Как можно видеть, точность распознавания текста достаточно высока.*

*Также, в приложении имеется возможность распознавания текста по камере.*

*Нейросеть (MLP) не обучалась на примерах символов, полученных с камеры, но, несмотря на это, работая на высокой скорости (30 FPS), алгоритму удаётся распознавать некоторые отдельные символы.*

*Пример работы на изображении с камеры показан на рис. 7.*



*Рисунок 7 – Пример распознавания текста с камеры*

# ***Генератор обучающей выборки для нейросети.***

*Для корректной классификации изображений, требуется достаточно точно обучить многослойный перцептрон. Для этого, предусмотрен специальный генератор обучающих изображений 20x30*

## ***Разделение изображения***

*На вход генератора подаётся обычное изображение (640x480), содержащие все необходимые символы (см рис. 8)*

**

*Рисунок 8 – Исходное изображение для генератора*

*По алгоритму, описанному выше, происходит разделение изображения на отдельные символы.*

## ***3.2. Создание обучающей выборки***

*Далее, из разделённых изображений составляется необходимое количество классов. А именно, создаётся ряд папок, в каждой содержится требуемое количество обучающих изображений.*

*Для повышения точности распознавания, на форме имеется checkbox, включив который, в момент генерации изображений будет подмешиваться яркостный шум, изменять яркость изображения, поворот и пиксельный сдвиг.*

*Результат генерации см. на рис. 9*



*Рисунок 9 – Сгенерированные изображения для обучения*

# ***Вывод.***

*В результате выполнения данной курсовой работы, было создано приложение, реализующее алгоритм Split & Merge, а также, приложение, способное, с достаточно высокой точностью классифицировать символы (текст) на изображении с последующей группировкой распознанных символов в слова и строки.*

# 

Инв. № подп

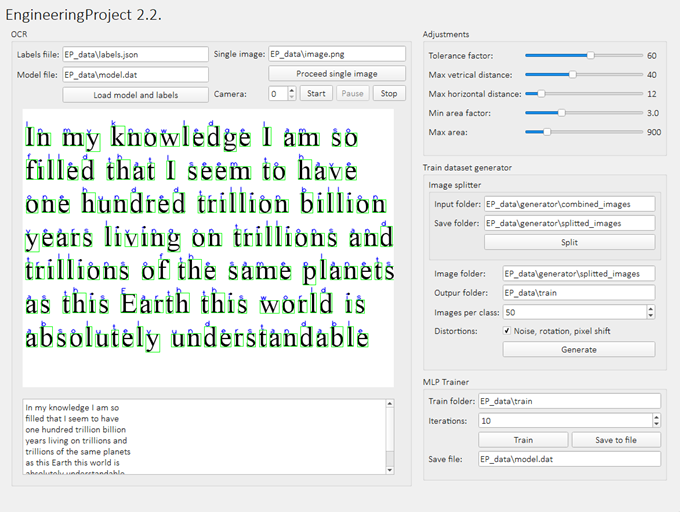
Подп. и дата

Взам. инв. №

Инв. № дубл.

Подп. и дата

*Скриншот основной формы приложения:*



Лит

Лист

Листов

1

1

Группа: 181-311

# *Приложение A. Скриншот основной формы приложения EngineeringProject*

Московский политехнический университет

Лит

№ докум.

Изм.

Подп.

Дата

<ФИО>

Разраб.

<ФИО препод.>

Пров.

Т. контр.

Н. контр.

<ФИО препод.>

Утв.

# 

Инв. № подп

Подп. и дата

Взам. инв. №

Инв. № дубл.

Подп. и дата

# *Скриншот формы приложения Split & Merge:*



Лит

Лист

Листов

1

1

Группа: 181-311

# *Приложение B. Скриншот формы приложения Split & Merge*

Московский политехнический университет

Лит

№ докум.

Изм.

Подп.

Дата

<ФИО>

Разраб.

<ФИО препод.>

Пров.

Т. контр.

Н. контр.

<ФИО препод.>

Утв.

# 

Инв. № подп

Подп. и дата

Взам. инв. №

Инв. № дубл.

Подп. и дата

*"""  
 Licensed under the Unlicense License;  
 you may not use this file except in compliance with the License.  
 You may obtain a copy of the License at  
  
 https://unlicense.org  
  
 Unless required by applicable law or agreed to in writing, software  
 distributed under the License is distributed on an "AS IS" BASIS,  
 WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
 See the License for the specific language governing permissions and  
 limitations under the License.  
"""*

**import** os  
**import** threading  
  
**import** numpy **as** np  
**import** cv2  
**import** sys  
  
**import** qimage2ndarray  
**from** PyQt5 **import** QtWidgets  
**from** PyQt5.QtGui **import** QPixmap  
**import** random  
**import** json  
**import** os.path  
**from** pathlib **import** Path  
  
**import** ImageSplitter  
**import** Network\_DIY  
**import** gui\_ep\_ocr  
  
  
**class** EngineeringProject(QtWidgets.QMainWindow, gui\_ep\_ocr.Ui\_MainWindow):  
 **def** \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.setupUi(self)  
  
 self.DEBUG = **True** self.font = cv2.FONT\_HERSHEY\_PLAIN  
  
 self.btn\_split.clicked.connect(self.split)  
 self.btn\_generate.clicked.connect(self.generate)  
 self.btn\_mlp\_train\_start.clicked.connect(self.mlp\_train\_start)  
 self.btn\_mlp\_train\_save.clicked.connect(self.mlp\_train\_save)  
 self.btn\_ocr\_load\_model.clicked.connect(self.ocr\_load\_model)  
 self.btn\_ocr\_single.clicked.connect(self.ocr\_single)  
 self.btn\_ocr\_camera\_start.clicked.connect(self.ocr\_camera\_start)  
 self.btn\_ocr\_camera\_pause.clicked.connect(self.ocr\_camera\_pause)  
 self.btn\_ocr\_camera\_stop.clicked.connect(self.ocr\_camera\_stop)  
  
 self.slider\_tolerance.valueChanged.connect(self.tolerance\_changed)  
 self.slider\_height.valueChanged.connect(self.height\_changed)  
 self.slider\_width.valueChanged.connect(self.width\_changed)  
 self.slider\_min\_area.valueChanged.connect(self.min\_area\_changed)  
 self.slider\_max\_area.valueChanged.connect(self.max\_area\_changed)  
  
 self.label\_tolerance.setText(str(self.slider\_tolerance.value()))  
 self.label\_height.setText(str(self.slider\_height.value()))  
 self.label\_width.setText(str(self.slider\_width.value()))  
 self.label\_min\_area.setText(str(self.slider\_min\_area.value() / 10))  
 self.label\_max\_area.setText(str(self.slider\_max\_area.value()))  
  
 self.camera\_running = **False** self.camera\_paused = **False** self.cv\_cap = **None** self.network = **None** self.text\_labels = []  
 self.train\_images = np.array([])  
 self.train\_labels = np.array([])  
  
 **def** tolerance\_changed(self):  
 self.label\_tolerance.setText(str(self.slider\_tolerance.value()))  
  
 **def** height\_changed(self):  
 self.label\_height.setText(str(self.slider\_height.value()))  
  
 **def** width\_changed(self):  
 self.label\_width.setText(str(self.slider\_width.value()))  
  
 **def** min\_area\_changed(self):  
 self.label\_min\_area.setText(str(self.slider\_min\_area.value() / 10))  
  
 **def** max\_area\_changed(self):  
 self.label\_max\_area.setText(str(self.slider\_max\_area.value()))

Лит

Лист

Листов

1

3

Группа: 181-311

# *Приложение C. Исходный код EngineeringProject.py*

Московский политехнический университет

Лит

№ докум.

Изм.

Подп.

Дата

<ФИО>

Разраб.

<ФИО препод.>

Пров.

Т. контр.

Н. контр.

<ФИО препод.>

Утв.

# 

Инв. № подп

Подп. и дата

Взам. инв. №

Инв. № дубл.

Подп. и дата

**def** split(self):  
 **if** self.DEBUG:  
 print(**'Splitting images...'**)  
 entries = Path(self.splitter\_input\_folder.text())  
 entry\_counter = 0  
 **for** entry **in** entries.iterdir():  
 splitter = ImageSplitter.ImageSplitter(cv2.imread(str(entry), 0),  
 min\_area\_threshold=self.slider\_min\_area.value() / 10,  
 max\_area=self.slider\_max\_area.value(),  
 tolerance\_factor=self.slider\_tolerance.value())  
  
 **for** i **in** range(len(splitter.letters)):  
 cv2.imwrite(self.splitter\_save\_folder.text() +  
 **'/'** + str(entry\_counter) + **'\_'** + str(i) + **'.png'**, splitter.letters[i])  
 entry\_counter += 1  
 **if** self.DEBUG:  
 print(**'Splitting done.'**)  
  
**def** generate(self):  
 **if** self.DEBUG:  
 print(**'Generating dataset...'**)  
 entries = Path(self.generator\_image\_folder.text())  
 images = []  
 classes = 0  
 last\_classes = []  
 **for** entry **in** entries.iterdir():  
 class\_num = int(os.path.splitext(entry.name)[0].split(**'\_'**)[1])  
 **if** class\_num > classes:  
 classes = class\_num  
 **if** class\_num **not in** last\_classes:  
 **if not** os.path.exists(self.generator\_output.text() + **'/'** + str(class\_num)):  
 os.mkdir(self.generator\_output.text() + **'/'** + str(class\_num))  
 images.append([])  
 last\_classes.append(class\_num)  
 classes += 1  
  
 **for** entry **in** entries.iterdir():  
 class\_num = int(os.path.splitext(entry.name)[0].split(**'\_'**)[1])  
 images[class\_num].append(cv2.imread(str(entry), 0))  
  
 **for** i **in** range(classes):  
 generations\_per\_image = int(self.generator\_images\_n.value() / len(images[i]))  
 generated\_n = 0  
 **for** image **in** images[i]:  
 **for** k **in** range(generations\_per\_image):  
 temp\_image = image.copy()  
 **if** self.generator\_distortions.isChecked():  
  
 *# Rotation* temp\_image = 255 - temp\_image  
 num\_rows, num\_cols = temp\_image.shape[:2]  
 rotation\_matrix = cv2.getRotationMatrix2D(  
 (num\_cols / 2, num\_rows / 2),  
 random.randrange(-7, 7), 1)  
 temp\_image = cv2.warpAffine(temp\_image, rotation\_matrix, (num\_cols, num\_rows))  
  
 *# Pixel shift* shift\_size\_x = random.randrange(-2, 2)  
 shift\_size\_y = random.randrange(-2, 2)  
 rows, cols = temp\_image.shape  
 M = np.float32([[1, 0, shift\_size\_x], [0, 1, shift\_size\_y]])  
 temp\_image = cv2.warpAffine(temp\_image, M, (cols, rows))  
 temp\_image = 255 - temp\_image  
  
 *# Brightness noise* temp\_image = temp\_image.astype(**'float32'**)  
 temp\_image += random.randrange(-60, 60)  
 temp\_image = np.clip(temp\_image, 0, 255)  
 temp\_image = temp\_image.astype(**'uint8'**)  
  
 *# Pixel noise* **for** y **in** range(image.shape[0]):  
 **for** x **in** range(image.shape[1]):  
 **if** random.randrange(0, 2) **and** temp\_image[y][x] > 0:  
 brightness\_pixel = temp\_image[y][x] + random.randrange(-20, 20)  
 **if** brightness\_pixel > 255: brightness\_pixel = 255  
 **if** brightness\_pixel < 0: brightness\_pixel = 0  
 temp\_image[y][x] = brightness\_pixel  
  
 cv2.imwrite(self.generator\_output.text() +  
 **'/'** + str(i) + **'/'** + str(generated\_n) + **'.png'**, temp\_image)  
 generated\_n += 1  
 **if** self.DEBUG:  
 print(**'Generating done.'**)  
  
**def** mlp\_train\_start(self):  
 self.train\_images = []  
 self.train\_labels = []  
 **with** open(self.mlp\_labels.text()) **as** f:  
 self.text\_labels = json.load(f)  
 self.text\_labels = np.asarray(self.text\_labels)  
  
 **for** i **in** range(len(self.text\_labels)):  
 entries = Path(self.mlp\_train\_folder.text() + **'/'** + str(i))  
 **for** entry **in** entries.iterdir():  
 self.train\_images.append(1.0 - (cv2.imread(str(entry), 0).flatten() / 255.0))  
 self.train\_labels.append(i)

Лит

Лист

Листов

2

3

Группа: 181-311

# *Приложение C. Исходный код EngineeringProject.py*

Московский политехнический университет

Лит

№ докум.

Изм.

Подп.

Дата

<ФИО>

Разраб.

<ФИО препод.>

Пров.

Т. контр.

Н. контр.

<ФИО препод.>

Утв.

# *Прило*

Инв. № подп

Подп. и дата

Взам. инв. №

Инв. № дубл.

Подп. и дата

combined\_lists = list(zip(self.train\_images, self.train\_labels))  
 random.shuffle(combined\_lists)  
 train\_images, train\_labels = zip(\*combined\_lists)  
  
 self.network = Network\_DIY.Network(images=np.array(train\_images), labels=np.array([train\_labels]),  
 train\_output\_labels=len(self.text\_labels),  
 train\_iterations=self.mlp\_train\_iterations.value())  
 self.network.start\_training()  
  
 **def** mlp\_train\_save(self):  
 self.network.save\_weights(self.mlp\_train\_save\_file.text())  
  
 **def** ocr\_load\_model(self):  
 **with** open(self.mlp\_labels.text()) **as** f:  
 self.text\_labels = np.asarray(json.load(f))  
 self.network = Network\_DIY.Network()  
 self.network.load\_weights(self.ocr\_model\_file.text())  
  
 **def** ocr\_single(self):  
 self.ocr\_text.setPlainText(self.proceed\_frame(cv2.imread(str(self.ocr\_test\_image.text()), 0)))  
  
 **def** proceed\_frame(self, image):  
 splitter = ImageSplitter.ImageSplitter(image, min\_area\_threshold=self.slider\_min\_area.value() / 10,  
 max\_area=self.slider\_max\_area.value(),  
 tolerance\_factor=self.slider\_tolerance.value())  
 debug\_image = cv2.cvtColor(splitter.image.copy(), cv2.COLOR\_GRAY2RGB)  
 text\_combined = **''  
 if** len(splitter.contours) > 0:  
 self.network.images = []  
 **for** letter **in** splitter.letters:  
 self.network.images.append(1.0 - letter.flatten() / 255.0)  
 self.network.images = np.array(self.network.images)  
  
 predicted\_classes = self.network.predict()  
  
 letters = []  
 **for** i **in** range(len(predicted\_classes)):  
 letters.append(self.text\_labels[predicted\_classes[i]])  
 cv2.putText(debug\_image, letters[i], *# letters[i], str(i)* (splitter.contours[i][0][0][0], splitter.contours[i][0][0][1]),  
 self.font, 1, (255, 0, 0), 1, cv2.LINE\_AA)  
 text\_combined = **''** lines = splitter.make\_text(self.slider\_height.value(), self.slider\_width.value())  
  
 **for** line **in** lines:  
 **for** word **in** line:  
 **for** letter **in** word:  
 text\_combined += letters[letter]  
 cv2.drawContours(debug\_image, splitter.contours, letter, (0, 255, 0), 1)  
 text\_combined += **' '** text\_combined += **'\n'** self.ocr\_image.setPixmap(QPixmap.fromImage(qimage2ndarray.array2qimage(  
 cv2.cvtColor(debug\_image, cv2.COLOR\_BGR2RGB))))  
 **return** text\_combined  
  
 **def** ocr\_camera\_start(self):  
 self.ocr\_text.setPlainText(**'See console for output.'**)  
 **if not** self.camera\_paused:  
 self.camera\_running = **True** self.cv\_cap = cv2.VideoCapture(self.ocr\_camera\_id.value(), cv2.CAP\_DSHOW)  
 thread = threading.Thread(target=self.camera\_process)  
 thread.start()  
 self.camera\_paused = **False** self.ocr\_camera\_id.setEnabled(**False**)  
 self.btn\_ocr\_camera\_start.setEnabled(**False**)  
 self.btn\_ocr\_camera\_pause.setEnabled(**True**)  
 **if** self.DEBUG:  
 print(**'Camera started.'**)  
  
 **def** ocr\_camera\_pause(self):  
 self.camera\_paused = **True** self.btn\_ocr\_camera\_start.setEnabled(**True**)  
 self.btn\_ocr\_camera\_pause.setEnabled(**False**)  
  
 **def** camera\_process(self):  
 **while** self.camera\_running:  
 **if not** self.camera\_paused:  
 ret, img = self.cv\_cap.read()  
 img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)  
 print(self.proceed\_frame(img))  
 *# self.ocr\_text.setPlainText* self.cv\_cap.release()  
  
 **def** ocr\_camera\_stop(self):  
 self.camera\_paused = **False** self.camera\_running = **False** self.ocr\_camera\_id.setEnabled(**True**)  
 self.btn\_ocr\_camera\_start.setEnabled(**True**)  
 self.btn\_ocr\_camera\_pause.setEnabled(**False**)  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 app = QtWidgets.QApplication(sys.argv)  
 app.setStyle(**"fusion"**)  
 window = EngineeringProject()  
 window.show()  
 app.exec\_()

Лит

Лист

Листов

3

3

Группа: 181-311

# *Приложение C. Исходный код EngineeringProject.py*

Московский политехнический университет

Лит

№ докум.

Изм.

Подп.

Дата

<ФИО>

Разраб.

<ФИО препод.>

Пров.

Т. контр.

Н. контр.

<ФИО препод.>

Утв.

# 

Инв. № подп

Подп. и дата

Взам. инв. №

Инв. № дубл.

Подп. и дата

*"""  
 Licensed under the Unlicense License;  
 you may not use this file except in compliance with the License.  
 You may obtain a copy of the License at  
  
 https://unlicense.org  
  
 Unless required by applicable law or agreed to in writing, software  
 distributed under the License is distributed on an "AS IS" BASIS,  
 WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
 See the License for the specific language governing permissions and  
 limitations under the License.  
"""*

**import** cv2  
**import** numpy **as** np  
  
  
**class** ImageSplitter:  
 **def** \_\_init\_\_(self, image, binary\_threshold=127,  
 splitted\_width=20,  
 splitted\_height=30,  
 min\_area\_threshold=4,  
 max\_area=500,  
 tolerance\_factor=61):  
  
 self.image = image  
 self.letters = []  
 self.contours = []  
 self.tolerance\_factor = tolerance\_factor  
 ret, self.binarized = cv2.threshold(self.image, binary\_threshold, 255, cv2.THRESH\_BINARY\_INV)  
 contours, \_ = cv2.findContours(self.binarized, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)  
  
 avg\_contour\_area = 0  
 **for** contour **in** contours:  
 avg\_contour\_area += cv2.contourArea(contour)  
 avg\_contour\_area /= len(contours)  
  
 **for** contour **in** contours:  
 **if** max\_area > cv2.contourArea(contour) > avg\_contour\_area / min\_area\_threshold:  
 [x, y, w, h] = cv2.boundingRect(contour)  
 tl = [[x, y]]  
 tr = [[x + w, y]]  
 br = [[x + w, y + h]]  
 bl = [[x, y + h]]  
 self.contours.append(np.array([tl, tr, br, bl]))  
  
 self.contours.sort(key=**lambda** sort\_x: self.get\_contour\_precedence(sort\_x, self.image.shape[1]))  
  
 **for** contour **in** self.contours:  
 letter = self.image[contour[0][0][1]: contour[2][0][1], contour[0][0][0]: contour[2][0][0]]  
 letter = cv2.resize(letter, (splitted\_width, splitted\_height), interpolation=cv2.INTER\_NEAREST)  
 self.letters.append(letter)  
  
 **def** get\_contour\_precedence(self, contour, cols):  
 origin = cv2.boundingRect(contour)  
 **return** ((origin[1] // self.tolerance\_factor) \* self.tolerance\_factor) \* cols + origin[0]  
  
 **def** make\_text(self, height\_threshold, width\_threshold, debug=**False**):  
  
 last\_contour\_right\_center = [0, 0]  
 word = []  
 words = []  
 lines = []  
  
 **for** i **in** range(len(self.contours)):  
 **if** i == 0:  
 word = [i]  
 **if** debug:  
 print(**'New letter found: '**, i)  
 last\_contour\_right\_center = self.contour\_right\_center(i)  
 **else**:  
 contour\_left\_center = self.contour\_left\_center(i)  
 diff\_x = abs(last\_contour\_right\_center[0] - contour\_left\_center[0])  
 diff\_y = abs(last\_contour\_right\_center[1] - contour\_left\_center[1])  
 **if** debug:  
 print(**'Diff\_x:'**, diff\_x, **' Diff\_y:'**, diff\_y)  
  
 **if** diff\_y < height\_threshold: *# In one line* **if** diff\_x < width\_threshold: *# In one word* word.append(i)  
 **if** debug:  
 print(**'New letter found: '**, i)  
 **else**:  
 words.append(word)  
 word = [i]  
 **if** debug:  
 print(**'New word found: '**, word):

Лит

Лист

Листов

1

2

Группа: 181-311

# *Приложение D. Исходный код ImageSplitter.py*

Московский политехнический университет

Лит

№ докум.

Изм.

Подп.

Дата

<ФИО>

Разраб.

<ФИО препод.>

Пров.

Т. контр.

Н. контр.

<ФИО препод.>

Утв.

Инв. № подп

Подп. и дата

Взам. инв. №

Инв. № дубл.

Подп. и дата

**else**:  
 words.append(word)  
 **if** debug:  
 print(**'New line found: '**, words)  
 lines.append(words)  
 words = []  
 word = [i]  
 **if** debug:  
 print(**'New letter found: '**, i)  
 last\_contour\_right\_center = self.contour\_right\_center(i)  
  
 words.append(word)  
 lines.append(words)  
 **if** debug:  
 print(**'Last line: '**, words)  
  
 **return** lines  
  
**def** contour\_left\_center(self, index):  
 c\_x = self.contours[index][0][0][0]  
 c\_y = int((self.contours[index][0][0][1] + self.contours[index][3][0][1]) / 2)  
 **return** [c\_x, c\_y]  
  
**def** contour\_right\_center(self, index):  
 c\_x = self.contours[index][1][0][0]  
 c\_y = int((self.contours[index][0][0][1] + self.contours[index][3][0][1]) / 2)  
 **return** [c\_x, c\_y]

Лит

Лист

Листов

2

2

Группа: 181-311

# *Приложение D. Исходный код ImageSplitter.py*

Московский политехнический университет

Лит

№ докум.

Изм.

Подп.

Дата

<ФИО>

Разраб.

<ФИО препод.>

Пров.

Т. контр.

Н. контр.

<ФИО препод.>

Утв.

# 

Инв. № подп

Подп. и дата

Взам. инв. №

Инв. № дубл.

Подп. и дата

*"""  
 Licensed under the Unlicense License;  
 you may not use this file except in compliance with the License.  
 You may obtain a copy of the License at  
  
 https://unlicense.org  
  
 Unless required by applicable law or agreed to in writing, software  
 distributed under the License is distributed on an "AS IS" BASIS,  
 WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
 See the License for the specific language governing permissions and  
 limitations under the License.  
"""*

**import** pickle  
**import** random  
**import** sys  
**import** threading  
  
**import** numpy **as** np  
  
  
**def** sigmoid(x):  
 **return** 1 / (1 + np.exp(-x))

**def** dot\_0\_layer(input\_layer, synaptic\_weights):  
 **return** layer\_0\_activator(np.dot(input\_layer, synaptic\_weights.T))

**def** layer\_0\_activator(weights\_sum):  
 *# return max(0, weights\_sum)* result = [[0] \* weights\_sum[0]] \* weights\_sum  
 **for** i **in** range(len(weights\_sum)):  
 sample\_result = [0] \* weights\_sum[0]  
 **for** k **in** range(len(weights\_sum[i])):  
  
 threshold = 1.8 *# 1.79* **if** weights\_sum[i][k] >= threshold:  
 sample\_result[k] = weights\_sum[i][k] - 2.2 *# 0.79* **else**:  
 sample\_result[k] = 0  
  
 result[i] = sample\_result  
 result = np.array(result)**return** np.array(result)  
  
  
**def** progress(count, total, suffix=**''**):  
 bar\_len = 60  
 filled\_len = int(round(bar\_len \* count / float(total)))  
  
 percents = round(100.0 \* count / float(total), 1)  
 bar = **'='** \* filled\_len + **'-'** \* (bar\_len - filled\_len)  
  
 print(**'[%s] %s%s ...%s\r'** % (bar, percents, **'%'**, suffix))  
  
  
**class** Network:  
 **def** \_\_init\_\_(self, synaptic\_weights\_0=np.array([]), synaptic\_weights\_1=np.array([]),  
 images=np.array([]), labels=np.array([]),  
 train\_iterations=20, train\_output\_labels=4, debug=**True**):  
 self.synaptic\_weights\_0 = synaptic\_weights\_0  
 self.synaptic\_weights\_1 = synaptic\_weights\_1  
 self.images = images  
 self.labels = labels  
 self.train\_iterations = train\_iterations  
 self.debug = debug  
 self.train\_output\_labels = train\_output\_labels  
  
 **def** save\_weights(self, file):  
 **if** self.debug:  
 print(**'Saving weights...'**)  
 compressed\_data = [self.synaptic\_weights\_0, self.synaptic\_weights\_1]  
 **with** open(file, **'wb'**) **as** filehandle:  
 pickle.dump(compressed\_data, filehandle)  
 **if** self.debug:  
 print(**'Done. File'**, file, **'saved.'**)  
  
 **def** load\_weights(self, file):  
 **if** self.debug:  
 print(**'Loading weights...'**)  
 **with** open(file, **'rb'**) **as** filehandle:  
 compressed\_data = pickle.load(filehandle)  
 self.synaptic\_weights\_0 = np.array(compressed\_data[0])  
 self.synaptic\_weights\_1 = np.array(compressed\_data[1])  
 **if** self.debug:  
 print(**'Done. Weights from file'**, file, **'loaded.'**)

Лит

Лист

Листов

1

2

Группа: 181-311

# *Приложение E. Исходный код Network\_DIY.py*

Московский политехнический университет

Лит

№ докум.

Изм.

Подп.

Дата

<ФИО>

Разраб.

<ФИО препод.>

Пров.

Т. контр.

Н. контр.

<ФИО препод.>

Утв.

# 

Инв. № подп

Подп. и дата

Взам. инв. №

Инв. № дубл.

Подп. и дата

**def** start\_training(self):  
 **if** len(self.synaptic\_weights\_0) == 0 **or** len(self.synaptic\_weights\_1) == 0:  
 **if** self.debug:  
 print(**'Weights are empty! Generating...'**)  
 *# Genarate weights* images\_flatten\_size = len(self.images[0])  
 self.synaptic\_weights\_0 = []  
 **for** i **in** range(512): *# 4096* string\_array = [int(random.randrange(-1, 2)) **for** \_ **in** range(3)] \  
 + [0 **for** \_ **in** range(images\_flatten\_size - 3)]  
 random.shuffle(string\_array)  
 self.synaptic\_weights\_0.append(string\_array)  
 self.synaptic\_weights\_0 = np.array(self.synaptic\_weights\_0)  
 self.synaptic\_weights\_1 = np.array(2 \* np.random.random((self.train\_output\_labels, 512)) - 1) *# 4096* **if** self.debug:  
 print(**'-------------------- TRAIN DATA --------------------'**)  
 print(**'Shape of images: '** + str(self.images.shape))  
 print(**'Shape of labels: '** + str(self.labels.shape))  
 print(**'Arrays:'**)  
 print(self.images)  
 print()  
 print(self.labels)  
 print(**'--------------------- WEIGHTS ----------------------'**)  
 print(**'Shape of synaptic\_weights\_0: '** + str(self.synaptic\_weights\_0.shape))  
 print(**'Shape of synaptic\_weights\_1: '** + str(self.synaptic\_weights\_1.shape))  
 print(**'Arrays:'**)  
 print(self.synaptic\_weights\_0)  
 print()  
 print(self.synaptic\_weights\_1)  
 print(**'----------------------------------------------------'**)  
  
 *# TRAINING* thread = threading.Thread(target=self.training)  
 thread.start()  
  
**def** predict(self):  
 output\_l0 = self.images  
 output\_l1 = dot\_0\_layer(output\_l0, self.synaptic\_weights\_0)  
 output\_l2 = sigmoid(np.dot(output\_l1, self.synaptic\_weights\_1.T))  
 predicted\_classes = []  
 **for** output **in** output\_l2:  
 predicted\_classes.append(np.argmax(output))  
 **return** predicted\_classes  
  
**def** training(self):  
 *# noinspection PyBroadException* **try**:  
 i = 0  
 **while** i < self.train\_iterations:  
 output\_l0 = self.images  
  
 output\_l1 = dot\_0\_layer(output\_l0, self.synaptic\_weights\_0)  
 output\_l2 = sigmoid(np.dot(output\_l1, self.synaptic\_weights\_1.T))  
  
 *# Layer 2 error calculations* error\_l2 = []  
 **for** k **in** range(len(output\_l2)):  
 a = []  
 **for** m **in** range(self.train\_output\_labels):  
 **if** m == self.labels[0][k]:  
 a.append(1 - output\_l2[k][m])  
 **else**:  
 a.append(0 - output\_l2[k][m])  
 error\_l2.append(a)  
 error\_l2 = np.array(error\_l2)  
  
 adjustments\_l2 = output\_l1.T.dot(error\_l2 \* (output\_l2 \* (1 - output\_l2)))  
 self.synaptic\_weights\_1 += adjustments\_l2.T  
  
 *# accuracy calculations* predicted = []  
 accuracy = 0  
 **for** k **in** range(len(output\_l2)):  
 predicted.append(np.argmax(output\_l2[k]))  
 **if** np.argmax(output\_l2[k]) == self.labels[0][k]:  
 accuracy += 1  
 accuracy /= len(output\_l2)  
 predicted = np.array(predicted)  
  
 **if** i % 1 == 0 **and** self.debug:  
 print(**'iteration:'**, i)  
 print(**'predicted:'**, predicted)  
 print(**'accuracy:'**, accuracy)  
 print()  
 i += 1  
 progress(i, self.train\_iterations, **'Training...'**)  
  
 **if** self.debug:  
 print(**'Training done. Don\'t forget to save the weights!'**)  
  
 **except**:  
 print(sys.exc\_info())

Лит

Лист

Листов

2

2

Группа: 181-311

# *Приложение E. Исходный код Network\_DIY.py*

Московский политехнический университет

Лит

№ докум.

Изм.

Подп.

Дата

<ФИО>

Разраб.

<ФИО препод.>

Пров.

Т. контр.

Н. контр.

<ФИО препод.>

Утв.

# 

Инв. № подп

Подп. и дата

Взам. инв. №

Инв. № дубл.

Подп. и дата

*"""  
 Licensed under the Unlicense License;  
 you may not use this file except in compliance with the License.  
 You may obtain a copy of the License at  
  
 https://unlicense.org  
  
 Unless required by applicable law or agreed to in writing, software  
 distributed under the License is distributed on an "AS IS" BASIS,  
 WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
 See the License for the specific language governing permissions and  
 limitations under the License.  
"""*

**import** threading  
  
**import** numpy **as** np  
**import** cv2  
**import** sys  
**from** PyQt5 **import** QtWidgets  
**from** PyQt5.QtGui **import** QPixmap  
**import** qimage2ndarray  
**import** SAM\_worker  
  
**import** gui\_ep  
  
PROJECT\_FOLDER = **'EP\_data\_SAM'  
  
  
class** EngineeringProject(QtWidgets.QMainWindow, gui\_ep.Ui\_MainWindow):  
 **def** \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.setupUi(self)  
 self.btn\_camera\_start.clicked.connect(self.camera\_start)  
 self.btn\_camera\_stop.clicked.connect(self.camera\_stop)  
 self.btn\_image\_save.clicked.connect(self.image\_save)  
 self.btn\_image\_load.clicked.connect(self.image\_load)  
 self.btn\_split\_and\_merge.clicked.connect(self.split\_and\_merge)  
  
 self.cv\_cap = **None** self.current\_frame = **None** self.loaded\_image = **None** self.camera\_running = **False** self.image\_running = **False  
  
 def** camera\_start(self):  
 self.camera\_running = **True  
 if** self.check\_dshow.isChecked():  
 self.cv\_cap = cv2.VideoCapture(self.camera\_id.value(), cv2.CAP\_DSHOW)  
 **else**:  
 self.cv\_cap = cv2.VideoCapture(self.camera\_id.value())  
 thread = threading.Thread(target=self.cv\_thread)  
 thread.start()  
  
 **def** camera\_stop(self):  
 self.camera\_running = **False** self.image\_running = **False  
  
 def** image\_save(self):  
 **if** self.camera\_running:  
 cv2.imwrite(PROJECT\_FOLDER + **'/image.png'**, self.current\_frame)  
 **else**:  
 print(**'Camera not running!'**)  
  
 **def** image\_load(self):  
 self.image\_running = **True** self.loaded\_image = cv2.imread(PROJECT\_FOLDER + **'/image.png'**)  
 thread = threading.Thread(target=self.cv\_thread)  
 thread.start()  
  
 **def** cv\_thread(self):  
 **while** self.camera\_running **or** self.image\_running:  
 img = np.zeros((640, 480, 3), dtype=np.uint8)  
 **if** self.camera\_running:  
 ret, img = self.cv\_cap.read()  
 **elif** self.image\_running:  
 img = self.loaded\_image  
 self.current\_frame = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)  
 self.cvl\_image.setPixmap(QPixmap.fromImage(qimage2ndarray.array2qimage(self.current\_frame)))  
 self.cv\_cap.release()  
  
 **def** split\_and\_merge(self):  
 self.loaded\_image = cv2.imread(PROJECT\_FOLDER + **'/image.png'**)  
 self.current\_frame = cv2.cvtColor(self.loaded\_image, cv2.COLOR\_BGR2GRAY)  
 sam\_worker = SAM\_worker.SAM(self.current\_frame)  
 self.cvl\_image.setPixmap(QPixmap.fromImage(qimage2ndarray.array2qimage(sam\_worker.image)))  
 sam\_worker.split\_and\_merge()  
 self.cvl\_image\_2.setPixmap(QPixmap.fromImage(qimage2ndarray.array2qimage(sam\_worker.debug)))  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 app = QtWidgets.QApplication(sys.argv)  
 app.setStyle(**"fusion"**)  
 window = EngineeringProject()  
 window.show()  
 app.exec\_()

Лит

Лист

Листов

1

1

Группа: 181-311

# *Приложение G. Исходный код SplitAndMerge.py*

Московский политехнический университет

Лит

№ докум.

Изм.

Подп.

Дата

<ФИО>

Разраб.

<ФИО препод.>

Пров.

Т. контр.

Н. контр.

<ФИО препод.>

Утв.

# 

Инв. № подп

Подп. и дата

Взам. инв. №

Инв. № дубл.

Подп. и дата

*"""  
 Licensed under the Unlicense License;  
 you may not use this file except in compliance with the License.  
 You may obtain a copy of the License at  
  
 https://unlicense.org  
  
 Unless required by applicable law or agreed to in writing, software  
 distributed under the License is distributed on an "AS IS" BASIS,  
 WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
 See the License for the specific language governing permissions and  
 limitations under the License.  
"""*

**import** random  
**import** threading  
  
**import** cv2  
**import** numpy **as** np  
  
  
**def** progress(count, total, suffix=**''**):  
 bar\_len = 60  
 filled\_len = int(round(bar\_len \* count / float(total)))  
  
 percents = round(100.0 \* count / float(total), 1)  
 bar = **'='** \* filled\_len + **'-'** \* (bar\_len - filled\_len)  
  
 print(**'[%s] %s%s ...%s\r'** % (bar, percents, **'%'**, suffix))  
  
  
**class** SAM:  
 **def** \_\_init\_\_(self, image):  
 self.image = image *# Input grayscale image* self.im\_height = self.image.shape[0] *# Height of input image* self.im\_width = self.image.shape[1] *# Width of input image* self.regions = np.zeros((self.im\_height, self.im\_width), dtype=np.uint8)  
 self.debug = np.zeros((self.im\_height, self.im\_width), dtype=np.uint8)  
 self.checked\_surface = np.zeros((self.im\_height, self.im\_width), dtype=np.uint8)  
 *# np.zeros((self.im\_height, self.im\_width, 3), dtype=np.uint8)* self.threshold = 120  
 self.checked\_surface\_mean = 0  
 self.checked\_surface\_mean\_last = 0  
 self.contours = []  
  
 **def** split\_and\_merge(self):  
 self.regions = np.zeros((self.im\_height, self.im\_width), dtype=np.uint8)  
 self.contours = []  
  
 *# Splitting* print(**'Splitting ...'**)  
 self.recursive\_splitting([0, 0], [self.im\_width - 1, 0],  
 [self.im\_width - 1, self.im\_height - 1], [0, self.im\_height - 1])  
 self.contours = np.array(self.contours)  
 print(**'Splitting done!'**)  
  
 *# Merging* print(**'Merging ...'**)  
 self.checked\_surface = np.zeros((self.im\_height, self.im\_width), dtype=np.uint8)  
 cv2.drawContours(self.regions, self.contours, -1, 255)  
 **for** i **in** range(6):  
 thread = threading.Thread(target=self.graphical\_merge)  
 thread.start()  
 **while** np.mean(self.checked\_surface) < 255:  
 self.debug = cv2.cvtColor(self.image.copy(), cv2.COLOR\_GRAY2BGR)  
 regions\_rgb = cv2.merge(  
 [np.zeros((self.im\_height, self.im\_width), dtype=np.uint8), self.regions,  
 np.zeros((self.im\_height, self.im\_width), dtype=np.uint8)])  
 self.debug = cv2.add(self.debug, regions\_rgb)  
  
 cv2.imshow(**'checked\_surface'**, self.checked\_surface)  
 cv2.imshow(**'debug'**, self.debug)  
 cv2.waitKey(1)  
 print(**'Merging done!'**)  
  
 *# Finding contours  
 # print('Finding contours ...')  
 # self.found\_contours()  
 # print('Done! Threshold: ' + str(self.threshold) + ' Contours: ' + str(len(self.contours)))  
  
 # self.contours, hierarchy = cv2.findContours(self.regions, cv2.RETR\_LIST, cv2.CHAIN\_APPROX\_SIMPLE)* **def** recursive\_splitting(self, tl, tr, br, bl): *# tl[0] = X (height), tl[1] = Y (width)* segment = self.image[tl[1]: br[1], tl[0]: br[0]]  
  
 **if** segment.max() - segment.min() > self.threshold:  
 **if** segment.shape[0] >= 6 **and** segment.shape[1] >= 8:

Лит

Лист

Листов

1

3

Группа: 181-311

# *Приложение H. Исходный код SAM\_worker.py*

Московский политехнический университет

Лит

№ докум.

Изм.

Подп.

Дата

<ФИО>

Разраб.

<ФИО препод.>

Пров.

Т. контр.

Н. контр.

<ФИО препод.>

Утв.

Инв. № подп

Подп. и дата

Взам. инв. №

Инв. № дубл.

Подп. и дата

tl\_tr\_mid = [int((tr[0] + tl[0]) / 2), tl[1]]  
 tr\_br\_mid = [tr[0], int((br[1] + tr[1]) / 2)]  
 br\_bl\_mid = [int((br[0] + bl[0]) / 2), br[1]]  
 bl\_tl\_mid = [bl[0], int((bl[1] + tl[1]) / 2)]  
 all\_mid = [tl\_tr\_mid[0], bl\_tl\_mid[1]]  
  
 self.recursive\_splitting(tl, tl\_tr\_mid, all\_mid, bl\_tl\_mid) *# TL* self.recursive\_splitting(tl\_tr\_mid, tr, tr\_br\_mid, all\_mid) *# TR* self.recursive\_splitting(all\_mid, tr\_br\_mid, br, br\_bl\_mid) *# BR* self.recursive\_splitting(bl\_tl\_mid, all\_mid, br\_bl\_mid, bl) *# BL* **else**:  
 self.contours.append([tl, tr, br, bl])  
  
 **else**:  
 self.contours.append([tl, tr, br, bl])  
  
**def** graphical\_merge(self):  
 kernel\_full = np.ones((3, 3), np.uint8)  
 kernel\_big = np.ones((5, 5), np.uint8)  
  
 self.checked\_surface = np.zeros((self.im\_height, self.im\_width), dtype=np.uint8)  
 expansion\_seed = [1, 1]  
 self.checked\_surface\_mean\_last = 0  
 **while True**:  
 filled = self.regions.copy()  
 cv2.floodFill(filled, **None**, (expansion\_seed[0], expansion\_seed[1]), 127)  
 fill\_mask = cv2.inRange(filled, 127, 127)  
 region\_mask = cv2.dilate(fill\_mask, kernel\_full)  
 combined\_region\_mask = region\_mask.copy()  
  
 contours, hierarchy = cv2.findContours(cv2.dilate(fill\_mask, kernel\_big), cv2.RETR\_LIST,  
 cv2.CHAIN\_APPROX\_SIMPLE)  
  
 contours = np.array(contours[0])  
 horizontal\_expansion = []  
 vertical\_expansion = []  
  
 **for** i **in** range(len(contours)):  
 dot\_1\_x = contours[i][0][0]  
 dot\_2\_x = contours[i + 1][0][0] **if** i < len(contours) - 1 **else** contours[0][0][0]  
 dot\_1\_y = contours[i][0][1]  
 dot\_2\_y = contours[i + 1][0][1] **if** i < len(contours) - 1 **else** contours[0][0][1]  
 **if** dot\_1\_x == dot\_2\_x **and** dot\_1\_x != self.image.shape[1] - 1:  
 *# Vertical* **for** y **in** range(min(dot\_1\_y, dot\_2\_y), max(dot\_1\_y, dot\_2\_y)):  
 **if** self.regions[y, dot\_1\_x] == 255:  
 vertical\_expansion.append([dot\_1\_x, y])  
 **else**:  
 *# Horizontal* **for** x **in** range(min(dot\_1\_x, dot\_2\_x), max(dot\_1\_x, dot\_2\_x)):  
 **if** self.regions[dot\_1\_y, x] == 255:  
 horizontal\_expansion.append([x, dot\_1\_y])  
  
 last\_y = -1  
 **for** i **in** range(len(horizontal\_expansion)):  
 **if** horizontal\_expansion[i][1] <= 0 **or** horizontal\_expansion[i][1] >= self.image.shape[0] - 1:  
 **continue  
 elif** horizontal\_expansion[i][1] != last\_y:  
 last\_y = horizontal\_expansion[i][1]  
 **else**:  
 diff\_x = int((horizontal\_expansion[i - 1][0] + horizontal\_expansion[i][0]) / 2)  
  
 *# Create new region around horizonal lines* new\_region = self.regions.copy()  
 cv2.floodFill(new\_region, **None**, (diff\_x, last\_y), 127)  
 new\_region = cv2.dilate(cv2.inRange(new\_region, 127, 127), kernel\_full)  
 region\_max = cv2.bitwise\_and(region\_mask, self.image).max()  
 region\_min = cv2.bitwise\_or(cv2.bitwise\_and(region\_mask, self.image),  
 cv2.bitwise\_not(region\_mask)).min()  
 new\_region\_max = cv2.bitwise\_and(new\_region, self.image).max()  
 new\_region\_min = cv2.bitwise\_or(cv2.bitwise\_and(new\_region, self.image),  
 cv2.bitwise\_not(new\_region)).min()  
  
 **if** (new\_region\_max - region\_min <= self.threshold  
 **and** region\_max - new\_region\_min <= self.threshold):  
 combined\_region\_mask = cv2.bitwise\_or(combined\_region\_mask, new\_region)  
  
 last\_x = -1  
 **for** i **in** range(len(vertical\_expansion)):  
 **if** vertical\_expansion[i][0] <= 0 **or** vertical\_expansion[i][0] >= self.image.shape[1] - 1:  
 **continue  
 elif** vertical\_expansion[i][0] != last\_x:  
 last\_x = vertical\_expansion[i][0]  
 **else**:  
 diff\_y = int((vertical\_expansion[i - 1][1] + vertical\_expansion[i][1]) / 2)  
  
 *# Create new region around vertical lines* new\_region = self.regions.copy()  
 cv2.floodFill(new\_region, **None**, (last\_x, diff\_y), 127)  
 new\_region = cv2.dilate(cv2.inRange(new\_region, 127, 127), kernel\_full)  
 region\_max = cv2.bitwise\_and(region\_mask, self.image).max()  
 region\_min = cv2.bitwise\_or(cv2.bitwise\_and(region\_mask, self.image),  
 cv2.bitwise\_not(region\_mask)).min()  
 new\_region\_max = cv2.bitwise\_and(new\_region, self.image).max()  
 new\_region\_min = cv2.bitwise\_or(cv2.bitwise\_and(new\_region, self.image),  
 cv2.bitwise\_not(new\_region)).min()

Лит

Лист

Листов

2

3

Группа: 181-311

# *Приложение H Исходный код SAM\_worker.py*

Московский политехнический университет

Лит

№ докум.

Изм.

Подп.

Дата

<ФИО>

Разраб.

<ФИО препод.>

Пров.

Т. контр.

Н. контр.

<ФИО препод.>

Утв.

# 

Инв. № подп

Подп. и дата

Взам. инв. №

Инв. № дубл.

Подп. и дата

**"""  
 if (new\_region\_max - region\_min <= self.merge\_threshold  
 and region\_max - new\_region\_min <= self.merge\_threshold) \  
 or (np.mean(new\_region) < 0.03 or np.mean(region\_mask) < 0.03):  
 """** self.checked\_surface = cv2.bitwise\_or(self.checked\_surface, combined\_region\_mask)  
  
 combined\_region\_mask = cv2.erode(combined\_region\_mask, kernel\_full)  
  
 self.regions = cv2.subtract(self.regions, combined\_region\_mask)  
  
 self.checked\_surface\_mean = np.mean(self.checked\_surface)  
 **if** self.checked\_surface\_mean == 255:  
 **break  
 elif** self.checked\_surface\_mean\_last != self.checked\_surface\_mean:  
 self.checked\_surface\_mean\_last = self.checked\_surface\_mean  
 expansion\_seed\_founded = **True  
 else**:  
 expansion\_seed\_founded = **False  
  
 while not** expansion\_seed\_founded:  
 expansion\_seed = [random.randrange(self.image.shape[1]), random.randrange(self.image.shape[0])]  
 **if not** self.checked\_surface[expansion\_seed[1], expansion\_seed[0]] \  
 **and not** self.regions[expansion\_seed[1], expansion\_seed[0]]:  
 expansion\_seed\_founded = **True  
 """  
 for ex\_y in range(self.image.shape[0]):  
 if expansion\_seed\_founded:  
 break  
 for ex\_x in range(self.image.shape[1]):  
 if not self.checked\_surface[ex\_y, ex\_x] and not self.regions[ex\_y, ex\_x]:  
 expansion\_seed = [ex\_x, ex\_y]  
 expansion\_seed\_founded = True  
 break  
 """** cv2.destroyAllWindows()  
  
**def** found\_contours(self):  
 self.contours = []  
 kernel\_full = np.ones((3, 3), np.uint8)  
  
 expansion\_seed = [1, 1]  
 **while True**:  
 filled = self.regions.copy()  
 cv2.floodFill(filled, **None**, (expansion\_seed[0], expansion\_seed[1]), 127)  
 fill\_mask = cv2.inRange(filled, 127, 127)  
 region\_mask = cv2.dilate(fill\_mask, kernel\_full)  
  
 contours, hierarchy = cv2.findContours(region\_mask, cv2.RETR\_LIST, cv2.CHAIN\_APPROX\_SIMPLE)  
  
 self.contours.append(np.array(contours[0]))  
  
 self.checked\_surface = cv2.bitwise\_or(self.checked\_surface, region\_mask)  
  
 self.checked\_surface\_mean = np.mean(self.checked\_surface)  
 **if** self.checked\_surface\_mean == 255:  
 **break** expansion\_seed\_founded = **False  
  
 while not** expansion\_seed\_founded:  
 expansion\_seed = [random.randrange(self.image.shape[1]), random.randrange(self.image.shape[0])]  
 **if not** self.checked\_surface[expansion\_seed[1], expansion\_seed[0]] \  
 **and not** self.regions[expansion\_seed[1], expansion\_seed[0]]:  
 expansion\_seed\_founded = **True  
 """  
 for ex\_y in range(self.image.shape[0]):  
 if expansion\_seed\_founded:  
 break  
 for ex\_x in range(self.image.shape[1]):  
 if not self.checked\_surface[ex\_y, ex\_x] and not self.regions[ex\_y, ex\_x]:  
 expansion\_seed = [ex\_x, ex\_y]  
 expansion\_seed\_founded = True  
 break  
 """** cv2.imshow(**'self.checked\_surface'**, self.checked\_surface)  
 cv2.waitKey(1)  
  
 cv2.destroyAllWindows()

Лит

Лист

Листов

3

3

Группа: 181-311

# *Приложение H. Исходный код SAM\_worker.py*

Московский политехнический университет

Лит

№ докум.

Изм.

Подп.

Дата

<ФИО>

Разраб.

<ФИО препод.>

Пров.

Т. контр.

Н. контр.

<ФИО препод.>

Утв.