ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
ФГАОУ ВО НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Образовательная программа «Прикладная математика и информатика»

Отчет о командном программном проекте на тему:

Детекция и распознавание графиков

Выполнили студенты:

группы #БПМИ213, 3 курса	Кокоева Мария Райбеговна
группы #БПМИ213, 3 курса	Сенкевич Евгений Иванович

Принял руководитель проекта:

Федоров Михаил Антонович
Сотрудник ФКН
Факультет компьютерных наук НИУ ВШЭ

Москва 2024

Содержание

Аннотация

Детекция и распознавание графиков на изображениях, то есть хотим распознавать основную информацию про графики с изображения, а именно тип графика (столбчатый, круговой, линейный), названия столбцов (у тех графиков где они есть), осей (у тех графиков где они есть), и численные значения.

Ключевые слова

Глубинное обучение, машинное обучение, компьютерное зрение

1 Введение

В современном мире для анализа все чаще и чаще требуют обработки изображений с целью извлечения информации из графиков.

Целью данного проекта является разработка системы детекции и распознавания графиков на изображениях с последующим извлечением основной информации из этих графиков. Основные компоненты информации, которые мы стремимся распознавать, включают в себя тип графика (столбчатые, круговые, линейные), названия столбцов (для графиков, где они присутствуют), оси (для графиков, где они присутствуют) и численные значения, представленные на графиках.

Распознавание и анализ данных на графиках имеет важное практическое значение в большом количестве областей, например, для цифровизации книг, учебников, статей (для этого в том числе используется OCR для распознавания текста на изображении). При помощи анализа данных на графиках мы сможем помочь как и в оцифровывании некоторых изданий, так и служить в качестве признаков для моделей, которые анализируют текст (например, для задачи автогенерации краткого содержания).

Как пример можно привести Рисунок 6.2. Мы бы хотели извлечь из данного графика название ("About as simple as it gets, folks"), названия осей ("voltage(mV)"и "time(s)"), и значения которые принимает график.

В нашем проекте мы планируем использовать современные методы компьютерного зрения и глубинного обучения: сегментацию изображений, выделение признаков, классификацию и обнаружение(пока что планируем использовать свёрточные нейронные сети для детекции и классификации графиков, такую как YOLO [2]. Далее опробовать различные модели, которые позволят декодировать изображение в нужный нам набор параметров), OCR, RNN, интерпретация и аннотация графиков.

Для ввода графика и вывода результата мы планируем создать веб-сайт с использованием фреймворка Flask или Django.

Итого всю задачу можно представить как последовательность более простых подзадач:

1 Нахождение графика на изображении.

2 Классификация графика

3 Извлечение информации из графика

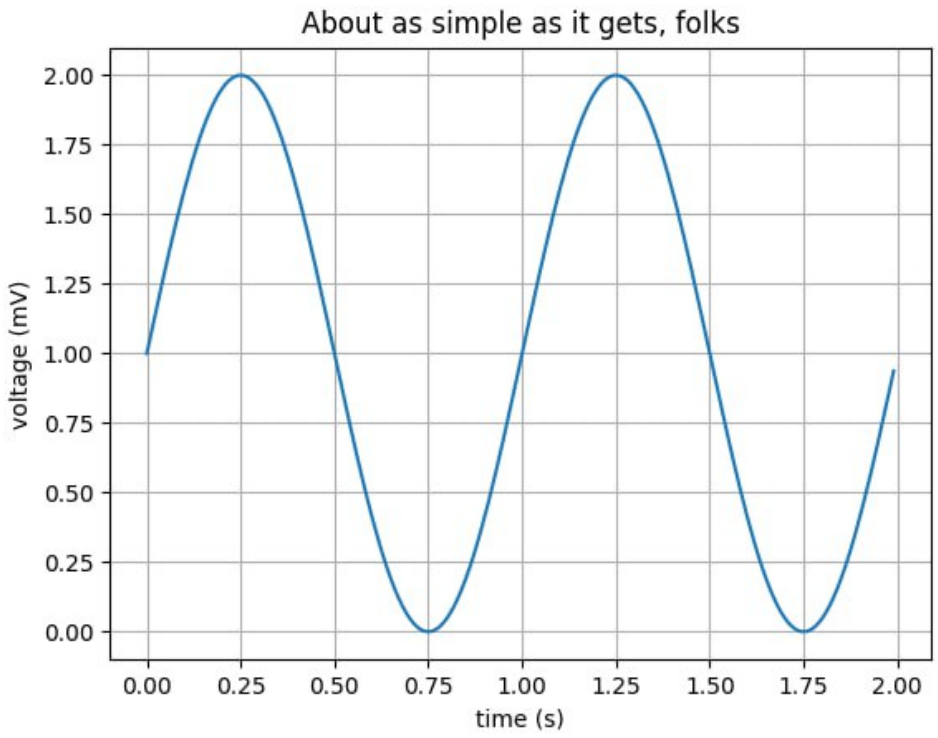

Рис. 1.1: Пример графика

2 Обзор литературы

2.1 COCO

В статье [1] представлен набор данных MS COCO. Авторы предложили новый набор данных с целью улучшения существующих моделей и общего понимания визуальных сцен. Изображения содержат 91 класс объектов, разбитых на 11 больших категорий, 328000 изображений на которых суммарно находится 2.5 миллиона объектов. Для каждого изображения есть маска, то есть для каждого пикселя изображения указано к объекту какого класса он относится.

Так же в статье приводится сравнение с другими наборами данных такими как PASCAL VOC и ImageNet. В сравнении становится заметно, что наборы данных PASCAL VOC и ImageNet в основном имеют изображения с $1-2$ категориями объектов, в то время как COCO содержит в среднем 3.5 категории на изображении. Схожая ситуация и с средним количеством объектов на изображении: PascalVOC (2.3), ImageNet (3.0), COCO (7.7). Исходя из этого можно сделать вывод, что набор данных COCO позволяет моделям лучше обучаться под изображения на которых находится много объектов.

2.2 Faster-RCNN

В статье [4] представляется модель faster-RCNN. Модель применяется для детекции объектов на изображении. Основная идея модели заключается в том, что мы сначала разбиваем изображение на регионы, для каждого региона предсказываем есть ли в этом регионе объект. Далее на каждом регионе, для которого мы предсказали наличие объекта, запускаем классификатор, который предсказывает класс объекта в этом регионе.

В статье описывается архитектура модели, описан процесс обучения модели, функция потерь, которая равна $L(p_i, t_i) = \frac{1}{N_{class}} \sum L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum p^* L_{reg}(t_i, t_i^*)$. В данной функции потерь p_i это предсказание есть ли какой-нибудь из искомых объектов в данном регионе, p_i^* равно 1, если объект действительно есть в регионе и 0 иначе, t_i это предсказание координат bounding box, а t_i^* значения настоящих координат.

Важными понятиями являются Intersection over Union (далее IoU) и mean Average Precision (далее mAP). IoU показывает насколько точно мы определили bounding boxes. $IoU = \frac{\text{Область пересечения}}{\text{Область объединения}}$. Для того, чтобы определить правильно или неправильно мы определили bounding box вводится значение $IoU_{threshold}$ и если посчитанное значение $IoU < IoU_{threshold}$, то считается, что границы объекта найдены неправильно, иначе правильно. Для того, чтобы считать mAP, мы берём некоторый набор $IoU_{threshold}$ (авторы статьи берут $IoU_{threshold}$ с шагом 0.05) и считают для каждого $IoU_{threshold}$ AveragePrecision, далее усредняют по количеству $IoU_{threshold}$ в наборе

Так же в данной статье описываются эксперименты на наборе данных MS COCO [1]. MS COCO [1] содержит 164000 изображений, с размеченными bounding boxes объектов и классами этих объектов. По результатам экспериментов получили значение $mAP@.5 = 41.5$

2.3 YOLO

В статье [2] впервые представляется модель YOLO. Основная идея заключалась в том, чтобы учиться предсказывать вероятность класса для каждой области изображения, в отличие от того как это было в модели faster-RCNN [4], где мы сначала предсказывали наличие объекта, а далее вероятность класса.

На наборе данных Pascal VOC 2007 модель YOLO показывает качество $mAP = 63.4$ при скорости работы $45 fps$(frames per second), в то время как у модели fasterRCNN $mAP = 62.1$ и скорость работы $18 fps$. Таким образом модель YOLO показывает себя как более качественная и быстрая.

2.4 YOLOv8 пример обучения

В статье [3] описан пример обучения модели YOLOv8 для задачи обнаружения и классификации летающих объектов. Авторы статьи выбрали модель YOLOv8 так как она считается state-of-the-art (наилучшей в своей области). Так же авторы использовали предобученную на наборе данных COCO [1] модель YOLOv8 и дообучали её под свою задачу.

Общая функция потерь выглядит так: $L(\theta) = \frac{\lambda_{box}}{N_{pos}} L_{box}(\theta) + \frac{\lambda_{cls}}{N_{pos}} L_{cls}(\theta) + \frac{\lambda_{dfl}}{N_{pos}} L_{dfl}(\theta) + \phi||\theta||_2^2$. Здесь L_{cls} функция потерь для задачи классификации, L_{box} функция потерь для задачи нахождения границ объекта, L_{dfl} dual focal loss функция потерь, которая помогает бороться с несбалансированными классами.

Функция потерь, которая используется в YOLOv8 выглядит вот так:

$$\begin{aligned}\mathcal{L} = \frac{\lambda_{box}}{N_{pos}} \sum_{x,y} 1_{c^*_{x,y}} \Big[1 - q_{x,y} + \frac{\|b_{x,y} - \hat{b}_{x,y}\|_2^2}{\rho^2} + \alpha_{x,y}\nu_{x,y}\Big] \\ + \frac{\lambda_{cls}}{N_{pos}} \sum_{x,y} \sum_{c \in classes} y_c log(\hat{y}_c) + (1 - y_c) log(1 - \hat{y}_c) \\ + \frac{\lambda_{dfl}}{N_{pos}} \sum_{x,y} 1_{c^*_{x,y}} \Big[- (q_{(x,y)+1} - q_{x,y}) log(\hat{q}_{x,y}) \\ + (q_{x,y} - q_{(x,y)-1}) log(\hat{q}_{(x,y)+1})\Big]\end{aligned}$$

, где:

$$q_{x,y} = IoU_{x,y} = \frac{\hat{\beta}_{x,y} \cap \beta_{x,y}}{\hat{\beta}_{x,y} \cup \beta_{x,y}}$$

$$\nu_{x,y} = \frac{4}{\pi^2}(arctan(\frac{w_{x,y}}{h_{x,y}}) - arctan(\frac{\hat{w}_{x,y}}{\hat{h}_{x,y}}))^2$$

$$\alpha_{x,y} = \frac{\nu}{1 - q_{x,y}}$$

$$\hat{y}_c = \sigma(\cdot)$$

$$\hat{q}_{x,y} = softmax(\cdot)$$

Чтобы подчеркнуть успех модели авторы статьи поставили себе 3 сложные задачи: классифицировать и находить очень маленькие объекты; объекты, которые сливаются с окружением; классифицировать разные типы летающих объектов. В результате авторам удалось достичь поставленных задач и обнаруживать объекты, которые тяжелы для обнаружения человеческим глазом.

Таким образом после прочтения статьи у нас было понимание как обучать модель, а так же понимание того, какого качества мы можем добиться при её использовании

3 Генерация данных

Для первого этапа нам нужно обучить классификатор и детектор графика на входном изображении. Для обучения модели требуются размеченные данные. Мы решили, что так как мы умеем сами строить графики, то можем сами сгенерировать обучающую выборку, похожую на реальные изображения. Основными преимуществами сгенерированных данных над размеченными является то, что мы можем сгенерировать большую выборку, про которую мы всё знаем абсолютно точно (отсутствуют ошибки связанные с человеческим фактором, которые неизбежно возникают при разметке). При генерации входных изображений нам потребуется сохранять: координаты графика на изображении (для обучения детекции), тип графика (для обучения классификатора), данные графика (для обучение декодера).

Сначала надо было определиться с выбором библиотеки. Мы рассматривали три варианта: matplotlib, seaborn, plotly.

- matplotlib:

  Библиотека matplotlib является самой популярной библиотекой для построения графиков. График с помощью данной библиотеки строится по шагам: сначала задаются объекты-фигуры, объекты-оси, далее на объекты-оси добавляются объекты-графики, далее к объектам-графикам добавляются название графика и название осей. За счёт такого подхода код для построоения графика выглядит перегруженно, однако благодаря этому имеется доступ к параметрам каждого объекта, что позволяет доставать информацию о расположении каждого объекта графика на итоговом изображении. Генерация данных происходит быстро, что позволяет создать датасет на несколько десятков тысяч изображений имея ограниченные вычислительные ресурсы. Так же стоит отметить, что так как эта библиотека является самой популярной библиотекой для построения графиков, то данные полученные при помощи данной библиотеки наиболее похожи на те данные для распознавания которых мы хотим обучить модель.

- seaborn:

  Библиотека seaborn является надстройкой над библиотекой matplotlib. Популярность данная библиотека приобрела за счёт того что позволяет в одну строчку кода, строить

графики, которые потребовали бы большого количества строк кода при использовании исходной библиотеки matplotlib в плюсах библиотеки так же входит приятная глазу стандартная цветовая гамма графиков в то время как в matplotlib её нужно явно задавать. Генерация изображений происходит так же быстро как и при помощи библиотеки matplotlib. Однако, так как библиотека писалась в первую очередь для облегчения построения графиков и уменьшения размера кода, то из полученных объектов не получается достать информацию о отрисовке деталей графика, необходимой для генерации обучающей выборки

- plotly:

  Библиотека получила популярность за счёт построения интерактивных графиков. Все визуализации производятся с помощью инструкций JavaScript за счёт этого в отрисованном окне можно динамически изменять масштаб графика, при наведении курсора смотреть значения точек включать и отключать фигуры на графике. При всех данных преимуществах в plotly тяжело получить доступ к функциям отрисовщикам, а, значит, тяжело получить разметку для данных графика, так же plotly значительно дольше строит график. Так как plotly используется для построения интерактивных графиков, то для статических изображений данная библиотека не используется, мы хотим расспознавать графики со статических изображений, поэтому plotly не подходит по данному пункту

Вся генерация данных будет происходить поэтапно: сначала мы генерируем данные, затем обучаем модель так, чтобы она хорошо работала на этих и предыдущих данных, далее делаем генерацию визуально более похожей на реальные данные и повторяем всё по циклу, пока не достигнем желаемого качества у решаемой задачи.

Сначала мы просто научились строить графики в matplotlib и вставлять их на белый фон. Про процесс обучения модели будет в следующей секции, пока отметим, что у нас получилось находить и распознавать графики для таких простых примеров.

Далее мы изменяли фон: на случайные шумы, на различный фиксированный набор фонов, на случайно сгенерированный текст вокруг графика. Убедившись, что модель работает на таких данных, перешли к следующему этапу

Последний этап заключался в том, чтобы добавить аугментированных изображений, чтобы повысить обобщающую способность модели. Мы использовали цветовые аугментации: ColorJitter, GrayScale, Contrast, а так же применяли к изображениям преобразование GaussianBlur. Обучив модель на таких данных и убедившись, что не потеряли в качестве

на неаугментрованных изображениях, заключили, что этап с классификацией и детекцией выполнен успешно.

Для генерации данных для обучения моделей распознавания общей информации с графиков и специализированной информации у нас было два подхода:

- Первый подход заключался в том, чтобы просто генерировать изображение графика и подавать такие изображения с разметкой для обучения модели. Плюсом такого подхода была скорость генерации обучающей выборки. Минусом являлось то, что мы действуем в предположении, что модель-детектор идеально распознает границы графика, что может быть не так

- Второй подход заключался в том, чтобы взять имеющийся набор данных, подавать изображения на вход обученной модели-детектору и вырезать область, предсказанную моделью. Плюсом является то, что таким образом данные становятся похожими на те, которые будут приходить моделям после обучения во время работы на реальных данных. Минусом же является долгая генерация данных, так как для построения каждого обучающего примера нужно сделать предскание с помощью модели-детектора

Мы решили использовать второй подход, так как с помощью него получаются данные близкие к реальным.

4 Обучение модели для детекции и классификации

Для начала надо было выбрать модель под нашу задачу. Мы хотим уметь находить на изображении график и классифицировать его. Так же мы бы хотели уметь работать с ситуациями. когда у нас на одном изображении несколько графиков, поэтому мы решили использовать модели из семейства YOLO [2]. Решив, что наша задача не должна быть тяжёлой мы выбрали предварительно обученную на наборе данных COCO [1] модель YOLOv8n [3], как самую небольшую по параметрам модель YOLOv8 [3].

Сравнение девайсов для обучения Далее буду приводиться сравнения для размера батча 32, модели yolov8s и 10000 изображений в обучающей выборке

- mps:

  Обучение на mps происходило быстрее чем на cpu, Однако при обучении на mps наблюдается проблема, при которой модель начинает расходиться. Мы нашли несколько

сообщений о данной проблеме. В некоторых случаях помогало увеличить размер батча, однако у нас при увеличении размера батча, качество действительно несколько повышалось, но процесс обучения всё ещё сходился. Итого преимуществом является быстрота относительно cpu: одна эпоха занимала 50 минут, минусом является крайне нестабильный процесс обучения

- cpu:

  Обучения на cpu является самым долгим: эпоха по приблизительной оценке занимает 13,5 часов. К преимуществам использования cpu можно отнести стабильный процесс обучения, а так же небольшая стоимость вычислительных часов. Главным минусом является очень долгое обучение

- gpu:

  Обучение на gpu является самым быстрым: одна эпоха занимает всего 2 минуты. К преимуществам gpu можно отнести крайне быстрое время работы и стабильный процесс обучения. Главным минусом является дороговизна вычислительных часов

Далее мы обучали модель согласно плану описанному в предыдущей секции. Проблем с обучением почти не возникало. Из проблем мы обнаружили, что модель плохо работает и обучается на случайном фоне, поэтому решили отказаться от изображений со случайным фоном, оставив все предыдущие идеи.

Для обучения модели использовался API от ultralytics

В итоге мы получили небольшую модель, которая, точно классифицирует и обнаруживает графики на изображениях

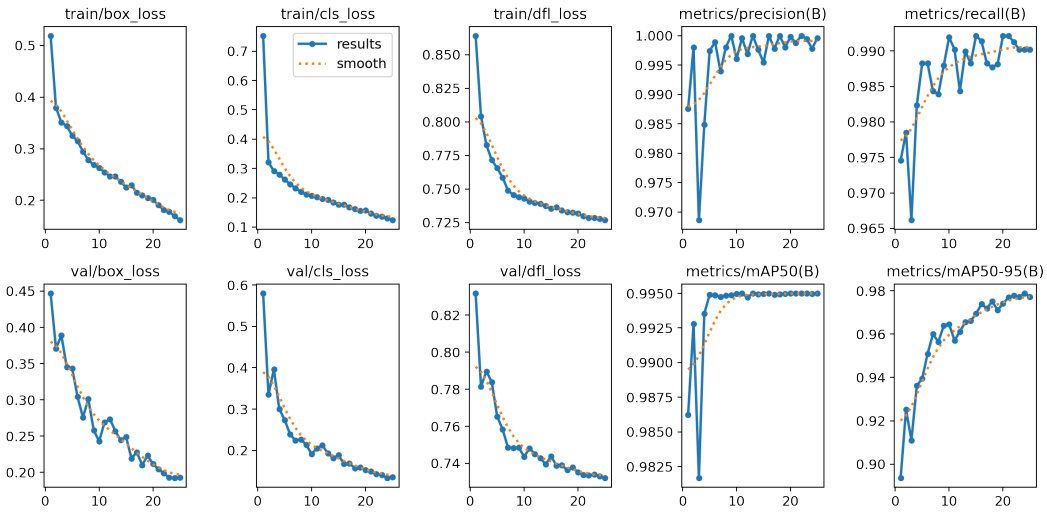

Рис. 4.1: График обучения

5 Обучение модели для распознавания общей информации

К общей информации относятся те элементы, которые есть на каждом графике не зависимо от его типа: название графика, названия осей, численные значения делений, сами деления. Сначала мы обучили модель YOLOv8 для детекции и классификации Модель обучилась крайне точно детектировать названия графика и осей, а так же численные значения делений. Однако не очень точно распознаёт сами деления: это связано с тем, что деление является мелкой частью изображения и для семейства YOLO это может являться сложностью. Однако для наших дальнейших целей данного качества хватало, поэтому мы остановились на этом Для распознавания текста и численных значений мы решили использовать OCR движок Texteract и его API для python PyTexteract. В результате получили модель, которая точно и быстро распознаёт общую информацию с графиков

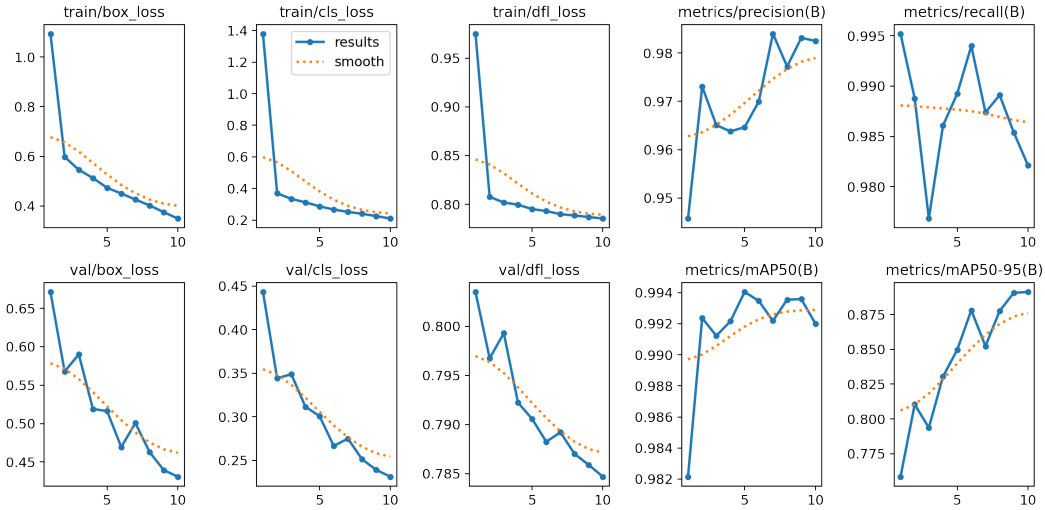

Рис. 5.1: График обучения

6 Обучение моделей для распознавания специализированной информации

К специализированной информации относятся точки для типа scatter и столбики для столбчатых графиков.

Для двух типов обучили модель YOLOv8. С обучением на столбцы проблем не возникло, получилось хорошее качество

С обучением на точки получили качество хуже, так как точки это небольшой объект. Так же некоторые точки могут наслаиваться друг на друга или сливаться с графиком. Для улучшения качества повышали размерность входных изображений, что дало небольшой прирост к качеству.

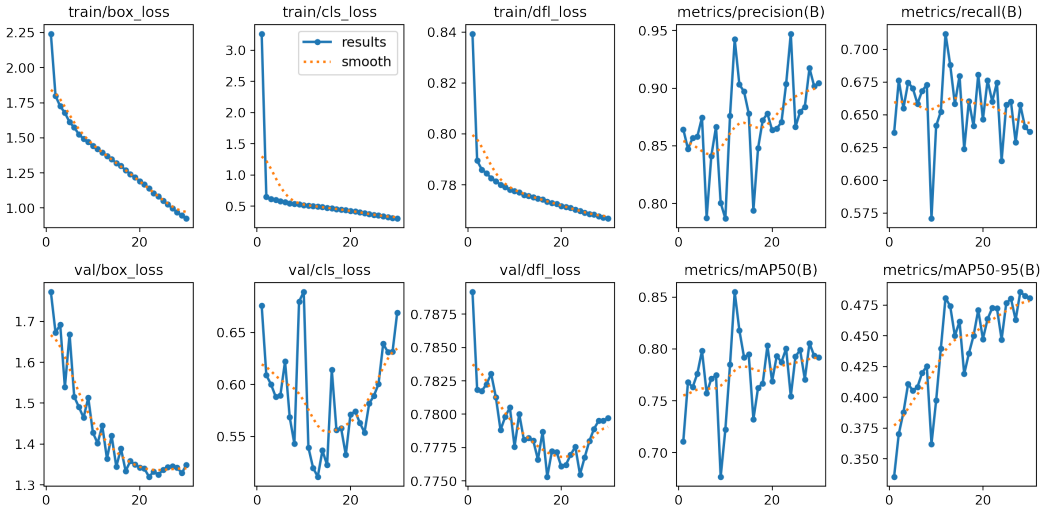

Рис. 6.1: График обучения

7 Итоговая схема работы модели

Итоговая схема работы выглядит следующим образом. Сначала запускаем модель-детектор, которая находит и классифицирует график. Далее вырезаем предсказанную данной моделью часть изображения и подаём её на вход модели, которая распознаёт общую информацию, и модели, которая распознаёт специализировпнную информацию (для каждого типа графика своя модель).

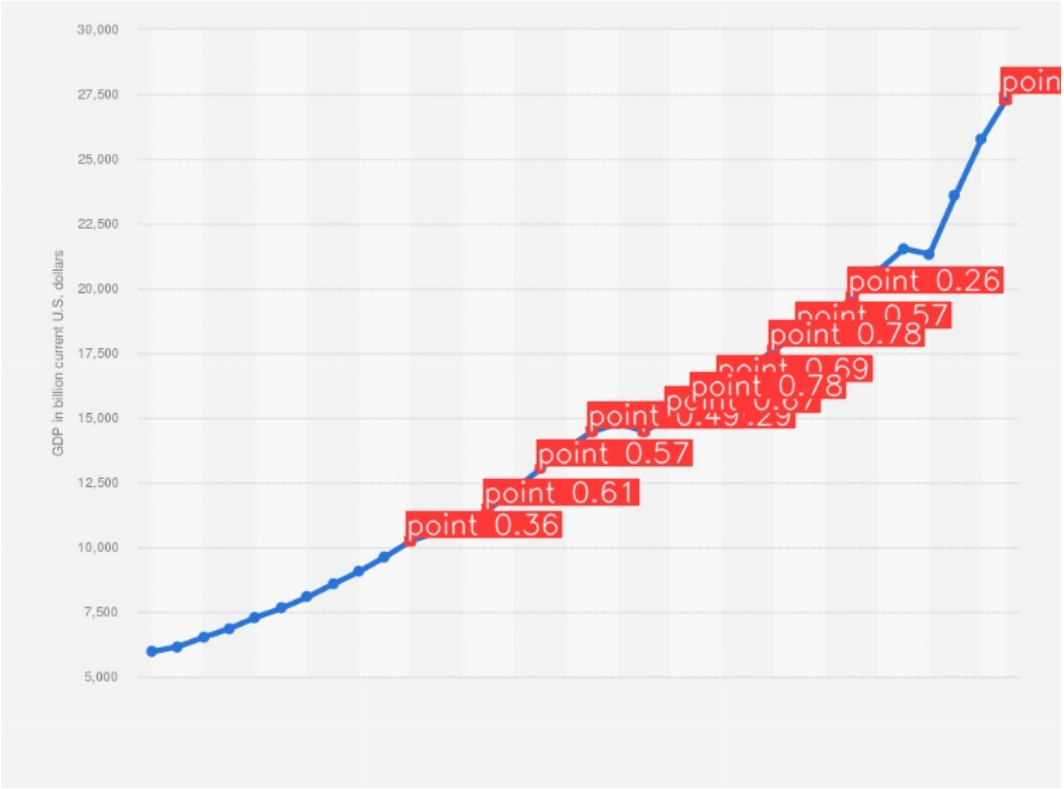

Рис. 6.2: Пример распознавания

8 План выполнения работы

1. Сгенерировать данные, на которых планируем далее обучаться (делает Мария Кокоева)

2. Обучение детектора и классификатора (делает Сенкевич Евгений)

3. Обучение модели под каждый тип графиков (делает Сенкевич Евгений)

4. Создание веб-сайта, как интерфейса для работы с моделью (делает Мария Кокоева)

Список литературы

[1] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick и Piotr Dollár. “Microsoft COCO: Common Objects in Context”. B: *arXiv:1405.0312* (2014).

[2] Joseph Redmon, Santosh Divvala, Ross Girshick и Ali Farhadi. “You Only Look Once: Unified, Real-Time Object Detection”. B: *arXiv:1506.02640* (2015).

[3] Dillon Reis, Jordan Kupec, Jacqueline Hong и Ahmad Daoudi. “Real-Time Flying Object Detection with YOLOv8”. B: *arXiv:2305.09972* (2023).

[4] Shaoqing Ren, Kaiming He, Ross Girshick и Jian Sun. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”. B: *arXiv:1506.01497* (2015).