Санкт-Петербургский государственный университет

Кафедра информационно-аналитических систем

Группа 23.Б08-мм

Анализ и модификация решений детекции животных с использованием глубокого обучения

Речкин Вадим Сергеевич

Отчёт по учебной практике
в форме «Эксперимент»

Научный руководитель:
ст. преподаватель кафедры ИАС, к.ф.-м.н., Азимов Р. Ш.

Санкт-Петербург
2025

Оглавление

Введение

В настоящее время появляется всё больше технологий, который используют в своей работе искусственный интеллект. Это достаточно широкая сфера, которая имеет большое количество более узких областей. Глубокое обучение — одна из областей искусственного интеллекта, которая в современном мире применяется всеми сферами деятельности, так как решает множество полезных задач.

Одна из основных задач глубокого обучения — задача детекции[1]. В настоящее время всё больше проектов используют модели, решающие задачу детекции: системы автопилота, камеры наблюдения, сканирование области и многие другие. Крайне важно решить задачу детекции как можно точнее, так как неверные предсказания могут привести к серьезным проблемам.

Один из популярных вариантов — решение задачи детекции на наборе данных с животными. Эта задача наиболее часто является учебной, но также применяется и на практике. Приемы, которые позволяют увеличить точность на этом наборе данных, зачастую могут помочь увеличить точность и на других, ведь часто наборы данных имеют довольно схожие проблемы.

В данной работе будут рассмотрены различные способы улучшения точности уже имеющихся решений задачи детекции на наборе данных с животными.

[1]Описание задачи детекции: https://en.wikipedia.org/wiki/Object_detection (дата обращения: 8.12.2024)

1. Постановка задачи

Целью работы являются анализ и модификация имеющихся решений для повышения точности предсказаний модели на наборе данных с животными. Для её выполнения были поставлены следующие задачи:

1. Провести обзор имеющихся решений и технологий, которые использовались для решения задачи детекции.
2. Провести обзор техник улучшения точности предсказаний модели, выбрать подходящие для существующих решений.
3. Применить выбранные техники улучшения точности предсказаний модели.
4. Сравнить результаты точности моделей после модификаций.

2. Обзор

В этом разделе будут описаны задача детекции, существующие решения, технологии, которые они используют, и различные методы, усовершенствующие решения.

2.1. Задача детекции

Задача детекции — задача машинного обучения, цель которого найти все объекты на изображении, распознать их и найти рамки, в которой эти объекты расположены. Задача детекции отличается от задачи классификации, потому что нужно не просто определить, есть ли объект на изображении, но и понять, где он находится. Это значительно усложняет подход к решению задачи. При обучении на вход модели подаются изображение и разметка с классами объектов. Дальше в обзоре будут приведены способы решения этой задачи.

2.2. Обзор набора данных

Данный набор данных[2] состоит из 80 классов животных. Он разделён на тренированый и тестовый наборы данных: в тренировочном 22566 изображений, в тестовом — 6505. В главной директории набора данных есть две папки: ”train” и ”test”. В них находятся папки каждого класса, а внутри каждой из них находятся изображения и папка с текстовыми документами, в которых находятся размеченные рамки для каждой фотографии. Имя изображения и текстового файла совпадают, отличаются лишь расширения. Каждый текстовый файл содержит строки, в которых сказано, какому классу принадлежит животное на фотографии, минимальные и максимальные координаты рамки x и y, в которой находится данный объект. Поскольку животных на фотографии может быть несколько, может быть несколько строчек, каждая из которых отвечает за одно животное. Также можно заметить, что этот набор данных не

[2]Ссылка на набор данных, который используется в работе: https://www.kaggle.com/datasets/antoreepjana/animals-detection-images-dataset/data (дата обращения: 25.11.2024)

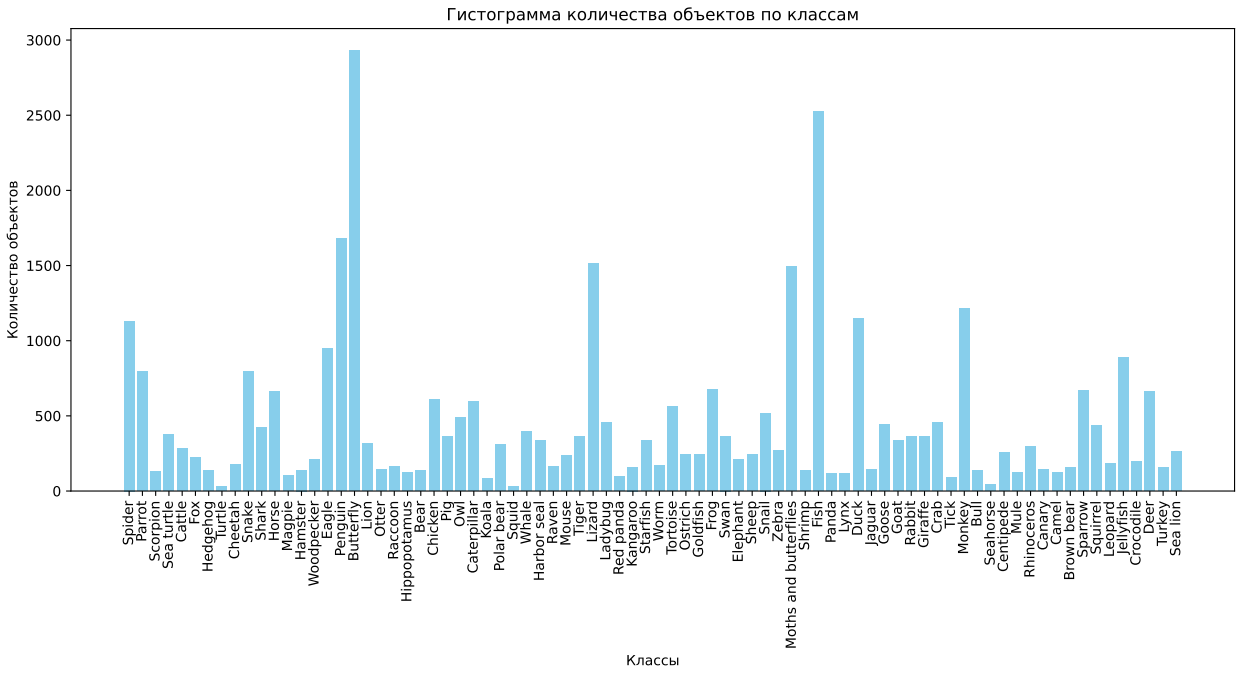

Рис. 1: Распределение количества животных по классам

сбалансирован, как видно на рисунке 1. Это создаёт сложности для увеличение точности модели, далее в этой работе будут рассказаны способы решения этой проблемы.

2.3. Обзор существующих решений

На данном наборе данных есть много работ, но в большей части в работах решают задачу классификации. В работах, где решают задачу детекции, применяются различные версии модели YOLO. [16] Это можно объяснить тем, что эти модели — одни из популярнейших во всём мире и над ними ведуться работы, поэтому они становятся лучше с течением времени. Также преимуществом этих моделей является лёгкий пользовательский интерфейс, а большой спектр настроек позволяет использовать модели YOLO на любом наборе данных.

В таблице 1 будут представлены результаты имеющихся решений:

Решение	Модель	Эпохи	Precision	Recall	mAP50	mAP50-95
1[3]	YOLOV8	100	0.64	0.56	0.62	0.51
2[4]	YOLOV5	30	0.56	0.60	0.56	0.42
3[5]	YOLOV5	30	0.59	0.71	0.65	0.47

Таблица 1: Результаты имеющихся решений задачи детекции

2.4. Обзор моделей YOLO

Модели YOLO — одни из самых популярных решений для задачи детекции и многих других. В последних версиях модели были добавлены различные инструменты, которые улучшают модель, повышая её точность. Для этих моделей нужна специальная структура набора данных, поэтому дальше в работе будет разбор этой структуры. Также в работе будет представленные основные принципы работы моделей YOLO

2.4.1. Структура набора данных для моделей YOLO

Чтобы модель yolo могла обучаться на пользовательском наборе данных, он должен иметь определённую структуру[6]. В главной директории должно быть две папки: ”train” и ”test”. Далее в каждой из этих папок должны быть две папки и один файл: ”images”, ”labels” и ”config.yaml”. В этих папках соответственно содержаться изображения и текстовые файлы с разметкой. Название фотографии и текстового файла должны совпадать с точностью до расширения. В текстовом файле должны быть данные в следующем порядке: индекс класса - индивидуальный номер у каждого класса, нормализованные координаты центра рамки x и y и ширина с высотой. В файле ”config.yaml” должны быть перечислены имена классов, их количество, путь к подготовленному набору данных, пути к тренировочным и тестовым выборкам. Также классы должны быть перечислены в порядке индексов классов. В этой работе будет приведён алгоритм, который подготавливает пользовательский набор данных к набору данных для модели YOLO.

[6]Описание структуры набора данных для моделей YOLO: https://docs.ultralytics.com/ (дата обращения: 25.11.2024)

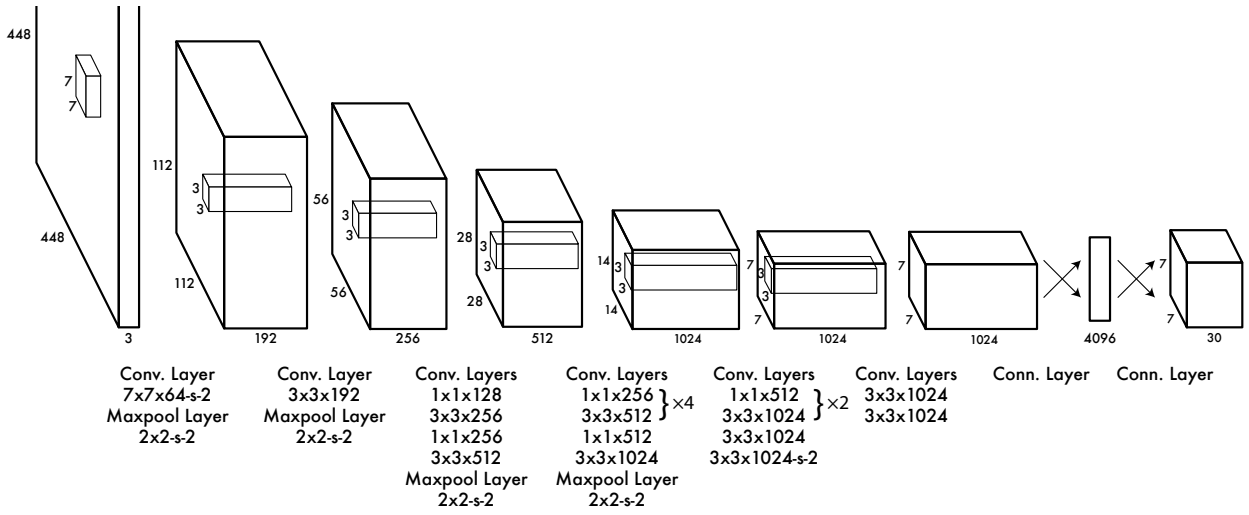

Рис. 2: Архитектура модели YOLO

2.4.2. Принцип работы моделей YOLO

Чтобы понимать, как работают современные модели YOLO, нужно разобраться с первой моделью YOLO. Создатели модели YOLO [6] решили использовать предобученную модель для задачи классификации и заменить последнии слои так, чтобы они выдавали не вероятность наличия определённого класса, а тензор, содержащий информацию о сегментах (в данном контексте — квадратных частях изображения) и классах. Этот подход превосходил подход в алгоритме Faster R-CNN [14] по скорости обучения, но немного проигрывал по точности модели. Разберём этот подход подробнее.

В начале строиться архитектура модели, основанная на модели GoogLeNet [12]. После обучения этой модели для задачи классификации, замораживаются первые 20 слоёв модели, а после добавляются четыре необученных свёрточных слоя [7] и один полносвязный [7]. На выходе получается тензор, который содержит информацию о сегментах и о соответствующих классах.

Функция потерь также была изменена, теперь её вид это:

$$
\begin{aligned}
\text{Loss} = \lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
+ \lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 \\
+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 \\
+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2 \quad (1)
\end{aligned}
$$

где:

- $\lambda_{\textbf{coord}}$ — коэффициент, отвечающий за вес ошибки, связанной с координатами,
- λ_{noobj} — коэффициент, отвечающий за вес ошибки, связанной с отсутствием объекта,
- S — количество сегментов вдоль одной стороны изображения, всего сегментов S^2,
- $\mathbb{1}_{i}^{\text{obj}}$ — индикатор наличия объекта в сегменте i,
- $\mathbb{1}_{ij}^{\text{obj}}$ — индикатор наличия ограничивающей рамки j в сегменте i,
- $\mathbb{1}_{ij}^{\text{noobj}}$ — индикатор отсутствия ограничивающей рамки j в сегменте i,
- x_i y_i — координата центра эталонной ограничивающей рамки в сегменте i,
- $\hat{x}_i$ и $\hat{y}_i$ — координаты центра предсказанной ограничивающей рамки в сегменте i,
- C_i — эталонный класс объекта в сегменте i,

- $\hat{C}_i$ — предсказанный класс объекта в сегменте i,
- $p_i(c)$ — эталонная вероятность класса c в сегменте i,
- $\hat{p}_i(c)$ — предсказанная вероятность класса c в сегменте i.

Каждая новая версия моделей yolo становилась точнее и обучалась ещё быстрее. На данный момент новейшая версия модели YOLO - это YOLO11[7] [11]. Её можно настроить под любую конкретную задачу из-за большого количества различных гиперпараметров, поэтому было решено использовать именно эту модель в своём решении.

2.5. Обзор оптимизаторов

Настройка гиперпараметров — один из самых важных процессов перед запуском модели. Одним из таких параметров является оптимизатор. Оптимизатор — метод, позволяющий усовершенствовать обучение модели, изменяя её параметры (веса модели и скорость обучения). Если грамотно выбрать оптимизатор, можно значительно ускорить обучение модели, что в условиях ограниченной мощности персонального компьютера может значительно увеличить точность модели.

2.5.1. Пакетный градиентный спуск

Пакетный градиентный спуск — метод оптимизации функции потерь, который для вычисления градиента функции использует полный набор данных. Этот метод имеет низкую производительность на больших наборах данных и требует выделение памяти под хранения всех градиентов. У этого метода есть усовершенствованные алгоритмы, которые позволяют ускорить процесс обучения и сократить использование ресурсов.

[7]Основная информация о YOLO11: https://docs.ultralytics.com/models/yolo11/ (дата обращения: 25.11.2024)

2.5.2. Стохастический градиентный спуск

Стохастический градиентный спуск [10] (SGD) — метод оптимизации функции потерь, который отличается от предыдущего тем, что на каждом выбирает лишь один объект из обучающей выборки, считает градиент и изменяет веса модели. Такой подход позволяет значительно ускорить работу алгоритма, но он менее стабильный, что может увеличить время сходимости алгоритма.

2.5.3. Мини-пакетный градиентный спуск

Мини-пакетный градиентный спуск [15] обеспечивает баланс между стохастическим и пакетным градиентными спусками. Этот метод за раз просматривает подмножество обучающей выборки и после пересчитывает веса модели. Этот подход позволяет добиться более высокой скорости сходимости, чем у SGD, и уменьшить вычислительную нагрузку, чем в случае пакетного градиентного спуска. Но функция потерь в данном случае будет менятся меньшими рывками, чем в SGD, что может затруднить проход локальных минимумов. В настоящее время могут подразумевать под SGD мини-пакетный градиентный спуск, но в данной работе это два различных алгоритма. Далее все методы оптимизации будут на основе мини-пакетного градиентного спуска.

2.5.4. Оптимизатор Momentum

Оптимизатор Momentum [9] — оптимизационный алгоритм, который учитывает скорость предыдущего шага, что помогает преодолеть локальные минимумы и прийти к глобальному.

$$v_t = \beta * v_{t-1} - \alpha * dw_{t-1}$$

$$w_t = w_{t-1} + v_t$$

где:

- v_t — вектор первого момента,

- β — коэффициент ослабления импульса,
- α — скорость обучения,
- dw_t - значения градиента функции потерь,
- w_t - веса модели.

Такой подход позволяет обойти "седловые точки" (точки, в которых по одним направлениям значение функции максимальна, по другим - минимальна), также это уменьшает разброс мини пакетного градиентного спуска. Но возникает проблема, если всё же алгоритм попал в локальный минимум, то преодолеть его будет сложно, что увеличит время сходимости.

2.5.5. Оптимизатор Nesterov Momentum

Оптимизатор Nesterov Momentum [5] — усовершенствование метода Momentum. Этот алгоритм позволяет решить проблему оптимизатора Momentum, описанную в 2.5.4. Градиент вычисляется не в локальной позиции, а немного дальше в направлении момента.

$$v_t = \beta * v_{t-1} - \alpha * (dw_{t-1} + \beta * v_{t-1})$$

$$w_t = w_{t-1} + v_t$$

где:

- v_t — вектор первого момента,
- β — коэффициент ослабления импульса,
- α — скорость обучения,
- dw_t - значения градиента функции потерь,
- w_t - веса модели.

Такой подход позволяет обеспечить более быструю сходимость, но создаёт чувствительность к гиперпараметрам.

2.5.6. Оптимизатор AdaGrad

Метод AdaGrad [2] — один из первых алгоритмов оптимизаций градиентного спуска с адаптивным шагом.

$$s_t = s_{\text{t-1}} + dw^2_{\text{t-1}}$$

$$w_t = w_{t-1} - \frac{\alpha}{\sqrt{s_t + \epsilon}} dw_{t-1}$$

где:

- s_t — коэффициент, вкладывающий значение dw^2_{t-1},
- w_t — веса модели,
- α — скорость обучения,
- ϵ — небольшое смещение, чтобы избежать деления на ноль.

Такой подход позволяет двигаться быстро вначале, постепенно замедляясь, когда градиент становиться меньше. Но возникает проблема при обучении нейронных сетей, алгоритм слишком быстро замедляется и не достигает глобального минимума.

2.5.7. Оптимизатор RMSProp

Оптимизатор RMSProp [13] — модификация AdaGrad, которая лучше работает в нейронных сетях.

$$s_t = \beta s_{t-1} + (1 - \beta) dw^2_{t-1}$$

$$w_t = w_{t-1} - \frac{\alpha}{\sqrt{s_t} + \epsilon} dw_{t-1}$$

где:

- s_t — коэффициент, вкладывающий значение dw^2_{t-1},
- β — коэффициент,
- w_t — веса модели,

- α — скорость обучения,
- ϵ — небольшое смещение, чтобы избежать деления на ноль.

RMSProp - успешная оптимизация, но есть недочёты. Алгоритм не очень хорошо работает в многомерных пространствах.

2.5.8. Оптимизатор Adam

Оптимизатор Adam [3] совмещает принципы оптимизаторов Momentum и RMSProp, описанных в 2.5.4 и 2.5.7 соответственно. Этот метод один из самых успешных оптимизаторов на данный момент. Из-за смещения Adam в среднем сходиться быстрее, чем предыдущие оптимизаторы, и более устойчив к выбору гиперпараметров. Это делает Adam стабильным и мощным решением. У Adam есть много модернизаций, которые могут увеличить скорость сходимости для конкретных задач. Напрашивается вывод, что для решения задач нужно использовать либо Adam, либо его модернизацию.

$$v_t = \beta_1 v_{t-1} + (1 - \beta_1) dw_{t-1}$$

$$s_t = \beta_2 s_{t-1} + (1 - \beta_2) dw_{t-1}^2$$

$$w_t = w_{t-1} - \alpha \frac{v_t}{\sqrt{s_t} + \epsilon}$$

Поправка на смещение:

$$v_t \leftarrow \frac{v_t}{1 - \beta_1^t}$$

$$s_t \leftarrow \frac{s_t}{1 - \beta_2^t}$$

Но у оптимизатора Adam есть недостатки: алгоритм чувствителен к шуму, это может привести к проблемам со сходимостью, алгоритм требует использование дополнительной памяти, потому что хранит информацию о градиентах и накопленном импульсе. Также оптимизатору Adam для продолжения обучения после остановки программы, нужно восстановить не только веса модели, но и всю информацию о накопленных параметрах.

2.5.9. Различные версии Adam

NAdam [1] — улучшенная версия Adam, использующая принцип Nesterov Momentum, описанный в 2.5.5. Это позволяет ему сходиться быстрее, чем Adam, но делает его более чувствительным к гиперпараметрам.

$$\hat{v}_t = \frac{{\beta_1}^{t+1} v_t}{1 - \prod_{t=1}^{t+1} \beta^i} + \frac{(1 - \beta_1^t) dw_{t-1}}{1 - \prod_{t=1}^{t} \beta^i}$$

AdamW [8] — добавляет ridge регуляризацию [4], что позволяет избежать переобучения. Может быть не таким эффективным по сравнению с Adam.

$$w_t = w_{t-1} - \alpha \frac{v_t}{\sqrt{s_t} + \epsilon} + \lambda * w_{t-1}$$

где,

- λ — коэффициент влияния регуляризации.

2.6. Проблема несбалансированности набора данных

Несбалансированность набора данных может сильно влиять на предсказания модели. Способов решения достаточно не мало, но в работе будет применён баланс классов. В функцию потерь передаётся информация о количестве объектов в каждом из классов и у функции потерь каждого класса появлявется вес. В зависимости от количества объектов в классе, ошибка будет варьироваться, не смещаясь в сторону классов, с большим количеством элементов.

2.7. Обзор метрик

Чтобы сравнивать результаты решений, нужно понимать, что показывают различные метрики. После проверки на тестовом наборе данных yolo для каждого класса выдаёт четыре числа: precision, recall, mAP50,

mAP50-95. Посмотрим на эти метрики точности.

$$precision = \frac{TP}{TP + FP}$$

где:

- $precision$ — точность обнаружения объектов. Отвечает за верное количество опознанных объектов к количеству всех опознанных объектов,
- TP — количество верно определённых объектов,
- FP — количество ложно определённых объектов.

$$recall = \frac{TP}{TP + FN}$$

где:

- $recall$ — величина, показывающая способность находить все объекты на изображении,
- FN - количество верных пропущенных объектов.

$$\text{IoU} = \frac{\text{Area of Intersection}}{\text{Area of Union}}$$

где:

- IoU — Мера, показывающая, насколько предсказанная ограничевающая рамка отличается от заданной,
- Area of Intersection — площадь пересечения предсказанной и заданной ограничевающих рамок,
- Area of Union — площадь объединения предсказанной и заданной ограничевающих рамок.

mAP50 - средняя точность, рассчитанная при пороге IoU, равному 0,5.

mAP50-95 - среднее значение средней точности, где порог менялся от 0,5 до 0,95.

Для сравнения результатов между решениями будут использованы *precision*, *recall*, mAP50, mAP50-95, причём mAP50, mAP50-95 показывают более общую работу модели, поэтому это самые важные метрики в данной работе.

3. Описание решения

В данном разделе будут представлены различные модификации уже имеющихся решений задачи детекции. Реализация будет использовать последнюю версию модели YOLO, но для её использования нужно будет изменить структуру набора данных, решить проблему несбалансированности набора данных, а после грамотно подобрать гиперпараметры модели.

3.1. Используемые инструменты

Данная работа писалась в Jupyter Notebook, потому что эта среда разработки содержит удобный интерфейс для работы: можно выполнять отдельные фрагменты кода, удобно выводить и анализировать результаты работы. Также Jupyter Notebook поддерживает язык Python, который имеет в наличие большое количество библиотек по работе с машинным обучением, что является нужный фактом для задачи Детекции. Выбор модели YOLO уже был описан в 2.4. Также в работе было использовано готовое решение для балансировки набора данных YOLO [8].

3.2. Работа с набором данных

Работу с набором данных можно поделить на два этапа: приведение к формату набора данных YOLO и решение проблемы несбалансированности набора данных.

3.2.1. Преобразование набора данных

Структура набора данных для модели YOLO была описана выше в 2.4.1, поэтому нужно написать программу, конвертирующую файлы из исходного набора данных в нужный формат. Для конвертирования был создан класс PreparationDatasetToYolo.

[8] Баланс классов для моделей YOLO https://y-t-g.github.io/tutorials/yolo-class-balancing/ (Дата обращения: 12.12.2024)

Объект данного класса преобразует исходный набор данных, а именно переименовывает изображения, в разметке рассчитывает новые нормализованные координаты, создавая подходящую структуру набора данных для модели YOLO. Также с помощью этого объекта можно получить основную информацию о наборе данных, часть из неё записывается в файл "config.yaml".

3.2.2. Балансировка набора данных

Набор данных, используемый в работе, не сбалансирован. Существует несколько способов решения этой проблемы, один из них - передать веса классов в модель. С помощью этого приёма модель сможет равномерно обучаться, не отклоняясь в сторону отдельных классов. В данной работе использовано готовое решение. Готовое решение имеет две реализации: для CPU и одного GPU и для нескольких GPU. В данной работе реализован автоматический выбор реализации из технических характеристи устройства.

3.3. Работа с гиперпараметрами модели

Подбор гиперпараметров — важнейшая часть перед обучение модели, ведь от этого зависит скорость сходимости и значения метрик.

В данной работе будут исследованы не все гиперпараметры модели yolo, так как их большое количество. Многие из них заданы по умолчанию, что обычно не делает модель хуже. Их полная настройка занимает большее количество времени, что в условиях ограниченных ресурсах не актуально.

3.3.1. Выбор размера пакета и разрешения изображений

Размер пакета и разрешение изображений — главные потребители ресурсов видеокарт. Для задачи детекции важно разрешение изображений, чем оно больше, тем модель может точнее найти рамку. Также лучшим вариантом будет как можно больший размер пакета, в таком

случае алгоритм сойдётся быстрее и модель за одно и то же количество итераций достигнет лучших результатов.

3.3.2. Выбор оптимизатора

В обзоре уже приводилось объяснение по поводу выбора оптимизатора в разделе 2.5, поэтому на данный момент нужно выбрать версию оптимизатора Adam. Были рассмотрены несколько вариантов: стандартный Adam, AwamW и NAdam. Их стоит попробовать использовать при обучении на некотором количестве эпох.

3.3.3. Выбор скорости обучения

На данном наборе данных есть решения, у которых заданы гиперпараметры. Было принято решение, использовать значения скоростей обучения из существующего решения.

4. Эксперимент

Цель данной работы - модифицировать решения для повышения точности модели, поэтому эксперимент можно считать успешным, если метрики обученной модели превзойдут метрики в таблице 1.

4.1. Условия эксперимента

Эксперимент проходил на платформе kaggle с акселератором, технические характеристики:

- CPU: Intel Xeon 2.00GHz
- CPU count: 4
- GPU: Tesla T4, 15095MiB
- GPU count: 2
- RAM: 31.35 GB

Перебрались следующие гиперпараметры модели:

- Оптимизаторы: Adam, NAdam, AdamW
- Размеры пакета: 64, 96
- Разрешения изображения: 640
- Начальные скорости обучения: 0.01, 0.001
- Отношения начальных скоростям обучения к конечным: 2
- Количества эпох: 50
- Значения dropout: 0

Также были опыты со взвешенным и невзвешенным набором данных.

4.2. Метрики

Метрики, используемые для сравнения решений:

- `precision`
- `recall`
- `mAP50`
- `mAP50-95`

4.3. Подбор гиперпараметров

Первым шагом были протестированы оптимизаторы Adam, NAdam и AdamW в одинаковых условиях. Далее будут перечислены остальные гиперпараметры модели.

- Количество эпох: 50
- Размер пакета: 64
- Размер изображения: 640
- Начальная скорость обучения: 0.001
- Конечная скорость обучения: 0.0005

Были получены следующие результаты:

Оптимизатор	**Precision**	**Recall**	**mAP50**	**mAP50-95**
AdamW	0.606	0.631	0.633	0.547
NAdam	0.601	0.636	0.643	0.553
Adam	0.592	0.635	0.630	0.545

Таблица 2: Сравнение результатов при разных оптимизаторов

Из таблицы 2 видно, что оптимизатор NAdam достиг практически во всех метриках лучшего результата, что и ожидалась. Было решено дальше в работе использовать оптимизатор NAdam.

Далее были протестированы различные разрешения изображений также в одинаковых условиях, далее они будут перечислены.

- Оптимизатор: NAdam
- Количество эпох: 10
- Размер пакета: 64
- Начальная скорость обучения: 0.001
- Конечная скорость обучения: 0.0005

В следующей таблице приведены результаты

Разрешение изображений	**Precision**	**Recall**	**mAP50**	**mAP50-95**
640	0.51	0.50	0.50	0.42
960	0.50	0.50	0.49	0.40

Таблица 3: Сравнение результатов метрик при различных разрешениях изображений

Увеличения разрешения изображение не дало прироста в точности модели. Одна из возможных причин — разрешение изображений набора данных. У достаточно большого количества изображений разрешения меньше, чем 960 на 960 пикселей, что не даёт преимущественно при увеличении изображения до нужного качества. Вследствие этого, было решено использовать разрешение изображений 640 на 640.

Размер изображений не был увеличен, что позволяет увеличить размер одного пакета. Соответственно были протестированы различные размеры пакетов в одинаковых условиях. Гиперпараметры для эксперимента:

- Оптимизатор NAdam
- Количество эпох: 10
- Разрешение изображений: 640
- Начальная скорость обучения: 0.001
- Конечная скорость обучения: 0.0005

Размер пакета	**Precision**	**Recall**	**mAP50**	**mAP50-95**
64	0.51	0.50	0.50	0.42
96	0.53	0.55	0.54	0.45

Таблица 4: Сравнение результатов метрик при различных размерах пакета

Видно, что увеличение размера пакета повлекло за собой увеличения значения метрик модели. Было решено использовать размер изображения 96

Скорость обучения была выбрана из существующего решения. При этих значениях скорости обучения модель выдава приемлемые значения метрик.

Был проведён эксперимент со всеми выводами, со следующими гиперпараметрами:

- Оптимизатор NAdam
- Количество эпох: 50
- Разрешение изображений: 640
- Начальная скорость обучения: 0.001
- Конечная скорость обучения: 0.0005

Результаты представлены в таблице 5

Размер пакета	**Precision**	**Recall**	**mAP50**	**mAP50-95**
64	0.60	0.64	0.64	0.55
96	0.62	0.64	0.64	0.55

Таблица 5: Сравнение лучших результатов работы

Из таблицы 5 видно, что после применённых модернизаций значения метрик не возросли. Одна из причин — попадание в локальный минимум функции потерь. Больший размер пакета позволяет меньше ”колебаться” весам модели во время градиентного спуска, из-за чего локальный минимум становиться достаточно сложной преградой для

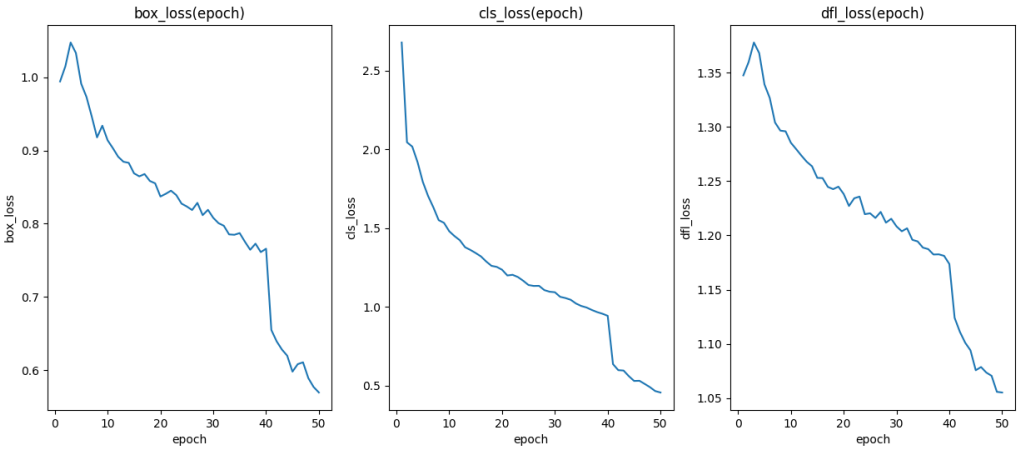

Рис. 3: Графики зависимости функции потерь от количества эпох на тренировочной выборке

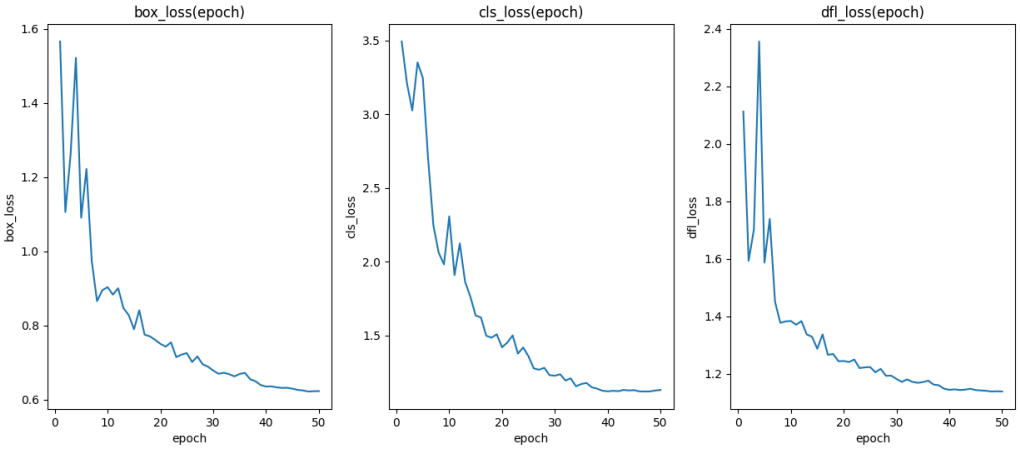

Рис. 4: Графики зависимости функции потерь от количества эпох на тестовой выборке

Решение	**Precision**	**Recall**	**mAP50**	**mAP5095**
YOLOV8	0.64	0.56	0.62	0.51
YOLO11 (64)	0.60	0.64	0.64	0.55
YOLO11 (96)	0.61	0.64	0.64	0.55

Таблица 6: Сравнение результатов

дальнейшего обучения. Это можно увидеть на графиках под номером 3. Также можно подумать, что начался процесс переобучения, но если взглянуть на графики под номером 4, то можно сделать вывод, что не было переобучения у модели.

4.4. Результаты

После проведения всех экспериментов сравним результаты с результатами существующих решений.

В таблице 6 представлены лучшие резуоьтаты данной работы и лучший результат существуюших. Видно, что удалось улучшить точность модели. Решения из этой работы превосходят практически по всем метрикам существующие решения.

4.5. Вывод

Из эксперимента видно, что грамотный подроб гиперпараметров и балансировка набора данных позволяет увеличить точность модели. Не на каждом наборе данных метод, балансирующий набор данных, может увеличить точность, но всё же большинство имеет проблемы с балансом классов. В свою очередь подбор гиперпараметров может требовать вычислительных ресурсов и времени, поэтому стоит предварительно изучить основную информацию о них, чтобы ресурсы на эксперименты были потрачены разумно.

Заключение

В ходе данной работы были проведены анализ и модификация существующих решений задачи детекции на наборе данных. Были использованы достаточно общие методы, позволяющие повысить точность предсказаний модели. В большинстве случаев они смогут помочь повысить точность на другим наборе данных. В процессе работы были выполнены поставленные задачи.

1. Выполнен обзор имеющихся решений. В большинстве существующих решений использовались разные версии модели YOLO.

2. Проведён обзор техник улучшения точности предсказаний модели. Подбор гиперпараметров и балансировка набора данных — основные методы повышения точности.

3. Применены техники улучшения точности предсказаний модели.

4. Сопоставлены результаты данной работы с результатами существующих решений. Удалось превзойти практически по всем метрикам.

Исходный код данной работы: https://github.com/VadosikRRR/Modification_OfSolutionsToDetection

Список литературы

[1] Ange Tato Roger Nkambou. IMPROVING ADAM OPTIMIZER. — URL: https://openreview.net/forum?id=HJfpZq1DM (дата обращения: 18 декабря 2024 г.).

[2] Chang Jen-Kuang Fang; Cher-Min Fong; Peng Yang; Chao-Kai Hung; Wen-Long Lu; Chien-Wei. AdaGrad Gradient Descent Method for AI Image Management. — URL: https://ieeexplore.ieee.org/abstract/document/9258085 (дата обращения: 8 декабря 2024 г.).

[3] Diederik P. Kingma Jimmy Ba. Adam: A Method for Stochastic Optimization. — URL: https://arxiv.org/abs/1412.6980 (дата обращения: 25 ноября 2024 г.).

[4] Dmitry Kobak Jonathan Lomond Benoit Sanchez. The Optimal Ridge Penalty for Real-world High-dimensional Data Can Be Zero or Negative due to the Implicit Ridge Regularization. — URL: https://www.jmlr.org/papers/v21/19-844.html (дата обращения: 18 декабря 2024 г.).

[5] Guan Lei. AdaPlus: Integrating Nesterov Momentum and Precise Stepsize Adjustment on Adamw Basis. — URL: https://ieeexplore.ieee.org/abstract/document/10447337 (дата обращения: 18 декабря 2024 г.).

[6] Joseph Redmon Santosh Divvala Ross Girshick Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. — URL: https://arxiv.org/abs/1506.02640 (дата обращения: 25 ноября 2024 г.).

[7] Keiron O'Shea Ryan Nash. An Introduction to Convolutional Neural Networkss. — URL: https://arxiv.org/abs/1511.08458 (дата обращения: 8 декабря 2024 г.).

[8] Lupera Ricardo Llugsi; Samira El Yacoubi; Allyx Fontaine; Pablo. Comparison between Adam, AdaMax and Adam W optimizers to implement a Weather Forecast based on Neural Networks for the Andean city

of Quito. — URL: https://openreview.net/forum?id=HJfpZq1DM (дата обращения: 18 декабря 2024 г.).

[9] Mohammad Dehghani Haidar Samet. Momentum search algorithm: a new meta-heuristic optimization algorithm inspired by momentum conservation law. — URL: https://link.springer.com/article/10.1007/s42452-020-03511-6 (дата обращения: 25 ноября 2024 г.).

[10] Netrapalli Praneeth. Stochastic Gradient Descent and Its Variants in Machine Learning. — URL: https://en.wikipedia.org/wiki/Gradient_descent (дата обращения: 25 ноября 2024 г.).

[11] Nidhal Jegham Chan Young Koh Marwan Abdelatti Abdeltawab Hendawii. Evaluating the Evolution of YOLO (You Only Look Once) Models: A Comprehensive Benchmark Study of YOLO11 and Its Predecessors. — URL: https://arxiv.org/abs/2411.00201 (дата обращения: 8 декабря 2024 г.).

[12] R. Anand T. Shanthi M. S. Nithish S. Lakshman. Face Recognition and Classification Using GoogleNET Architecture. — URL: https://link.springer.com/chapter/10.1007/978-981-15-0035-0_20 (дата обращения: 25 ноября 2024 г.).

[13] Reham Elshamy Osama Abu-Elnasr Mohamed Elhoseny Samir Elmougy. Improving the efficiency of RMSProp optimizer by utilizing Nestrove in deep learning. — URL: https://www.nature.com/articles/s41598-023-35663-x (дата обращения: 8 декабря 2024 г.).

[14] Shaoqing Ren Kaiming He Ross Girshick Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. — URL: https://ieeexplore.ieee.org/abstract/document/7485869 (дата обращения: 25 ноября 2024 г.).

[15] Stochastic gradient descent (mini-branch gradient descent). — URL: https://en.wikipedia.org/wiki/Stochastic_gradient_descent (дата обращения: 25 ноября 2024 г.).

[16] Ultralytics YOLO Docs. — URL: https://docs.ultralytics.com/ (дата обращения: 25 ноября 2024 г.).