



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

ФАКУЛЬТЕТ «Информатика, искусственный интеллект и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

ОТЧЕТ
по лабораторной работе № 6
по курсу «Основы искусственного интеллекта»
на тему: «Компьютерное зрение»

Студент ИУ7-13М
(Группа)

(Подпись, дата)

Орду М. А.
(И. О. Фамилия)

Преподаватель

(Подпись, дата)

Строганов Ю. В.
(И. О. Фамилия)

СОДЕРЖАНИЕ

1	Теоретическая часть	4
2	Практическая часть	5
2.1	Методика	5
2.2	Результаты	6
	ПРИЛОЖЕНИЕ А Код программы для EasyOCR	10
	ПРИЛОЖЕНИЕ Б Код программы для TesseractOCR . . .	23
	ПРИЛОЖЕНИЕ В Код программы для PaddleOCR	32

ВВЕДЕНИЕ

Целью данной лабораторной работы является сравнительный анализ трёх локально разворачиваемых библиотек оптического распознавания символов (OCR), поддерживающих русский язык, при работе с рукописным текстом. Исследование проводилось на наборе данных “Тест для французских булочек и чаю”, содержащем изображения рукописных текстов. Актуальность работы обусловлена растущей потребностью в автоматизации извлечения информации из рукописных источников в различных областях. В качестве объектов сравнения выбраны библиотеки: PyTesseract, EasyOCR и PaddleOCR с моделью `cyrillic_PP-OCRv5_mobile_rec`.

1 Теоретическая часть

Оптическое распознавание символов (OCR) — это технология преобразования изображений печатного или рукописного текста в машиночитаемый текст. Современные системы OCR используют глубокое обучение, в частности, сверточные нейронные сети для детекции текста и рекуррентные нейронные сети для его распознавания.

- **TesseractOCR** – открытая библиотека, изначально разработанная Hewlett-Packard. В настоящее время поддерживается Google. Использует LSTM-сети. Позволяет работать с множеством языков, включая русский.
- **EasyOCR** – библиотека на основе PyTorch, использующая архитектуры CRAFT для детекции текста и CRNN (CNN + BiLSTM) с механизмом внимания для распознавания. Поддерживает более 80 языков, включая русский.
- **PaddleOCR** – фреймворк от Baidu, предлагающий современные предобученные модели. Модель `cyrillic_PP-OCRv5_mobile_rec`, используемая в рамках работы, оптимизирована для распознавания кириллицы.

Основная метрика для оценки качества OCR – точность распознавания – отношение правильно распознанных символов к общему количеству символов в эталонном тексте.

2 Практическая часть

2.1 Методика

В качестве тестового набора данных использован “Тест для французских булочек и чаю.zip”, содержащий изображения рукописных текстов на русском языке.

На каждой странице рукописного текста имеются метки позиционирования, пример которой представлен на рисунке 2.1, однако использовать их для увеличения точности распознавания не удалось. Эти метки позиционирования система распознавания ошибочно интерпретировала как символы текста. Это снижало общую точность распознавания. Для устранения данной проблемы было принято решение предварительно удалять эти метки с изображений перед обработкой.

Стоит отметить, что основная задача этих меток - определение положения текста в пространстве - может быть выполнена всеми библиотеками, примененными в ходе данной работы, исключительно анализом текста, т.е. необходимости применения подобных меток в рамках данной работы нет.



Рисунок 2.1 – Метка позиционирования, распознанная как символ ©

Для каждого файла выполнялись следующие шаги:

1. Предобработка изображения.

2. Подача на вход каждого изображения, заранее обрезанного таким образом, чтобы в изображении остался только текст без меток позиционирования.
3. Извлечение распознанного текста.
4. Сравнение полученного текста с эталонным.

2.2 Результаты

Распознавание текста, написанного низкоконтрастными чернилами, ожидаемо плохо выполнялось всеми тремя моделями. Результаты распознавания представлены на рисунках 2.2 и 2.3.



Рисунок 2.2 – Результат разпознавания текста, полученный моделью `cyrillic_PP-OCRv5_mobile_rec`

- **PyTesseract** показал среднюю точность при работе с рукописным текстом. Библиотека часто ошибалась в распознавании похожих по начертанию символов кириллицы (например, “и” и “н”, “л” и “п”).
- **EasyOCR** точность распознавания была выше, чем у Tesseract, однако библиотека требовала больше времени на обработку. Показал лучшие результаты среди рассмотренных трех моделей.

- PaddleOCR с моделью `cyrillic_PP-OCRv5_mobile_rec` показал худшие результаты среди рассмотренных трех моделей. Стоит отметить, что несмотря на то, что модель обучена на тексте с кириллицей, результатом распознавания отдельных символов являлись латинские буквы. Так буква Ш, часто распознавалась как W.

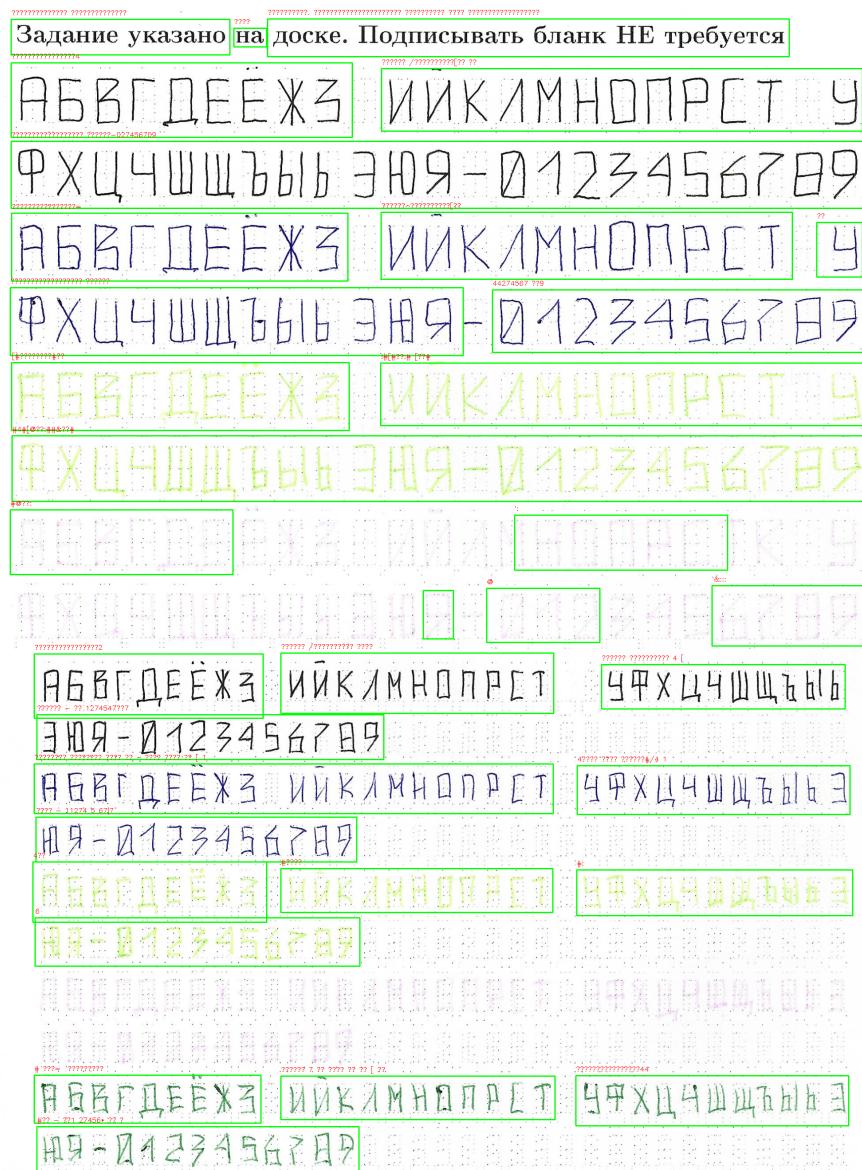


Рисунок 2.3 – Результат распознавания текста, полученный библиотекой EasyOCR

Все изображения перед проведением процедуры распознавания текста были бинаризированы, как представлено на рисунке 2.4.

Задание указано на доске. Подписывать бланк НЕ требуется

АБВГДЕЁЖЗ ИЙКЛМНОПРСТ Ч
ФХЦЧЩЬЫЬ ЭЮЯ-0123456789
АБВГДЕЁЖЗ ИЙКЛМНОП РСТУФХЦЧЩЬЫЬ Э
ЮЯ-0123456789
АБВГДЕЁЖЗИЙКЛМНОП РСТУФХЦЧЩЬЫЬ Э
ЮЯ-0123456789
АБВГДЕЁЖЗИЙКЛМНОП РСТУФХЦЧЩЬЫЬ Э
ЮЯ-0123456789
АБВГДЕЁЖЗИЙКЛМНОП РСТУФХЦЧЩЬЫЬ Э
ЮЯ-0123456789
АБВГДЕЁЖЗИЙКЛМНОП РСТУФХЦЧЩЬЫЬ Э
ЮЯ-0123456789

Рисунок 2.4 – Бинаризированное изображение рукописного текста

Сравнительная таблица рассмотренных библиотек представлена в таблице 2.1.

Таблица 2.1 – Сравнительная таблица библиотек

Критерий	PyTesseract	EasyOCR	PaddleOCR
Средняя символьная точность	53%	55%	45%

ЗАКЛЮЧЕНИЕ

В данной работе был проведён сравнительный анализ трёх библиотек для оптического распознавания текста: PyTesseract, EasyOCR и PaddleOCR. Тестирование выполнялось на наборе изображений рукописных текстов на русском языке.

Результаты эксперимента показали, что библиотека EasyOCR показала наилучшую точность распознавания (55%), опередив PyTesseract (53%) и PaddleOCR (45%). PaddleOCR, несмотря на специализированную кириллическую модель, демонстрировал характерные ошибки, такие как замена кириллических символов на латинские.

Основные выводы исследования:

1. Все рассмотренные библиотеки испытывают значительные трудности с распознаванием рукописного текста, что указывает на сложность самой задачи.
2. Качество распознавания критически зависит от предобработки изображений. Удаление посторонних элементов и бинаризация являются необходимыми этапами.
3. Для задач локального распознавания русской рукописи библиотека EasyOCR является наиболее предпочтительным выбором из рассмотренных.

Как правило, обучение моделей машинного обучения для распознавания рукописного текста проводят на словах и предложениях, а не просто на совокупности букв. Этим можно объяснить низкую точность распознавания, полученную в ходе работы.

ПРИЛОЖЕНИЕ А

Код программы для EasyOCR

Листинг А.1 – Исходный код программы

```
1 import cv2
2 import numpy as np
3 from jiwer import cer, wer
4 from difflib import SequenceMatcher
5 import os
6 import glob
7 import pandas as pd
8 import easyocr
9 import warnings
10 from typing import List, Tuple
11
12 # Игнорируем предупреждения от EasyOCR
13 warnings.filterwarnings('ignore')
14
15 # Инициализация EasyOCR для русского языка
16 reader = easyocr.Reader(['ru'], gpu=False) # Установите
17     gpu=True если есть поддержка CUDA
18
19 # Папка с изображениями
20 FOLDER_PATH = "1"
21
22 def crop_by_marks(image, offset=50):
23     """
24         Обрезает изображение, удаляя верхнюю левую метку
25             позиционирования.
26
27     Args:
28         image: исходное изображение
29         offset: дополнительный отступ после метки
30             (положительный = отступ от метки)
31
32     Returns:
33         cropped_image: обрезанное изображение
34         mark_coords: координаты найденной метки (x, y, w, h)
35     """
36
37     # Создаем копию изображения
38     original = image.copy()
```

```

35
36     # Конвертируем в grayscale
37     gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
38     h, w = gray.shape
39
40     # Бинаризуем изображение
41     _, binary = cv2.threshold(gray, 0, 255,
42                               cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)
43
44     # Находим контуры
45     contours, _ = cv2.findContours(binary, cv2.RETR_EXTERNAL,
46                                    cv2.CHAIN_APPROX_SIMPLE)
47
48     marks = []
49
50     # Определяем область поиска для верхней левой метки
51     search_area_w = w // 2
52     search_area_h = h // 2
53
54     for cnt in contours:
55         x, y, cw, ch = cv2.boundingRect(cnt)
56         area = cv2.contourArea(cnt)
57
58         # Фильтр по размеру
59         if 200 < area < 3000:
60             # Фильтр по положению: только в верхней левой части
61             if x < search_area_w and y < search_area_h:
62                 marks.append((x, y, cw, ch, area))
63
64     if not marks:
65         print("Метки в верхнем левом углу не найдены,
66               возвращаю оригинальное изображение")
67     return original, []
68
69     marks.sort(key=lambda m: (m[1], m[0]))
70
71     # Выбираем самую верхнюю левую метку
72     x, y, mark_w, mark_h, area = marks[0]
73
74     # Вычисляем координаты для обрезки
75     crop_start_x = x + mark_w + offset

```

```

73     crop_start_y = y + mark_h + offset
74
75     # Обрезаем изображение
76     roi = image[crop_start_y:, crop_start_x:]
77
78     if roi.size == 0:
79         print("ROI пустой, возвращаю оригинальное изображение")
80         return original, [(x, y, mark_w, mark_h)]
81
82     return roi, [(x, y, mark_w, mark_h)]
83
84 def draw_marks(image, marks, color=(0, 255, 0), thickness=2):
85     debug_img = image.copy()
86     for x, y, w, h in marks:
87         cv2.rectangle(debug_img, (x, y), (x + w, y + h),
88                       color, thickness)
89     return debug_img
90
91 def char_accuracy(gt, pred):
92     return SequenceMatcher(None, gt, pred).ratio()
93
94 def extract_text_lines_with_easyocr(image) -> List[str]:
95     """
96     Извлекает текст из изображения с помощью EasyOCR, сохраняя
97     построчную структуру.
98
99     Args:
100        image: изображение в формате BGR
101
102     Returns:
103        List[str]: список строк распознанного текста
104     """
105
106     # Получаем детальные результаты от EasyOCR (с координатами)
107     results = reader.readtext(image, paragraph=False, detail=1)
108
109     # Если результатов нет, возвращаем пустой список
110     if not results:
111         return []
112
113     # Группируем текст по строкам на основе Y-координаты
114     bounding_box

```

```

111     lines_dict = {}
112
113     for bbox, text, confidence in results:
114         # Вычисляем среднюю Y-координату bounding box
115         y_coords = [point[1] for point in bbox]
116         avg_y = sum(y_coords) / len(y_coords)
117
118         # Находим ближайшую группу строк (в пределах порога)
119         found_group = None
120         for group_y in lines_dict.keys():
121             if abs(avg_y - group_y) < 15: # Порог для
122                 объединения в одну строку
123                 found_group = group_y
124                 break
125
126         if found_group is not None:
127             # Добавляем к существующей строке
128             lines_dict[found_group].append((bbox, text, avg_y))
129         else:
130             # Создаем новую строку
131             lines_dict[avg_y] = [(bbox, text, avg_y)]
132
133         # Сортируем строки по Y-координате (сверху вниз)
134         sorted_groups = sorted(lines_dict.items(), key=lambda x:
135             x[0])
136
137         # Для каждой группы сортируем слова по X-координате (слева
138             направо)
139         lines = []
140         for group_y, elements in sorted_groups:
141             # Сортируем элементы в строке по X-координате
142             elements_sorted = sorted(elements, key=lambda elem:
143                 min(point[0] for point in elem[0]))
144
145             # Объединяем слова в строку
146             line_text = ', '.join([text for _, text, _ in
147                 elements_sorted])
148             lines.append(line_text.strip())
149
150     return lines

```

```

147 def process_image(image_path, save_intermediate=False):
148     print(f"\n{'='*50}")
149     print(f"Обработка файла: {os.path.basename(image_path)}")
150     print('='*50)
151
152     img = cv2.imread(image_path)
153     if img is None:
154         print(f"Ошибка: Не удалось загрузить изображение
155             {image_path}")
156         return None
157
158     # Обрезка по меткам
159     roi, marks = crop_by_marks(img)
160
161     if save_intermediate:
162         os.makedirs("debug", exist_ok=True)
163         base_name =
164             os.path.splitext(os.path.basename(image_path))[0]
165
166         draw = draw_marks(img, marks)
167         cv2.imwrite(f"debug/{base_name}_marks_easyocr.png",
168                     draw)
169
170     if roi is None or roi.size == 0:
171         print("Ошибка: ROI пустой, используем оригинальное
172             изображение")
173         roi = img.copy()
174
175     # Пробуем разные варианты предобработки для лучшего
176     # распознавания
177     lines_list = []
178
179     # Вариант 1: оригинальное изображение
180     print("Распознавание на оригинальном изображении... ")
181     lines1 = extract_text_lines_with_easyocr(roi)
182     lines_list.append(("original", lines1))
183
184     # Вариант 2: предобработанное бинарное изображение
185     gray = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
186     gray = cv2.blur(gray, (3, 3)) # Уменьшаем blur для
187         сохранения деталей

```

```

182     _, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY
183                             + cv2.THRESH_OTSU)
184
185     print("Распознавание на бинарном изображении...")
186     lines2 = extract_text_lines_with_easyocr(processed_img)
187     lines_list.append(("binary", lines2))
188
189     # Вариант 3: адаптивная бинаризация
190     adaptive = cv2.adaptiveThreshold(gray, 255,
191                                       cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
192                                       cv2.THRESH_BINARY, 11, 2)
193
194     adaptive_img = cv2.cvtColor(adaptive, cv2.COLOR_GRAY2BGR)
195
196     print("Распознавание на адаптивно бинаризованном
197           изображении...")
198
199     lines3 = extract_text_lines_with_easyocr(adaptive_img)
200     lines_list.append(("adaptive", lines3))
201
202     # Выбираем вариант с максимальным количеством строк
203     best_variant = max(lines_list, key=lambda x: len(x[1]))
204     print(f"Выбран вариант: {best_variant[0]} с
205           {len(best_variant[1])} строками")
206
207     lines = best_variant[1]
208
209     if save_intermediate:
210         cv2.imwrite(f"debug/{base_name}_binary_easyocr.png",
211                     thresh)
212         cv2.imwrite(f"debug/{base_name}_adaptive_easyocr.png",
213                     adaptive)
214         cv2.imwrite(f"debug/{base_name}_roi_easyocr.png", roi)
215
216         # Сохраняем изображение с bounding boxes для отладки
217         debug_img = roi.copy()
218         results = reader.readtext(roi, paragraph=False,
219                               detail=1)
220
221         for bbox, text, confidence in results:
222             # Конвертируем bounding box в нужный формат
223             bbox = np.array(bbox, dtype=np.int32)
224             cv2.polylines(debug_img, [bbox], True, (0, 255,

```

```

0) , 2)
# Добавляем текст
cv2.putText(debug_img, text, (bbox[0][0],
                                bbox[0][1] - 10),
            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0,
                                             255), 1)
cv2.imwrite(f"debug/{base_name}_bboxes_easyocr.png",
            debug_img)

# Сохраняем распознанный текст
raw_text = '\n'.join(lines)
text_file = os.path.splitext(image_path)[0] + "_easyocr" +
".txt"
with open(text_file, "w", encoding="utf-8") as file:
    file.write(raw_text)

# Эталонные строки
expected_lines = []
expected_lines.append("Задание указано на доске.
Подписывать бланк НЕ требуется")

for _ in range(5):
    expected_lines.append("АБВГДЕЁЖЗ ИЙКЛМНОПРСТ У")
    expected_lines.append("ФХЦЧЩЫЫЭЮЯ -0123456789")

for _ in range(5):
    expected_lines.append("АБВГДЕЁЖЗИЙ КЛМНОПРСТ
УФХЦЧЩЫЫЭ")
    expected_lines.append("ЮЯ -0123456789")

# Вычисление метрик
acc_list, cer_list, wer_list = [], [], []
line_results = []

# Для каждой ожидаемой строки находим наиболее подходящую
# распознанную
for i, gt in enumerate(expected_lines):
    best_match = ""
    best_cer = float('inf')

    # Ищем лучшую совпадающую строку среди распознанных

```

```

249     for pred in lines:
250         if pred: # Пропускаем пустые строки
251             current_cer = cer(gt, pred)
252             if current_cer < best_cer:
253                 best_cer = current_cer
254                 best_match = pred
255
256             # Если нашли подходящую строку, используем ее
257             if best_match and best_cer < 0.8: # Порог для
258                 принятия решения
259                 pred = best_match
260             else:
261                 pred = "" # Не нашли подходящую строку
262
263             acc = char_accuracy(gt, pred)
264             cer_val = cer(gt, pred)
265             wer_val = wer(gt, pred)
266
267             acc_list.append(acc)
268             cer_list.append(cer_val)
269             wer_list.append(wer_val)
270
271             line_results.append({
272                 'file': os.path.basename(image_path),
273                 'line_num': i,
274                 'gt': gt,
275                 'pred': pred,
276                 'accuracy': acc,
277                 'cer': cer_val,
278                 'wer': wer_val,
279                 'matched': 1 if pred else 0
280             })
281
282             print(f"Строка {i:2d} | ACC: {acc:.3f} | CER:
283                 {cer_val:.3f} | WER: {wer_val:.3f}")
284             print(f"  Эталон: {gt}")
285             print(f"  Распознано: {pred if pred else '(не
286                 распознано)'}")
287
288             # Вычисляем средние метрики для файла
289             file_metrics = {

```

```

287     'file': os.path.basename(image_path),
288     'avg_accuracy': np.mean(acc_list),
289     'avg_cer': np.mean(cer_list),
290     'avg_wer': np.mean(wer_list),
291     'total_lines': len(expected_lines),
292     'matched_lines': sum(r['matched'] for r in
293                           line_results),
294     'line_results': line_results
295   }
296
297   print(f"\nИтоги для файла {os.path.basename(image_path)}:")
298   print(f"Средняя Accuracy:
299         {file_metrics['avg_accuracy']:.4f}")
300   print(f"Средний CER: {file_metrics['avg_cer']:.4f}")
301   print(f"Средний WER: {file_metrics['avg_wer']:.4f}")
302   print(f"Ожидалось строк: {file_metrics['total_lines']}"))
303   print(f"Совпало строк: {file_metrics['matched_lines']}"))
304   print(f"Всего распознано строк EasyOCR: {len(lines)}")
305
306   return file_metrics
307
308 def main():
309   # Получаем список всех файлов X_Y.png в папке
310   pattern = os.path.join(FOLDER_PATH, "*_*_.png")
311   image_files = glob.glob(pattern)
312
313   # Фильтруем только файлы с паттерном X_Y.png (где X и Y -
314   # числа)
315   filtered_files = []
316   for file in image_files:
317     basename = os.path.basename(file)
318     name_without_ext = os.path.splitext(basename)[0]
319     parts = name_without_ext.split('_')
320
321     if len(parts) == 2 and parts[0].isdigit() and
322       parts[1].isdigit():
323       filtered_files.append(file)
324
325   print(f"Найдено файлов для обработки:
326         {len(filtered_files)})"
327
328

```

```

323     if not filtered_files:
324         print(f"Файлы с паттерном X_Y.png не найдены в папке
325             '{FOLDER_PATH}'")
326
327     # Сортируем файлы по X и Y
328     filtered_files.sort(key=lambda x: (
329         int(os.path.splitext(os.path.basename(x))[0].split('_')[0]),
330         int(os.path.splitext(os.path.basename(x))[0].split('_')[1])
331     ))
332
333     # Обрабатываем все файлы
334     all_metrics = []
335     all_line_results = []
336
337     for i, image_file in enumerate(filtered_files, 1):
338         print(f"\n{('#'*60)}")
339         print(f"Обработка файла {i}/{len(filtered_files)}")
340         metrics = process_image(image_file,
341             save_intermediate=True)
342         if metrics:
343             all_metrics.append(metrics)
344             all_line_results.extend(metrics['line_results'])
345
346     if not all_metrics:
347         print("Не удалось обработать ни одного файла")
348         return
349
350     # Выводим общую статистику
351     print("\n" + "="*80)
352     print("ОБЩАЯ СТАТИСТИКА ПО ВСЕМ ФАЙЛАМ")
353     print("="*80)
354
355     df_metrics = pd.DataFrame([{'k': v for k, v in m.items() if
356         k != 'line_results'} for m in all_metrics])
357     df_lines = pd.DataFrame(all_line_results)
358
359     print("\nМетрики по файлам:")
360     print(df_metrics.to_string(index=False))
361
362     print("\n" + "="*80)

```

```

361     print("СРЕДНИЕ МЕТРИКИ ПО ВСЕМ ФАЙЛАМ:")
362     print("=". * 80)
363     print(f"Средняя Accuracy по всем файлам:
364         {df_metrics['avg_accuracy'].mean():.4f}")
365     print(f"Средний CER по всем файлам:
366         {df_metrics['avg_cer'].mean():.4f}")
367     print(f"Средний WER по всем файлам:
368         {df_metrics['avg_wer'].mean():.4f}")
369     print(f"\nОбщее количество обработанных файлов:
370         {len(df_metrics)}")
371     print(f"Общее количество обработанных строк:
372         {len(df_lines)})")

373     print("\nТочность (Accuracy) по строкам:")
374     print(f"    Среднее: {accuracy_stats['mean']:.4f}")
375     print(f"    Медиана: {accuracy_stats['50%']:.4f}")
376     print(f"    Стандартное отклонение:
377         {accuracy_stats['std']:.4f}")
378     print(f"    Минимум: {accuracy_stats['min']:.4f}")
379     print(f"    Максимум: {accuracy_stats['max']:.4f}")

380     print("\nCER по строкам:")
381     print(f"    Среднее: {cer_stats['mean']:.4f}")
382     print(f"    Медиана: {cer_stats['50%']:.4f}")
383     print(f"    Стандартное отклонение: {cer_stats['std']:.4f}")

384     print("\nWER по строкам:")
385     print(f"    Среднее: {wer_stats['mean']:.4f}")
386     print(f"    Медиана: {wer_stats['50%']:.4f}")
387     print(f"    Стандартное отклонение: {wer_stats['std']:.4f}")

388     print("\nПроцент совпадения строк:
389         {(df_lines['matched'].mean() * 100):.1f}%)")

```

```

395
396     print("\nРаспределение строк по уровню CER:")
397     bins = [0, 0.1, 0.2, 0.3, 0.5, 1.0, float('inf')]
398     labels = ['0-0.1', '0.1-0.2', '0.2-0.3', '0.3-0.5',
399                 '0.5-1.0', '>1.0']
400     df_lines['cer_bin'] = pd.cut(df_lines['cer'], bins=bins,
401                                   labels=labels, right=False)
402     cer_distribution =
403         df_lines['cer_bin'].value_counts().sort_index()
404
405     for bin_label, count in cer_distribution.items():
406         percentage = (count / len(df_lines)) * 100
407         print(f" CER {bin_label}: {count} строк
408             ({percentage:.1f}%)")
409
410     print("\n" + "="*80)
411     print("СОХРАНЕНИЕ РЕЗУЛЬТАТОВ:")
412     print("="*80)
413
414     os.makedirs("results", exist_ok=True)
415
416     df_metrics.to_csv("results/file_metrics_easyocr.csv",
417                         index=False, encoding='utf-8-sig')
418     df_lines.to_csv("results/line_metrics_easyocr.csv",
419                     index=False, encoding='utf-8-sig')
420
421     with open("results/summary_report_easyocr.txt", "w",
422               encoding="utf-8") as f:
423         f.write("ОТЧЕТ ПО ОБРАБОТКЕ OCR (EasyOCR)\n")
424         f.write("=". * 50 + "\n\n")
425         f.write(f"Обработано файлов: {len(df_metrics)}\n")
426         f.write(f"Обработано строк: {len(df_lines)}\n")
427         f.write(f"Процент совпадения строк:
428             {(df_lines['matched'].mean() * 100):.1f}%\n\n")
429
430         f.write("СРЕДНИЕ МЕТРИКИ:\n")
431         f.write(f"Средняя Accuracy:
432             {df_metrics['avg_accuracy'].mean():.4f}\n")
433         f.write(f"Средний CER:
434             {df_metrics['avg_cer'].mean():.4f}\n")
435         f.write(f"Средний WER:

```

```
426     {df_metrics['avg_wer'].mean():.4f}\n\n")
427 f.write("МЕТРИКИ ПО ФАЙЛАМ:\n")
428 f.write(df_metrics.to_string(index=False))
429
430 print("\nОбработка завершена успешно!")
431 print("Результаты сохранены в папке 'results' с суффиксом
432      '_easyocr'")
433 if __name__ == "__main__":
434     main()
```

ПРИЛОЖЕНИЕ Б

Код программы для TesseractOCR

Листинг Б.1 – Исходный код программы

```
1 import cv2
2 import pytesseract
3 import numpy as np
4 from jiwer import cer, wer
5 from difflib import SequenceMatcher
6 import os
7 import glob
8 import pandas as pd
9
10 pytesseract.pytesseract.tesseract_cmd =
11     r'C:\Users\ordum\AppData\Local\Programs\Tesseract-OCR\tesseract.exe'
12
13 # Папка с изображениями
14 FOLDER_PATH = "1"
15
16 def crop_by_marks(image, offset=50):
17     """
18         Обрезает изображение, удаляя верхнюю левую метку
19             позиционирования.
20
21     Args:
22         image: исходное изображение
23         offset: дополнительный отступ после метки
24             (положительный = отступ от метки)
25
26     Returns:
27         cropped_image: обрезанное изображение
28         mark_coords: координаты найденной метки (x, y, w, h)
29     """
30
31     # Создаем копию изображения
32     original = image.copy()
33
34     # Конвертируем в grayscale
35     gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
36     h, w = gray.shape
37
38     # Бинаризуем изображение
```

```

35     _, binary = cv2.threshold(gray, 0, 255,
36                               cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)
37
38     # Находим контуры
39     contours, _ = cv2.findContours(binary, cv2.RETR_EXTERNAL,
40                                    cv2.CHAIN_APPROX_SIMPLE)
41
42     marks = []
43
44     # Определяем область поиска для верхней левой метки
45     search_area_w = w // 2
46     search_area_h = h // 2
47
48     for cnt in contours:
49         x, y, cw, ch = cv2.boundingRect(cnt)
50         area = cv2.contourArea(cnt)
51
52         # Фильтр по размеру
53         if 200 < area < 3000:
54             # Фильтр по положению: только в верхней левой части
55             if x < search_area_w and y < search_area_h:
56                 marks.append((x, y, cw, ch, area))
57
58     if not marks:
59         print("Метки в верхнем левом углу не найдены,
60               возвращаю оригинальное изображение")
61     return original, []
62
63     marks.sort(key=lambda m: (m[1], m[0]))
64
65     # Выбираем самую верхнюю левую метку
66     x, y, mark_w, mark_h, area = marks[0]
67
68     # Вычисляем координаты для обрезки
69     crop_start_x = x + mark_w + offset
70     crop_start_y = y + mark_h + offset
71
72     # Обрезаем изображение
73     roi = image[crop_start_y:, crop_start_x:]
74
75     if roi.size == 0:

```

```

73     print("ROI пустой, возвращаю оригинальное изображение")
74     return original, [(x, y, mark_w, mark_h)]
75
76     return roi, [(x, y, mark_w, mark_h)]
77
78 def draw_marks(image, marks, color=(0, 255, 0), thickness=2):
79     debug_img = image.copy()
80     for x, y, w, h in marks:
81         cv2.rectangle(debug_img, (x, y), (x + w, y + h),
82                       color, thickness)
83     return debug_img
84
85 def char_accuracy(gt, pred):
86     return SequenceMatcher(None, gt, pred).ratio()
87
88 def process_image(image_path, save_intermediate=False):
89     print(f"\n{'='*50}")
90     print(f"Обработка файла: {os.path.basename(image_path)}")
91     print('='*50)
92
93     img = cv2.imread(image_path)
94     if img is None:
95         print(f"Ошибка: Не удалось загрузить изображение
96             {image_path}")
97         return None
98
99     # Обрезка по меткам
100    roi, marks = crop_by_marks(img)
101
102    if save_intermediate:
103        os.makedirs("debug", exist_ok=True)
104        base_name =
105            os.path.splitext(os.path.basename(image_path))[0]
106
107        draw = draw_marks(img, marks)
108        cv2.imwrite(f"debug/{base_name}_marks.png", draw)
109
110    if roi is None or roi.size == 0:
111        print("Ошибка: ROI пустой, используем оригинальное
112             изображение")
113    roi = img.copy()

```

```

110
111     # Предобработка
112     gray = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
113     gray = cv2.blur(gray, (9, 9))
114     _, thresh = cv2.threshold(gray, 100, 255,
115                               cv2.THRESH_BINARY + cv2.THRESH_OTSU)
116
117     if save_intermediate:
118         cv2.imwrite(f"debug/{base_name}_binary_tes.png",
119                     thresh)
120         cv2.imwrite(f"debug/{base_name}_roi_tes.png", roi)
121
122     # OCR
123     config = r"--oem 3 --psm 1 -l rus"
124     raw_text = pytesseract.image_to_string(thresh,
125                                            config=config)
126     lines = [l.strip() for l in raw_text.splitlines() if
127              l.strip()]
128
129     # Сохраняем распознанный текст
130     text_file = os.path.splitext(image_path)[0] + "_tes" +
131                 ".txt"
132     with open(text_file, "w", encoding="utf-8") as file:
133         file.write(raw_text)
134
135     # Эталонные строки
136     expected_lines = []
137     expected_lines.append("Задание указано на доске.
138                         Подписывать бланк НЕ требуется")
139
140     for _ in range(5):
141         expected_lines.append("АБВГДЕЁЖ З ИЙКЛМНОПРСТ У")
142         expected_lines.append("ФХЦЧЩЪЫЬЭЮЯ -0123456789")
143
144     for _ in range(5):
145         expected_lines.append("АБВГДЕЁЖЗИЙ КЛМНОПРСТ
146                         УФХЦЧЩЪЫЬЭ")
147         expected_lines.append("ЮЯ -0123456789")
148
149     # Вычисление метрик
150     acc_list, cer_list, wer_list = [], [], []

```

```

144     line_results = []
145
146     min_lines = min(len(expected_lines), len(lines))
147
148     if min_lines == 0:
149         print(f"Ошибка: Нет строк для сравнения в файле
150               {image_path}")
151
152     for i in range(min_lines):
153         gt = expected_lines[i]
154         pred = lines[i] if i < len(lines) else ""
155         acc = char_accuracy(gt, pred)
156         cer_val = cer(gt, pred)
157         wer_val = wer(gt, pred)
158
159         acc_list.append(acc)
160         cer_list.append(cer_val)
161         wer_list.append(wer_val)
162
163     line_results.append({
164         'file': os.path.basename(image_path),
165         'line_num': i,
166         'gt': gt,
167         'pred': pred,
168         'accuracy': acc,
169         'cer': cer_val,
170         'wer': wer_val
171     })
172
173     print(f"Строка {i:2d} | ACC: {acc:.3f} | CER:
174                   {cer_val:.3f} | WER: {wer_val:.3f}")
175
176     # Вычисляем средние метрики для файла
177     file_metrics = {
178         'file': os.path.basename(image_path),
179         'avg_accuracy': np.mean(acc_list),
180         'avg_cer': np.mean(cer_list),
181         'avg_wer': np.mean(wer_list),
182         'total_lines': min_lines,
183         'matched_lines': sum(1 for i in range(min_lines) if

```

```

        cer_list[i] < 0.5), # линии с CER < 0.5
183     'line_results': line_results
184 }
185
186 print(f"\nИтоги для файла {os.path.basename(image_path)}:")
187 print(f"Средняя Accuracy:
188     {file_metrics['avg_accuracy']:.4f}")
189 print(f"Средний CER: {file_metrics['avg_cer']:.4f}")
190 print(f"Средний WER: {file_metrics['avg_wer']:.4f}")
191 print(f"Обработано строк: {file_metrics['total_lines']}")

192 return file_metrics
193
194 def main():
195     # Получаем список всех файлов X_Y.png в папке
196     pattern = os.path.join(FOLDER_PATH, "*_*_.png")
197     image_files = glob.glob(pattern)

198     # Фильтруем только файлы с паттерном X_Y.png (где X и Y -
199         # числа)
200     filtered_files = []
201     for file in image_files:
202         basename = os.path.basename(file)
203         name_without_ext = os.path.splitext(basename)[0]
204         parts = name_without_ext.split('_')

205         if len(parts) == 2 and parts[0].isdigit() and
206             parts[1].isdigit():
207             filtered_files.append(file)

208     print(f"Найдено файлов для обработки:
209         {len(filtered_files)})")

210     if not filtered_files:
211         print(f"Файлы с паттерном X_Y.png не найдены в папке
212             '{FOLDER_PATH}'")
213     return

214
215     # Сортируем файлы по X и Y
216     filtered_files.sort(key=lambda x:
217         int(os.path.splitext(os.path.basename(x))[0].split('_')[0])),

```

```

218         int(os.path.splitext(os.path.basename(x))[0].split('_')[1])
219     ))
220
221     # Обрабатываем все файлы
222     all_metrics = []
223     all_line_results = []
224
225     for image_file in filtered_files:
226         metrics = process_image(image_file,
227                               save_intermediate=True)
228
229         if metrics:
230             all_metrics.append(metrics)
231             all_line_results.extend(metrics['line_results'])
232
233     if not all_metrics:
234         print("Не удалось обработать ни одного файла")
235         return
236
237     # Выводим общую статистику
238     print("\n" + "="*80)
239     print("ОБЩАЯ СТАТИСТИКА ПО ВСЕМ ФАЙЛАМ")
240     print("="*80)
241
242     df_metrics = pd.DataFrame([k: v for k, v in m.items() if
243                                k != 'line_results'} for m in all_metrics])
244     df_lines = pd.DataFrame(all_line_results)
245
246     print("\nМетрики по файлам:")
247     print(df_metrics.to_string(index=False))
248
249     print("\nСРЕДНИЕ МЕТРИКИ ПО ВСЕМ ФАЙЛАМ:")
250     print("="*80)
251     print(f"Средняя Accuracy по всем файлам:
252           {df_metrics['avg_accuracy'].mean():.4f}")
253     print(f"Средний CER по всем файлам:
254           {df_metrics['avg_cer'].mean():.4f}")
255     print(f"Средний WER по всем файлам:
256           {df_metrics['avg_wer'].mean():.4f}")
257     print(f"Общее количество обработанных файлов:
258           {len(df_metrics)}")

```

```

253     print(f"Общее количество обработанных строк:
254         {len(df_lines)}")
255
255     print("\n" + "="*80)
256     print("ДОПОЛНИТЕЛЬНАЯ СТАТИСТИКА:")
257     print("="*80)
258
259     accuracy_stats = df_lines['accuracy'].describe()
260     cer_stats = df_lines['cer'].describe()
261     wer_stats = df_lines['wer'].describe()
262
263     print(f"\nТочность (Accuracy) по строкам:")
264     print(f"    Среднее: {accuracy_stats['mean']:.4f}")
265     print(f"    Медиана: {accuracy_stats['50%']:.4f}")
266     print(f"    Стандартное отклонение:
267         {accuracy_stats['std']:.4f}")
268     print(f"    Минимум: {accuracy_stats['min']:.4f}")
269     print(f"    Максимум: {accuracy_stats['max']:.4f}")
270
271     print(f"\nCER по строкам:")
272     print(f"    Среднее: {cer_stats['mean']:.4f}")
273     print(f"    Медиана: {cer_stats['50%']:.4f}")
274     print(f"    Стандартное отклонение: {cer_stats['std']:.4f}")
275
276     print(f"\nWER по строкам:")
277     print(f"    Среднее: {wer_stats['mean']:.4f}")
278     print(f"    Медиана: {wer_stats['50%']:.4f}")
279     print(f"    Стандартное отклонение: {wer_stats['std']:.4f}")
280
281     print("\nРаспределение строк по уровню CER:")
282     bins = [0, 0.1, 0.2, 0.3, 0.5, 1.0, float('inf')]
283     labels = ['0-0.1', '0.1-0.2', '0.2-0.3', '0.3-0.5',
284         '0.5-1.0', '>1.0']
285     df_lines['cer_bin'] = pd.cut(df_lines['cer'], bins=bins,
286         labels=labels, right=False)
287     cer_distribution =
288         df_lines['cer_bin'].value_counts().sort_index()
289
290     for bin_label, count in cer_distribution.items():
291         percentage = (count / len(df_lines)) * 100
292         print(f"    CER {bin_label}: {count} строк

```

```

289         (f'{percentage:.1f}%)')
290
291     print("\n" + "="*80)
292     print("СОХРАНЕНИЕ РЕЗУЛЬТАТОВ:")
293     print("="*80)
294
295     os.makedirs("results", exist_ok=True)
296
297     df_metrics.to_csv("results/file_metrics_tes.csv",
298                       index=False, encoding='utf-8-sig')
299     df_lines.to_csv("results/line_metrics_tes.csv",
300                      index=False, encoding='utf-8-sig')
301
302     with open("results/summary_report_tes.txt", "w",
303                encoding="utf-8") as f:
304         f.write("ОТЧЕТ ПО ОБРАБОТКЕ OCR\n")
305         f.write("=="*50 + "\n\n")
306         f.write(f"Обработано файлов: {len(df_metrics)}\n")
307         f.write(f"Обработано строк: {len(df_lines)}\n\n")
308
309         f.write("СРЕДНИЕ МЕТРИКИ:\n")
310         f.write(f"Средняя Accuracy:
311                 {df_metrics['avg_accuracy'].mean():.4f}\n")
312         f.write(f"Средний CER:
313                 {df_metrics['avg_cer'].mean():.4f}\n")
314         f.write(f"Средний WER:
315                 {df_metrics['avg_wer'].mean():.4f}\n\n")
316
317         f.write("МЕТРИКИ ПО ФАЙЛАМ:\n")
318         f.write(df_metrics.to_string(index=False))
319
320
321
322
323
324     print("\nОбработка завершена успешно!")
325
326 if __name__ == "__main__":
327     main()

```

ПРИЛОЖЕНИЕ В

Код программы для PaddleOCR

Листинг В.1 – Исходный код программы

```
1 import cv2
2 import numpy as np
3 from jiwer import cer, wer
4 from difflib import SequenceMatcher
5 import os
6 import glob
7 import pandas as pd
8 from paddleocr import PaddleOCR
9 import json
10
11 ocr = PaddleOCR(
12     text_recognition_model_name="cyrillic_PP-OCRv5_mobile_rec",
13     use_doc_orientation_classify=False,
14     use_doc_unwarping=False,
15     use_textline_orientation=True,
16     device="cpu",
17 )
18
19 FOLDER_PATH = "1"
20
21 def crop_by_marks(image, offset=50):
22     """
23         Обрезает изображение, удаляя верхнюю левую метку
24         позиционирования.
25
26     Args:
27         image: исходное изображение
28         offset: дополнительный отступ после метки
29             (положительный = отступ от метки)
30
31     Returns:
32         cropped_image: обрезанное изображение
33         mark_coords: координаты найденной метки (x, y, w, h)
34     """
35
36     original = image.copy()
37
38     gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

```

36     h , w = gray . shape
37
38     _ , binary = cv2 . threshold (gray , 0 , 255 ,
39                               cv2 . THRESH_BINARY_INV + cv2 . THRESH_OTSU)
40
41     contours , _ = cv2 . findContours (binary , cv2 . RETR_EXTERNAL ,
42                                         cv2 . CHAIN_APPROX_SIMPLE)
43
44     marks = []
45
46
47     for cnt in contours :
48         x , y , cw , ch = cv2 . boundingRect (cnt)
49         area = cv2 . contourArea (cnt)
50
51         if 200 < area < 3000:
52             if x < search_area_w and y < search_area_h:
53                 marks . append ((x , y , cw , ch , area))
54
55     if not marks:
56         print ("Метки в верхнем левом углу не найдены ,
57               возвращаю оригинальное изображение")
58
59     return original , []
60
61     marks . sort (key = lambda m: (m [1] , m [0]))
62
63     x , y , mark_w , mark_h , area = marks [0]
64
65     crop_start_x = x + mark_w + offset
66     crop_start_y = y + mark_h + offset
67
68     roi = image [crop_start_y: , crop_start_x:]
69
70     if roi . size == 0:
71         print ("ROI пустой , возвращаю оригинальное изображение")
72         return original , [(x , y , mark_w , mark_h)]
73
74     return roi , [(x , y , mark_w , mark_h)]

```

```

74     def draw_marks(image, marks, color=(0, 255, 0), thickness=2):
75         debug_img = image.copy()
76         for x, y, w, h in marks:
77             cv2.rectangle(debug_img, (x, y), (x + w, y + h),
78                           color, thickness)
79         return debug_img
80
81     def char_accuracy(gt, pred):
82         return SequenceMatcher(None, gt, pred).ratio()
83
84     def paddleocr_to_text(result):
85         """
86             Преобразует результат PaddleOCR в текст с разделением на
87             строки.
88
89             Args:
90                 result: результат от ocr.ocr()
91
92             Returns:
93                 list: список строк текста
94
95             """
96
97         if result is None or len(result) == 0:
98             return lines
99
100        all_boxes = []
101        for line in result:
102            boxes = line[0]
103            text = line[1][0]
104            confidence = line[1][1]
105
106            y_center = np.mean([point[1] for point in boxes])
107
108            all_boxes.append({
109                'y_center': y_center,
110                'text': text,
111                'confidence': confidence,
112                'box': boxes

```

```

113     })
114
115     all_boxes.sort(key=lambda x: x['y_center'])
116
117     line_threshold = 20
118     current_line_items = []
119
120     for item in all_boxes:
121         if current_y is None:
122             current_y = item['y_center']
123             current_line_items.append(item)
124         elif abs(item['y_center'] - current_y) <
125             line_threshold:
126             current_line_items.append(item)
127         else:
128             current_line_items.sort(key=lambda x:
129                 np.mean([point[0] for point in x['box']])))
130             line_text = ' '.join([item['text'] for item in
131                 current_line_items])
132             lines.append(line_text.strip())
133
134             current_line_items = [item]
135             current_y = item['y_center']
136
137             if current_line_items:
138                 current_line_items.sort(key=lambda x:
139                     np.mean([point[0] for point in x['box']])))
140                 line_text = ' '.join([item['text'] for item in
141                     current_line_items])
142                 lines.append(line_text.strip())
143
144             return lines
145
146     def process_image(image_path, save_intermediate=False,
147     save_json=False):
148         print(f"\n{'='*50}")
149         print(f"Обработка файла: {os.path.basename(image_path)}")
150         print('='*50)
151
152         img = cv2.imread(image_path)
153         if img is None:

```

```

148     print(f"Ошибка: Не удалось загрузить изображение
149         {image_path}")
150
151     # Обрезка по меткам
152     roi, marks = crop_by_marks(img)
153
154     if save_intermediate:
155         os.makedirs("debug", exist_ok=True)
156         base_name =
157             os.path.splitext(os.path.basename(image_path))[0]
158
159         draw = draw_marks(img, marks)
160         cv2.imwrite(f"debug/{base_name}_marks_paddle.png",
161                     draw)
162
163     if roi is None or roi.size == 0:
164         print("Ошибка: ROI пустой, используем оригинальное
165             изображение")
166         roi = img.copy()
167
168     gray = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
169     gray = cv2.blur(gray, (9, 9))
170     _, thresh = cv2.threshold(gray, 100, 255,
171                               cv2.THRESH_BINARY + cv2.THRESH_OTSU)
172
173     if save_intermediate:
174         cv2.imwrite(f"debug/{base_name}_binary_paddle.png",
175                     thresh)
176         cv2.imwrite(f"debug/{base_name}_roi_paddle.png", roi)
177
178     temp_path = f"temp_{os.path.basename(image_path)}"
179     cv2.imwrite(temp_path, roi)
180
181     try:
182         result = ocr.predict(roi)
183         # for res in result:
184         #     res.print()
185         #     res.save_to_img("output")
186         #     res.save_to_json("output")

```

```

183     all_texts = []
184     for res in result:
185         if hasattr(res, 'rec_texts'):
186             all_texts.extend(res.rec_texts)
187         elif 'rec_texts' in res:
188             all_texts.extend(res['rec_texts'])
189
190     for text in all_texts:
191         print(text)
192
193     full_text = "\n".join(all_texts)
194     print("\nПолный текст:")
195     print(full_text)
196
197     raw_text = full_text
198
199     lines = full_text.split('\n')
200
201 finally:
202     if os.path.exists(temp_path):
203         os.remove(temp_path)
204
205     text_file = os.path.splitext(image_path)[0] + "_paddle.txt"
206     with open(text_file, "w", encoding="utf-8") as file:
207         file.write(raw_text)
208
209     print(f"Распознано строк PaddleOCR: {len(lines)}")
210     for i, line in enumerate(lines[:5]):
211         print(f"  Стока {i}: {line}")
212
213     expected_lines = []
214     expected_lines.append("Задание указано на доске."
215                           " Подписывать бланк НЕ требуется")
216
217     for _ in range(5):
218         expected_lines.append("АБВГДЕЁЖЗ ИЙКЛМНОПРСТ У")
219         expected_lines.append("ФХЦЧШЩЫҮЭЮЯ -0123456789")
220
221     for _ in range(5):
222         expected_lines.append("АБВГДЕЁЖЗИЙ КЛМНОПРСТ"
223                               " УФХЦЧШЩЫҮЭ")

```

```

222     expected_lines.append("ЮЯ-0123456789")
223
224     acc_list, cer_list, wer_list = [], [], []
225     line_results = []
226
227     min_lines = min(len(expected_lines), len(lines))
228
229     if min_lines == 0:
230         print(f"Ошибка: Нет строк для сравнения в файле
231             {image_path}")
232         return None
233
234     for i in range(min_lines):
235         gt = expected_lines[i]
236         pred = lines[i] if i < len(lines) else ""
237         acc = char_accuracy(gt, pred)
238         cer_val = cer(gt, pred)
239         wer_val = wer(gt, pred)
240
241         acc_list.append(acc)
242         cer_list.append(cer_val)
243         wer_list.append(wer_val)
244
245         line_results.append({
246             'file': os.path.basename(image_path),
247             'line_num': i,
248             'gt': gt,
249             'pred': pred,
250             'accuracy': acc,
251             'cer': cer_val,
252             'wer': wer_val
253         })
254
255         print(f"Строка {i:2d} | ACC: {acc:.3f} | CER:
256             {cer_val:.3f} | WER: {wer_val:.3f}")
257
258     file_metrics = {
259         'file': os.path.basename(image_path),
260         'avg_accuracy': np.mean(acc_list),
261         'avg_cer': np.mean(cer_list),
262         'avg_wer': np.mean(wer_list),

```

```

261     'total_lines': min_lines,
262     'matched_lines': sum(1 for i in range(min_lines) if
263         cer_list[i] < 0.5),
264     'ocr_engine': 'PaddleOCR',
265     'line_results': line_results
266 }
267
268 print(f"\nИтоги для файла {os.path.basename(image_path)}:")
269 print(f"Средняя Accuracy:
270     {file_metrics['avg_accuracy']:.4f}")
271 print(f"Средний CER: {file_metrics['avg_cer']:.4f}")
272 print(f"Средний WER: {file_metrics['avg_wer']:.4f}")
273 print(f"Обработано строк: {file_metrics['total_lines']}")

274
275 return file_metrics

276
277 def main():
278     pattern = os.path.join(FOLDER_PATH, "*_*_.png")
279     image_files = glob.glob(pattern)
280
281     filtered_files = []
282     for file in image_files:
283         basename = os.path.basename(file)
284         name_without_ext = os.path.splitext(basename)[0]
285         parts = name_without_ext.split('_')
286
287         if len(parts) == 2 and parts[0].isdigit() and
288             parts[1].isdigit():
289             filtered_files.append(file)

290     print(f"Найдено файлов для обработки:
291         {len(filtered_files)})")

292     if not filtered_files:
293         print(f"Файлы с паттерном X_Y.png не найдены в папке
294             '{FOLDER_PATH}'")
295
296     return

297
298     filtered_files.sort(key=lambda x: (
299         int(os.path.splitext(os.path.basename(x))[0].split('_')[0]),
300         int(os.path.splitext(os.path.basename(x))[0].split('_')[1]))

```

```

297     ))
298
299     all_metrics = []
300     all_line_results = []
301
302     for image_file in filtered_files:
303         metrics = process_image(image_file,
304             save_intermediate=True, save_json=True)
305         if metrics:
306             all_metrics.append(metrics)
307             all_line_results.extend(metrics['line_results'])
308
309     if not all_metrics:
310         print("Не удалось обработать ни одного файла")
311         return
312
313     print("\n" + "="*80)
314     print("ОБЩАЯ СТАТИСТИКА ПО ВСЕМ ФАЙЛАМ (PaddleOCR)")
315     print("="*80)
316
317     df_metrics = pd.DataFrame([{'k': v for k, v in m.items() if
318                               k != 'line_results'} for m in all_metrics])
319     df_lines = pd.DataFrame(all_line_results)
320
321
322     print("\nМетрики по файлам:")
323     print(df_metrics.to_string(index=False))
324
325     print("\n" + "="*80)
326     print("СРЕДНИЕ МЕТРИКИ ПО ВСЕМ ФАЙЛАМ:")
327     print("="*80)
328     print(f"Средняя Accuracy по всем файлам:
329           {df_metrics['avg_accuracy'].mean():.4f}")
330     print(f"Средний CER по всем файлам:
331           {df_metrics['avg_cer'].mean():.4f}")
332     print(f"Средний WER по всем файлам:
333           {df_metrics['avg_wer'].mean():.4f}")
334     print(f"Общее количество обработанных файлов:
335           {len(df_metrics)}")
336     print(f"Общее количество обработанных строк:
337           {len(df_lines)}")

```

```

331     print("\n" + "="*80)
332     print("ДОПОЛНИТЕЛЬНАЯ СТАТИСТИКА:")
333     print("="*80)
334
335     accuracy_stats = df_lines['accuracy'].describe()
336     cer_stats = df_lines['cer'].describe()
337     wer_stats = df_lines['wer'].describe()
338
339     print(f"\nТочность (Accuracy) по строкам:")
340     print(f"    Среднее: {accuracy_stats['mean']:.4f}")
341     print(f"    Медиана: {accuracy_stats['50%']:.4f}")
342     print(f"    Стандартное отклонение:
343             {accuracy_stats['std']:.4f}")
344     print(f"    Минимум: {accuracy_stats['min']:.4f}")
345     print(f"    Максимум: {accuracy_stats['max']:.4f}")
346
347     print(f"\nCER по строкам:")
348     print(f"    Среднее: {cer_stats['mean']:.4f}")
349     print(f"    Медиана: {cer_stats['50%']:.4f}")
350     print(f"    Стандартное отклонение: {cer_stats['std']:.4f}")
351
352     print(f"\nWER по строкам:")
353     print(f"    Среднее: {wer_stats['mean']:.4f}")
354     print(f"    Медиана: {wer_stats['50%']:.4f}")
355     print(f"    Стандартное отклонение: {wer_stats['std']:.4f}")
356
357     print("\nРаспределение строк по уровню CER:")
358     bins = [0, 0.1, 0.2, 0.3, 0.5, 1.0, float('inf')]
359     labels = ['0-0.1', '0.1-0.2', '0.2-0.3', '0.3-0.5',
360               '0.5-1.0', '>1.0']
361     df_lines['cer_bin'] = pd.cut(df_lines['cer'], bins=bins,
362                                   labels=labels, right=False)
363     cer_distribution =
364         df_lines['cer_bin'].value_counts().sort_index()
365
366     for bin_label, count in cer_distribution.items():
367         percentage = (count / len(df_lines)) * 100
368         print(f"    CER {bin_label}: {count} строк
369             ({percentage:.1f}%)")
370
371     print("\n" + "="*80)

```

```

367     print("СОХРАНЕНИЕ РЕЗУЛЬТАТОВ:")
368     print("="*80)
369
370     os.makedirs("results", exist_ok=True)
371
372     df_metrics.to_csv("results/file_metrics_paddle.csv",
373         index=False, encoding='utf-8-sig')
373     df_lines.to_csv("results/line_metrics_paddle.csv",
374         index=False, encoding='utf-8-sig')
374
375     with open("results/summary_report_paddle.txt", "w",
376     encoding="utf-8") as f:
376         f.write("ОТЧЕТ ПО ОБРАБОТКЕ OCR (PaddleOCR)\n")
377         f.write("="*50 + "\n\n")
378         f.write(f"Обработано файлов: {len(df_metrics)}\n")
379         f.write(f"Обработано строк: {len(df_lines)}\n\n")
380
381         f.write("СРЕДНИЕ МЕТРИКИ:\n")
382         f.write(f"Средняя Accuracy:
383             {df_metrics['avg_accuracy'].mean():.4f}\n")
383         f.write(f"Средний CER:
384             {df_metrics['avg_cer'].mean():.4f}\n")
384         f.write(f"Средний WER:
385             {df_metrics['avg_wer'].mean():.4f}\n\n")
386
386         f.write("МЕТРИКИ ПО ФАЙЛАМ:\n")
387         f.write(df_metrics.to_string(index=False))
388
389         print("\nОбработка завершена успешно!")
390         print(f"Результаты сохранены в папке 'results' с суффиксом
391             '_paddle'")
392
392     if __name__ == "__main__":
393         main()

```