ПРОФЕССИОНАЛЬНОЕ ЗАВТРА

VI Всероссийский сетевой конкурс студенческих проектов с участием студентов с инвалидностью

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФГБОУ ВО «Приамурский государственный университет имени Шолом-Алейхема»

Факультет математики, информационных технологий и техники

Направление «Профессиональное завтра в науке»

Номинация «Научная статья»

«Разработка программы распознавания символов художественного шрифта методами машинного обучения»

Выполнил:
Черкашин Александр Михайлович

Руководитель:
зав. кафедрой информационных систем,
математики и правовой информатики
Баженов Руслан Иванович

Биробиджан, 2023 г.

Потребность в использовании модели для распознавания дизайнерских символов и художественного шрифта с разными символами возникла при получении получения текста из изображения. В исследовании предлагается модель машинного обучения, которая выполняет извлекает текст из изображений с разными символами. В наборе данных представлено меню ресторанов и столовых, содержащиеся изображения с готовой разметкой области текста и текст в виде символов. Целью работы является разработка программы и модели по распознаванию и извлечению текста из изображений. Для нахождения текста в изображении использовалась модель ResNet50, а для извлечения текста применялась модель DenseNet. Предлагаемый метод обеспечивает высокие показатели точности при распознавании и нахождении текста в изображениях. Таким образом, предложенная модель извлечения символов художественных шрифтов с использованием машинного обучения является весьма перспективной и имеет большое значение для разработки новых моделей распознавания в области компьютерной зрения. Масштабы результатов широки, и полученные результаты могут быть применены ко многим практическим приложениям, включая графический дизайн, типографику и цифровые медиа.

Ключевые слова: Python, Pytorch, Tensorflow, OCR, Recognizer, Detector, ResNet, DenseNet.

1. Обзор исследований

И. Цуляк и др. описывают различные компоненты OpenCV и его применения в обработке изображений и видео, компьютерном зрении и машинном обучении. Исследовательская работа также включает практические примеры и фрагменты кода, которые подчеркивают использование функций и функций OpenCV для задач обработки изображений и видео. В целом авторы стремятся дать новичкам в области компьютерного зрения базовое представление о библиотеке OpenCV и ее потенциальных приложениях [1].

К. Пулли и др. исследуют важность эффективного и точного компьютерного зрения в реальном времени в различных областях, таких как робототехника, видеонаблюдение, анализ видео и медицинская визуализация. Затем они дают обзор различных методов и инструментов, используемых в компьютерном зрении в реальном времени, и того, как библиотека OpenCV может использоваться в этих приложениях [2].

Исследование, проведенное Г. Хие, В. Лу посвящено обнаружению границ на изображениях с использованием технологии OpenCV. Авторы представляют важность обнаружения краев в различных приложениях обработки изображений. Затем объясняют концепцию краев изображения и общие методы, используемые для их обнаружения [3].

Г. Агам предоставляет пошаговое руководство по установке OpenCV, настройке среды разработки и началу программирования на C++. Автор обсуждает фундаментальные концепции обработки изображений, обнаружения объектов и извлечения признаков, а также показывает, как реализовать эти методы с помощью различных функций и модулей OpenCV. Кроме того, учебник включает примеры реальных приложений компьютерного зрения, таких как распознавание лиц, обнаружение движения и отслеживание транспортных средств. В целом автор стремится предоставить исчерпывающее введение в программирование OpenCV,

доступное для начинающих, а также полезное для опытных разработчиков [4].

Л. Яиао и др. предоставляет обзор различных инструментов, включенных в DavarOCR, включая инструменты распознавания текста, инструменты распознавания таблиц и инструменты распознавания изображений. Инструментарий разработан таким образом, чтобы быть простым в использовании и адаптируемым к широкому спектру вариантов использования. Авторы описывают ряд тематических исследований и экспериментов, демонстрирующих эффективность DavarOCR. Эти исследования показывают, что DavarOCR способен точно распознавать и понимать широкий спектр документов и может работать с различными языками. В целом цель авторов состоит в том, чтобы предоставить мощный и гибкий инструмент, который может помочь исследователям и практикам лучше понять и проанализировать широкий спектр документов [5].

В исследовании М. Висинтин и А. Девер. по оптическому распознаванию символов (OCR) представлен обзор технологии, используемой для автоматического распознавания печатного или рукописного текста в цифровых изображениях или отсканированных документах. Авторы обсуждают различные методы и алгоритмы, которые используются для извлечения, анализа и интерпретации текста, включая предварительную обработку, сегментацию, извлечение признаков и машинное обучение. Они также изучают применение оптического распознавания символов в различных областях, таких как оцифровка документов, индексация и поиск текста, а также автоматический перевод [6].

Н. Лу и др. предлагают многоаспектную нелокальную сеть (Master) для оптимального распознавания текста сцены. Они утверждают, что существующие методы распознавания текста сцены ограничены, поскольку они рассматривают задачу распознавания как простую проблему классификации изображений, которая не учитывает сложные вариации и искажения, возникающие в естественных сценах. Чтобы устранить это

ограничение, авторы предлагают новую структуру, включающую нелокальные нейронные сети, которые позволяют сети фиксировать долгосрочные зависимости между различными частями входного изображения. Система обучается на большом наборе данных текстовых изображений с использованием комбинации контролируемых и неконтролируемых методов обучения для повышения точности и обобщения. [7].

В. Ю и др. представляет метод извлечения ключевой информации из документов с использованием графовых сверточных сетей (GLCN). Авторы предлагают полуконтролируемую структуру, которая сочетает в себе явное распространение сигнала с выделением признаков для повышения точности извлечения информации. Предлагаемый метод оценивается на нескольких эталонных наборах данных, и результаты показывают, что он превосходит несколько современных методов с точки зрения точности, полноты и оценки F1. В документе также представлен подробный анализ предлагаемого метода, включая изучение влияния различных гиперпараметров на производительность. В целом исследование показывает, что предложенный ими метод является эффективным подходом к сложной проблеме извлечения ключевой информации из документов [8].

Авторы исследования П. Х. Лин и Д. Ван стремятся повысить точность и эффективность систем OCR, разработав модель, позволяющую редактировать шаблоны, то есть процесс манипулирования шаблонами OCR для исправления ошибок распознавания текста. Система включает в себя редактор шаблонов, который позволяет пользователям вручную редактировать шаблоны OCR, и алгоритм проецирования слов, который предсказывает следующее слово в предложении, повышая точность системы распознавания. В целом исследование направлено на повышение точности и эффективности систем OCR, делая их более полезными и надежными инструментами для распознавания текста [9].

Авторы исследования З. Куанг и др. стремятся создать гибкую и настраиваемую структуру, которую можно применять к целому ряду приложений, от оцифровки документов до поиска изображений. Исследование включает в себя разработку ряда моделей глубокого обучения, которые могут точно и эффективно обнаруживать, и распознавать текст на изображениях с использованием различных методов, включая сверточные нейронные сети, механизмы внимания и пространственные преобразователи. В целом, исследование направлено на совершенствование современных технологий распознавания текста и предоставление исследователям и разработчикам комплексного набора инструментов для работы с текстом в изображениях [10].

О. Зайен и др. приняли участие в конкурсе ICPR2020 и разработали подход, основанный на глубоком обучении, для обнаружения текстовых областей в новостных видеокадрах на арабском языке. Они применили методы предварительной и последующей обработки для повышения производительности модели. Кроме того, авторы предложили систему распознавания для распознавания и сегментации текстовых областей для достижения оптимальных результатов. Основная цель исследования — улучшить систему обнаружения и распознавания арабского текста в наборе данных новостных видеороликов путем разработки надежного подхода, основанного на глубоком обучении [11].

Г. Яуме, Х. К. Екенел, Й. П. Тхиран представляют новый набор данных под названием FUNSD, что означает «Понимание форм в зашумленных отсканированных документах». Этот набор данных предназначен для продвижения исследований в области понимания формы и извлечения информации из отсканированных документов. В исследовании приводится подробное описание набора данных, включая его содержание и методологию аннотирования. Авторы также представляют результаты тестов, основанные на различных подходах к машинному обучению с использованием набора данных FUNSD. Кроме того, они дают представление о проблемах,

связанных с набором данных, и потенциальных будущих направлениях исследований в этой области. В целом, исследование направлено на то, чтобы предоставить ценный ресурс для исследователей, заинтересованных в развитии области понимания формы и извлечения информации из отсканированных документов [12].

Авторы П. Батра и др. стремятся повысить точность и скорость систем ALPR с помощью алгоритма You Only Look Once (YOLO). Этот алгоритм был модифицирован и обучен на большом наборе данных номерных знаков для распознавания номерных знаков в режиме реального времени с высокой точностью. Исследователи провели эксперименты, чтобы продемонстрировать эффективность и действенность предложенной ими системы ALPR. В целом, представляют новый подход к системам ALPR, который может быть полезен в различных приложениях, таких как управление дорожным движением, управление парковками и системами безопасности [13].

М. Памнани рассказывает о студенческом проекте по разработке приложения для Android под названием DOT-HAZMAT. Приложение предназначено для использования технологий машинного обучения и компьютерного зрения для поиска и классификации табличек HAZMAT во время спасательных операций. Автор описывает приложение как инновационное решение для повышения безопасности спасателей во время аварий с опасными материалами. Проект подробно объясняется, демонстрируя технические аспекты технологии и ее потенциальное использование в будущем. В целом, автор представляет обзор студенческого проекта и его вклада в улучшение мер безопасности во время инцидентов HAZMAT [14].

А. Весалаинен фокусируется на разработке и оценке различных методов сегментации изображений для детального анализа макета документа OCR. Сначала автор проводит литературный обзор предыдущих исследований по анализу макета документа и сегментации изображений, а

также определяют ограничения и проблемы существующих методов. Затем предлагает и сравнивает различные подходы к сегментации изображений документа на области текста, изображения и фона с использованием таких методов, как адаптивное пороговое значение, обнаружение границ и анализ связанных компонентов. Автор оценивает точность и эффективность каждого метода, используя большой набор данных документов OCR, и обсуждает свои выводы и рекомендации для будущих исследований в этой области. В целом исследование направлено на то, чтобы внести свой вклад в разработку более точных и универсальных систем оптического распознавания символов, способных обрабатывать сложные макеты документов [15].

Авторы работы К. Олейничак и М. Шульц фокусируются на выявлении и анализе проблемы обнаружения текста в системах оптического распознавания символов (OCR). Авторы утверждают, что большинство современных систем оптического распознавания текста в первую очередь ориентированы на распознавание текста из отсканированных документов, но не обязательно на обнаружение текста в изображениях документов. В результате эти системы могут пропускать или неправильно распознавать текст на изображениях с низким разрешением или сложным фоном, что приводит к снижению точности оптического распознавания символов. Для решения этой проблемы авторы предлагают новый подход, сочетающий методы обнаружения текста с системами OCR. Они проводят эксперименты, чтобы сравнить точность этого подхода с существующими системами OCR и демонстрируют, что предлагаемый ими подход повышает точность распознавания для изображений со сложным фоном или разрешением низкого качества. В исследовании подчеркивается важность включения распознавания текста в системы OCR для более точного распознавания текста на изображениях документов [16].

2. Исходные данные

Исходные данные представлены на сайте конкурса Huawei digix global ai challenge 2021 в категории Menu identification leveraging few-shot learning models [17].

Исходный датасет содержится в виде двух файлов архива 2021_5_data.zip и 2021_finals_data_5.zip. В файле архива включаются каталоги train_image_common (3561 изображения), test_image (500 изображения) и train_image_special (93 изображения), в которых содержится изображением с наложенным текстом, а файлы train_label_common.json и train_label_special.json содержат указатели с выделенными изображениями, с размеченными областями и текстом. Всего содержится 4154 изображения.

Архивы 2021_5_data.zip и 2021_finals_data_5.zip отличается только в каталоге test_image тестовые данные (Рис 1).

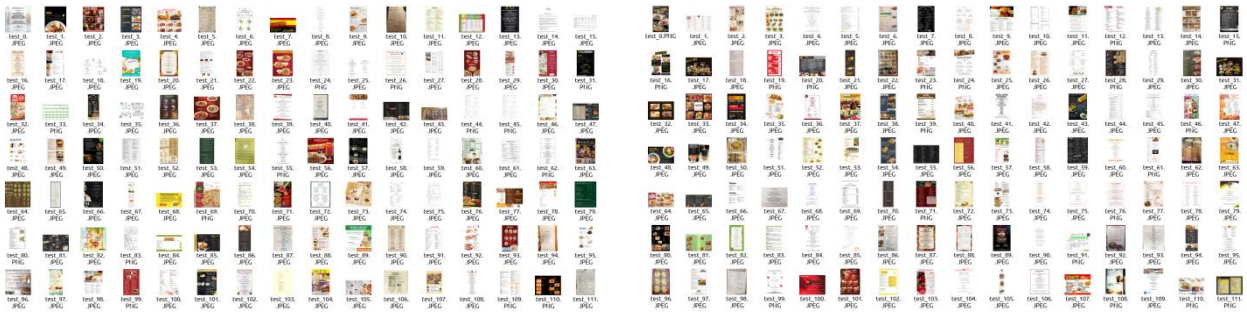


Рисунок 1. Отличие набор данных, слева набор данных из архива 2021_5_data.zip, а справа 2021_finals_data_5.zip, в каталоге test_image

Таблица 1. Архив 2021_5_data.zip и 2021_finals_data_5.zip

Название	Описание
train_image_common	Тренировочные данные для машинного обучения
test_image	Тестовые данные для оценки качества модели
train_image_special	Специальные тренировочные данные для оценки качества модели
train_label_common.json	Файл выделенных тренировочных данных с изображениями текста

train_label_special.json	Файл выделенных специальных тренировочных данных с изображениями текста

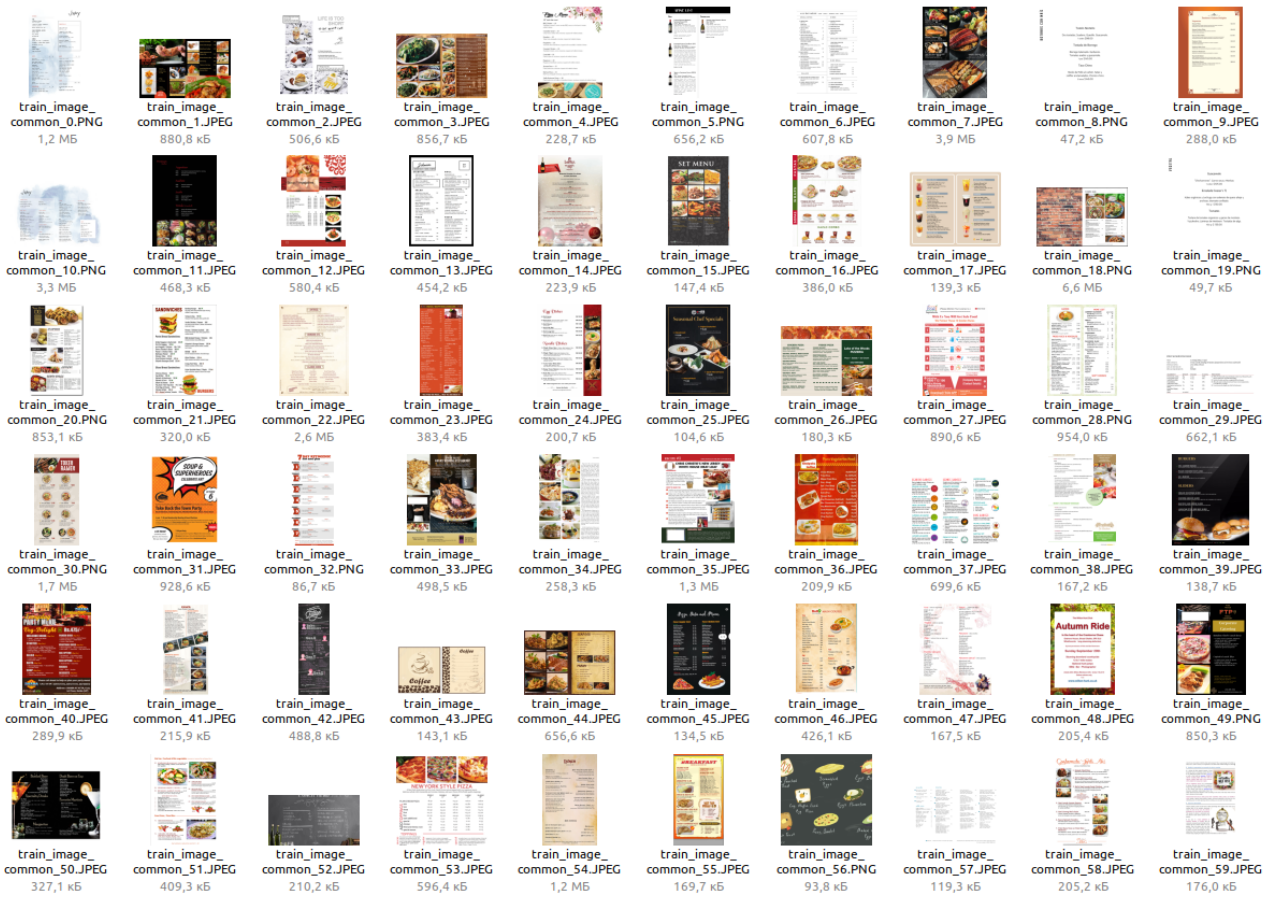


Рисунок 2. Исходные данные в каталоге train_image_common

Файл train_label_common.json содержит ассоциативный массив, ключ — название файла изображения из каталога train_image_common, значение структура (табл 2).

Таблица 2 Файлы train_label_common.json и train_label_special.json

Название	Описание
label	Текст
points	Вершины с координатами x, y с плавающей точкой

Рисунок 3. Чтение файла train_label_common.json

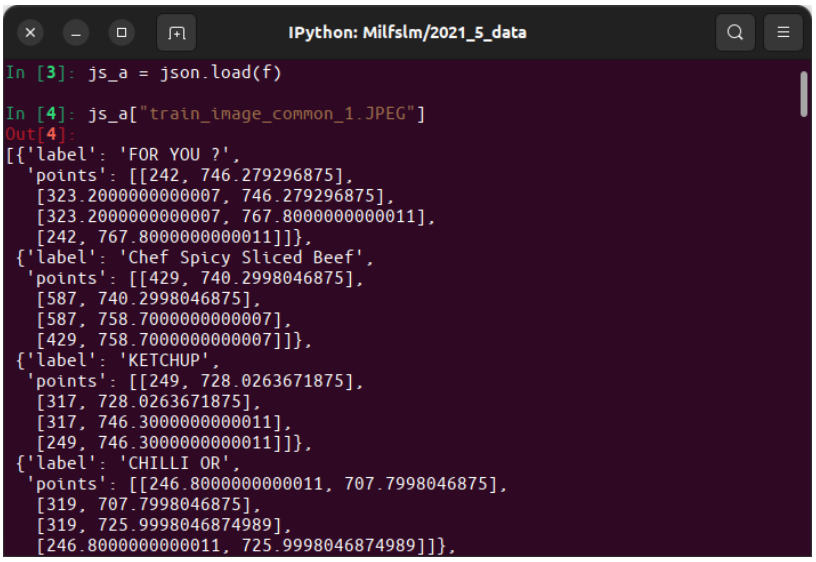


Рисунок 4. Вывод структуры из файла train_label_common.json. Выбранный «train_image_common_1.JPEG»

Рисунок 5. Файл train_image_common_1.JPEG. Расположенный 2021_5_data/train_image_common

Структура файлов train_label_common.json и train_label_special.json:

- Имя файла изображение: Словарь
  - Список
    - label: Строка
    - points: Список
      - Список: по координаты X и Y.

3. Структура программы

Для работы получено базовое решение, исходный код программы 2021-DIGIX-BASELINE в каталоге baseline-game5 [18].

Таблица 3. Файловая структура вывода в программе содержится

Адрес	Описание
output/checkpoints	Содержатся сохраненные и обученные модели
output/checkpoints/detector/checkpoint.pth.tar	Файл, обученный модель (Detector) для поиска текста изображения
output/checkpoints/recognizer	Содержатся серии сохраненные и обученные модели для извлечение текста из изображения (Recognizer)
output/detector_test_output/menu	Содержатся серии изображения выведено выделенный области (нарисованные квадратики) нахождения текста. Как результат

	обученный модель по поискам текста изображения (Detector).
output/recognizer/labels	Содержатся файлы формата json, в структуре файла содержится область нахождения (вершины) текста и текст в результате полученная обученная модель по извлечению текста из изображения (Recognizer).
output/recognizer_images	Содержатся серии изображения, вырезанные из разметки в файла train_label_common.json и train_label_special.json содержащие вершины. Изображение содержит только текст.
output/recognizer_log/train	Содержатся серии журнала обучения модели по извлечению текста из изображения (Recognizer).
output/recognizer_txts/train.txt	Файл содержит список в виде строки, имя изображение (из серии output/recognizer_images) и текст значение.
output/recognizer_txts/real_train.txt	Файл содержит список в виде строки, имя изображение (из серии output/recognizer_images) и текст значение в виде коды символы.
output/test_null.json	Файл в формата json, в структуре файла содержится область нахождения (вершины) текста и пустой текст.

Программа написана на языке Python, которая использует библиотеки PyTorch, OpenCV, Keras, Tensorflow, для ускорения обработки изображения программная часть была написано на языке C++ с использованием библиотеки OpenCV. Для ускорения работы по предсказанию применяется библиотека multiprocessing.

3.1. Запуск detector/train.py

В этой программе выполняется поиск текст изображений (Detector).

Использовали модель ResNet50 и оптимизацию SGD для обучения по нахождению текста в изображения, для оценки обучения применяется метрика Dice (Рис. 2) [19].

$$\text{Dice} = \frac{2 * \text{TP}}{2 * \text{TP} + \text{FP} + \text{FN}}$$

Рисунок 6. Формула метрики Dice

Параметры оптимизация
SGD (Parameter Group 0 dampening: 0 lr: 0.0001 maximize: False momentum: 0.99 nesterov: False weight_decay: 0.0005)

Модель ResNet50

```
OCR_DETECTOR(
  (backbone): ResNet(
    (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu1): ReLU(inplace=True)
    (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
    (layer1): Sequential(
      (0): Bottleneck(
        (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (downsample): Sequential(
          (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
      )
      (1): Bottleneck(
        (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
      )
      (2): Bottleneck(
        (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
```

```
    (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
  )
)
(layer2): Sequential(
  (0): Bottleneck(
    (conv1): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (downsample): Sequential(
      (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
      (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (1): Bottleneck(
    (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
  )
  (2): Bottleneck(
    (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
  )
  (3): Bottleneck(
    (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
  )
)
(layer3): Sequential(
  (0): Bottleneck(
    (conv1): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (downsample): Sequential(
      (0): Conv2d(512, 1024, kernel_size=(1, 1), stride=(2, 2), bias=False)
      (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (1): Bottleneck(
```

```
    (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
  )
  (2): Bottleneck(
    (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
  )
  (3): Bottleneck(
    (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
  )
  (4): Bottleneck(
    (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
  )
  (5): Bottleneck(
    (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
  )
)
(layer4): Sequential(
  (0): Bottleneck(
    (conv1): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (downsample): Sequential(
      (0): Conv2d(1024, 2048, kernel_size=(1, 1), stride=(2, 2), bias=False)
      (1): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (1): Bottleneck(
    (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
```

```
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
  )
)
(neck): FPN(
  (reduce_layers): Sequential(
    (0): Sequential(
      (0): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
      (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU(inplace=True)
    )
    (1): Sequential(
      (0): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1))
      (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU(inplace=True)
    )
    (2): Sequential(
      (0): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1))
      (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU(inplace=True)
    )
    (3): Sequential(
      (0): Conv2d(2048, 256, kernel_size=(1, 1), stride=(1, 1))
      (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU(inplace=True)
    )
  )
  (smooth_layers): Sequential(
    (0): Sequential(
      (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU(inplace=True)
    )
    (1): Sequential(
      (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU(inplace=True)
    )
    (2): Sequential(
      (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU(inplace=True)
    )
  )
)
(head): SIMPLE_DILATE_HEAD(
  (conv1): Conv2d(1024, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu1): ReLU(inplace=True)
```

```
    (conv2): Conv2d(256, 7, kernel_size=(1, 1), stride=(1, 1))
  )
)
```

Аугментация данных выполняется перед каждый шаг обучения.

Этапы аугментация изображения:

1. Используем трех-канальные RGB.
2. Изменяем случайные размер изображение по масштабу от 0.5 до 3.0.
3. Создаем два изображения, текст и маска. Текст — черный, маска (область изображения) — белый фон.
4. Рисуем выделенный область для текста (points), если текст пустой или # (решетка) то накладываем черную область в изображение маска.
5. Создаем несколько изображений маски с названием ядро, в данном случае 7 слоев (Рис 5).
6. Обрабатываем изображение, уменьшаем размер до 640 пикселей, случайно поворачиваем и обрезаем изображение (Рис 6).
7. Изменяем случайно яркость (в значение 32 / 255) и насыщенность (в значение 0.5).
8. Преобразуем в тензор.
9. Нормализуем изображения.

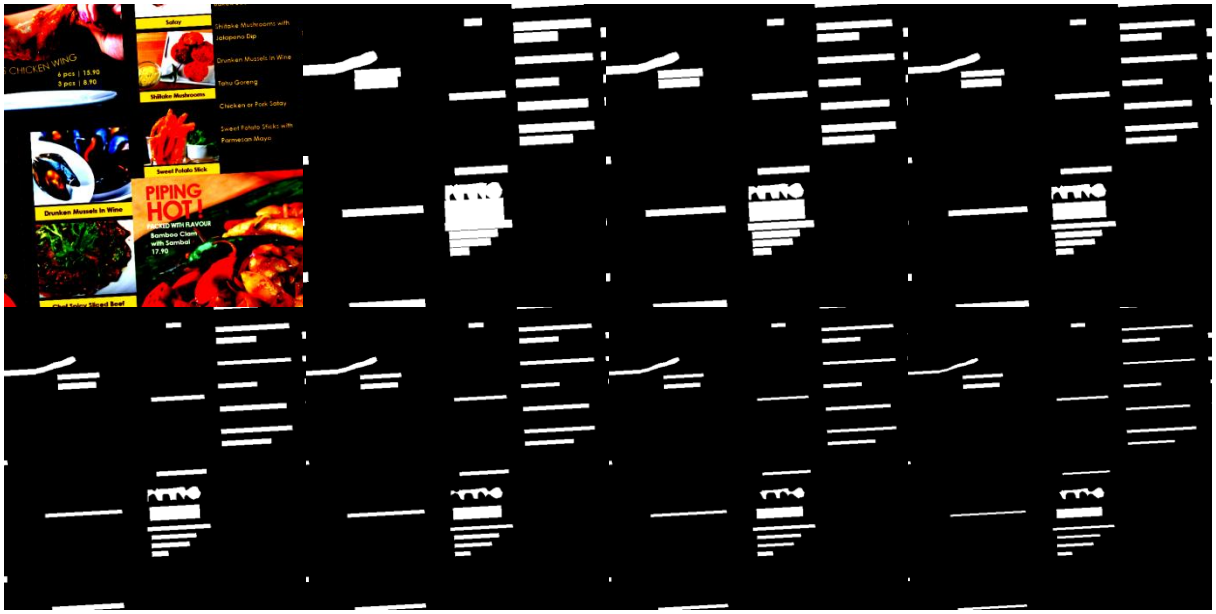


Рисунок 7. Серия изображение вырезано размером 640x640. Вверхни слева стороны изображение

Левый и верхний рисунок - обработанное исходное изображения (готов к обучению), следом черно-белый рисунок — маска под текст, выделенное белое (Маска) геометрический вершины уменьшается в центр, начиная от 0.9 заканчивая до 0.3 (Рис 7). Область обучения (Маска обучение), цвет области устанавливается, если исходные изображение обработано и пересекается за край, белый — распознаваемый область, черный — не распознаваемая область, то есть исходное изображение не лежит в этой области (Рис 8).

Рисунок 8. Левой исходный изображений, справа область распознаваемый изображений (маска распознавания)

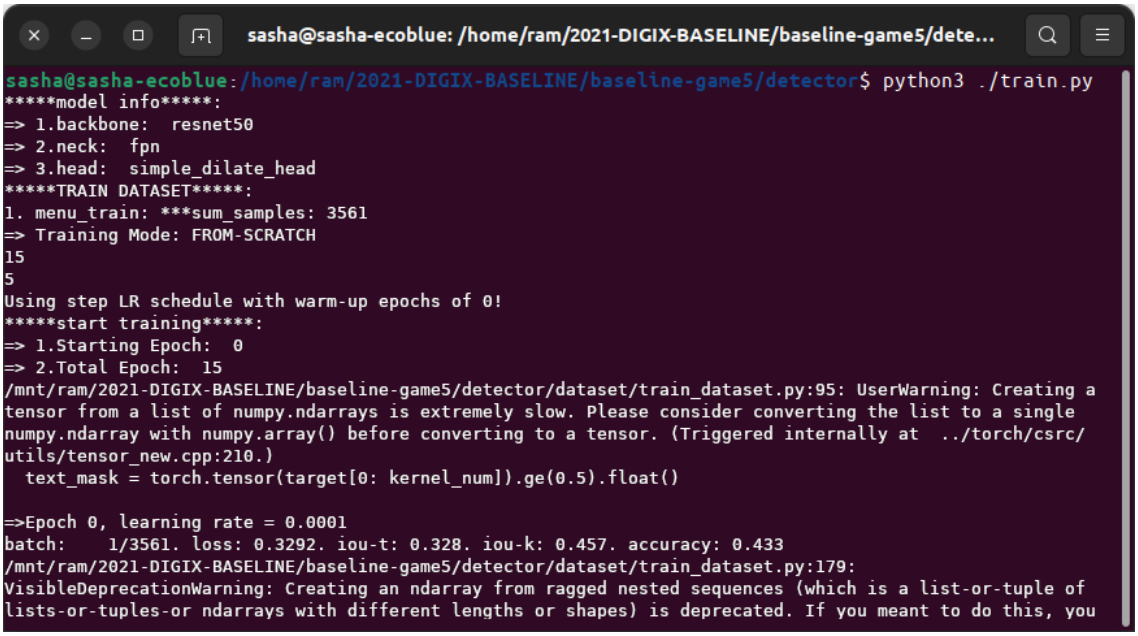


Рисунок 9. Обучения модели

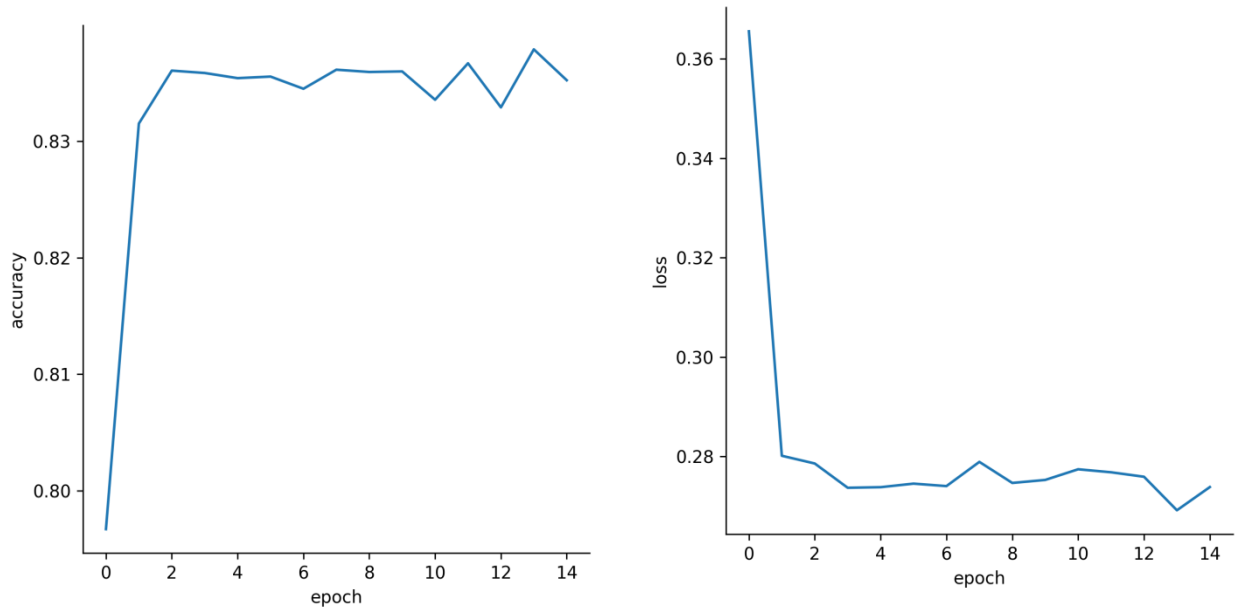
Таблица 4. Вывода данные метрики, batch 3561.

Epoch (Эпоха)	Loss (Dice)	iou-t	iou-k	accuracy
0	0.365571	0.634908	0.623926	0.796715
1	0.280158	0.714952	0.657267	0.831510
2	0.278601	0.719787	0.660720	0.836051
3	0.273721	0.723036	0.661050	0.835848
4	0.273836	0.721631	0.659525	0.835405

Epoch (Эпоха)	Loss (Dice)	iou-t	iou-k	accuracy
5	0.274550	0.721451	0.659668	0.835540
6	0.274054	0.720512	0.661375	0.834495
7	0.278926	0.721446	0.662046	0.836130
8	0.274689	0.722450	0.660161	0.835934
9	0.275296	0.721505	0.661585	0.835982
10	0.277442	0.718966	0.659482	0.833552
11	0.276836	0.721942	0.662015	0.836680
12	0.275936	0.719033	0.658878	0.832896
13	0.269190	0.724763	0.663797	0.837875
14	0.273860	0.720973	0.661448	0.835225

Iou-t (Intersection over Union - text) — оценка сходства текстовое область изображение (маска текста) в модели обучение (табл. 4).

Iou-k (Intersection over Union - kernel) — оценка сходства ядро область изображение (маска область распознавания) в модели обучения (табл. 4).



Accuracy — оценка точности модели для маски текста (табл. 4).

Рисунок 10. На графике представлены левой accuracy, Справа Loss. По оси Ox - epoch (Эпоха)

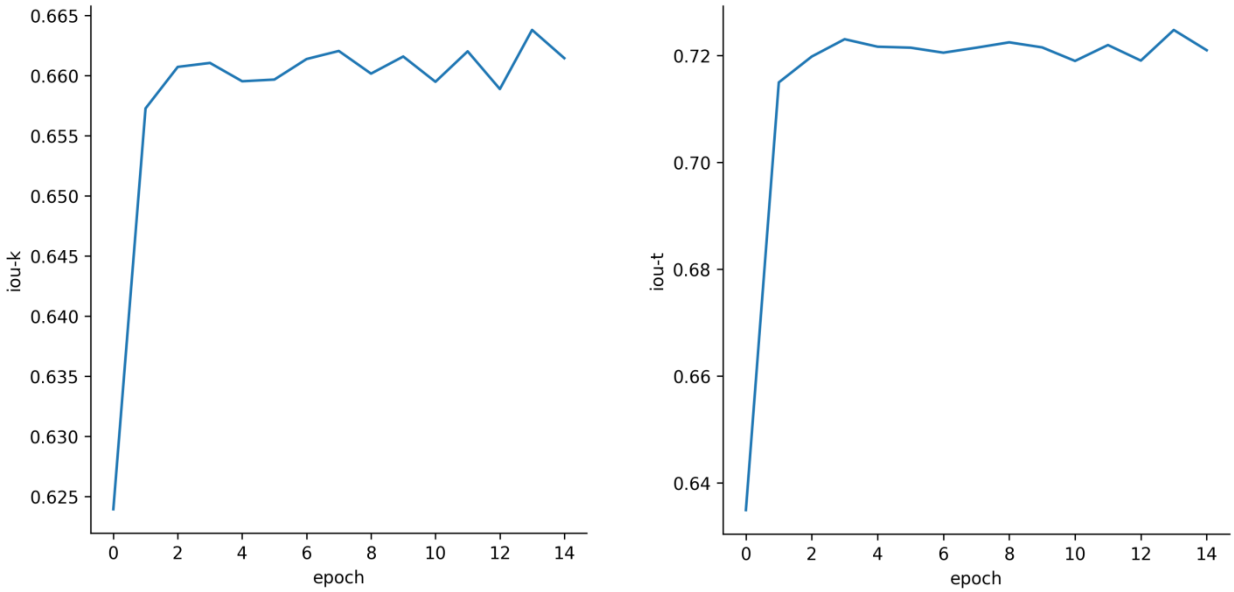


Рисунок 11. На графике представлены левой iou-k, Справа iou-t. По оси Ox - epoch (Эпоха)

Время выполнение программы (учитывается обучение и мониторинг метрики, вывод в консоль и аугментация данных) равно 32136.69 секунд (8 часов, 55.61 минут, 36.69 секунда). Время считается с начало и до конца цикла обучения в каждый эпохи обучения.

Средняя время шаг обучения 0,602 секунда.

Время выполнение обучения (не учитывается мониторинг метрики, вывод в консоль и аугментация данных) равно 15293.21 секунд (4 часов, 14 минут, 53,21 секунда). Время считается с начало и до конца шага обучения в каждый цикл обучения.

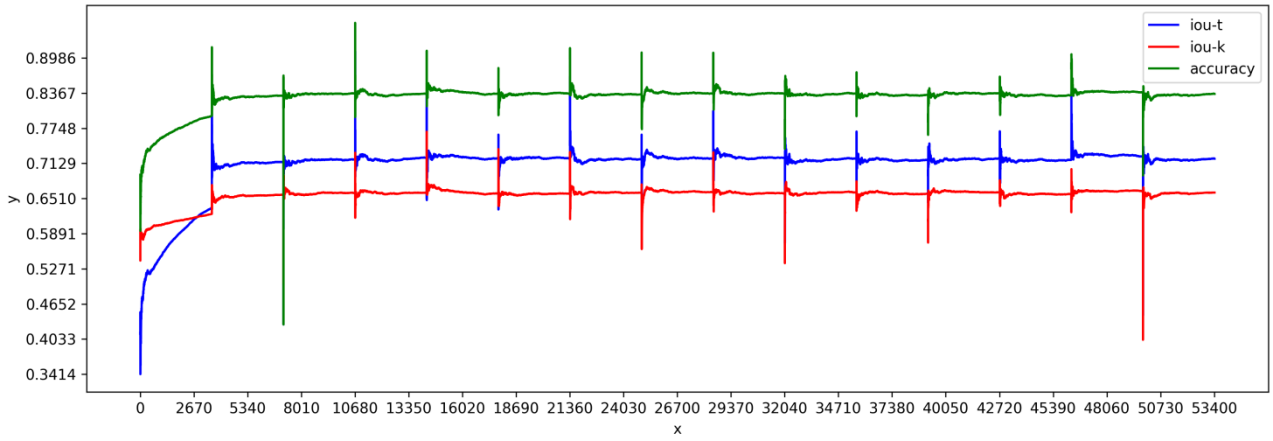


Рисунок 12. На графике представлены цикл обучение за все эпохи

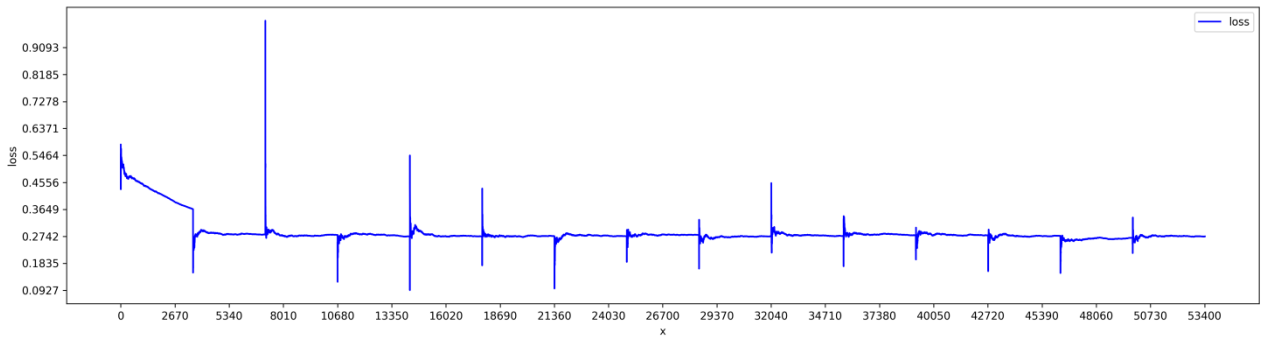


Рисунок 13. На графике представлены цикл обучение за все эпохи с функцией потерь (Dice)

На графике представлены y это метрики accuracy, iou-t, iou-k, loss, а x это цикл обучение за все эпохи (Рис 12, 13). Максимальный цикл обучение за все эпохи равен 53 415.

3.2. Подготовка данных к обучению recognizer

В этой программе подготавливается исходные данные для обучения по извлечению текста из изображений.

1. Чтение файла train_label_common.json и train_label_special.json в формате json.
2. Чтение изображения train_image_common и train_image_special указанный в train_label_common.json и train_label_special.json.
3. Получаем points и сортируем вершины по возрастанию.

4. Вырезаем изображений в области points и выводим в отдельный файл (Рис 15).

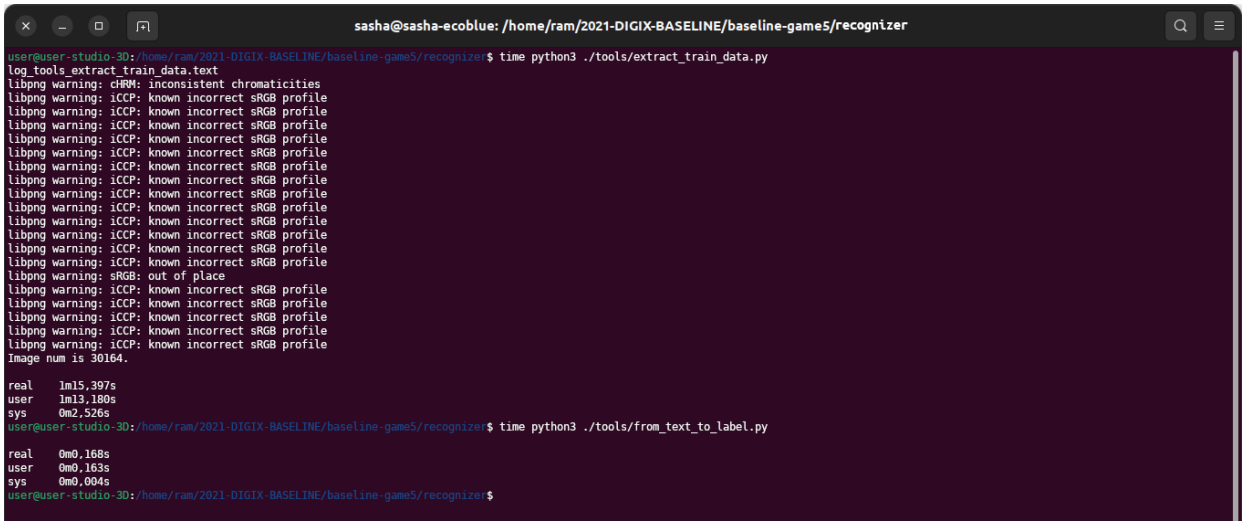


Рисунок 14. Выполнение программы

Файл train.txt

0.jpg	BILLECART SALMON BRUT ROSE
1.jpg	IPA
2.jpg	HITACHINO NEST DAI DAI
3.jpg	KIRIN DRAUGHT
4.jpg	Rich, fragrant, umami
5.jpg	G
6.jpg	Shiitake Mushrooms with
7.jpg	BARK CAFE FAMOUS CHICKEN WING
8.jpg	crazy

Рисунок 15. Вырезанный изображений (файлы 0.jpg — 8.jpg). Наложенный коллаж для представления

Таблица символов. Символы расположенный по порядку. Номер строки начинается с 0.

```
!
"
#
$
%
&
'
(
)
```

```
*
+
,
-
.
/
```

5. Кодируем символы в число.

6. Выводим текстовый файл real_train.txt.

```
0.jpg    34 41 44 44 37 35 33 50 52 0 51 33 44 45 47 46 0 34 50 53 52 0 50 47 51 37
1.jpg    41 48 33
2.jpg    40 41 52 33 35 40 41 46 47 0 46 37 51 52 0 36 33 41 0 36 33 41
3.jpg    43 41 50 41 46 0 36 50 33 53 39 40 52
4.jpg    50 72 66 71 12 0 69 81 64 70 81 64 77 83 12 0 84 76 64 76 72
5.jpg    39
6.jpg    51 71 72 72 83 64 74 68 0 45 84 82 71 81 78 78 76 82 0 86 72 83 71
7.jpg    34 33 50 43 0 35 33 38 37 0 38 33 45 47 53 51 0 35 40 41 35 43 37 46 0 55 41 46 39
8.jpg    66 81 64 89 88
```

Время выполнение программы (от запуска до завершения) равно 1 минуте и 15,565 секунд.

В результате вывода набор вырезанных изображений содержит 30 164 изображений (Рис 15).

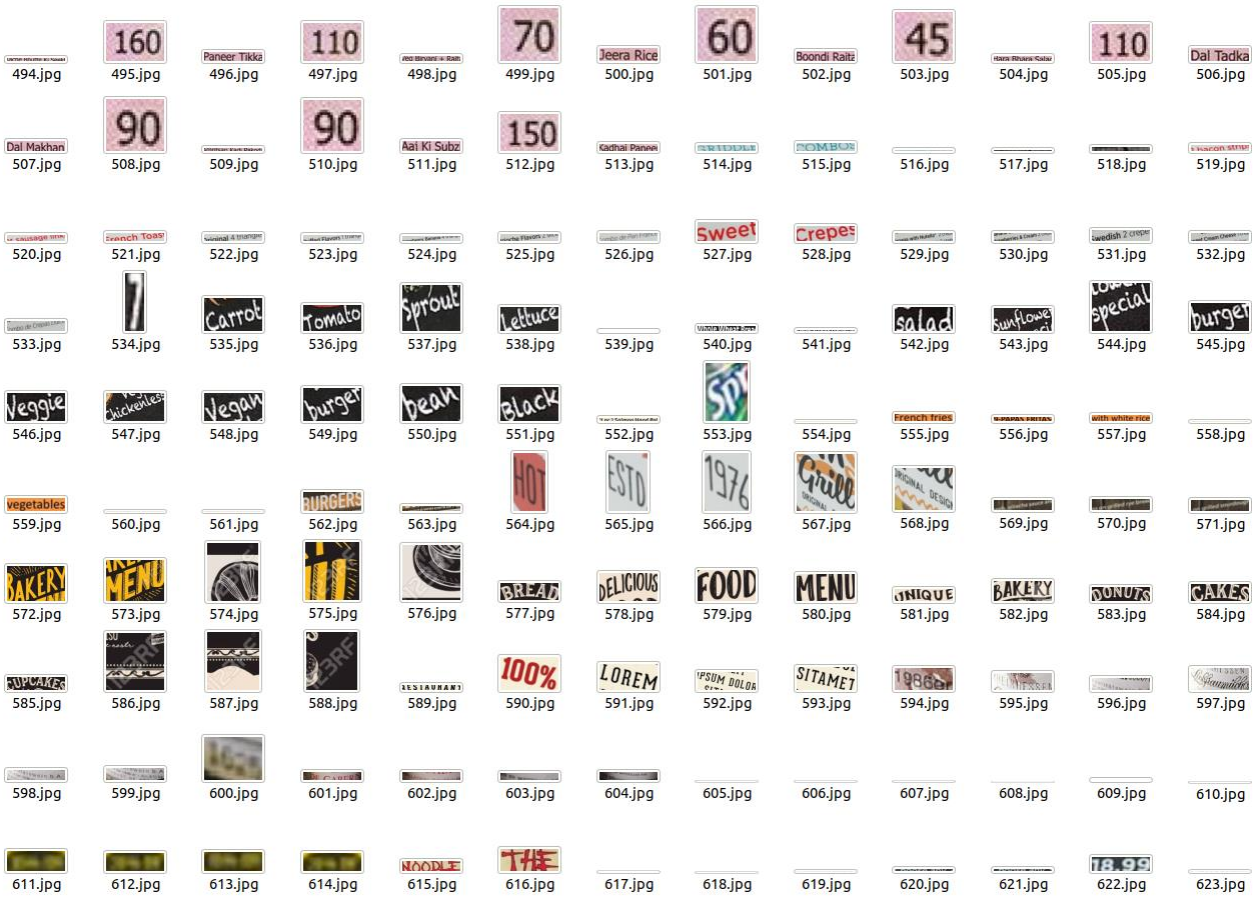


Рисунок 16. Вырезанный изображений

3.3. Запуск recognizer/train.py

В программе выполняется обучение модели по распознаванию текста в изображениях (Recognizer). Используется библиотека Keras.

Параметры обучения:

- Размер пакетов batch_size = 128.
- Максимальная длина строки текста max_label_length = 239.
- Эпоха обучения epochs = 50.
- Модель DenseNet.
- Размер входной свертки 280x32 и 3 компонентный.
- Оптимизатор Adam, скорость обучение 0.0005.
- Метрика для обучения CTC (Коннекционистская временная классификация) [20].

Модель DenseNet

```
_____
Layer (type)                 Output Shape         Param #     Connected to
==================================================================
input_data (InputLayer)      [(None, 32, 280, 3)  0]          []
zero_padding2d (ZeroPadding2D)  (None, 38, 286, 3)  0         ['input_data[0][0]']
conv1/conv (Conv2D)          (None, 16, 140, 64)  9408        ['zero_padding2d[0][0]']
conv1/bn (BatchNormalization)  (None, 16, 140, 64)  256       ['conv1/conv[0][0]']
conv1/relu (Activation)      (None, 16, 140, 64)  0           ['conv1/bn[0][0]']
zero_padding2d_1 (ZeroPadding2D)  (None, 18, 142, 64)  0       ['conv1/relu[0][0]']
pool1 (MaxPooling2D)         (None, 8, 70, 64)    0           ['zero_padding2d_1[0][0]']
conv2_block1_0_bn (BatchNormalization)  (None, 8, 70, 64)   256       ['pool1[0][0]']
conv2_block1_0_relu (Activation)  (None, 8, 70, 64)   0         ['conv2_block1_0_bn[0][0]']
conv2_block1_1_conv (Conv2D)   (None, 8, 70, 128)   8192      ['conv2_block1_0_relu[0][0]']
conv2_block1_1_bn (BatchNormalization)  (None, 8, 70, 128)  512
['conv2_block1_1_conv[0][0]']
conv2_block1_1_relu (Activation)  (None, 8, 70, 128)  0         ['conv2_block1_1_bn[0][0]']
conv2_block1_2_conv (Conv2D)   (None, 8, 70, 32)    36864     ['conv2_block1_1_relu[0][0]']
conv2_block1_concat (Concatenate)  (None, 8, 70, 96)   0
['pool1[0][0]',
'conv2_block1_2_conv[0][0]']
conv2_block2_0_bn (BatchNormalization)  (None, 8, 70, 96)  384
['conv2_block1_concat[0][0]']
conv2_block2_0_relu (Activation)  (None, 8, 70, 96)   0         ['conv2_block2_0_bn[0][0]']
conv2_block2_1_conv (Conv2D)   (None, 8, 70, 128)   12288     ['conv2_block2_0_relu[0][0]']
conv2_block2_1_bn (BatchNormalization)  (None, 8, 70, 128)  512
['conv2_block2_1_conv[0][0]']
conv2_block2_1_relu (Activation)  (None, 8, 70, 128)  0         ['conv2_block2_1_bn[0][0]']
conv2_block2_2_conv (Conv2D)   (None, 8, 70, 32)    36864     ['conv2_block2_1_relu[0][0]']
conv2_block2_concat (Concatenate)  (None, 8, 70, 128)  0         ['conv2_block1_concat[0][0]',
```

```
'conv2_block2_2_conv[0][0]']
conv2_block3_0_bn (BatchNormalization) (None, 8, 70, 128) 512 ['conv2_block2_concat[0][0]']
conv2_block3_0_relu (Activation) (None, 8, 70, 128) 0 ['conv2_block3_0_bn[0][0]']
conv2_block3_1_conv (Conv2D) (None, 8, 70, 128) 16384 ['conv2_block3_0_relu[0][0]']
conv2_block3_1_bn (BatchNormalization) (None, 8, 70, 128) 512 ['conv2_block3_1_conv[0][0]']
conv2_block3_1_relu (Activation) (None, 8, 70, 128) 0 ['conv2_block3_1_bn[0][0]']
conv2_block3_2_conv (Conv2D) (None, 8, 70, 32) 36864 ['conv2_block3_1_relu[0][0]']
conv2_block3_concat (Concatenate) (None, 8, 70, 160) 0 ['conv2_block2_concat[0][0]', 'conv2_block3_2_conv[0][0]']
conv2_block4_0_bn (BatchNormalization) (None, 8, 70, 160) 640 ['conv2_block3_concat[0][0]']
conv2_block4_0_relu (Activation) (None, 8, 70, 160) 0 ['conv2_block4_0_bn[0][0]']
conv2_block4_1_conv (Conv2D) (None, 8, 70, 128) 20480 ['conv2_block4_0_relu[0][0]']
conv2_block4_1_bn (BatchNormalization) (None, 8, 70, 128) 512 ['conv2_block4_1_conv[0][0]']
conv2_block4_1_relu (Activation) (None, 8, 70, 128) 0 ['conv2_block4_1_bn[0][0]']
conv2_block4_2_conv (Conv2D) (None, 8, 70, 32) 36864 ['conv2_block4_1_relu[0][0]']
conv2_block4_concat (Concatenate) (None, 8, 70, 192) 0 ['conv2_block3_concat[0][0]', 'conv2_block4_2_conv[0][0]']
conv2_block5_0_bn (BatchNormalization) (None, 8, 70, 192) 768 ['conv2_block4_concat[0][0]']
conv2_block5_0_relu (Activation) (None, 8, 70, 192) 0 ['conv2_block5_0_bn[0][0]']
conv2_block5_1_conv (Conv2D) (None, 8, 70, 128) 24576 ['conv2_block5_0_relu[0][0]']
conv2_block5_1_bn (BatchNormalization) (None, 8, 70, 128) 512 ['conv2_block5_1_conv[0][0]']
conv2_block5_1_relu (Activation) (None, 8, 70, 128) 0 ['conv2_block5_1_bn[0][0]']
conv2_block5_2_conv (Conv2D) (None, 8, 70, 32) 36864 ['conv2_block5_1_relu[0][0]']
conv2_block5_concat (Concatenate) (None, 8, 70, 224) 0 ['conv2_block4_concat[0][0]', 'conv2_block5_2_conv[0][0]']
conv2_block6_0_bn (BatchNormalization) (None, 8, 70, 224) 896 ['conv2_block5_concat[0][0]']
conv2_block6_0_relu (Activation) (None, 8, 70, 224) 0 ['conv2_block6_0_bn[0][0]']
conv2_block6_1_conv (Conv2D) (None, 8, 70, 128) 28672 ['conv2_block6_0_relu[0][0]']
conv2_block6_1_bn (BatchNormalization) (None, 8, 70, 128) 512 ['conv2_block6_1_conv[0][0]']
conv2_block6_1_relu (Activation) (None, 8, 70, 128) 0 ['conv2_block6_1_bn[0][0]']
conv2_block6_2_conv (Conv2D) (None, 8, 70, 32) 36864 ['conv2_block6_1_relu[0][0]']
conv2_block6_concat (Concatenate) (None, 8, 70, 256) 0 ['conv2_block5_concat[0][0]', 'conv2_block6_2_conv[0][0]']
pool2_bn (BatchNormalization) (None, 8, 70, 256) 1024 ['conv2_block6_concat[0][0]']
pool2_relu (Activation) (None, 8, 70, 256) 0 ['pool2_bn[0][0]']
conv2d (Conv2D) (None, 4, 70, 512) 3277312 ['pool2_relu[0][0]']
reshape (Reshape) (None, 280, 1, 512) 0 ['conv2d[0][0]']
flatten (TimeDistributed) (None, 280, 512) 0 ['reshape[0][0]']
output (Dense) (None, 280, 283) 145179 ['flatten[0][0]']
==================================================================
==
Total params: 3,771,483
Trainable params: 3,767,579
Non-trainable params: 3,904
```

load model done.

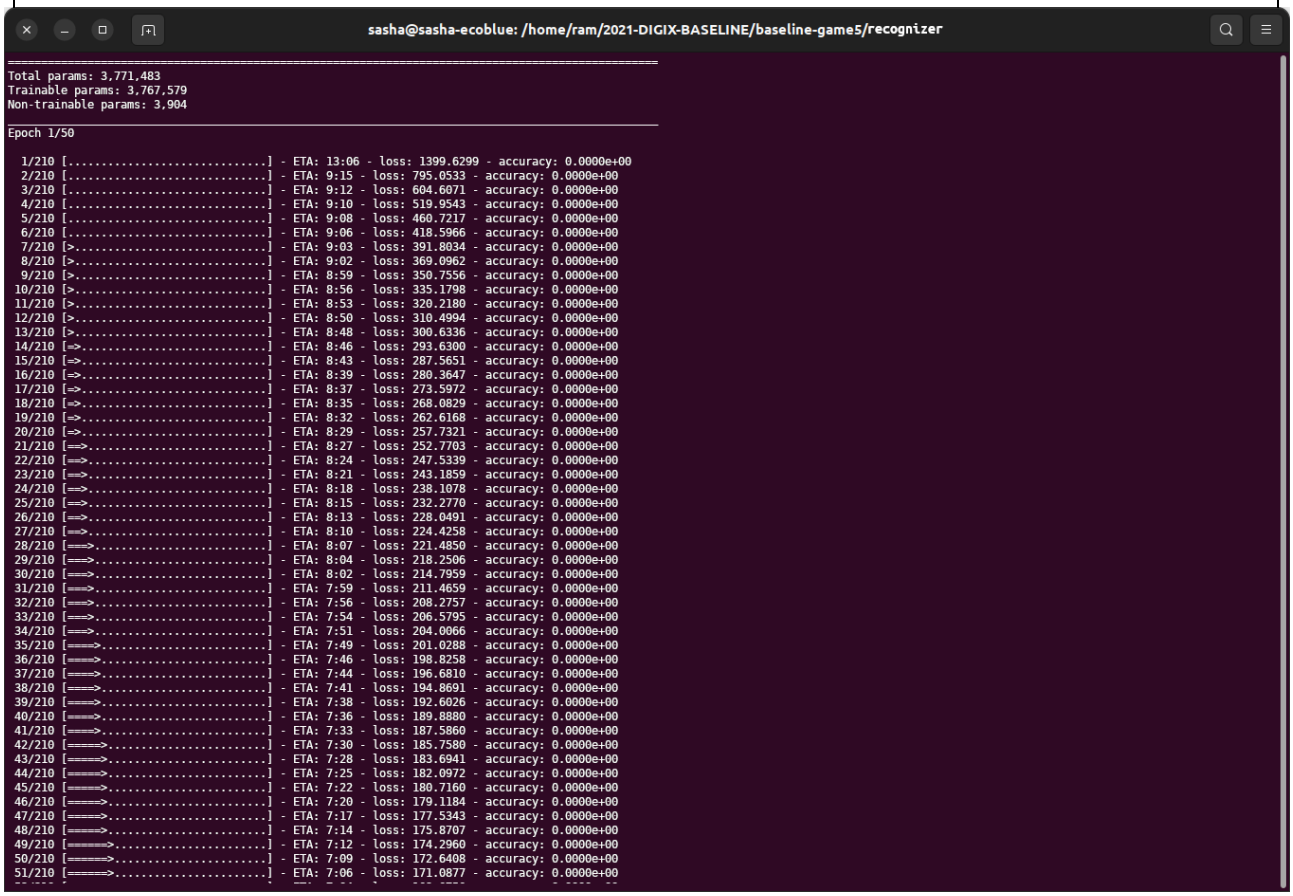


Рисунок 17. Выполнение программы обучение модели

Процесс аугментация данными:

1. Изменение размер изображение на 280x32
2. Преобразования в тензор и нормирования от -1 до 1

Таблица 5. Вывод процесс обучение модели

Epoch (Эпоха)	Loss (CTC)	accuracy	Lr (Скорость)	Время обучения (секунда)
1	102.3565	0.000000	5.000000e-04	569
2	70.6220	0.000000	5.000000e-04	569
3	54.2058	0.000075	5.000000e-04	573
4	40.1689	0.002700	5.000000e-04	573
5	34.6726	0.018500	5.000000e-04	571
6	31.1881	0.052000	5.000000e-04	570

Epoch (Эпоха)	Loss (CTC)	accuracy	Lr (Скорость)	Время обучения (секунда)
7	29.1256	0.081900	5.000000e-04	570
8	27.6356	0.106600	5.000000e-04	570
9	26.2397	0.128200	5.000000e-04	574
10	25.3960	0.146100	5.000000e-04	581
11	24.2791	0.162600	5.000000e-04	569
12	23.3033	0.178700	5.000000e-04	567
13	22.6908	0.195900	5.000000e-04	575
14	22.1181	0.201400	5.000000e-04	574
15	21.4371	0.216200	5.000000e-04	571
16	21.0644	0.226400	5.000000e-04	571
17	20.2271	0.237100	5.000000e-04	565
18	20.0330	0.242500	5.000000e-04	562
19	19.6939	0.255000	5.000000e-04	562
20	18.6772	0.264600	5.000000e-04	562
21	18.3791	0.270700	5.000000e-04	562
22	18.0687	0.279600	5.000000e-04	562
23	17.8101	0.284900	5.000000e-04	563
24	17.0091	0.296600	5.000000e-04	580
25	16.7885	0.302800	5.000000e-04	578
26	16.5632	0.310300	5.000000e-04	583
27	15.6259	0.322900	5.000000e-04	589
28	15.7555	0.329400	5.000000e-04	594
29	13.1467	0.413100	5.000000e-05	593
30	12.2423	0.441500	5.000000e-05	592
31	12.3565	0.444700	5.000000e-05	597
32	12.1558	0.455000	5.000000e-06	605
33	11.7739	0.462700	5.000000e-06	589
34	12.0154	0.459100	5.000000e-06	585
35	11.7770	0.462300	5.000000e-07	594
36	11.8297	0.461500	5.000000e-08	580
37	11.9964	0.463500	5.000000e-09	582

Epoch (Эпоха)	Loss (CTC)	accuracy	Lr (Скорость)	Время обучения (секунда)
38	11.6463	0.461100	5.000000e-10	597
39	12.0612	0.461600	5.000000e-10	583
40	11.9004	0.464100	5.000000e-11	590
41	11.8442	0.459000	5.000000e-12	589
42	11.8288	0.465000	5.000000e-13	584
43	11.5611	0.461900	5.000000e-14	580
44	12.0484	0.462600	5.000000e-14	578
45	11.8042	0.464700	5.000000e-15	572
46	11.9628	0.459400	5.000000e-16	570
47	11.6019	0.461400	5.000000e-17	569
48	11.8933	0.463800	5.000000e-18	570

Всего время обучения: 28 335 секунда (7 часов, 52 минута, 15 секунда).

Время выполнения обучения 27 708 секунда (7 часов, 41 минута, 48 секунда).

Средняя время обучения за одну эпоху: 577.25 (9 минут, 37.25 секунда).

Средняя время шаг обучения 2,699 секунда (3 секунда).

Использовалось CPU без графического ускорения, т.к. для графического ускорения не хватало памяти для выполнения обучения.

На этапе обучение эпоха 48 завершилось, т.к. функция потери перестала уменьшаться, произошло затухание градиента. Сработал критерий останова (табл. 5).

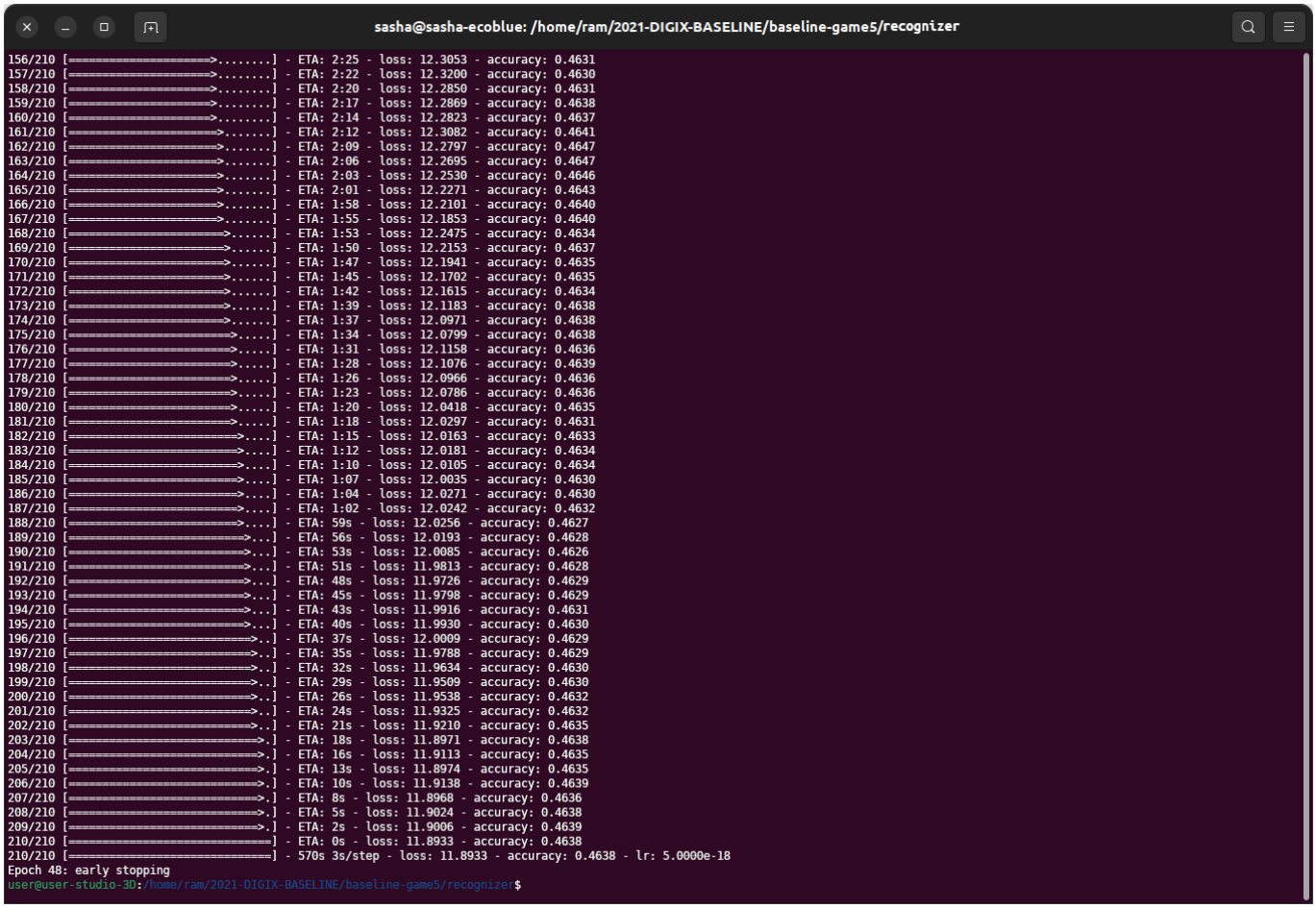


Рисунок 18. Программа завершила обучение, так как скорость обучения упала, градиент затух

Loss — функция потерь, метрика CTC.

Accuracy — оценка обученность модели на тренировочных данных.

Lr — скорость обучения с использование оптимизатора Adam.

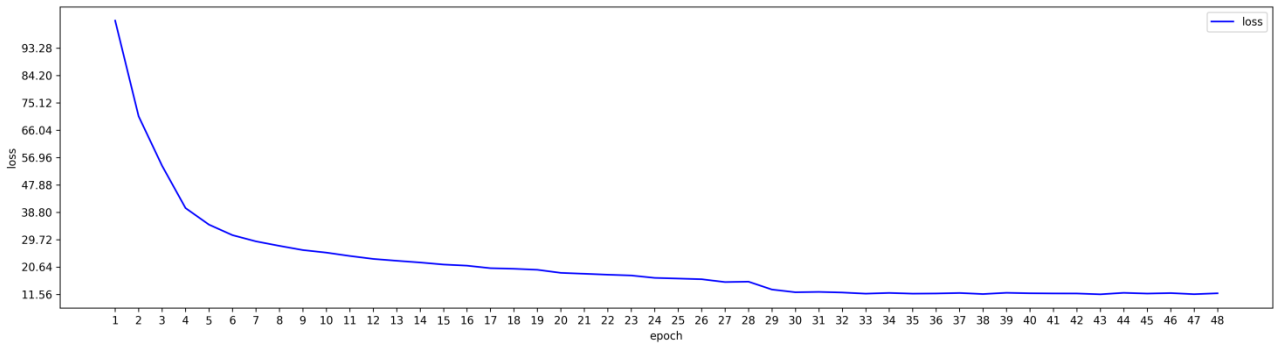


Рисунок 19. На графике представлены по оси x — epoch, по оси y — loss

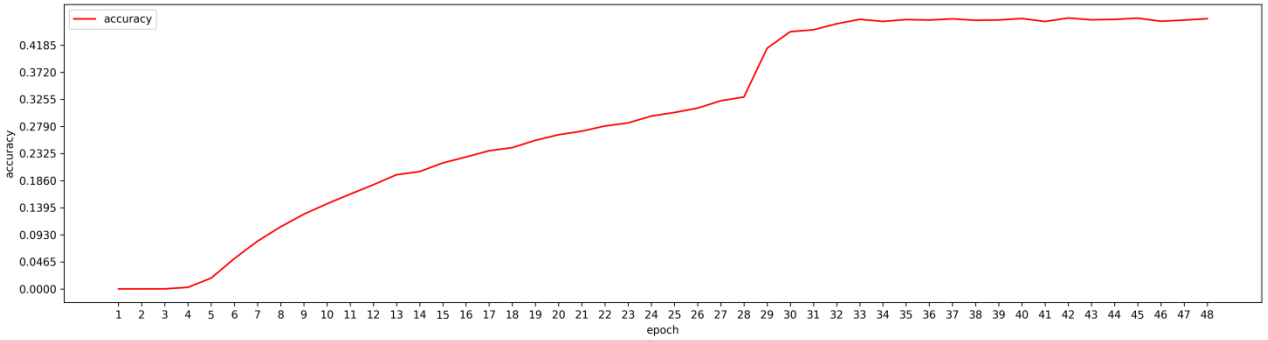


Рисунок 20. На графике представлены по оси x — epoch, по оси y — accuracy

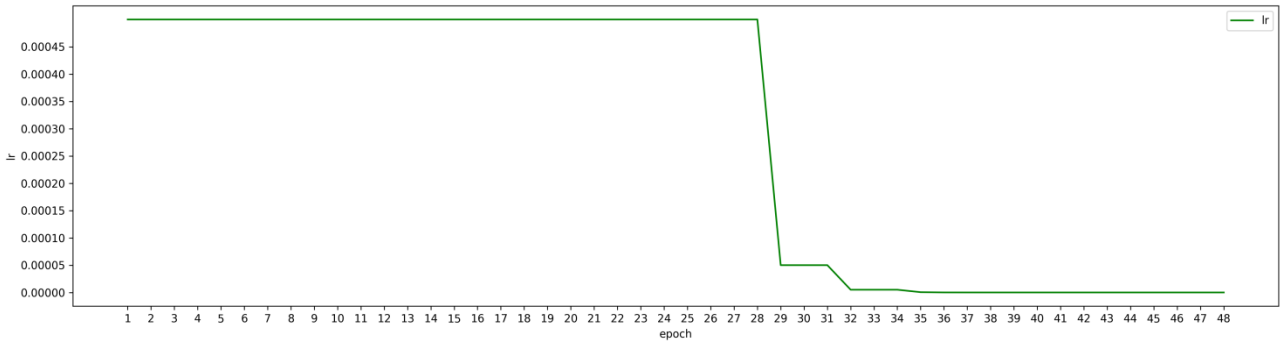


Рисунок 21. На графике представлены по оси x — epoch, по оси y — lr

На графике видно, что (Рис. 19) начиная с эпохи 28, качество модели резко улучшилось, а дальше держится примерно 0.46, а на другом графике (Рис 18), функция потерь держится примерно 11. На графике (Рис 20) скорость обучение начала падать, начиная с эпохи 29 и постепенно стало быстрее падать, т.к. модель стала хуже обучается.

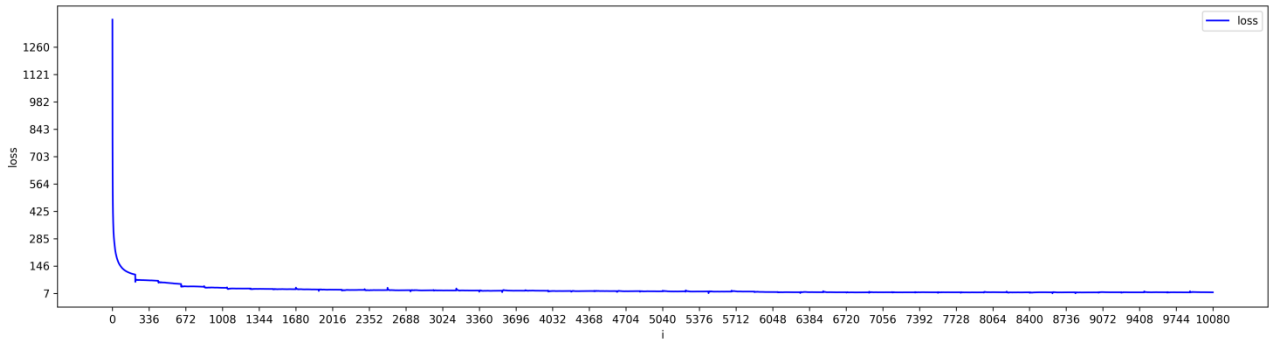


Рисунок 22. На графике представлены по оси x — цикл обучения, по оси y — loss. За весь период обучения

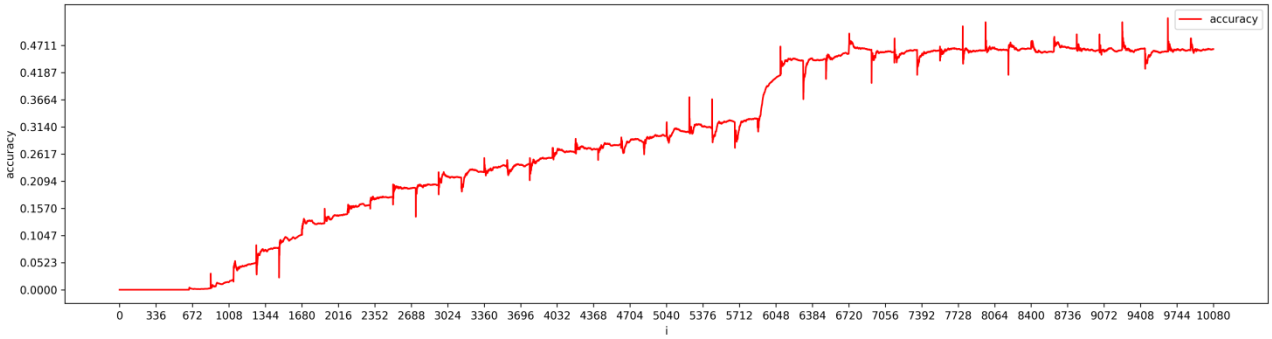


Рисунок 23. На графике представлены по оси x — цикл обучения, по оси y — accuracy. За весь период обучения

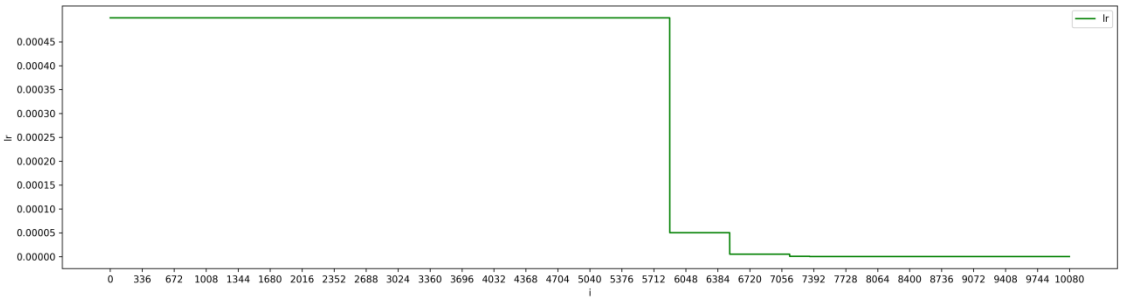


Рисунок 24. На графике представлены по оси x — цикл обучения, по оси y — lr. За весь период обучения

3.4. Запуск detector/predict.py

В программе выполняется предсказание модели по поиску текста в изображении (Detector).

Этапы подготовка изображения:

3. Изменяем размер изображения в приближенный 1920 пикселей.
4. Преобразуем в Tensor и RGB каналы.
5. Нормализация.

Рисунок 25. Сравнение качество обучение, слева — исходные разметки, а справа — разметка результат - обученный модель

Средняя время выполнение обработки одно изображение (учитывая распознавание, ввод модели, рисование область в изображение, вывод в файл) равно 3.601 секунде.

Время выполнение программы для всего изображения 14958.781 секунда (4 часа, 9 минута, 18.781 секунда).

3.5. Запуск recognizer/predict.py

В программе выполняется предсказание модели по извлечению текста в изображения (Recognizer).

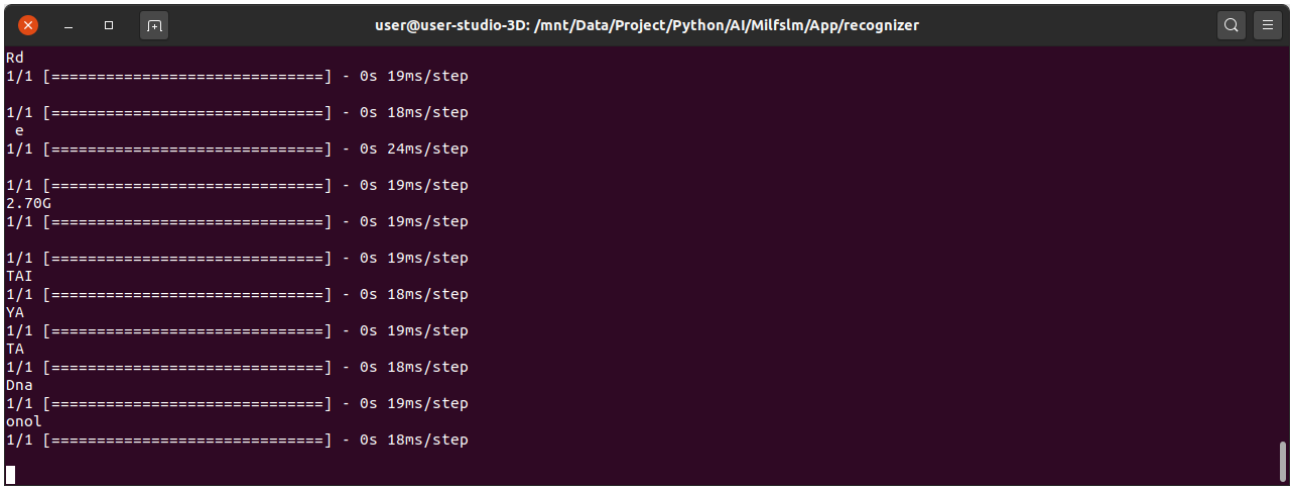


Рисунок 26. Выполнение программы

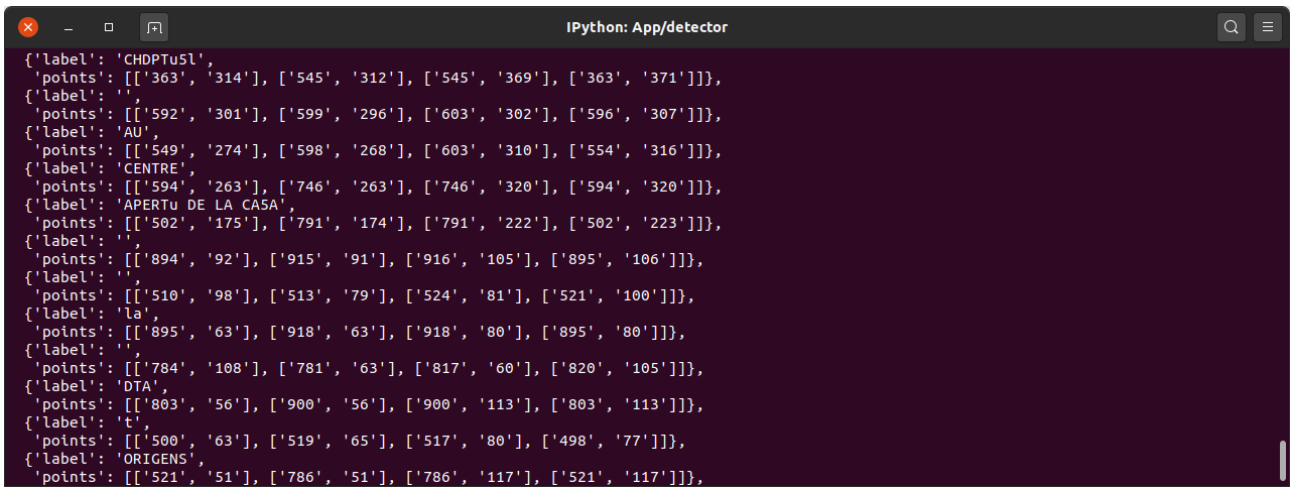


Рисунок 2.27. Пример чтение файла test_0.json

Рисунок 2.28. Файл test_0.JPEG. Выделенное квадратиком текста

Время выполнение программы равно 4292,502 секунд (1 час, 11 минут, 32.502 секунда).

Средняя время выполнение обработки каждого изображения составило 1.254 секунд.

Всего обработано изображений из 3423 файлов.

Заключение

В данной работе представлена разработанная программа для поиска текста в изображениях и получения текста из изображений. В результате обучена модель для оптического распознавания текста. В ходе обучения определена метрика для оценки модели. Несмотря на ошибки и неточности в ходе обучения модели искусственного интеллекта получилось неплохо, модель может при определенных входных данных выдать правильные предсказания. В данном случае модель машинного обучения показала хороший результат по поиску текста изображения с точностью 83,52%. Слабее получен результат модели по извлечению текста из изображения - результат с точностью 46,38%. Время выполнения программы составляет 79171,538 секунд (21 час, 59 минут, 31.538 секунда) (табл 6).

Таблица 6. Время выполнения программы

Программа	Время
detector/train.py	32136.69 секунд (8 часов, 55.61 минут, 36.69 секунда)
Подготовка данных к обучению recognizer	1 минута и 15,565 секунда.
recognizer/train.py	27708 секунд (7 часов, 41 минута, 48 секунда).
detector/predict.py	14958.781 секунд (4 часа, 9 минута, 18.781 секунда).
recognizer/predict.py	4292,502 секунд (1 час, 11 минут, 32.502 секунда).
Сумма	79171,538 секунд (21 час, 59 минут, 31.538 секунда)

Список использованной литературы

1. Culjak I. et al. A brief introduction to OpenCV //2012 proceedings of the 35th international convention MIPRO. – IEEE, 2012. – С. 1725-1730.
2. Pulli K. et al. Real-time computer vision with OpenCV //Communications of the ACM. – 2012. – Т. 55. – №. 6. – С. 61-69.
3. Xie G., Lu W. Image edge detection based on opencv //International Journal of Electronics and Electrical Engineering. – 2013. – Т. 1. – №. 2. – С. 104-106.
4. Agam G. Introduction to programming with OpenCV //Online Document. – 2006. – Т. 27.
5. Qiao L. et al. DavarOCR: A Toolbox for OCR and Multi-Modal Document Understanding //Proceedings of the 30th ACM International Conference on Multimedia. – 2022. – С. 7355-7358.
6. Visintin M., Dever A. Optical Character Recognition. Master Degree Thesis. 2018.
7. Lu N. et al. Master: Multi-aspect non-local network for scene text recognition //Pattern Recognition. – 2021. – Т. 117. – С. 107980.
8. Yu W. et al. PICK: processing key information extraction from documents using improved graph learning-convolutional networks //2020 25th International Conference on Pattern Recognition (ICPR). – IEEE, 2021. – С. 4363-4370.
9. Lin P. H., Wang D. A Optical Character Recognition System with Pattern Editing Functionalities.
10. Kuang Z. et al. MMOCR: a comprehensive toolbox for text detection, recognition and understanding //Proceedings of the 29th ACM International Conference on Multimedia. – 2021. – С. 3791-3794.
11. Zayene O. et al. ICPR2020 competition on text detection and recognition in arabic news video frames //Pattern Recognition. ICPR International Workshops and Challenges: Virtual Event, January 10-15, 2021, Proceedings, Part VIII. – Springer International Publishing, 2021. – С. 343-356.

12. Jaume G., Ekenel H. K., Thiran J. P. Funsd: A dataset for form understanding in noisy scanned documents //2019 International Conference on Document Analysis and Recognition Workshops (ICDARW). – IEEE, 2019. – Т. 2. – С. 1-6.
13. Batra P. et al. A novel memory and time-efficient ALPR system based on YOLOv5 //Sensors. – 2022. – Т. 22. – №. 14. – С. 5283.
14. Pamnani M. DOT-HAZMAT: An Android application for detection of hazardous materials (HAZMAT) A student article showcasing an Android application called DOT-HAZMAT based on Machine Learning and Computer Vision that locates and categorizes HAZMAT placards in perilous rescue operations.
15. Vesalainen A. Image Segmentation methods for fine-grained OCR Document Layout Analysis. Master's thesis. 2022.
16. Olejniczak K., Šulc M. Text Detection Forgot About Document OCR //arXiv preprint arXiv:2210.07903. 2022.
17. GLOBAL AI CHALLENGE URL: https://developer.huawei.com/consumer/en/activity/devStarAI/algo/ (дата обращения: 2021-10-04).
18. GitHub - luoda888/2021-DIGIX-BASELINE: 2021 huawei DIGIX competition baseline // GitHub URL: https://github.com/luoda888/2021-DIGIX-BASELINE (дата обращения: 2022-05-26).
19. Dice — PyTorch-Metrics 0.11.4 documentation // Dice URL: https://torchmetrics.readthedocs.io/en/stable/classification/dice.html (дата обращения: 2023-06-27).
20. CTCLoss — PyTorch 2.0 documentation // CTCLoss URL: https://pytorch.org/docs/stable/generated/torch.nn.CTCLoss.html (дата обращения: 2023-06-27).