ФЕДЕРАЛЬНОЕ АВТОНОМНОЕ УЧРЕЖДЕНИЕ
«ГОСУДАРСТВЕННЫЙ НАУЧНО-ИССЛЕДОВАТЕЛЬСКИЙ ИНСТИТУТ
АВИАЦИОННЫХ СИСТЕМ»

Унифицированная программная платформа для разработки конечно ориентированных программных комплексов автоматического распознавания объектов на основе нейросетевых подходов «Платформа-ГНС»

Библиотека PlatformAPI

Описание

Москва 2023

СОДЕРЖАНИЕ

1. Назначение библиотеки PlatformAPI

PlatformAPI – это библиотека, которая реализует обмен информацией между серверным программным обеспечением (ПО) и типовыми решениями унифицированной программной платформы «Платформа-ГНС» (далее – УПП «Платформа-ГНС»). Типовое решение (ТР) – это сценарий обучения, состоящий из формирования выборок, обучения и тестирования нейросетей, выгрузки полученных весовых коэффициентов) с подготовленными архитектурами и параметрами обучения или тестирования, вынесенными для настройки пользователем.

Разработчики могут использовать библиотеку PlatformAPI и отправлять запросы на сервер в соответствии с предложенным описанием. Ответ на запрос будет содержать данные в формате JSON.

Библиотека PlatformAPI ускоряет и облегчает процесс интеграции внешнего кода в Платформу и предоставляет пользователю следующие возможности:

– передача входных данных, установленных пользователем в автоматизированном рабочем месте разработчика (АРМ-Р), от серверного ПО задаче типового решения;

– формирование и передача типовому решению выборки, содержащей набор исходных файлов и соответствующих им аннотированных данных (при наличии);

– сохранение на серверном ПО прогресса выполнения задачи типового решения;

– сохранение на серверном ПО текстовых сообщений, сформированных в процессе выполнения задачи типового решения;

– сохранение на серверном ПО характеристик процесса обучения глубокой нейронной сети (ГНС) в задаче типового решения в виде графиков;

– сохранение результатов тестирования ГНС в виде метрик качества;

– сохранение результатов тестирования задачи типового решения по обработки данных ГНС из тестовой выборки.

На рисунке 1 представлена схема использования библиотеки PlatformAPI при встраивании решений в УПП «Платформа-ГНС».

Рисунок 2 – Схема использования библиотеки PlatformAPI

Базовая интеграция внешнего программного кода в пользовательское решение УПП «Платформа-ГНС» выполняется следующим образом:

– встраивание функций библиотеки PlatformAPI в исходный код решения;

– формирование файла параметров в формате JSON, описывающего необходимый пользовательский интерфейс решения;

– сборка контейнера для запуска вычислительных заданий в серверной части УПП «Платформа-ГНС», включающего данный код и все необходимые для его запуска библиотеки.

2. Вызов и основные настройки библиотеки PlatformAPI

Библиотека PlatformAPI реализована на языке программирования Python версии 3.7. Для её работы требуются следующие пакеты (python библиотеки):

– redis;

– pymongo;

– requests;

– aiofiles;

– nose2.

Для использования библиотеки PlatformAPI в коде необходимо выполнить ее импорт (см. листинг 1).

Листинг 1 – Команда импорта библиотеки PlatformAPI

```
import PlatformAPI
```

В момент импорта выполняется загрузка параметров, которые записаны кластер-менеджером УПП «Платформа-ГНС» в виде JSON файла. По умолчанию файл называется «settings.json» и располагается в каталоге /app/task_info/.

Если указного файла не существует, то будут использованы следующие настройки по умолчанию, приведенные в листинге 2, где:

– uuid – уникальный идентификатор задания, с которым происходит работа;

– STORAGE_HOST – IP адрес объектного хранилища;

– STORAGE_PORT – порт объектного хранилища;

– REDIS_HOST – IP адрес Redis в составе;

– REDIS_PORT – порт Redis в составе;

– DJANGO_HOST – IP адрес программного модуля реализации логики работы, хранения данных и API;

– DJANGO_PORT – порт программного модуля реализации логики работы, хранения данных и API;

– MONGO_HOST – IP адрес MongoDB в составе;

– MONGO_PORT – порт MongoDB в составе;

– USE_MONGO – флаг применения для записи логов или метрик MongoDB (значение «1») или Redis (значение «0»).

Листинг 2 – Настройки параметров по умолчанию

```
    {
      'uuid': 'default',
      'STORAGE_HOST': '127.0.0.1',
      'STORAGE_PORT': '24700',
      'REDIS_HOST': '127.0.0.1',
      'REDIS_PORT': '6379',
      'DJANGO_HOST': '127.0.0.1',
      'DJANGO_PORT': '8000',
      'MONGO_HOST': '127.0.0.1',
      'MONGO_PORT': '27017',
      'USE_MONGO': '0'
}
```

3. Перечень доступных функций

3.1. Функции получения входных параметров

Функция получения входных параметров, заданных пользователем в клиенте УПП «Платформа-ГНС» для стадии обучения (train), приведена в листинге 3. Возвращает словарь (dict).

Листинг 3 – Функция получения входных параметров на стадии обучения

```
PlatformAPI.get_train_params() → dict
```

Функция получения входных параметров, введенных пользователем в клиенте УПП «Платформа-ГНС» для стадии тестирования (test), приведен в листинге 4. Возвращает словарь (dict):

Листинг 4 – Функция получения входных параметров на стадии тестирования

```
PlatformAPI.get_test_params() → dict
```

3.2. Функции работы с выборками

Функция получения списка всех доступных выборок представлена в листинге 5. Возвращает список объектов класса Dataset.

Листинг 5 – Функция получения списка всех доступных выборок

```
PlatformAPI.get_available_datasets() → typing.List[Dataset]
```

Функция формирования для заданной выборки списка доступных данных: объектов (например, изображений), связанных объектов, разметки в формате УПП «Платформа-ГНС», приведена в листинге 6. В качестве аргумента передаётся объект класса Dataset. Результаты работы функции сохраняются в атрибутах переданного объекта.

Листинг 6 – Функция формирования для заданной выборки доступных данных

```
PlatformAPI.generate_dataset(dataset: Dataset) → None
```

В классе Dataset реализованы следующие основные методы:

– get_all_images_path – получение списка путей в файловой системе до основных объектов (элементов выборки);

– get_all_additional_images – получение списка путей до дополнительных изображений или файлов (элементов баз данных), расположенных в хранилище данных;

– get_all_markups – получение списка путей до файлов JSON, содержащих разметку данных (элемента базы данных).

Пример получения списка основных файлов и разметки в формате УПП «Платформа-ГНС» для обучающей выборки (train) представлен в листинге 7.

Листинг 7 – Пример получения списка основных файлов и разметки для обучающей выборки

```
      # получение списка имен доступных датасетов
     datasets = PlatformAPI.get_available_datasets()
     index = -1
for i in range(len(datasets)):
    name_data = datasets[i].name
    if name_data == 'train':
           index = i

# получение списка картинок и списка разметок
train_db = datasets[index]
PlatformAPI.generate_dataset(train_db)
images_path = train_db.get_all_images_path()
markups_path = train_db.get_all_markups ()
```

3.3. Функции вывода прогресса выполнения работы

Функция сохранения прогресса выполнения работы программы на стадии обучения (train), отображаемого в клиенте УПП «Платформа-ГНС», представлена в листинге 8. В качестве аргумента передаётся числовое значение прогресса.

Листинг 8 – Функция сохранения прогресса выполнения работы программы на стадии обучения

```
PlatformAPI.save_progress_for_platform(progress: float) → None
```

Функция сохранения прогресса выполнения работы программы на стадии тестирования (test), отображаемого в клиенте УПП «Платформа-ГНС», представлена в листинге 9. В качестве аргумента передаётся числовое значение прогресса.

Листинг 9 – Функция сохранения прогресса выполнения работы программы на стадии тестирования

```
PlatformAPI.save_test_progress_for_platform(progress: float) →
None
```

3.4. Функции вывода информационных сообщений

В клиенте УПП «Платформа-ГНС» сообщения разделяются на три типа: информационное, вывод предупреждения и сообщение об ошибке.

Функция получения объекта для записи сообщений, отображаемых в клиенте УПП «Платформа-ГНС» представлена в листинге 10. Возвращает сконфигурированный класс PlatformLogger.

Листинг 10 – Функция получения объекта для записи сообщений

```
PlatformAPI.get_logger() → typing.Type[PlatformLogger]
```

Класс PlatformLogger имеет следующие методы для записи сообщений:

– info(message: str) – для вывода информации со статусом «Информация»;

– warning(message: str) – для вывода информации со статусом «Предупреждение»;

– error(message: str) – для вывода информации со статусом «Ошибка».

Все методы принимают в качестве аргумента строковое значение.

Пример получения объекта для записи сообщений и вывода текстовой информации в клиент УПП «Платформа-ГНС» представлен в листинге 11

Листинг 11 – Пример получения объекта для записи сообщений и вывода текстовой информации

```
# инициализация объекта, принимающего информационные сообщения
logger = PlatformAPI.get_logger()
# Сохранение текстовой информации
lis = ('TRAIN ||Epoch: %d || iter: %d || max_iter_epoch: %d' %
(epoch + 1, i + 1, len(loader_train))
logger.info(lis)
```

3.5. Функции сохранения графиков обучения

Функция получения объекта для записи метрик обучения, отображаемых в клиенте УПП «Платформа-ГНС» приведена в листинге 12. Возвращает класс TrainMetrics.

Листинг 12 – Функция получения объекта для записи метрик обучения

```
PlatformAPI.get_train_metrics_logger() →
typing.Type[TrainMetrics]
```

Метод класса TrainMetrics сохранения точки на графике обучения представлена в листинге 13, где:

– name – имя графика;

– value – значение точки;

– epoch – номер эпохи;

– iter – номер итерации;

– info – дополнительная информация для точке на графике.

Листинг 13 – Метод сохранения точки на графике обучения

```
TrainMetrics.log_point(name: str, value: float, epoch: int,
iter: int, info: str) → None
```

Пример сохранения графика обучения приведен в листинге 14.

Листинг 14 – Пример сохранения графика обучения

```
# инициализация объекта, принимающего данные для отображения
графиков
train_logger = PlatformAPI.get_train_graph_logger()

for epoch in range(int(max_epoch)):
    # Тело цикла обучения
    for i, data in enumerate(loader_train):
        # Итерация
        # Подсчет функции ошибки loss_train
        #Сохранение точки для графика функции потерь
        train_logger.log_point("loss_L1", float(loss_train),
epoch + 1, i + 1, "L1 metric")
```

3.6. Функция сохранения метрик тестирования

Функция получения объекта записи метрик тестирования, отображаемых в клиенте УПП «Платформа-ГНС» представлен в листинге 15. Возвращает класс TestMetrics.

Листинг 15 – Функция получения объекта записи метрик тестирования

```
PlatformAPI.get_test_metrics_logger() →
typing.Type[TestMetrics]
```

Сохранение метрик тестирования осуществляется вызовом метода «append» класса TestMetrics. Принимает на вход файл JSON, содержащий метрики тестирования в формате УПП «Платформа-ГНС».

3.7. Функция сохранения результатов тестирования

Функция получения словаря разметки в формате УПП «Платформа-ГНС» для списка изображений в инициализированной выборке представлена в листинге 16. Возвращает словарь с разметкой в формате УПП «Платформа-ГНС».

Листинг 16 – Функция получения словаря разметки для списка изображений в инициализированной выборке

```
PlatformAPI.get_markup_dict_for_images(images: list,
description: str) → dict
```

Пример сохранения разметки, полученной на тестировании, для задачи устранение шумов на изображениях (Denoise) представлен в листинге 17.

Листинг 17 – Пример сохранения разметки

```
buffered = BytesIO()
im_out.save(buffered, format="PNG")
save_data = PlatformAPI.get_markup_dict_for_images([buffered],
"Denoise")
dataset.save_test_markup(s, [save_data], os.path.join('.',
'task_info', 'test_markup'))
```

4. Форматы данных при работе с сервером

4.1. Форматы и размеры пользовательских изображений

Типовые решения УПП «Платформы-ГНС» (и соответствующие им базы данных) используют изображения строго определенных форматов, с установленными ограничениями на размеры, представленными в таблице 1.

Таблица 1 – Форматы и размеры изображений для баз данных типовых решений

Типовое решение	Формат изображения	Размер изображения, пиксель
Обнаружение объектов по изображениям и видеопоследовательностям	bmp, jpg, jpeg, png	Без ограничений. Исходное изображение автоматически преобразуется до размера 300×300
Обнаружение объектов по многоспектральным данным	bmp, jpg, jpeg, png	Без ограничений. Исходное изображение автоматически преобразуется до размера 300×300
Дешифрирование	bmp, jpg, jpeg, png	Без ограничений. Исходное изображение масштабируется с сохранением соотношения сторон таким образом, чтобы минимальная сторона изображения была больше 800, а максимальная не превышала 1333. Если условия не выполняется, то масштабирование происходит для условия – максимальный размер изображения меньше 1333
Классификация	bmp, jpg, jpeg, png, tiff	Без ограничений. Исходные данные масштабируются под размеры, заданные пользователем: от 32 до 2048 по высоте и ширине
Семантическая сегментация	bmp, jpg, jpeg, png, tiff	Размер изображений должен быть больше, чем установленный пользователем. При обучении исходные изображения обрезаются под размеры, заданные пользователем: – по высоте от 64 до 1024, – по ширине от 64 до 2048. При тестировании исходное изображение обрабатывается целиком

Продолжение таблицы 1

Типовое решение	Формат изображения	Размер изображения, пиксель
Распознавание типов объекта	bmp, jpg, jpeg, png	Без ограничений. Исходные данные масштабируются под размеры, заданные пользователем: от 32 до 2048 по высоте и ширине
Сопровождение объектов на видеопоследовательностях	bmp, jpg, jpeg, png	Без ограничений. Исходное изображение объекта сопровождения автоматически преобразуется до размера 127×127, Изображение на котором необходимо обнаружить исходный объект преобразуется до размера 255×255
Обработка и комплексирование изображений различных диапазонов	bmp, jpg, jpeg, png, tiff	Без ограничений. Исходные данные, в зависимости от выбранного режима обучения, масштабируются под размеры, заданные пользователем: – от 32 до 1024 по ширине, – от 32 до 1024 по высоте; или вырезаются. При тестировании масштабируются или обрабатываются целиком.
Устранение шумов и помех на изображениях (денойзинг)	bmp, jpg, jpeg, png, tiff	Без ограничений. Исходные данные, в зависимости от выбранного режима обучения, масштабируются под размеры, заданные пользователем: – от 32 до 1024 по ширине, – от 32 до 1024 по высоте; или вырезаются. При тестировании масштабируются или обрабатываются целиком.
Устранение смаза и расфокусировки изображений (деблюр)	bmp, jpg, jpeg, png, tiff	Без ограничений. Исходные данные, в зависимости от выбранного режима обучения, масштабируются под размеры, заданные пользователем: – от 32 до 1024 по ширине, – от 32 до 1024 по высоте; или вырезаются. При тестировании масштабируются или обрабатываются целиком.

4.2. Формат разметки изображений

Разметка (аннотация) – это геометрические объекты, расположенные поверх изображения, которыми помечаются те или иные объекты на изображении. Список доступных видов объектов (классов) определен для всей базы данных (БД). Для каждого класса разметчиком задается свой цвет.

Существуют следующие виды разметки:

– классификация – класс на все изображение;

– обнаружение – класс на геометрический объект (точка, линия, кривая, прямоугольник, повернутый прямоугольник, многоугольник);

– сегментация – класс на подмножество пикселей, т.е. разметка представляет собой bitmap.

4.3. Формат входных данных

Заданные пользователем настройки для запуска решения передаются от серверного ПО программному модулю типового решения в виде JSON файла. Пример файла JSON с параметрами типового решения задачи классификации приведен в листинге 18. Подробная информация о выборке предоставляется в виде объектов класса Dataset (см. подраздел 3.2).

Листинг 18 – Пример файла JSON с параметрами типового решения задачи классификации

```
{
  "type": "Classification",
  "beta1": 0.9,
  "beta2": 0.999,
  "gamma": 0.1,
  "image": "classification_train",
  "policy": "StepLR",
  "solver": "Adam",
  "amsgrad": false,
  "input_c": 3,
  "input_h": 32,
  "input_w": 32,
  "loss_fn": "CrossEntropy",
  "num_cpu": 1,
  "version": 1,
  "backbone": "LeNet",
```

```
  "division": 0.9,
  "max_epoch": 2,
  "num_nodes": 1,
  "step_size": 20,
  "batch_size": 1,
  "num_workers": 4,
  "weight_decay": 0,
  "learning_rate": 0.001,
  "pending_limit": 300,
  "target_device": "cpu",
  "save_frequency": 1,
  "test_frequency": 5,
  "checkpoint_name": "",
  "uuid": "0130d82d935e4cb99d20510d09739440",
  "parent_uuid": "56d2c5b81f8a47f8a0758594fb7a41b3",
  "dataset": {
    "classes": [
      "airplane",
      "car",
      "ship",
      "truck"
    ],
    "count_classes": [
      100,
      100,
      100,
      100
    ],
    "count_images": 400,
    "count_items": {
      "a5f41eabbb5241f68af0de8a9ff17426": 400
    },
    "options": {}
  },
  "use_balancing": false,
  "balancing_mode": "EXACT",
  "balancing_value": 100,
  "preprocessing_type": "generic"
}
```


4.4. Форматы аннотированных данных

Общая структура разметки представлена в листинге 19, где:

– version – версия разметки. Его значение проверяется в первую очередь при открытии файла, и дальше чтение содержимого файла проводится в соответствии с его значением;

– annotation – список объектов разметки.

Листинг 19 – Структура разметки

```
{
    "version": 1,
    "annotation": [
        // список объектов аннотации
    ]
}
```

Каждый объект разметки выглядит в соответствии с листингом 20, где:

– description – описание объекта разметки, должно соответствовать типу БД (типу проекта). Например, «Segmentation», «ObjectDetection» и др.;

– class – класс объекта разметки;

– shape – геометрическая фигура, которой отмечен объект; если данное поле отсутствует или ему соответствует значение «null», то классом отмечено все изображение.

Листинг 20 – Структура объекта разметки

```
{
    "description": "<описание>",
    "class": "<имя класса>",
    "shape": {
        // объект геометрии
    }
}
```

Поддерживаются следующие объекты геометрии разметки:

– прямоугольник,

– линия,

– точка,

– многоугольник,

– бинарная (raster),

– изображение из хранилища.

Формат записи прямоугольника представлен в листинге 21, где:

– exterior – координаты вершин [[left, top], [right, bottom]] ([[левая верхняя вершина], [правая нижняя вершина]]);

– interior – пустой список.

Листинг 21 – Формат записи прямоугольника

```
    "type": "rectangle",
    "exterior": [
        [1075, 298], [1517, 681]
    ],
    "interior": []
}
```

Формат записи линии представлен в листинге 22, где:

– exterior – список координат точек линии [[x1, y1], [x2, y2], [x3, y3], [x4, y4], ...];

– interior – пустой список.

Листинг 22 – Формат записи линии

```
{
    "type": "line",
    "exterior": [
        [42, 43], [1075, 298], [1517, 681]
    ],
    "interior": []
}
```

Формат записи точки представлен в листинге 23, где:

– exterior – список координат точек [[x1, y1], [x2, y2], [x3, y3], [x4, y4], ...];

– interior – пустой список.

Листинг 23 – Формат записи точки

```
{
    "type": "point",
    "exterior": [
        [42, 43], [1075, 298], [1517, 681]
    ],
    "interior": []
}
```

Формат записи многоугольника представлен в листинге 24, где:

– exterior – список кординат точек внешнего контура многоугольника [[x1, y1], [x2, y2], [x3, y3], [x4, y4], ...];

– interior – список списков координат точек внутренних контуров многоугольника.

Листинг 24 – Формат записи многоугольника

```
{
    "type": "polygon",
    "exterior": [
        [1126, 753], [1123, 775], ..., [1136, 721], [1126, 753]
    ],
    "interior": [
        [[1, 2], [3, 4], ...],
        [[5, 6], [7, 8], ...],
    ]
}
```

Формат записи бинарного типа представлен в листинге 25, где:

– origin – координаты верхнего левого угла изображения;

– data – битовая карта (bitmap) разметки, закодированная в base64.

Листинг 25 – Формат записи бинарного типа

```
{
    "type": "raster",
    "origin": [
        1177,
        931
    ],
    "data": "eJwBvwJA/YlQTkcNC ... AEDm2GYAAAJn"
}
```

Формат записи изображения представлен в листинге 26, где:

– origin – координаты верхнего левого угла изображения;

– uuid – идентификатор битовой карты (bitmap) разметки, содержащейся в виде отдельного файла в хранилище.

Листинг 26 – Формат записи изображения

```
{
    "type": "image",
    "origin": [
        1177,
        931
    ],
    "uuid": "a74ae9a0-7e05-4bf1-8c41-c654c4983e62"
}
```

Началом отсчёта всех координат является верхний левый угол изображения.

Пример разметки для задачи обнаружения представлен в листинге 27.

Листинг 27 – Пример разметки для задачи обнаружения

```
{
    "version": 1,
    "annotation": [
        {
            "description": "ObjectDetection",
            "class": "Car",
            "shape": {
                "type": "rectangle",
                "exterior": [
                    [236, 47], [274, 68]
                ],
                "interior": []
            }
        },
        {
            "description": "ObjectDetection",
            "class": "Plane",
            "shape": {
                "type": "rectangle",
                "exterior": [
                    [137, 54], [265, 120]
                ],
                "interior": []
            }
        }
    ]
}
```

4.5. Форматы метрик обучения

В клиенте УПП «Платформа-ГНС» поддерживается один тип метрик обучения – график ("type" : "graphic"), который выводится двумя способами:

– график по эпохам (по умолчанию);

– график по итерациям внутри эпохи.

При отображении графика эпох используется специальная точка «iter: -1». Данный подход позволяет отображать среднее значение по эпохе, либо результат валидации.

4.6. Форматы метрик тестирования

Поддерживается четыре типа метрик тестирования:

– графики;

– матрица путанности (confusion matrix);

– гистограммы;

– таблицы.

4.6.1. Графики

Способ хранения графиков представлен в листинге 28.

Листинг 28 – Способ хранения графиков

```
{
  "graphics": [
    // список графиков
  ]
}
```

Каждый график в списке является словарем (см. листинг 29), в котором:

– display_name – название графика;

– name_x – название оси X;

– name_y – название оси Y;

– trace – словарь, содержащий в себе все виды кривых, отображаемых на одном графике.

Листинг 29 – Общий вид словаря графика

```
{
  "display_name": "Graphic Name",
  "name_x": "Axis X Name",
  "name_y": "Axis Y Name",
  "trace": {}
}
```

Общий вид словаря, содержащего в себе все виды кривых, представлен в листинге 30, где:

– name – ключ-название кривой (например, «loss»);

– display_name – отображаемое название кривой;

– data – упорядоченный список точек в формате словаря, где значения ключей «x» и «y» отвечают за координаты точки на графике, а «point_info» – строка с информацией о данной точке;

– info – строка с информацией о кривой.

Листинг 30 – Общий вид словаря, содержащего в себе все виды кривых

```
  {
    "name": {
      "display_name": "Line Name",
      "data": [
        {
          "x": 1,
          "y": 1,
          "point_info": "Point Info"
        }
      ],
      "info": "Line Info"
    }
  }
```

Пример формирования графика представлен в листинге 31.

Листинг 31 – Пример формирования графика

```
{
  "graphics": [
    {
      "display_name": "Graphic Name",
      "name_x": "Axis X Name",
      "name_y": "Axis Y Name",
      "trace": {
        "name": {
          "display_name": "Line Name",
          "data": [
            {
              "x": 1,
              "y": 1,
              "point_info": "Point Info"
            }
          ],
          "info": "Line Info"
        }
      }
    }
  ]
}
```

4.6.2. Матрица путанности

Способ хранения матрицы путанности представлен в листинге 32.

Листинг 32 – Способ хранения матрицы путанности

```
{
  "confusion_matrices": [
    //список значений
  ]
}
```

Каждый элемент списка значений является словарем (см. листинг 33), в котором:

– display_name – название матрицы путанности;

– name_x – название оси X;

– name_y – название оси Y;

– labels_x – подписи по оси X слева направо;

– labels_y – подписи по оси Y сверху вниз;

– grid – двумерный список значений ячеек матрицы путанности.

Листинг 33 – Общий вид словаря матрицы путанности

```
{
  "display_name": "Matrix Name",
  "name_x": "Axis X Name",
  "name_y": "Axis Y Name",
  "labels_x": ["X1 Name", "X2 Name"],
  "labels_y": ["Y1 Name", "Y2 Name", "Y3 Name"],
  "grid": [[]]
}
```

Двумерный список значений ячеек матрицы путанности имеет вид словаря, представленного в листинге 34, где:

– count – количество объектов в ячейке;

– percent – процент количества объектов от общего числа.

Листинг 34 – Общий вид двумерного списка значений ячеек матрицы путанности

```
{
    "count": 1,
    "percent": 0.8
  }
```

Пример формирования матрицы путанности представлен в листинге 35.

Листинг 35 – Пример формирования матрицы путанности

```
{
  "confusion_matrices": [{
    "display_name": "Matrix Name",
    "name_x": "Axis X Name",
    "name_y": "Axis Y Name",
    "labels_x": ["X1 Name", "X2 Name"],
    "labels_y": ["Y1 Name", "Y2 Name", "Y3 Name"],
    "grid": [
      [
        {
          "count":  25,
          "percent":  0.8
        }
      ]
    ]
  }]
}
```

4.6.3. Гистограммы

Способ сохранения гистограмм представлен в листинге 36.

Листинг 36 – Способ сохранения гистограмм

```
{
  "histograms": [
    // список гистограмм
  ]
}
```

Каждая гистограмма в списке является словарем (см. листинг 37), в котором:

– display_name – название гистограммы;

– name_x – название оси X;

– name_y – название оси Y;

– labels – подписи столбцов гистограммы;

– bins – список, содержащий в себе все значения гистограммы.

Листинг 37 – Общий вид словаря гистограммы

```
{
  "display_name": "Histogram Name",
  "name_x": "Axis X Name",
  "name_y": "Axis Y Name",
  "labels": ["L1 Name", "L2 Name"],
  "bins": []
}
```

Общий вид списка со значениями гистограммы представлен в листинге 38, где:

– value – значение столбца гистограммы;

– info – строка с информацией о столбце (необязательно).

Листинг 38 – Общий вид списка со значениями гистограммы

```
  {
    "value": 10,
    "info": "Bin Info"
  }
```

Пример формирования гистограммы представлен в листинге 39.

Листинг 39 – Пример формирования гистограммы

```
{
  "histograms": [
    {
      "display_name": "Histogram Name",
      "name_x": "Axis X Name",
      "name_y": "Axis Y Name",
      "labels": ["L1 Name", "L2 Name"],
      "bins": [
        {
          "value": 10,
          "info": "Bin Info"
        }
      ]
    }
  ]
}
```

4.6.4. Таблицы

Способ сохранения таблиц представлен в листинге 40.

Листинг 40 – Пример формирования таблиц

```
{
  "tables": [
    // список таблиц
  ]
}
```

Каждая таблица является словарем (см. листинг 41), в котором:

– display_name – название таблицы;

– labels_x – подписи по оси X слева направо;

– labels_y – подписи по оси Y сверху вниз;

– data – двумерный список значений ячеек таблицы.

Листинг 41 – Общий вид словаря таблицы

```
{
  "display_name": "Table Name",
  "labels_x": ["X1 Name", "X2 Name"],
  "labels_y": ["Y1 Name", "Y2 Name", "Y3 Name"],
  "data": [[]]
}
```

Пример формирования таблиц приведен в листинге 42.

Листинг 42 – Пример формирования таблиц

```
{
  "tables": [{
    "display_name": "Table Name",
    "labels_x": ["X1 Name", "X2 Name"],
    "labels_y": ["Y1 Name", "Y2 Name", "Y3 Name"],
    "data": [[1, 2, 3], [1, 2]]
  }]
}
```

4.7. Формат записи в журнал сообщений (лог)

Запись логов производится в БД по ключу «task_<uuid>_train_metrics», значение которого представляет собой набор записей, содержащихся в порядке их добавления. Общий формат записи лога приведен в листинге 43, где:

– type – тип метрики;

– name – название метрики;

– datetime – время записи в формате «год-месяц-день час:минута:секунда»;

– value – значение метрики;

– epoch – эпоха записи;

– iter – итерация записи;

– info – дополнительная информация о метрике (необязательно).

Листинг 43 – Общий формат записи лога

```
[
  {
    "type": "graphic",
    "name": "",
    "datetime": "",
    "value": "",
    "epoch": "",
    "iter": "",
    "info": ""
  }
]
```

5. Примеры использования

5.1. Формирование набора параметров

Параметры, которые задаются пользователем через интерфейс, влияют на ход выполнения программы. Добавление нового параметра осуществляется в разделе «Проект», в окне «Параметры обучение» (см. рисунок 3) после нажатия кнопки «+».

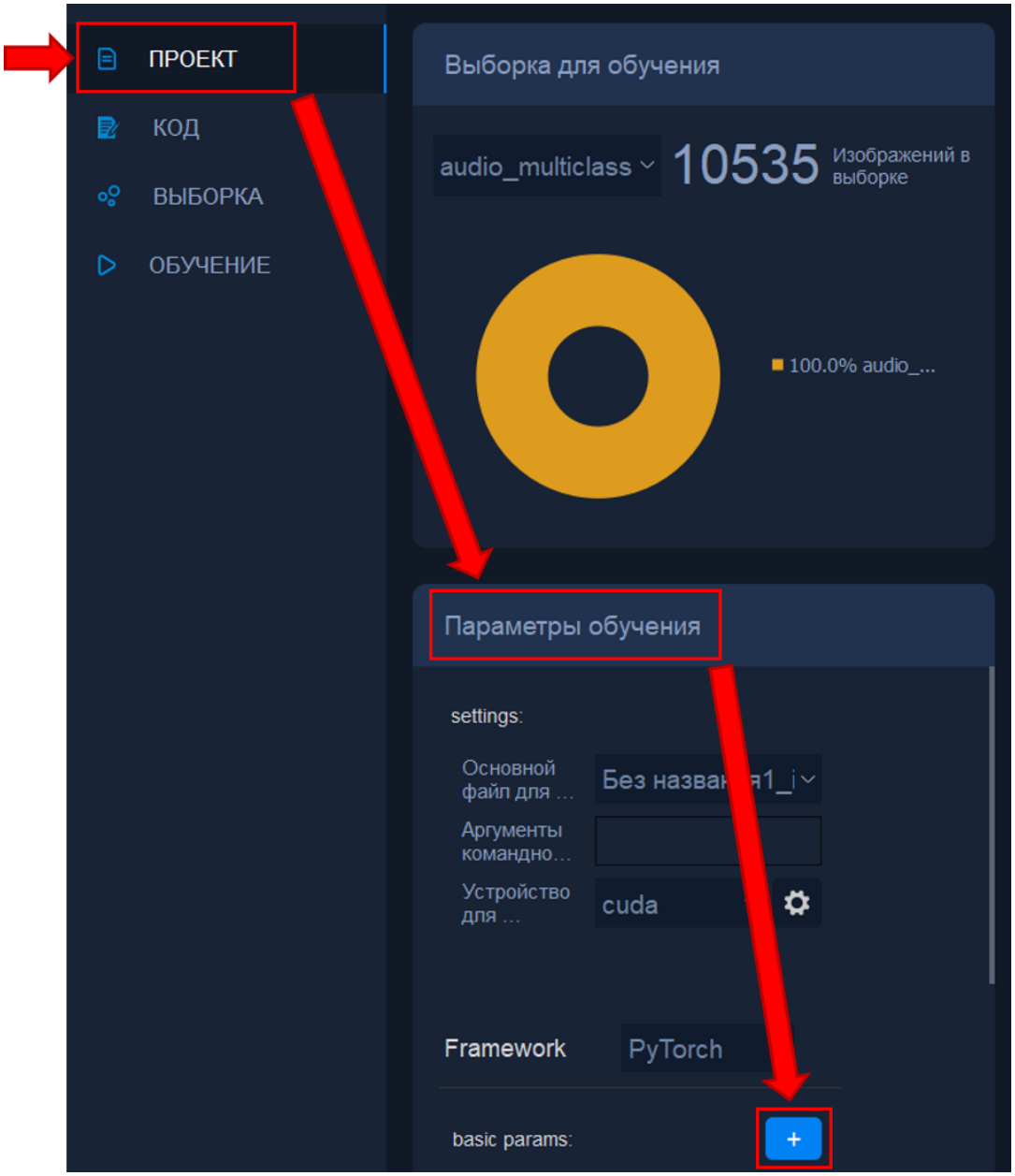

Рисунок 3 – Добавление нового параметра

В появившемся окне (см. рисунок 4) выбираются поля из выпадающих списков (basic_options или advanced_options и network или learning), задается имя параметра (английскими буквами и цифрами), выбирается тип данных («str» – строка, «int» – целое число, «float» – число с плавающей точкой, «bool» – логический тип, булев тип).

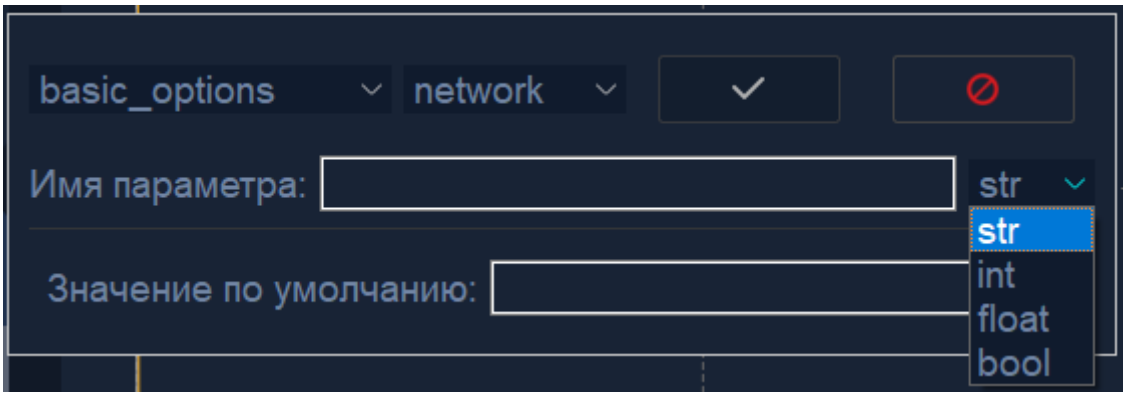

Рисунок 4 – Окно настройки нового параметра

В зависимости от выбранного типа данных параметра предоставляются разные возможности дальнейших настроек. Так, для строкового типа указывается значение по умолчанию (см. рисунок 4); для целочисленного – значение по умолчанию и, при желании, минимальное и максимальное значения и шаг (для получения возможности ввода поставьте «галочку» в белой ячейке (активируйте «чекбокс») напротив параметра) (см. рисунок 5); для типа данных с плавающей точкой – по аналогии целочисленным типом (см. рисунок 6); для логического типа – значение по умолчанию (для задания значения «True» необходимо активировать «чекбокс») (см. рисунок 7).

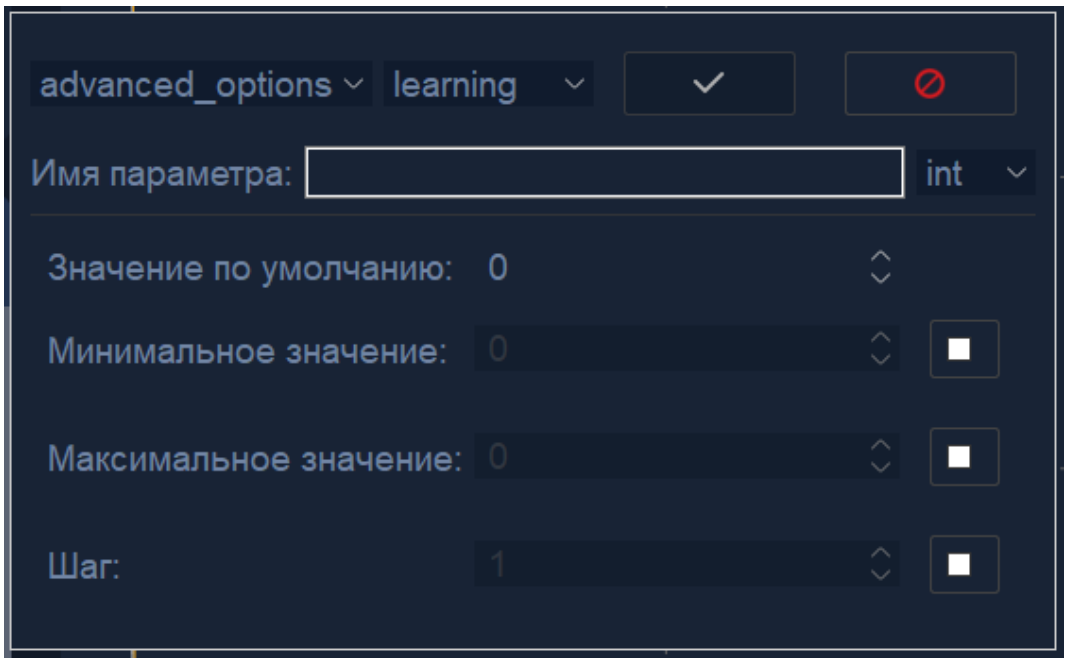

Рисунок 5 – Настройки параметра целочисленного типа данных

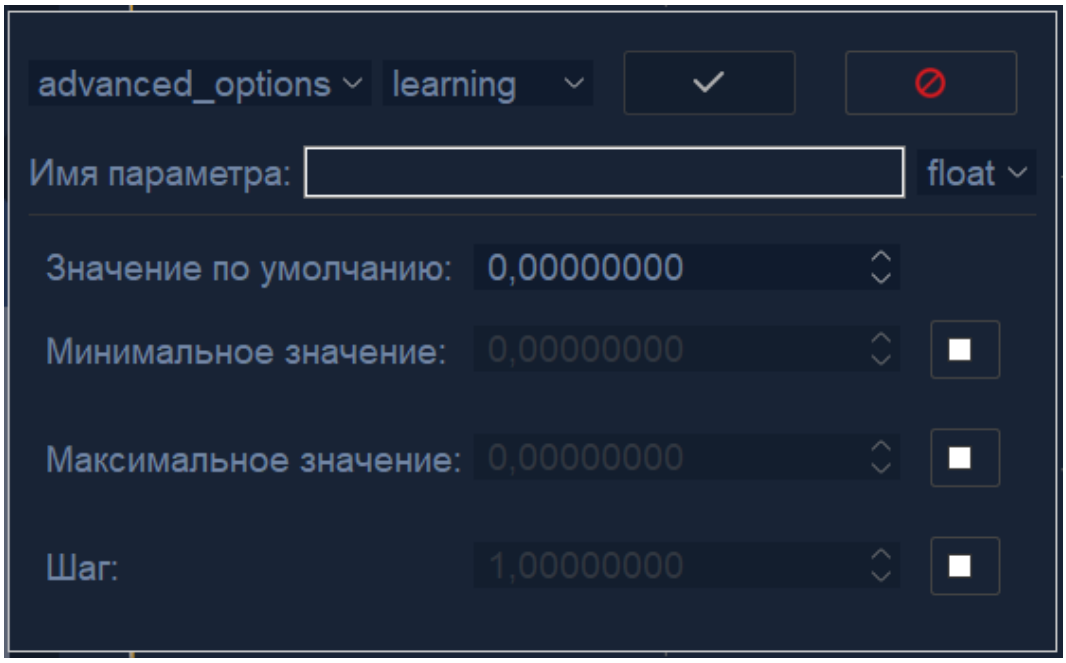

Рисунок 6 – Настройки параметра типа данных числа с плавающей точкой

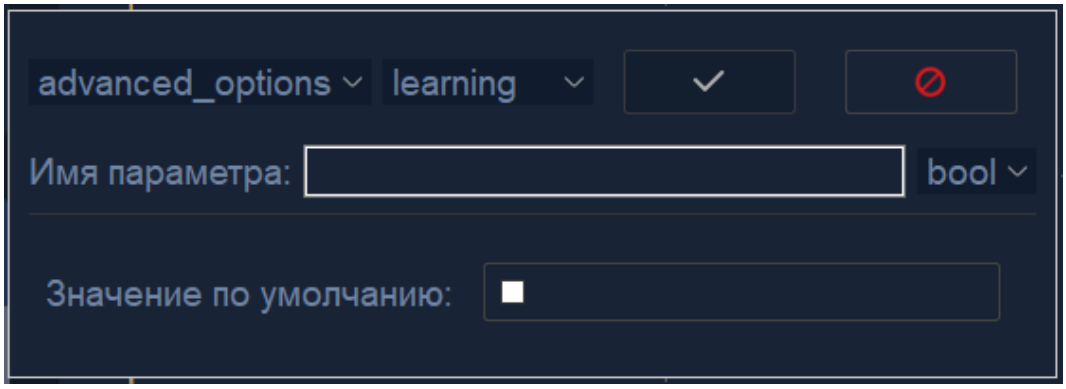

Рисунок 7 – Настройки параметра логического типа

Заданные параметры формируются в файле JSON и доступны пользователю при запуске подготовленного решения задачи.

В коде решения задачи (обучения или тестирования) входные параметры можно получить с помощью методов, описанных в подразделе 3.1.

На выходе данного метода формируется словарь с параметрами (см. листинг 44).

Листинг 44 – Пример словаря с параметрами

```
{'args': '',
 'main': 'train.py',
 'input_h': 32,
 'input_w': 32,
 'num_gpu': 1,
 'max_epoch': 5,
 'batch_size': 16,
 'target_device': 'cuda',
 'dataset': {
    'classes': [
      'Abrams',
      'Chellendger',
      'Leclerc',
      'Leopad1',
      'Leopard2',
      'Merkava',
      'T-90',
      'Zulficar'
    ],
    'count_classes': [
      251,
      251,
      251,
      251,
      251,
      251,
      251,
      251
    ],
    'count_images': 2008,
    'count_items': {
      '2b3279f874aa4191b944d9c5e9e3d37b': 2008
    },
    'options': {},
    'name': 'train'
  }
}
```

Сформированный словарь содержит в себе перечень как обязательных и необязательных параметров (см. таблицу), так и опциональные, зависящие от решаемой задачи и ее методов.

Таблица 2 – Обязательные и необязательные параметры словаря заданных параметров

Параметр	**Примечание**
Обязательные	
main	Основной файл для исполнения программы, который нужно выбрать в интерфейсе клиента
dataset	Вся информация о базе данных, формируется при добавлении выборки
target_device	Используемое устройство при запуске программы
Необязательные	
args	Включает в себя аргументы командной строки, в случае если их добавить в интерфейсе клиента

5.2. Получение данных из выборки

В разделе «Выборка» пользователь может сформировать выборки из доступных для него баз данных. База данных состоит из основного файла, дополнительного файла (при его наличии) и JSON-файла аннотации (при его наличии). Основной и дополнительный файлы могут содержать изображения, звуки, таблицы или другие пользовательские данные. JSON-файл аннотации имеет определенную структуру (см. подраздел 4.4).

Далее, в листингах 7 – 9, приведены примеры JSON-файлов для задач классификации и сегментации изображений, а также обнаружения объектов на изображениях:

Листинг 45 – Пример файла разметки для задачи «Классификация»

```
{
    "version": 1,
    "annotation": [
        {
            "description": "Classification",
            "class": "airplane",
            "shape": {}
        }
    ]
}
```

Листинг 46 – Пример файла разметки для задачи «Обнаружение»

```
{
    "version": 1,
    "annotation": [
        {
            "description": "ObjectDetection",
            "class": "Car",
            "shape": {
                "type": "rectangle",
                "exterior": [
                    [236, 47], [274, 68]
                ],
                "interior": []
            }
        },
        {
            "description": "ObjectDetection",
            "class": "Plane",
            "shape": {
                "type": "rectangle",
                "exterior": [
                    [137, 54], [265, 120]
                ],
                "interior": []
            }
        }
    ]
}
```

Листинг 47 – Пример файла разметки для задачи «Сегментация»

```
{
    "version": 1,
    "annotation": [
        {
            "description": "Segmentation",
            "class": "raster",
            "shape": {
                "type": "raster",
                "origin": [
                    0, 0
                ],
                "data": "..."
            }
        }
    ]
}
```

Для работы с выборкой необходимо получить список доступных выборок (см. подраздел 3.2).

Для обращения к выборке на стадии обучения необходимо обратиться к выборке с индексом «0», на стадии тестирования с индексом «2».

Подготовить выборку для дальнейшей работы: «train»: i = 0, «test»: i = 1.

Пример получения элемента выборки представлен в листинге 48.

Листинг 48 – Пример получения элемента выборки

```
train_db = datasets[i]
images_path = train_db.get_all_images_path()
add_images_path = train_db.get_all_add_images_path()
markups_path = train_db.get_all_markups()
#images_path - список путей для основных элементов базы
(изображения/файлы/таблицы)
#add_images_path - список путей до дополнительный элементов
базы (опционально)
#markups_path - список путей до JSON-файлов, содержащих
разметку (опционально)
```

5.3. Прогресс выполнения задания

При запуске программы необходимо прогрессу присвоить стартовое (нулевое) значение (см. листинг 49):

Листинг 49 – Пример присвоения переменной прогресса выполнения задания стартового значения

```
Progress = 0
```

В ходе выполнения программы передавать значение переменной прогресса выполнения задания от 0 до 100 (см. листинг 50)

Листинг 50 – Пример передачи значения переменной прогресса выполнения задания

```
# Пример описание переменных прогресса
percent = 100.0 / (len(loader_train) *
int(params['max_epoch']))
progress += percent

# Обновление прогресса обучения
PlatformAPI.save_progress_for_platform(progress)
```

Обновление статуса выполнения задания отображается на прогресс-баре (индикатор выполнения), соответствующем стадии выполнения решения (обучение, тестирование, отладка кода в работе с кодом).

5.4. Вывод сообщений

В ходе выполнения решения доступен вывод трех типов сообщений: информационные, предупреждения и сообщения об ошибке.

Инициализация объекта, передающего информационные сообщения, описана в подразделе 3.4.

В зависимости от типа сообщения используются разные методы для их вывода. Общий пример отправки сообщения в соответствующую вкадку информационного поля, приведен в листинге 51.

Листинг 51 – Пример отправки сообщения

```
message = ('TRAIN ||Epoch: %d || iter: %d' % (epoch + 1, i + 1)
logger.info(message) # информация
logger.error(message) # ошибка
logger.warning(message) # предупреждение
```

5.5. Построение графиков обучения и валидации

Для построения графиков обучения и валидации необходимо инициализировать объект, передающий информацию о графиках обучения и валидации (см. подраздел 3.5)

Пример записи точки на графике представлен в листинге 52.

Листинг 52 – Пример отправки сообщения

```
# Определение последней итерации в эпохе
    is_last = idx == len(dataloader) - 1

    # Сохранение точек для графика
    train_logger.log_point(
        params['loss_fn'] + "Loss",
        train_loss / (idx + 1),
        epoch + 1,
        idx + 1,
        params['loss_fn'] + "Metrics",
        last_iter_in_epoch=is_last)
# name – имя графика;
# value – значение точки;
# epoch – номер эпохи;
# iter – номер итерации;
# info – дополнительная информация для точки на графике;
# last_iter_in_epoch – флаг последней итерации в эпохе.
```

При формировании графиков валидации доступно сохранение только одной точки на эпоху. При этом значения «iter» должно быть равно единице, а «last_iter_in_epoch» присвоено значение «True».

Графиков обучения и валидации может быть несколько, для этого необходимо, чтобы они имели уникальные имена. Таким образом, для графика функции в цикле обучения необходимо задавать значения как для отображения по «эпохам» (когда записывается одно число для эпохи, где значению номера итерации присваивается значение единицы), так и по «итерациям». Для остальных графиков можно записывать и в режиме «эпохи».

Пример кода для формирования двух графиков представлен в листинге 53.

Листинг 53 – Пример кода для формирования двух графиков

```
train_logger.log_point(
                        "loss_L1",
                        float(los_L1),
                        epoch + 1,
                        i + 1,
                        "L1 metric",
                        last_iter_in_epoch=is_last)

train_logger.log_point(
                        "psnr",
                        float(psnr_train_total_epoch/(i+1)),
                        epoch + 1,
                        i + 1,
                        "PSNR metric",
                        last_iter_in_epoch=is_last)
```

Таким образом можно отобразить множество графиков.

Пример отображения графика в клиенте УПП «Платформа-ГНС» представлен на рисунке 8.

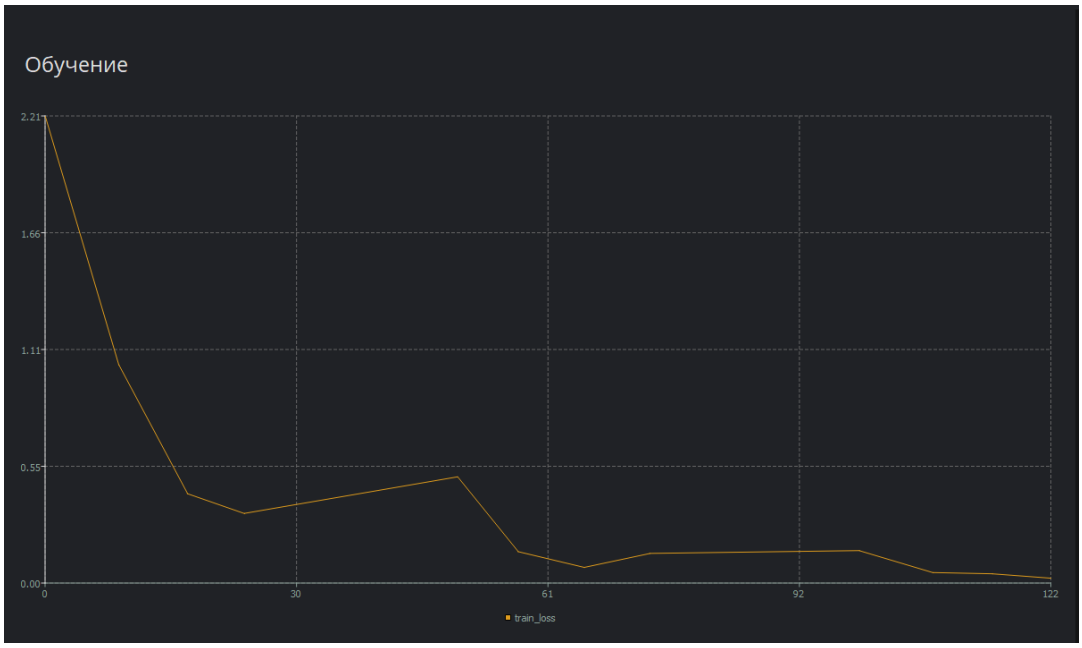

Рисунок 8 – Пример отображения графика в клиенте УПП «Платформа-ГНС»

5.6. Формирование метрик тестирования

Все метрики отображаются только при тестировании и будут доступны в разделе пользовательских решений.

Пример инициализации объекта, передающего информацию о метриках тестирования представлен в листинге 54.

Листинг 54 – Пример кода для формирования двух графиков

```
from PlatformAPI.testgraphlogger import TestGraphLogger

   test_logger = PlatformAPI.get_test_metrics_logger()
```

Для сохранения полученных значений метрик используются следующие классы:

– класс «Table» – для метрик в виде таблицы,

– класс «Matrix» – для метрик в виде матрицы,

– класс «ConfusionMatrix» – для метрик в виде матрицы путаницы,

– класс «Histogram» – для метрик в виде гистограммы.

Примеры вызова этих классов представлен в листинге 55.

Листинг 55 – Пример вызова классов типов метрик тестирования

```
from PlatformAPI import Table, Matrix, ConfusionMatrix, Histogram

Table.from_tensor(
                   data,
                   display_name,
                   labels_x,
                   labels_y)
# Метод, реализующий преобразование двумерного тензора формы [N
x M] (в виде листов), являющейся данными таблицы в формат
именованного кортежа.
# data - двумерная матрица, состоящая из списка;
# display_name - отображаемое имя матрицы;
# labels_x - название каждой строки или индексы;
# labels_y - название каждого  столбца.

Matrix.from_tensor(
                    matrix,
                    display_name,
                    name_x,
                    name_y,
                    labels_x,
                    labels_y)
# Возвращает трехмерный тензор в формате именованного кортежа.
# matrix - двумерная матрица, состоящая из списка;
# display_name - отображаемое имя матрицы;
# name_x - название столбцов;
```

```
# name_y - название строк;
# labels_x - название каждого столбца;
# labels_y - название каждой строки.

ConfusionMatrix.from_tensor(
                            matrix,
                            display_name,
                            name_x,
                            name_y,
                            labels)
# Возвращает трехмерный тензор в формате именованного кортежа.
# matrix - двумерная матрица состоящая из списка, элементом
будет 2 числа, "count" и "percent";
# display_name - отображаемое имя матрицы;
# name_x - название столбцов;
# name_y - название строк;
# labels - название каждой строки(столбца).

Histogram.from_tensor(
                      matrix,
                      display_name,
                      name_x,
                      name_y,
                      labels)
# Возвращает двумерный тензор в формате именованного кортежа.
# matrix - двумерная матрица состоящая из списка, элементом
будет 2 числа, "value" и "info";
# display_name - отображаемое имя гистограммы;
# name_x - название столбцов;
# name_y - название строк;
# labels - название каждого столбца.
```

Получение объекта для записи метрик тестирования, которые будут отображаться в клиенте УПП «Платформа-ГНС», представлено в листинге 56.

Листинг 56 – Пример вызова классов типов метрик тестирования

```
# Получение графиков
metric_logger.append(TestGraphLogger.get_json_for_graphics())
```

Примеры отображения метрик тестирования в виде матрицы путанности и гистограммы представлены на рисунках 9 и 10.

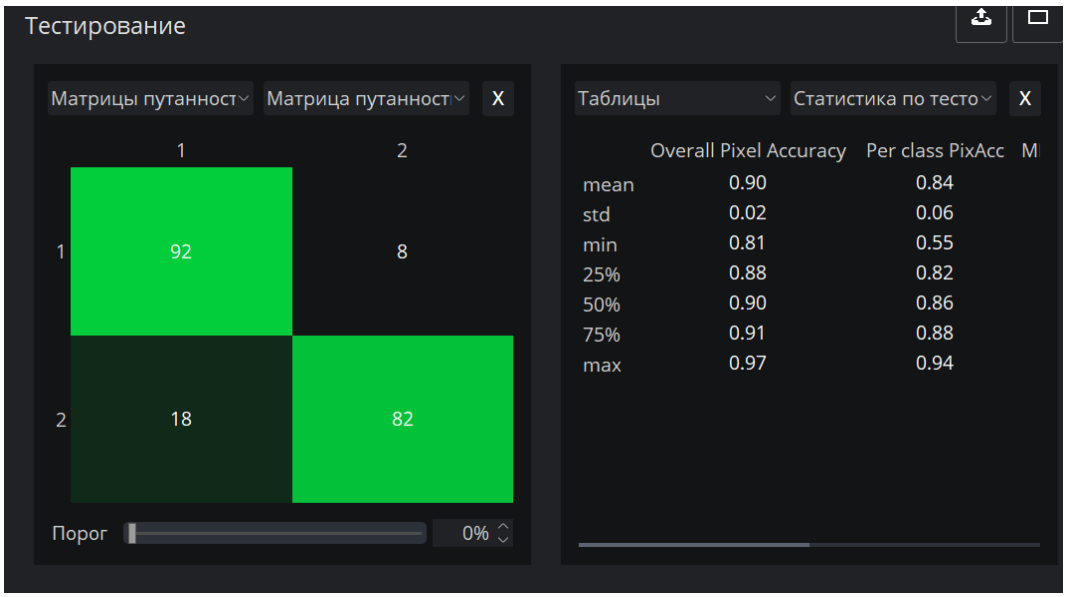

Рисунок 9 – Пример метрик тестирования в виде матрицы путанности в клиенте УПП «Платформа-ГНС»

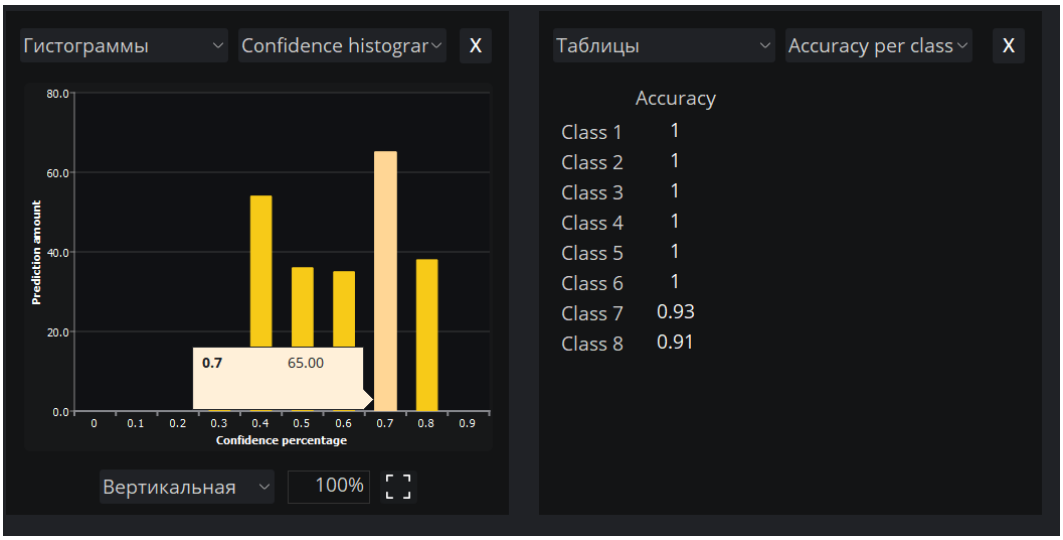

Рисунок 10 – Пример метрик тестирования в виде гистограммы в клиенте УПП «Платформа-ГНС»

5.7. Сохранение результатов обработки тестовой выборки

Результатом обработки элемента базы является JSON-файл. Его заполнение зависит от типа задачи и типа разметки. Для решения задачи классификации изображений пользователю необходимо сформировать JSON-файл и передать его с применением метода, чтобы получить словарь разметки в формате УПП «Платформа-ГНС» для списка классов (см. листинг 57).

Листинг 57 – Получение словаря разметки для списка классов

```
PlatformAPI.get_markup_dict_for_classes(json, description)
# description - тип решения
```

Для решения задачи сегментации изображений пользователю необходимо сформировать JSON-файл и передать его с применением метода, чтобы получить словарь разметки в формате УПП «Платформа-ГНС» для списка изображений (см. листинг 58).

Листинг 58 – Получение словаря разметки для списка изображений

```
PlatformAPI.get_markup_dict_for_images(json, description)
# description - тип решения
```

Для сохранения результатов обработки тестовой выборки при решении задач классификации и сегментации изображений, а также обнаружения объектов на изображении доступны методы, приведенные в листинге 59.

Листинг 59 – Методы сохранения результатов решения задач

```
from PlatformAPI.dataset import
save_test_markup_classification, save_test_markup_detection,
save_test_markup_segmentation

# Для классификации
save_test_markup_classification(predicted, data, num)
# predicted - список предсказанных классов
# data - список картинок
# num - номер батча

# Для обнаружения
save_test_markup_detection(predicted, map_cls, data)
# predicted - список предсказанных классов
# data - список картинок
# map_cls - список названий классов

# Для сегментации
save_test_markup_detection(predicted, data)
# predicted - список предсказанных классов
# data - список картинок
```

5.8. Экспорт готового пользовательского решения

В разделе проект при нажатии на кнопку «Экспорт» необходимо выбрать исполняемый файл и заполнить информационные поля (см. рисунок 11).

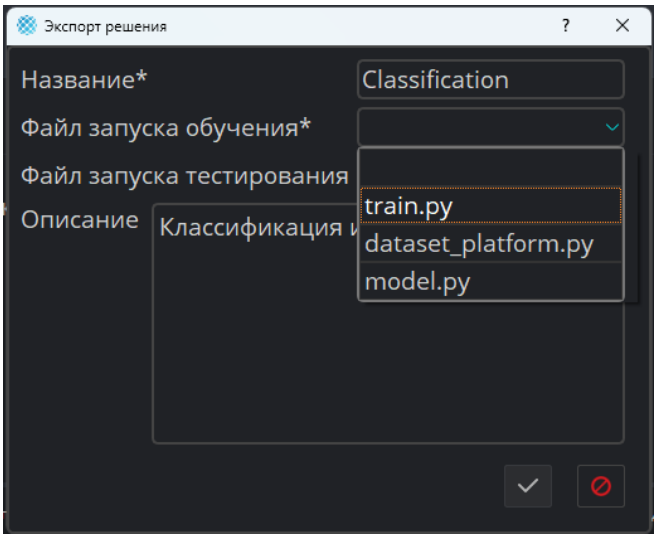

Рисунок 11 – Параметры настройки экспорта
в клиенте УПП «Платформа-ГНС»

Код будет доступен во вкладке «Решения» в окне «Пользовательские решения».