



PROGRAMLAB
INNOVATIVE DIGITAL SYSTEMS

РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

**ВИРТУАЛЬНЫЙ УЧЕБНЫЙ КОМПЛЕКС
«МОДЕЛИРОВАНИЕ И УПРАВЛЕНИЕ ДОРОЖНЫМ
ДВИЖЕНИЕМ С ПОМОЩЬЮ МАШИННОГО ОБУЧЕНИЯ»**



ОГЛАВЛЕНИЕ

Инструкция по установке и запуску проекта	4
Запуск и управление в программе.....	7
Введение в нейронные сети	9
История появления нейросети.....	10
Структура нейросети	11
Принцип работы	14
Функции нейросети	16
Задача.....	19
Методы обучения	20
Типы нейросетей	22
Применения нейросетей.....	24
Преимущества и недостатки нейросетей.....	26
Установка и настройка сервера	27
Работа в программе.....	34
Создание датасета	42
Обучение на созданном датасете.....	52
Спецификация API.....	59
Спецификация сервера управления городом.....	59
Описание лабораторной работы	66
Описание сервера управления городом.....	67
Описание сервера нейросети распознавание объектов	80
Устранение проблем и ошибок	91

Инструкция по установке и запуску проекта

1. Распакуйте, соберите и подключите к сети компьютер.
2. Установите «PLCore».

Модуль запуска программных комплексов PLCore предназначен для запуска, обновления и активации программных комплексов, поставляемых компанией «Програмлаб».

В случае поставки программного комплекса вместе с персональным компьютером модуль запуска PLCore устанавливается на компьютер перед отправкой заказчику.

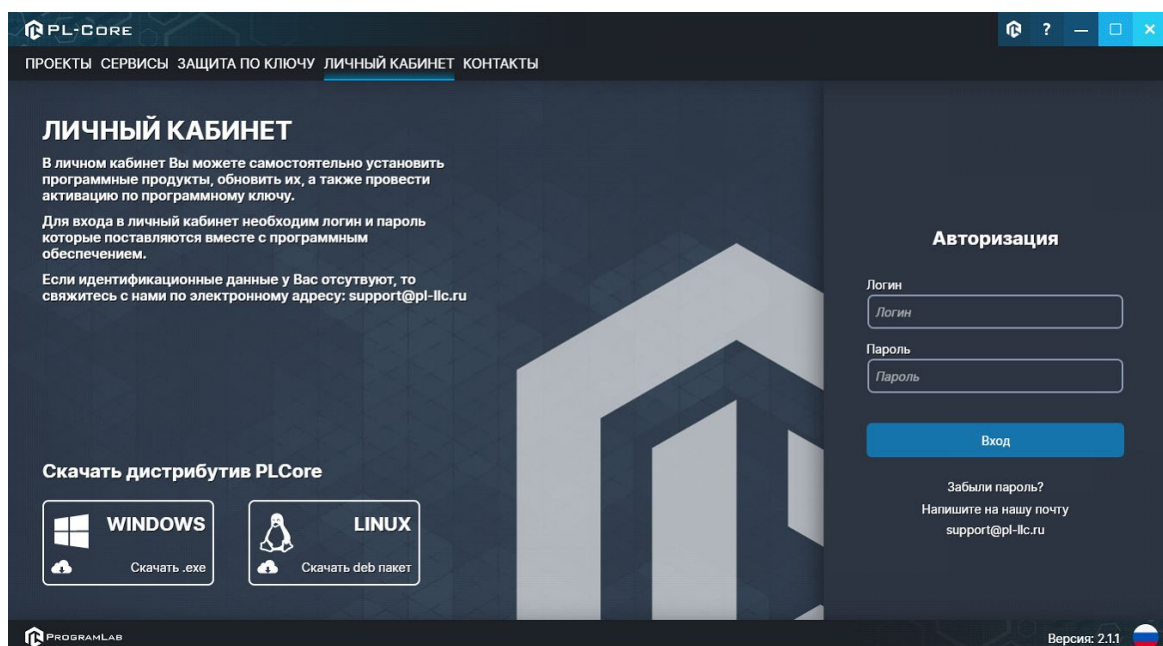
В случае поставки программного комплекса без ПК вам необходимо установить программное обеспечение с USB-носителя.

Перед установкой программного обеспечения установите модуль запуска учебных комплексов PLCORE. Для этого запустите файл с названием вида **PLCoreSetup_vX.X.X** на USB-носителе (Значения после буквы v в названии файла обозначают текущую версию ПО) и следуйте инструкциям.

3. Войдите в личный кабинет «PLCore».

В комплект поставки входит **конверт с идентификационными данными для личного кабинета**. Если конверта нет, то напишите нам на почту support@pl-llc.ru.

Во вкладке «Личный кабинет» располагается окно авторизации по уникальному логину и паролю. После прохождения авторизации в личном кабинете представляется информация о доступных программных модулях (описание, состояние лицензии, информация о версиях), с возможностями их удаленной загрузки, обновления и активации по сети интернет.



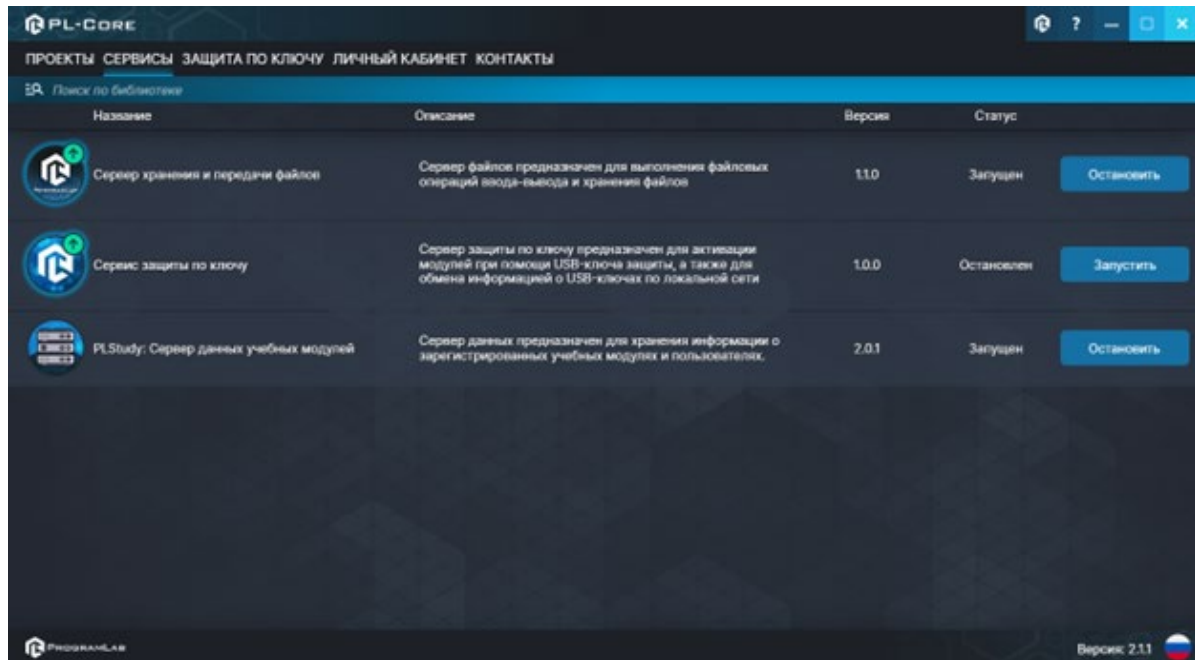
Вход в личный кабинет «PLCore»

4. Активируйте проект следуя руководству пользователя «PLCore».

5. Если ваш стенд предполагает автоматическую отправку результатов, установите «PLStudy» – программный комплекс, состоящий из двух модулей:

-Сервис «PLStudy: Сервер данных учебных модулей»

-Программный модуль «PLStudy: Администрирование»



Вкладка «Сервисы» с установленными и запущенными Сервером хранения и передачи файлов и PLStudy: Сервер данных учебных модулей

Установите сервер данных учебных модулей, если он ещё не установлен, на компьютер, который будет являться сервером. Для этого воспользуйтесь руководством пользователя «PLStudy».

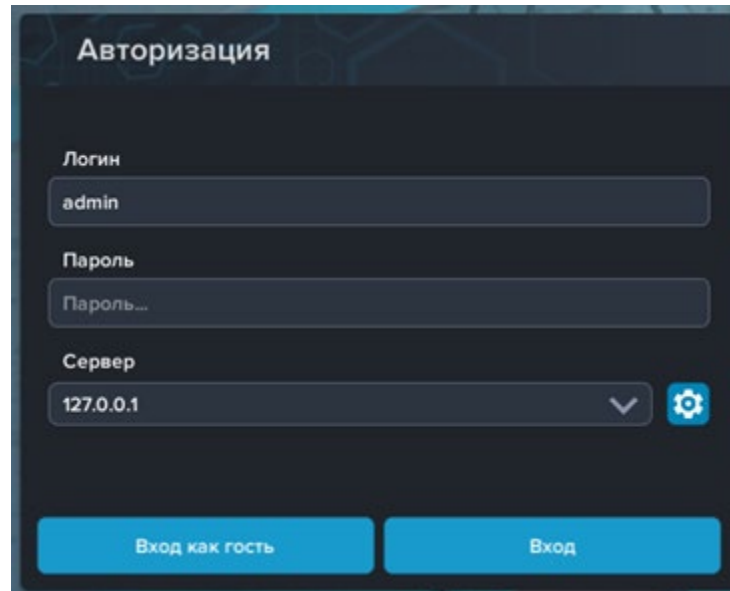
По умолчанию в системе создается пользователь с именем **Администратор** и ролью **Администратор**. Этот пользователь не может быть удален, но его параметры могут быть изменены. **По умолчанию логин пользователя: admin; Пароль: admin.**

6. Для некоторых проектов необходим сервис «**Сервер хранения и передачи файлов**». Сервер необходим для сохранения и загрузки с него файлов большого объема. Например, отчетов о прохождении тестирования в формате PDF.

7. Запустите проект.

Перед входом программа запросит логин, пароль. Здесь необходимо ввести параметры администратора или созданного на сервере («PLStudy») пользователя.

При авторизации в поле «Сервер» должен быть указан IP-адрес компьютера, на котором установлен сервис «**PLStudy: Сервер данных учебных модулей**».



Авторизация

Логин
admin

Пароль
Пароль...

Сервер
127.0.0.1

Вход как гость Вход

Окно авторизации

Запуск и управление в программе



— Левая кнопка мыши – действие, выбор объекта;



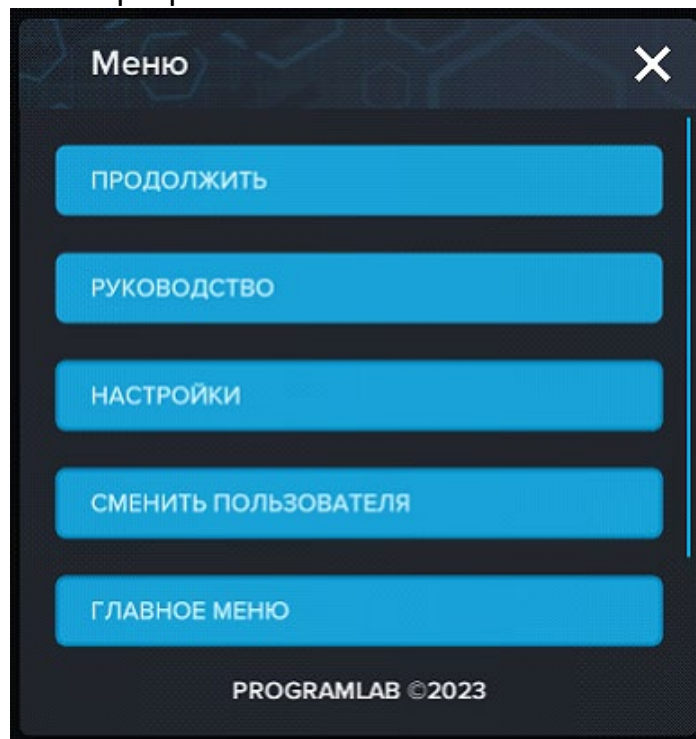
— Правая кнопка мыши – вращение камеры;



— Вращение колеса мыши – приближение\отдаление камеры;



— Вызов меню программы.



Меню программы

«**Продолжить**» – вернуться в программу;

«**Руководство**» – вызвать руководство пользователя;

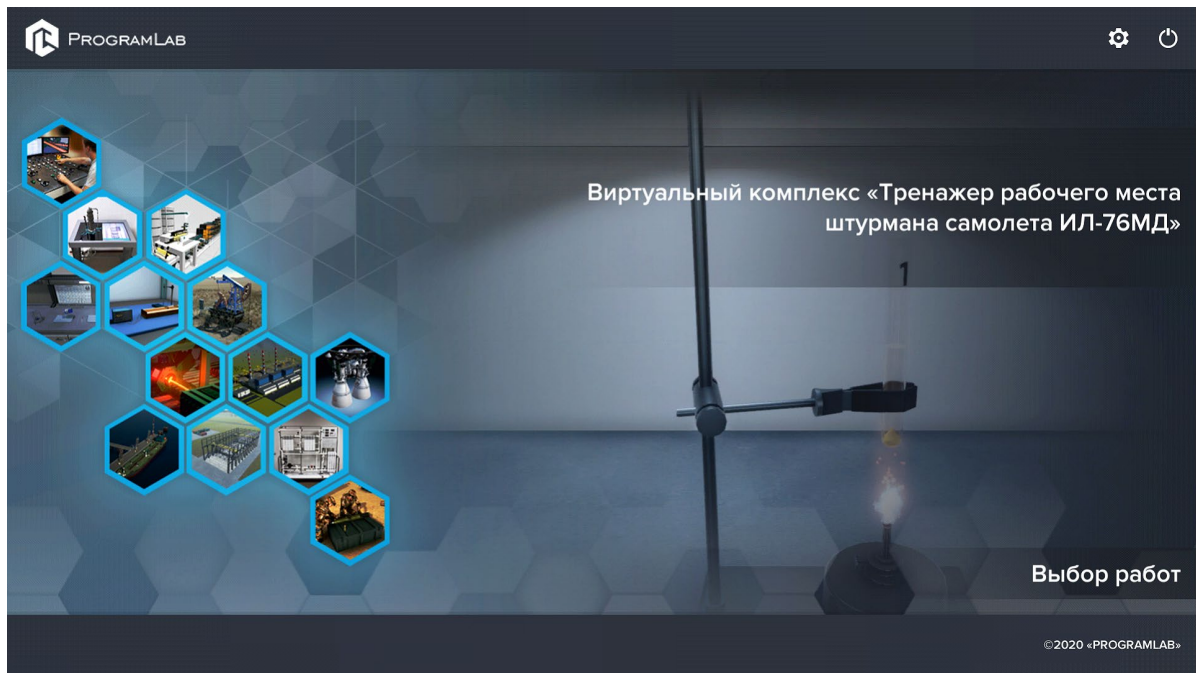
«**Настройки**» – настройки параметров графики;

«**Сменить пользователя**» – пройти авторизацию повторно;


«**Главное меню**» – выход в главное меню;

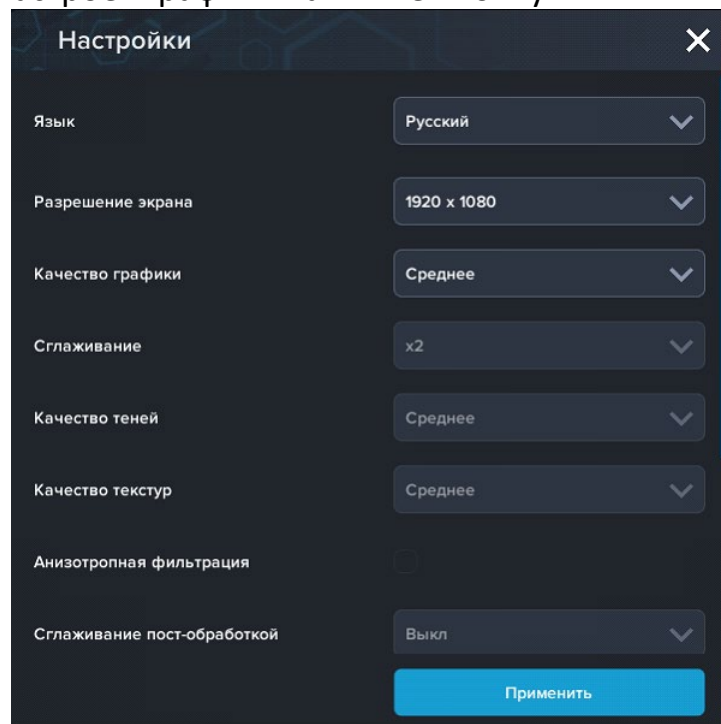
«**Выход**» – выход из программы.

Для запуска программы нажмите кнопку **«Загрузить»**, либо нажмите кнопку **«Выбор работ»** и выберите из открывшегося списка режим работы.




Окно запуска программного модуля

Для изменения настроек графики нажмите кнопку .



Окно настроек графики

Нажмите **«Применить»** чтобы закрыть окно.

Для выхода из программы нажмите .

Введение в нейронные сети

Нейронная сеть (neural network) – это компьютерный алгоритм, способный обрабатывать большие объемы данных, имитируя деятельность человеческого мозга. Как и человек, нейросеть изучает новые предметы, делает выводы и в дальнейшем использует полученную информацию. Нейросети представляют собой математические модели, созданные на основе биологических нейронных сетей, существующих в глубинах человеческого мозга.

Нервную систему человека образуют нейроны – клетки, которые получают информацию и транслируют ее в виде импульсов. Основная часть нейрона – аксон, а длинный отросток на его конце носит название дендрит, он выполняет роль своеобразного провода при передаче информации от одного нейрона к другому. Таким образом мозг, транслируя информацию, управляет всеми действиями человека.

На основе соответствующего принципа работают и компьютерные нейронные сети, ставшие цифровой моделью человеческого мозга. Главная же их особенность – **способность к обучению**. Стандартные компьютерные программы предполагают, что алгоритм для них пишет человек, то есть задает определенный набор действий, которые должны выполнить компьютеры. При использовании нейросети не нужно говорить ей, как решить задачу. Достаточно задать вводные данные, а способам решения задач нейронная сеть на основе искусственного интеллекта обучается сама, выявляя закономерности и обнаруживая на их основе способы решения задач.

История появления нейросети

Попытки математически описать сеть нейронов предпринимались еще в 1940-е годы. Идею создания нейронных сетей впервые предложили исследователи из Чикагского университета Уоррен Маккалоу и Уолтер Питтс. В 1950-е годы эта математическая модель была воссоздана психологом Корнеллского университета Фрэнком Розенблаттом с помощью компьютерного кода. Розенблатт был автор перцептрона – прототипа современных нейросетей. Даже такая элементарная структура в те годы могла обучаться и самостоятельно решать простые задачи.

Возрождение интереса к нейронным сетям и революция в глубоком обучении произошли лишь в последние годы благодаря индустрии компьютерных игр. Современные игры требуют сложных вычислений для обработки большого числа операций. В итоге производители начали выпускать **графические процессоры (GPU)**, которые объединяют тысячи относительно простых вычислительных ядер на одном чипе. Исследователи вскоре поняли, что архитектура графического процессора очень похожа на архитектуру нейросети.

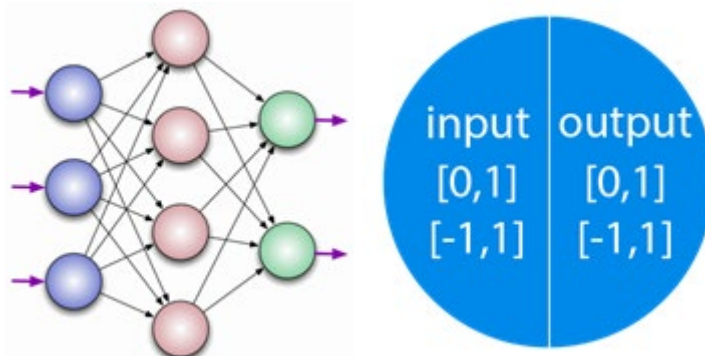
Современные GPU позволили развивать «глубокое обучение» — повышать глубину слоев нейросети. Именно благодаря ему появились самообучаемые нейросети, которые не требуют специальной настройки, а самостоятельно обрабатывают входящую информацию.

Структура нейросети



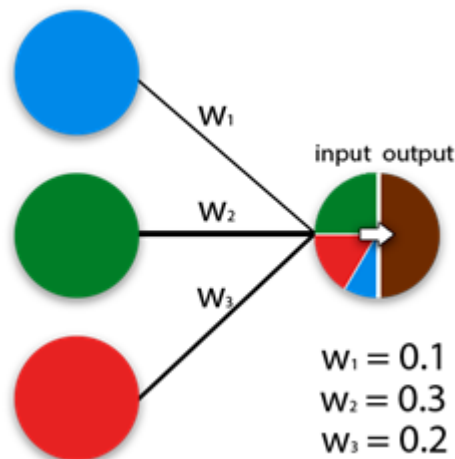
Главное отличие нейросетевых моделей от классических заключается в их структуре. Основные элементы, из которых он состоит – искусственные нейроны и связи между ними.

Нейрон — это вычислительная единица, которая получает информацию, производит над ней простые вычисления и передает ее дальше. Они делятся на три основных типа: входной (синий), скрытый (красный) и выходной (зеленый). В том случае, когда нейросеть состоит из большого количества нейронов, вводят термин слоя. У каждого из нейронов есть 2 основных параметра: входные данные (input data) и выходные данные (output data). В случае входного нейрона: $input=output$. В остальных, в поле input попадает суммарная информация всех нейронов с предыдущего слоя, после чего, она нормализуется, с помощью функции активации (представим ее $f(x)$) и попадает в поле output.



Важно помнить, что нейроны оперируют числами в диапазоне $[0,1]$ или $[-1,1]$. Числа, которые выходят из данного диапазона необходимо обрабатывать разделив 1 на это число. Этот процесс называется нормализацией, и он очень часто используется в нейронных сетях.

Синапс – это связь между двумя нейронами. У синапсов есть 1 параметр — вес. Благодаря ему, входная информация изменяется, когда передается от одного нейрона к другому. Допустим, есть 3 нейрона, которые передают информацию следующему. Тогда у нас есть 3 веса, соответствующие каждому из этих нейронов. У того нейрона, у которого вес будет больше, та информация и будет доминирующей в следующем нейроне (пример — смешение цветов).



На самом деле, совокупность весов нейронной сети или матрица весов — это своеобразный мозг всей системы. Именно благодаря этим весам, входная информация обрабатывается и превращается в результат.

Важно помнить, что во время инициализации нейронной сети, веса расставляются в случайном порядке.

Нейронов в нейросети много, поэтому они объединяются в **слои**:

- Входной, куда поступают данные. Они могут иметь любой формат – файлы, тексты, музыка, картинки, видео и другие.
- Скрытые, в которых производятся вычисления и обработка. Обычно скрытых слоев не больше трех.
- Выходной – отсюда выходят результаты.

Глобально нет разницы между искусственным интеллектом (ИИ) и нейросетями.

Нейросеть — это компьютерная система, которая имитирует работу нейронов в мозге человека. Она состоит из множества «нейронов», соединённых между собой и передающих информацию по цепочке. Нейросети используются во многих сферах для решения различных задач, в том числе для распознавания образов, обработки речи и прочего.

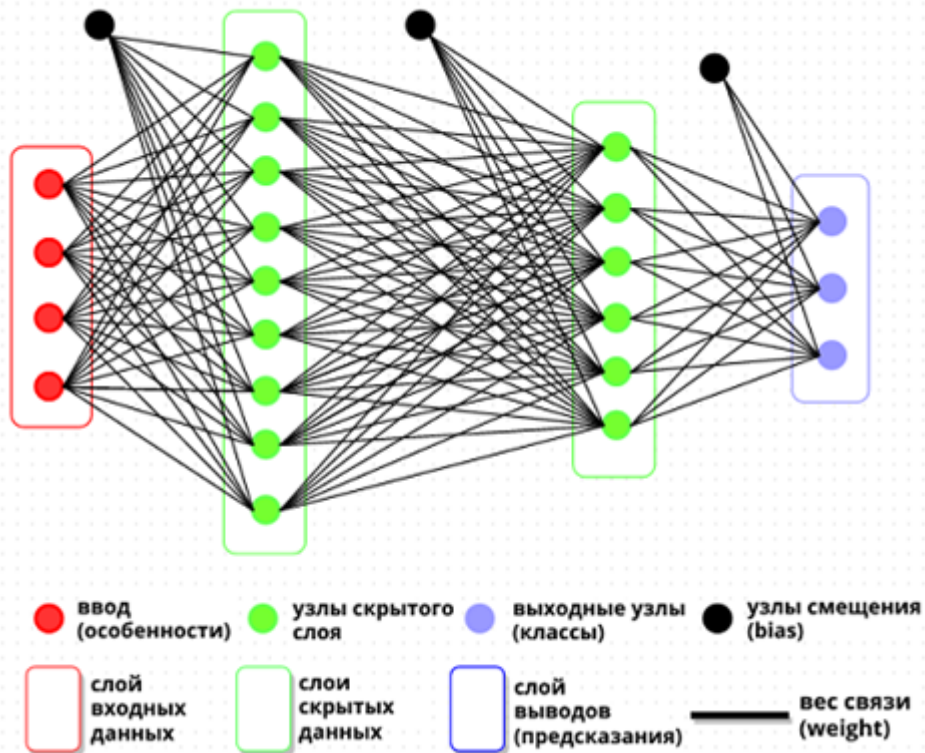
Искусственный интеллект — понятие более широкое. Оно включает в себя не только нейронные сети, но и другие методы обработки информации, в том числе экспертные и логические программы. Нейронные сети — один из видов искусственного интеллекта. Их отличительная особенность — обучение и адаптация в основе алгоритмов.

Искусственная нейронная сеть (ИНС) представляет собой систему соединённых и взаимодействующих между собой простых процессоров (искусственных нейронов). Такие процессоры обычно довольно просты (особенно в сравнении с процессорами, используемыми в персональных компьютерах). Каждый процессор подобной сети имеет дело только с сигналами, которые он периодически получает, и сигналами, которые он периодически посылает другим процессорам. И, тем не менее, будучи соединёнными в достаточно большую сеть с управляемым взаимодействием, такие по отдельности простые процессоры вместе способны выполнять довольно сложные задачи.

- С точки зрения машинного обучения, нейронная сеть представляет собой частный случай методов распознавания образов, дискриминантного анализа;
- С точки зрения математики, обучение нейронных сетей — это многопараметрическая задача нелинейной оптимизации;
- С точки зрения кибернетики, нейронная сеть используется в задачах адаптивного управления и как алгоритмы для робототехники;
- С точки зрения развития вычислительной техники и программирования, нейронная сеть — способ решения проблемы эффективного параллелизма;
- С точки зрения искусственного интеллекта, ИНС является основой философского течения коннекционизма и основным направлением в структурном подходе по изучению возможности построения (моделирования) естественного интеллекта с помощью компьютерных алгоритмов.

Принцип работы

Чем большее число слоев в нейронной сети, тем сложнее задачи, с которыми она может справляться.



- Входной слой нейронов воспринимает информацию. Это могут быть фото, видео, аудио, текстовые файлы — данные в любом формате и объёме.
- На скрытом слое происходит обработка и перевод данных в математические числовые коды. Количество скрытых слоёв не ограничено и зависит от объёма данных и поставленных задач, чаще всего их три.
- Ответ сети формируется в выходном слое. Формат ответа также может быть любым.

На входной слой поступает запрос и данные, которые необходимо обработать. На скрытом слое происходит непосредственно работа: сортировка, отбор по конкретному признаку и прочее. На выходном слое нейросеть выдаёт итог проделанной работы.

Например, для обучения и генерации конечного результата в виде изображения, сеть перерабатывает огромное количество текстовых данных и изображений. Это позволяет ей создавать красивые картинки на основе заданных параметров. Вот в чём состоит принцип действия:

1. Ввод запроса: пользователь вводит текст, который нейросети нужно преобразовать в изображение. Текст может быть любым: описание объекта, сцена, даже стихотворение.

2.Токенизация: нейросеть разбивает введённый текст на отдельные слова или фразы — токены. Каждый представляет собой часть информации, которую нейросеть может обрабатывать.

3.Представление токенов в числовом виде: сеть преобразует информацию в числовой формат. Этот процесс называется векторизацией. Она позволяет нейронной сети работать с токенами в скрытом слое.

4.Обработка токенов нейросетью: в зависимости от сложности задачи работа происходит на разных слоях. В результате многослойной обработки нейросеть формирует промежуточное представление токенов.

5.Генерация изображения: промежуточные токены преобразуются в изображение — подвергаются декодированию.

6.Вывод изображения: пользователь получает изображение, которое соответствует введённому тексту.

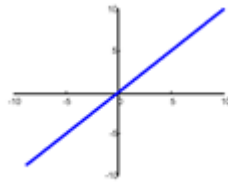
Чем точнее и подробнее запрос, тем быстрее и качественнее получится результат.

Функции нейросети

Функция активации — это способ нормализации входных данных. То есть, если на входе у вас будет большое число, пропустив его через функцию активации, вы получите выход в нужном вам диапазоне. Функций активации достаточно много поэтому мы рассмотрим самые основные: Линейная, Сигмоид (Логистическая) и Гиперболический тангенс. Главные их отличия — это диапазон значений.

Линейная функция

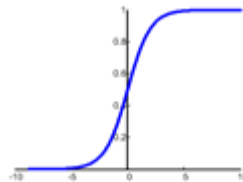
$$f(x) = x$$



Эта функция почти никогда не используется, за исключением случаев, когда нужно протестировать нейронную сеть или передать значение без преобразований.

Сигмоид

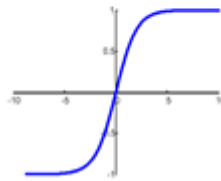
$$f(x) = \frac{1}{1+e^{-x}}$$



Это самая распространенная функция активации, ее диапазон значений $[0,1]$. Именно на ней показано большинство примеров в сети, также ее иногда называют логистической функцией.

Гиперболический тангенс

$$f(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$



Имеет смысл использовать гиперболический тангенс, только тогда, когда ваши значения могут быть и отрицательными, и положительными, так как диапазон функции $[-1,1]$. Использовать эту функцию только с положительными значениями нецелесообразно так как это значительно ухудшит результаты вашей нейросети.

Тренировочный сет

Тренировочный сет — это последовательность данных, которыми оперирует нейронная сеть.

Итерация

Это своеобразный счетчик, который увеличивается каждый раз, когда нейронная сеть проходит один тренировочный сет. Другими словами, это общее количество тренировочных сетов, пройденных нейронной сетью.

Эпоха

При инициализации нейронной сети эта величина устанавливается в 0 и имеет потолок, задаваемый вручную. Чем больше эпоха, тем лучше натренирована сеть и соответственно, ее результат.

✓ `for (int i=0;i<maxEpoch;i++)
for (int j=0;j<trainSet;j++)`

✗ `for (int j=0;j<trainSet;j++)
for (int i=0;i<maxEpoch;i++)`

Важно не путать итерацию с эпохой и понимать последовательность их инкремента. Сначала n раз увеличивается итерация, а потом уже эпоха и никак не наоборот. Другими словами, нельзя сначала тренировать нейросеть только на одном сете, потом на другом и тд. Нужно тренировать каждый сет один раз за эпоху. Так, вы сможете избежать ошибок в вычислениях.

Ошибка

Ошибка — это процентная величина, отражающая расхождение между ожидаемым и полученным ответами. Ошибка формируется каждую эпоху и

должна идти на спад. Если этого не происходит, значит, вы что-то делаете не так. Ошибку можно вычислить разными путями, но мы рассмотрим лишь три основных способа: Mean Squared Error (далее MSE), Root MSE и Arctan. Каждый метод считает ошибки по-разному. У Arctan, ошибка, почти всегда, будет больше, так как он работает по принципу: чем больше разница, тем больше ошибка. У Root MSE будет наименьшая ошибка, поэтому, чаще всего, используют MSE, которая сохраняет баланс в вычислении ошибки.

MSE:

$$\frac{(i_1-a_1)^2+(i_2-a_2)^2+\dots+(i_n-a_n)^2}{n}$$

Root MSE:

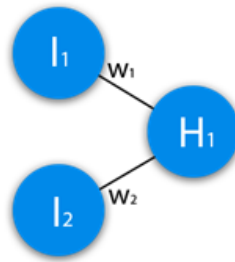
$$\sqrt{\frac{(i_1-a_1)^2+(i_2-a_2)^2+\dots+(i_n-a_n)^2}{n}}$$

Arctan:

$$\frac{\arctan^2(i_1-a_1)+\dots+\arctan^2(i_n-a_n)}{n}$$

Принцип подсчета ошибки во всех случаях одинаков. За каждый сет, мы считаем ошибку, отняв от идеального ответа, полученный. Далее, либо возводим в квадрат, либо вычисляем квадратный тангенс из этой разности, после чего полученное число делим на количество сетов.

Задача



$$1) H_{1input} = (I_1 * W_1) + (I_2 * W_2)$$

$$2) H_{1output} = f_{activation}(H_{1input})$$

В данном примере изображена часть нейронной сети, где буквами I обозначены входные нейроны, буквой H — скрытый нейрон, а буквой W — веса. Из формулы видно, что входная информация — это сумма всех входных данных, умноженных на соответствующие им веса.

Зададим на вход 1 и 0. Пусть $W_1=0.4$ и $W_2=0.7$

Входные данные нейрона H_1 будут следующими: $1*0.4+0*0.7=0.4$.

Теперь, когда у нас есть входные данные, мы можем получить выходные данные, подставив входное значение в функцию активации.

Теперь, когда у нас есть выходные данные, мы передаем их дальше. И так, мы повторяем для всех слоев, пока не дойдем до выходного нейрона. Запустив такую сеть в первый раз, мы увидим, что ответ далек от правильно, потому что сеть не натренирована. Чтобы улучшить результаты мы будем ее тренировать.

Эпоха увеличивается каждый раз, когда мы проходим весь набор тренировочных сетов, в нашем случае, 4 сетов или 4 итераций.

Теперь, чтобы проверить себя, подсчитайте результат, данной нейронной сети, используя сигмоид, и ее ошибку, используя MSE.

Данные: $I_1=1$, $I_2=0$, $W_1=0.45$, $W_2=0.78$, $W_3=-0.12$, $W_4=0.13$, $W_5=1.5$, $W_6=-2.3$.

Решение:

$$H_{1input} = 1*0.45+0*-0.12=0.45$$

$$H_{1output} = \text{sigmoid}(0.45)=0.61$$

$$H_{2input} = 1*0.78+0*0.13=0.78$$

$$H_{2output} = \text{sigmoid}(0.78)=0.69$$

$$O_{1input} = 0.61*1.5+0.69*-2.3=-0.672$$

$$O_{1output} = \text{sigmoid}(-0.672)=0.33$$

$$O_{1ideal} = 1 \text{ (0xor1=1)}$$

$$\text{Error} = ((1-0.33)^2)/1=0.45$$

Результат — 0.33, ошибка — 45%.

Методы обучения

Один из главных признаков нейросетей – способность к обучению. Перед началом обучения все веса нейронной сети определяются случайными значениями. Обучающие данные передаются на входной слой, проходят через следующие слои и достигают выходного. В процессе обучения данные постоянно подвергаются корректировке, и циклы повторяются до тех пор, пока данные обучения не станут показывать одинаковые результаты.

По сути, любая модель машинного обучения использует метод градиентного спуска. Он применяется и для обучения нейросетей и называется методом обратного распространения ошибки.

Существуют следующие методы обучения:

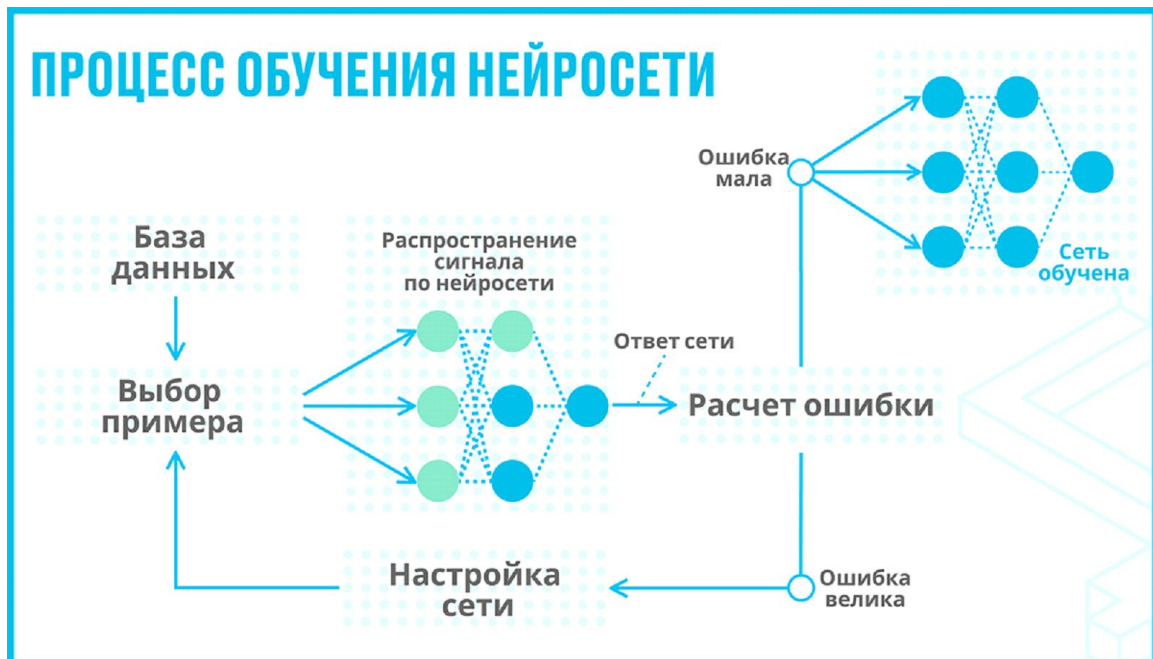


•**С учителем.** Пользователь дает сигнал на вход, получает на выходе ответ нейросети, затем сравнивает его с уже известным правильным. После этого с помощью специальных алгоритмов меняются веса связей и снова задается входной сигнал. Процесс продолжается до тех пор, пока нейросеть не начнет отвечать точно. Такое обучение называют также контролируемым.

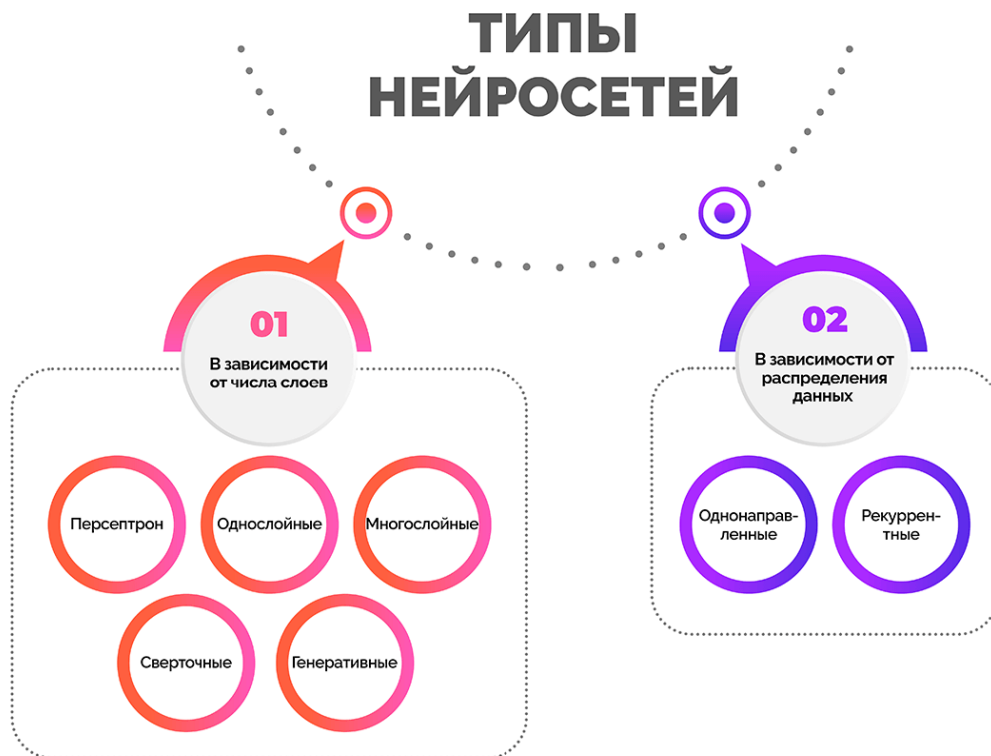
•**Без учителя.** Метод применяют, если нет правильных ответов на входные сигналы. Сеть в этом случае, используя собственную память, делит объекты на классы, то есть начинает кластеризацию. Эталонные ответы при этом не показаны. Данный тип обучения называют глубоким: система все время обучается сама.

•**С подкреплением.** Такие нейросети обучаются самостоятельно, но при этом взаимодействуют с окружающей средой, которая специально моделируется и становится обучающей. Чаще всего такой подход применяют в робототехнике и разработке игр.

В зависимости от типа входной информации выделяют аналоговые, двоичные и образные нейросети.



Типы нейросетей



В зависимости от числа слоев, в которых расположены нейроны, нейросети могут быть:

- **Перцептрон** – самая старая форма. Один нейрон принимает информацию, применяет активацию, в результате становится доступным вывод в двоичной системе. Перцептрон можно использовать только для классификации данных на две группы. Из-за ограниченных возможностей такие нейронные сети в наше время практически не используются.

- **Однослойные.** Сигнал поступает во входной слой и сразу же отправляется к выходному, где происходят вычисления. Связь между нейронами входного и выходного слоев обеспечивают синапсы.

- **Многослойные.** Помимо входного и выходного слоев, в таких нейронных сетях есть еще несколько скрытых промежуточных. Обработка информации и вычисления производятся на нескольких этапах, поэтому решения, предлагаемые такими сетями, более точные.

- **Сверточные.** В структуру таких нейросетей входят два дополнительных слоя - сверточные и объединяющие. Сверточные нейронные сети используются для обработки изображений, картинок и фото.

- **Генеративные.** В эту группу входят нейросети, способные что-то создавать. Это, к примеру, генераторы картинок или текстов.

Еще одна классификация делит нейросети на однонаправленные и рекуррентные в зависимости от распределения данных по синапсам:

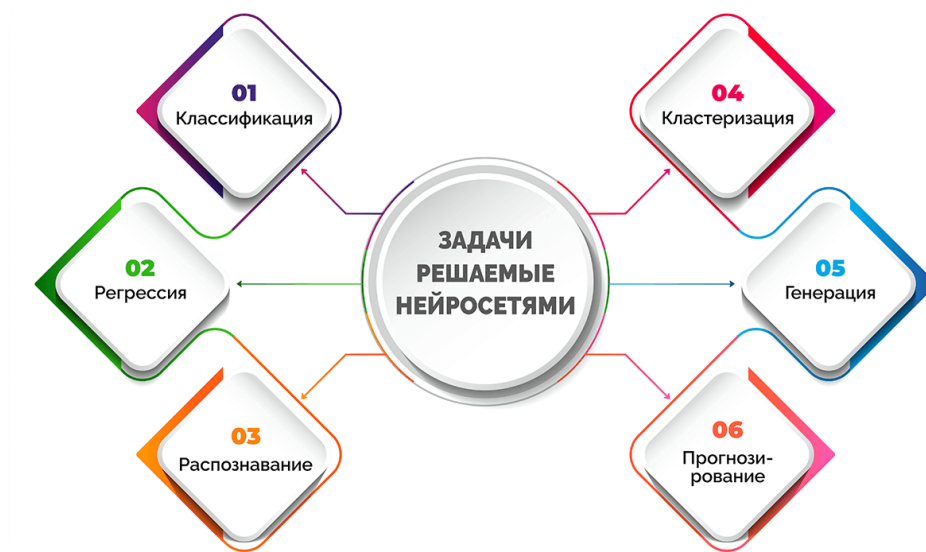
- **Однонаправленные** (прямого распространения). Сигнал движется от входного слоя к выходному, обратного движения нет. Нейросети такого типа используют для распознавания речи, кластеризации, составления прогнозов.

- **Реккурентные** (с обратными связями). Реккурентные нейронные сети предполагают, что любое количество сигналов может перемещаться в разных направлениях, в том числе от выхода к входу.

По типам нейронов сети могут быть однородными или гибридными. Первые состоят из нейронов одного типа, вторые сочетают несколько классов нейронов. По характеру настройки синапсов нейронные сети бывают с фиксированными либо с динамическими связями.

Применения нейросетей

Разные варианты нейросетей создаются для решения нескольких типов различных задач:



- Классификация – отнесение объектов к нужному классу.
- Регрессия – предсказывание результата в виде чисел (например, стоимости дома в зависимости от его площади и района, в котором он расположен).
- Распознавание – выделение объекта среди огромного множества других похожих (пример - сеть может выделить конкретное лицо в толпе).
- Кластеризация – разделение объектов на несколько групп по какому-либо признаку, неизвестному ранее. Это, например, разбивка документов на разные классы.
- Генерация – рождение чего-то нового в рамках заданной тематики.
- Прогнозирование – на основе полученных данных искусственный интеллект формулирует прогнозы по заданной теме на определенное время.

В зависимости от задачи, которую могут решать искусственные нейронные сети (она у каждого своя), они используются в разных областях. Перечислим сферы, где они наиболее востребованы:

1. Медицина. Искусственный интеллект помогает обрабатывать снимки и другие данные исследований и тем самым позволяет врачам устанавливать точный диагноз, при этом тратить меньше времени.

2. Образование. Преподаватели с помощью искусственных сетей имеют возможность быстрее проверять домашние задания, за короткое время составлять сложные презентации и планы уроков.

3. Искусство. Нейросети создают изображения, произведения литературы и музыку.

4. Строительство и архитектура. Искусственный интеллект полезен застройщикам, чтобы выбрать материалы, прогнозировать время выполнения работ.

5.Безопасность. Нейросети имеют возможность распознавать обычные лица и путем слежки в общественных местах вычислять преступников, которые находятся в розыске.

6.Банковская сфера. Нейронная сеть анализирует кредитную историю клиентов, создает прогнозы биржевых индексов.

7.Производство. Искусственный интеллект участвует в отслеживании производственных процессов, дают возможность контролировать продукции на предприятиях.

Применение:

Генерация и обработка изображений. Нейронные сети из этой категории рисуют на основе текста и пользовательских изображений с любым указанным стиле, в том числе используя вектор. Сервисы могут изменять фон картинки, дорисовывать изображения по описанию, генерировать картинку на основе фотографий, создавать визуальный контент для брендов и логотипы, а также реалистичные изображения в дополнение к текстовому описанию карточек товаров в интернет-магазинах и на маркетплейсов, фотографии для социальных сетей.

Генерация игровых миров и персонажей. Нейронные сети, создающие персонажей для игр, уровни, анимацию, видео, изображения для интерфейса. Упрощают разработку сюжетных линий и хода игры.

Работа с аудио. Нейронные сети могут просто преобразовать аудио в текст и обратно, расшифровывать в форме текста записи конференций, интервью и лекций. Используются для озвучивания роликов и прочего видеоконтента, для улучшения качества аудиозаписей и избавления их от шумов и посторонних звуков, для генерации музыки. Сервисы поддерживают несколько языков, включая русский. Многие подобные сети разработаны на основе языковой модели ChatGPT.

Музыка. Нейронные сети с ИИ могут создать музыку в разных стилях с нуля или обрабатывать и аранжировать мелодии.

Видео. Нейросети создают видео ролики с персонажами с возможностями настройки голоса и стиля речи. Источниками для видео роликов могут быть собственные сценарии или контент сайтов, соцсетей, приложений. Некоторые инструменты могут также создавать GIF-анимацию, озвучивать тексты, накладывать на видео фоновую музыку и даже делать фильмы.

Написание кода. Нейронные сети ускоряют разработку кода на разных языках программирования. Могут находить ошибки в уже написанных кодах, генерировать коды по текстовому запросу, создавать тесты.

Создание документов и презентаций. Умеют по запросу генерировать любой контент, структурировать информацию и разбивать ее по слайдам, добавлять диаграммы. Сеть генерирует изображения, обрабатывает фотографии и прочие визуальные элементы.

Преимущества и недостатки нейросетей

Преимущества нейросетей:

- незаменимы в сфере автоматизации процессов;
- спасают там, где может навредить человеческий фактор;
- экономят время на выполнении рутинных задач;
- постоянно обучаются.

Недостатки:

- напрямую зависят от вводимых данных, поэтому сильно подвержены влиянию;
- чтобы получить действительно хорошую рабочую сеть, нужно потратить много времени на её обучение;
- занимают много места на сервере и требуют больших вычислительных мощностей.

Учитывая то, с какой скоростью развивается искусственный интеллект сегодня, плюсы и минусы нейросетей достаточно относительно. Но их нельзя игнорировать.

Установка и настройка сервера

Перед установкой серверов вам необходимо скачать и установить на компьютер следующий список программ: python, anaconda, VSCOD

Данные программы можно скачать по следующим ссылкам:

<https://code.visualstudio.com/docs/?dv=win64user>

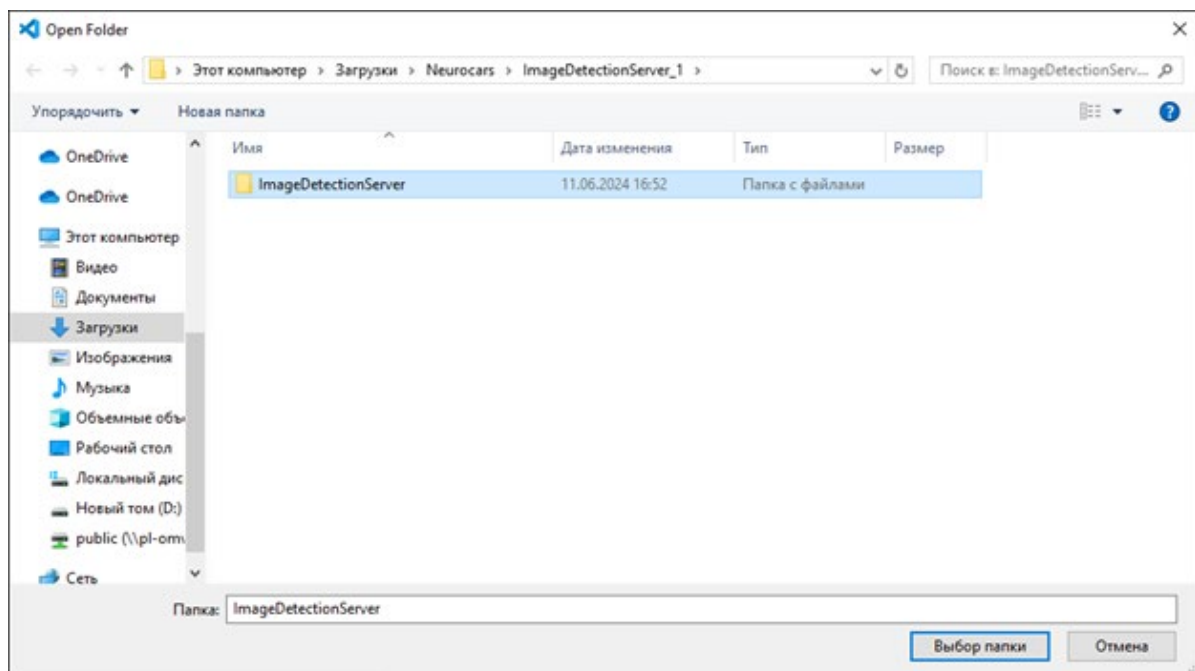
<https://www.python.org/downloads/release/python-3124/>

<https://docs.anaconda.com/miniconda/>

После установки всех программ перезапустите компьютер.

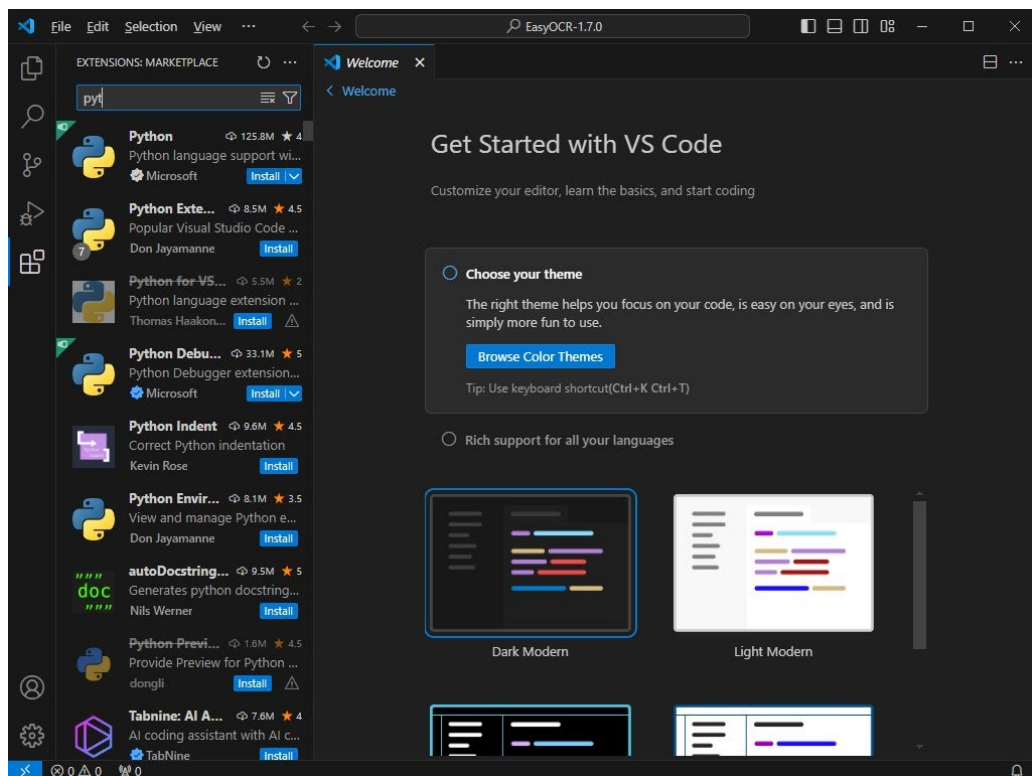
Когда все необходимые программы установлены, запустите Visual studio code, нажмите на вкладку File -> Open folder и выберите путь к папке где располагается необходимый вам сервер (при установке ПО сервер по умолчанию находится в папке проекта:

C:\Users\Public\Documents\ProgramLab\LaboratoryofTrafficManagement
\Base_M).



Открытие папки сервера

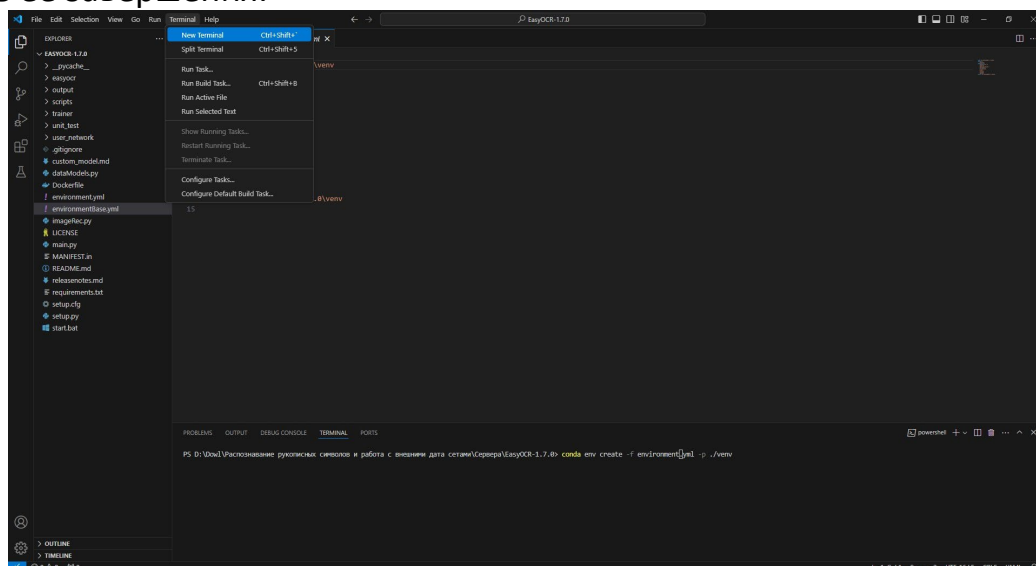
В открытом Visual studio code перейдите во вкладку Extensions и в нем установите Python.



Установка Python

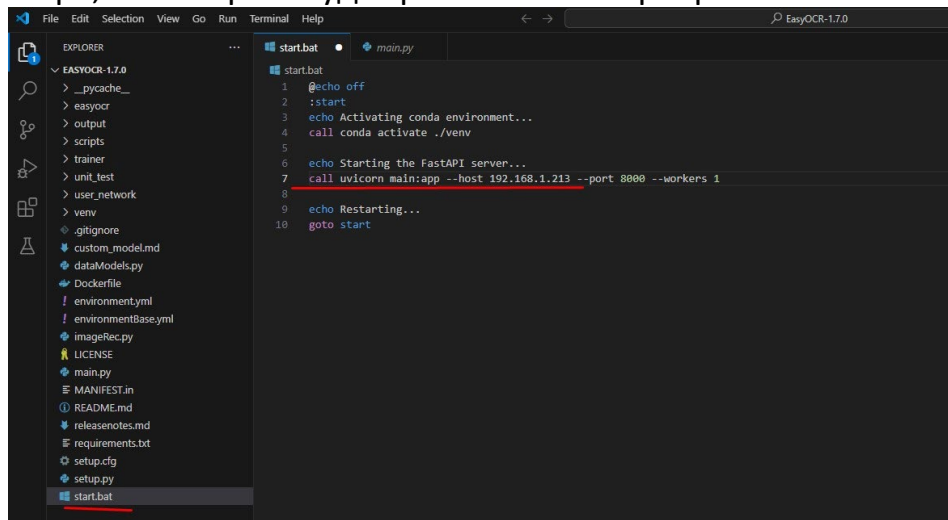
После установки питона нажмите на вкладку Terminal далее нажмите на New terminal.

В появившемся поле введите команду « `conda env create -f «Название файла.yml» -p ./venv` » (Можете использовать один из двух файлов а именно «environmentBase.Yml » или « `environment.Yml` »), после чего начнется установка, дождитесь её завершения.



Установка необходимых пакетов

Вы можете изменить IP адрес сервера, если захотите запустить сервер на отдельном компьютере, для этого зайдите в файл `start` и поменяйте IP адрес на IP адрес компьютера, на котором будет расположен сервер.

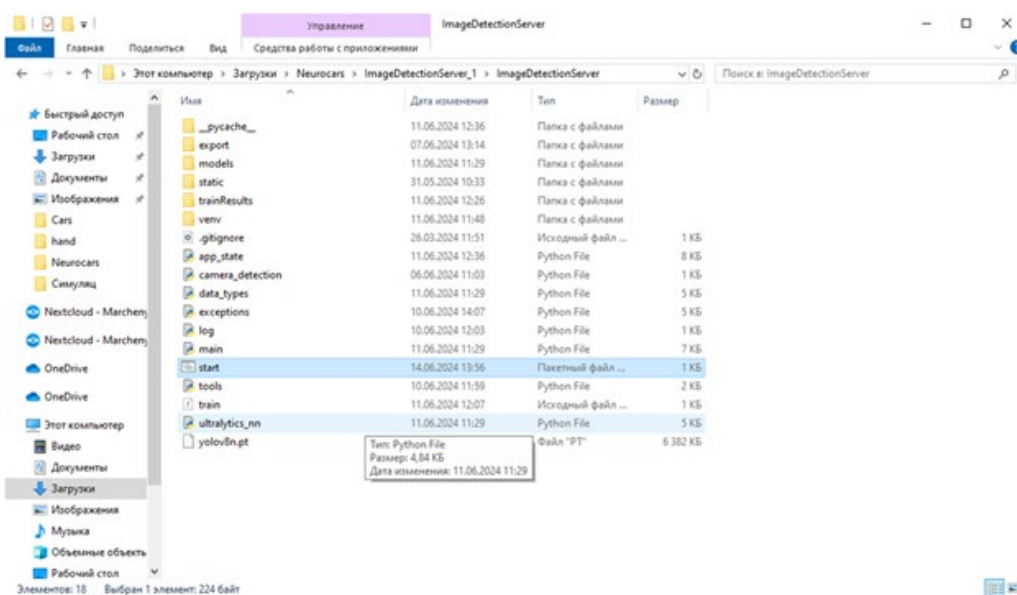


```

1 @echo off
2 :start
3 echo Activating conda environment...
4 call conda activate ./venv
5
6 echo Starting the FastAPI server...
7 call uvicorn main:app --host 192.168.1.213 --port 8000 --workers 1
8
9 echo Restarting...
10 goto start
  
```

Смена IP адреса

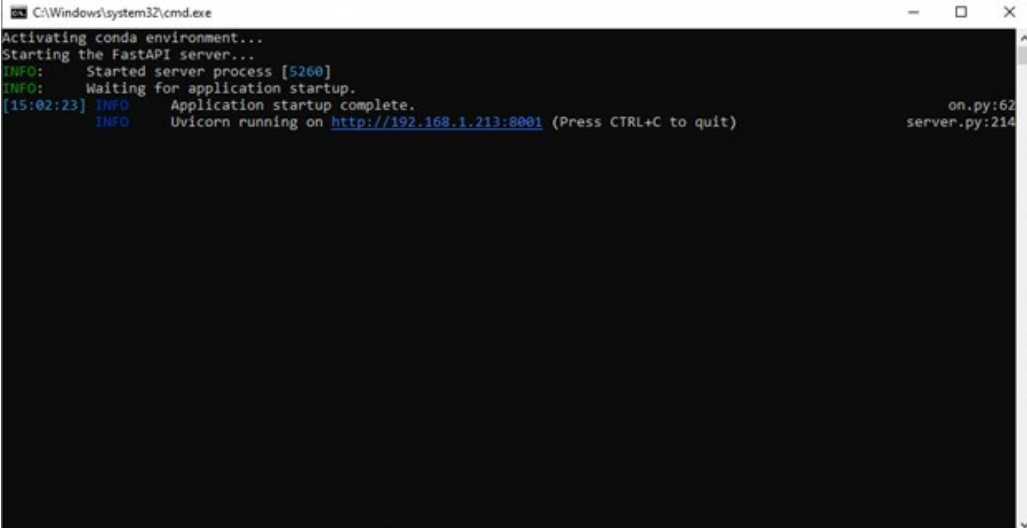
Сохраните изменения, теперь необходимо проверить правильность настроек и работы сервера, для чего перейдите в папку местонахождения сервера и запустите файл `start.bat`.



Если после запуска появляется ошибка о недостающем пакете, то его необходимо установить, для этого вернитесь в Visual studio и в терминале введите команду `conda init`, после чего введите команду `conda activate ./venv`, теперь вам необходимо ввести команду для установки необходимого пакета `conda install <name>`, где `name` это название необходимого вам пакета.

Если данные пакеты устанавливаются, то введите в терминале `pip install <name>`, где `name` название пакета.

Запустите сервер и проверьте, что все работает.



```

C:\Windows\system32\cmd.exe
Activating conda environment...
Starting the FastAPI server...
INFO: Started server process [5260]
INFO: Waiting for application startup.
[15:02:23] INFO Application startup complete.
INFO Uvicorn running on http://192.168.1.213:8001 (Press CTRL+C to quit)
on.py:62
server.py:214

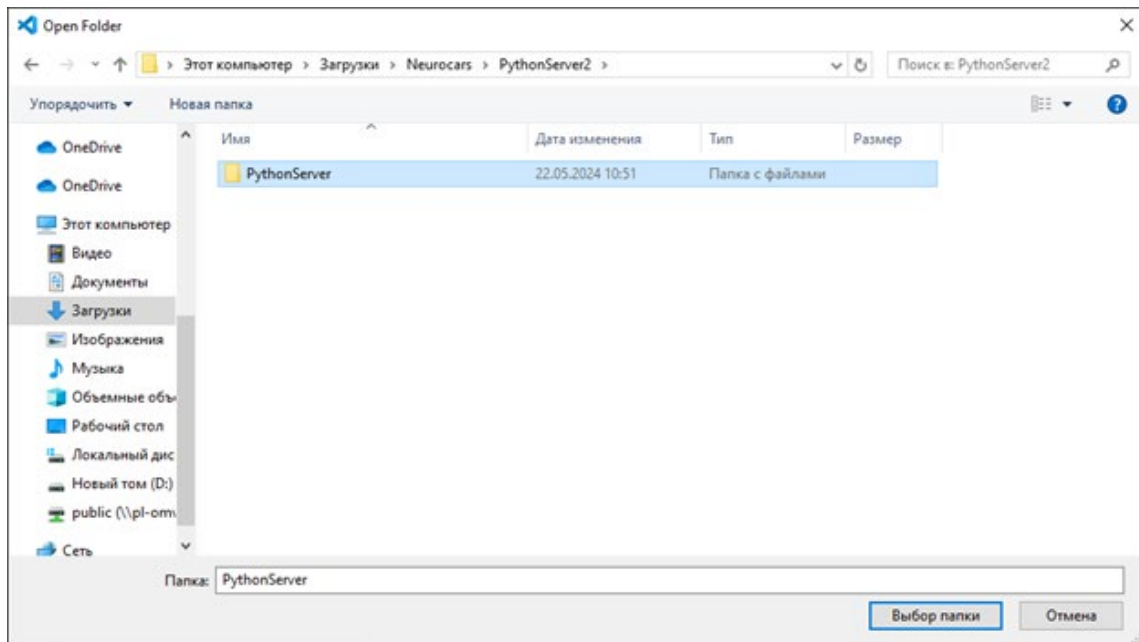
```

Запуск сервера

Установка и настройка сервера распознавания

После установки и настройки сервера распознавания, необходимо установить и настроить сервер лабораторной работы.

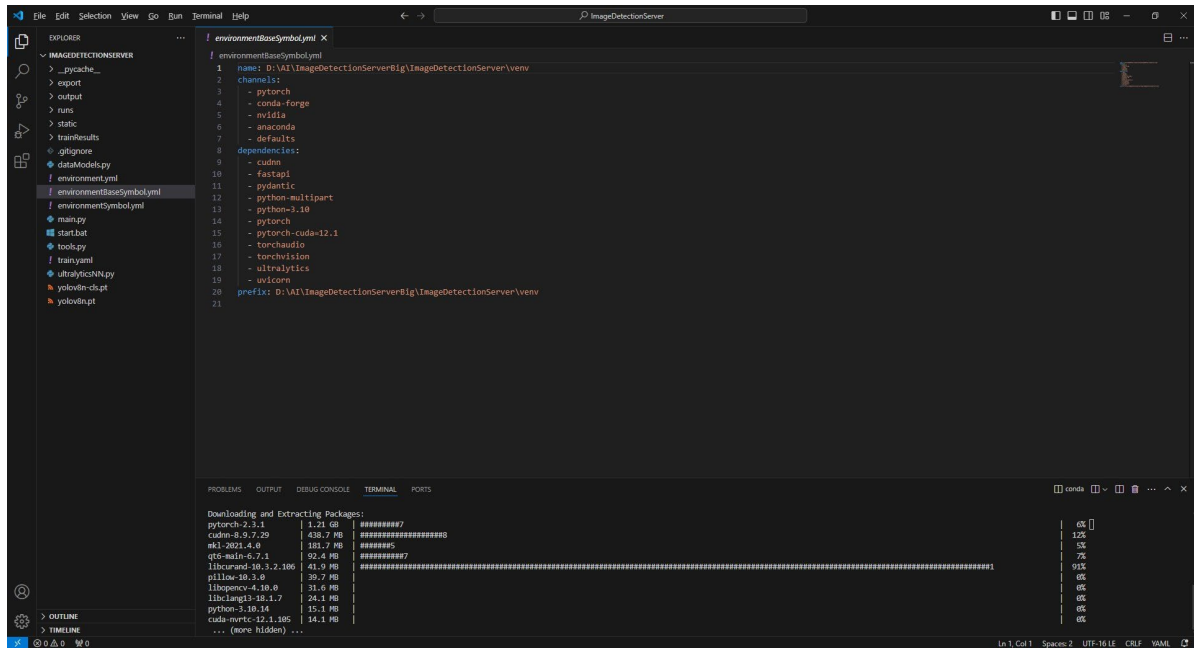
Запустите Visual studio code, нажмите на вкладку File ->Open folder и выберите путь к папке где располагается необходимый вам сервер (при установке ПО сервер по умолчанию находится в папке проекта: C:\Users\Public\Documents\ProgramLab\LaboratoryofTrafficManagement \Base_M).



Открытие папки с сервером

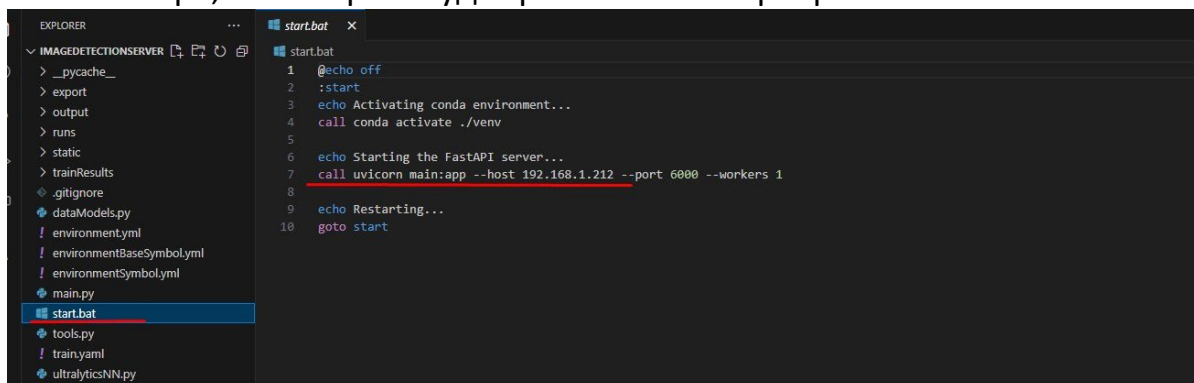
После установки питона нажмите на вкладку Terminal далее нажмите на New terminal.

В появившемся поле введите команду «conda env create -f «Название файла.yml» -p ./venv » (Можете использовать один из двух файлов а именно «environmentBase.Yml » или « environment.Yml »), после чего начнется установка, дождитесь её завершения.



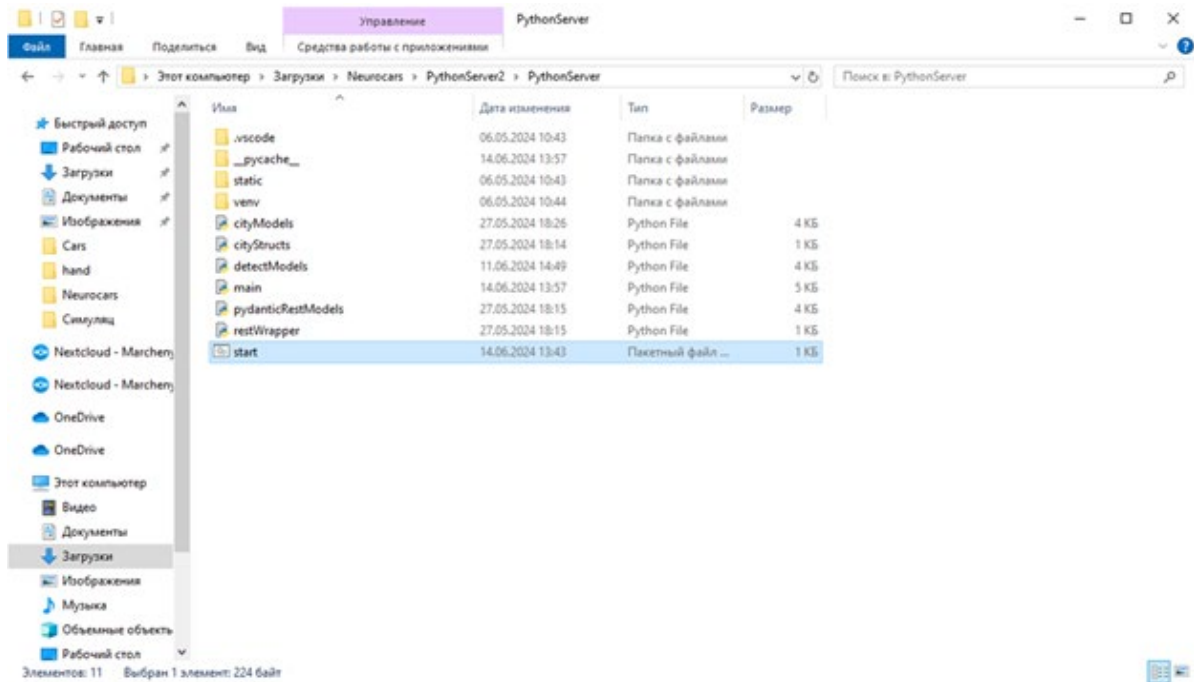
Установка необходимых библиотек

Вы можете изменить IP адрес сервера, если захотите запустить сервер на отдельном компьютере, для этого зайдите в файл start и поменяйте IP адрес на IP адрес компьютера, на котором будет расположен сервер.



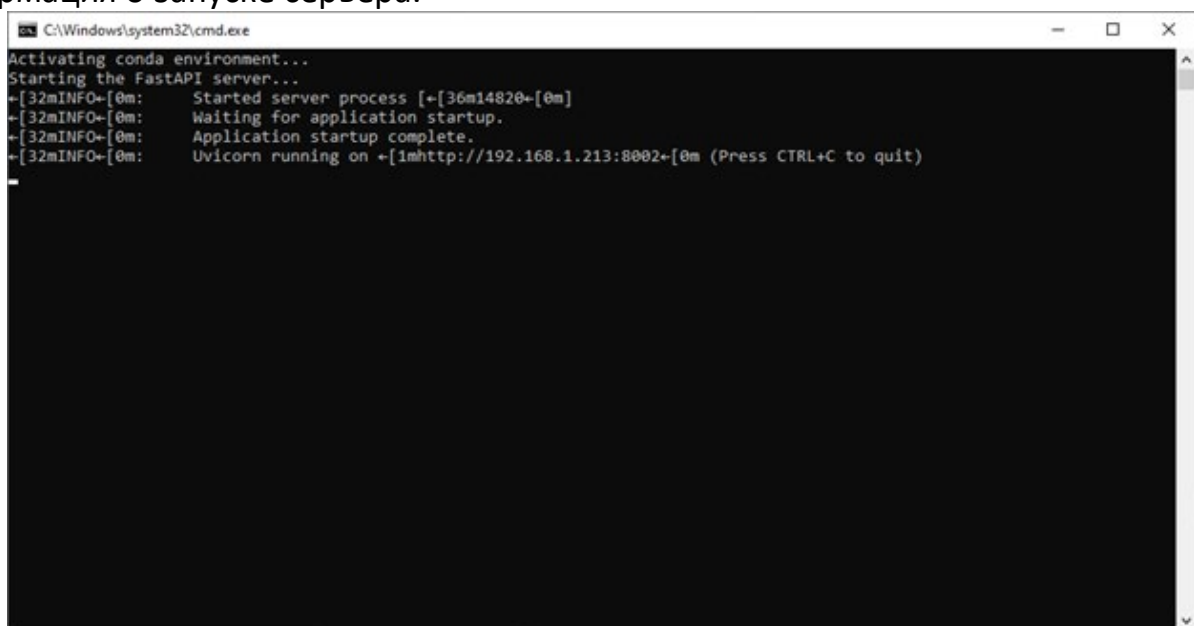
Смена IP адреса

Зайдите в папку сервера и запустите сервер запустив файл start.



Запуск сервера

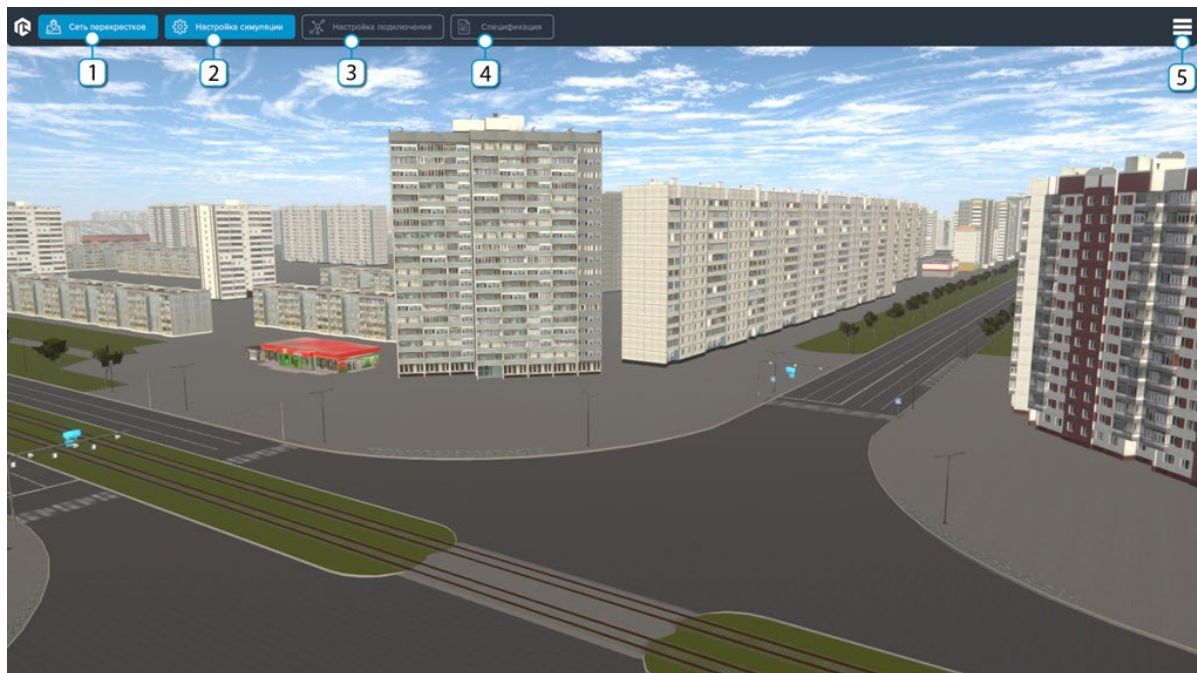
После запуска сервера откроется командная строка, в которой появятся информация о запуске сервера.



Успешный старт сервера

Работа в программе

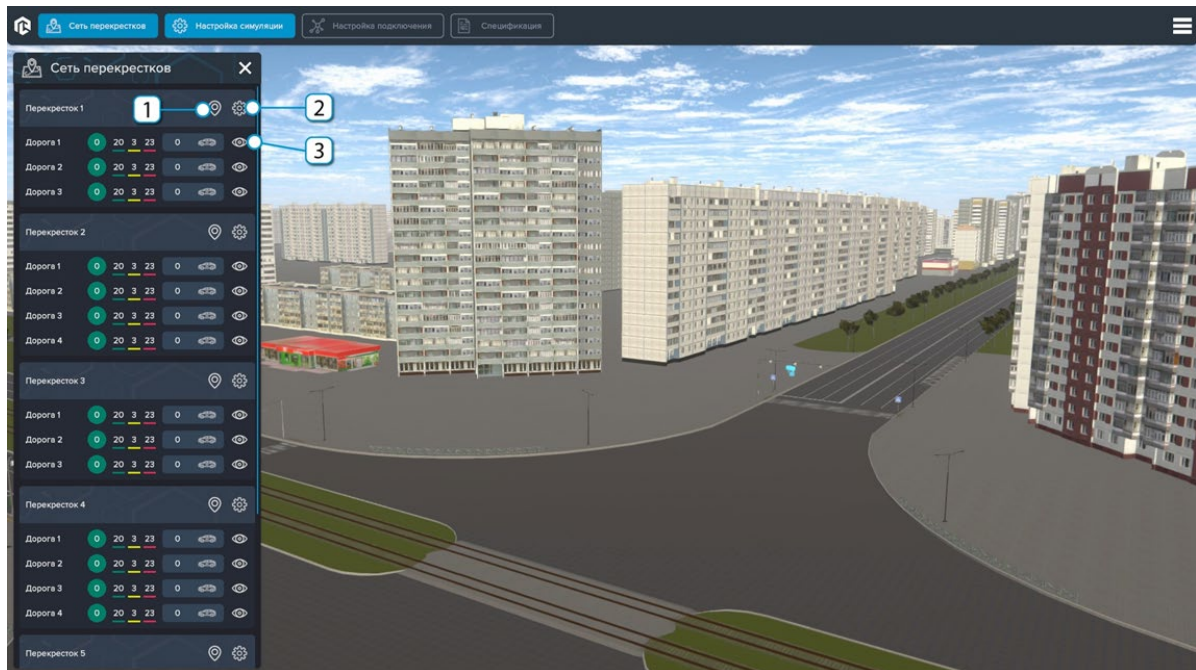
Перед запуском приложения запустите сервер управления дорожным движением. После запуска серверов запустите приложение. При запуске приложения откроется главный экран.



Главный экран

1. Вкладка списка и настроек перекрестков и камер.
2. Вкладка настроек и запуска симуляции автомобильного движения.
3. Кнопка выбора сервера и порта подключения.
4. Кнопка описании спецификации.
5. Кнопка вызова меню.

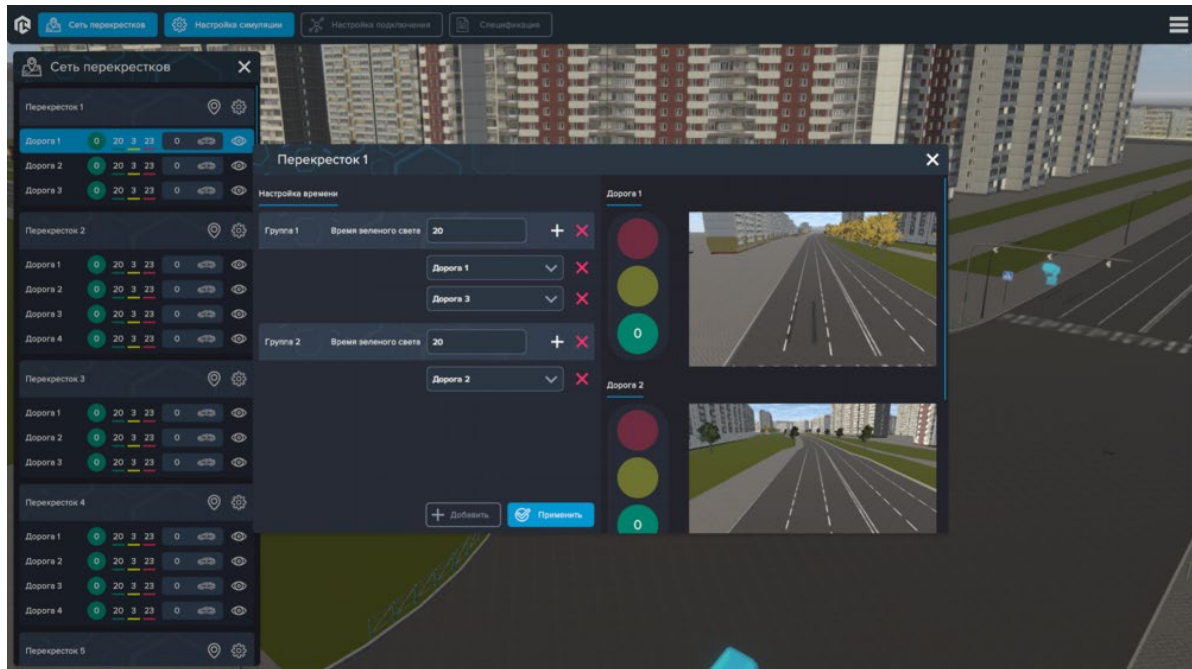
Во вкладке списка перекрестков и камер находится список всех перекрестков и камер, расположенных на данных перекрестках, в данной вкладке так же отображается количество автомобилей определенных нейросетью на каждом из светофоров, какой свет включен сейчас на светофоре, время на светофоре, так же по нажатию можно перейти к местоположению светофора или перекрестка.



Вкладка перекрестков

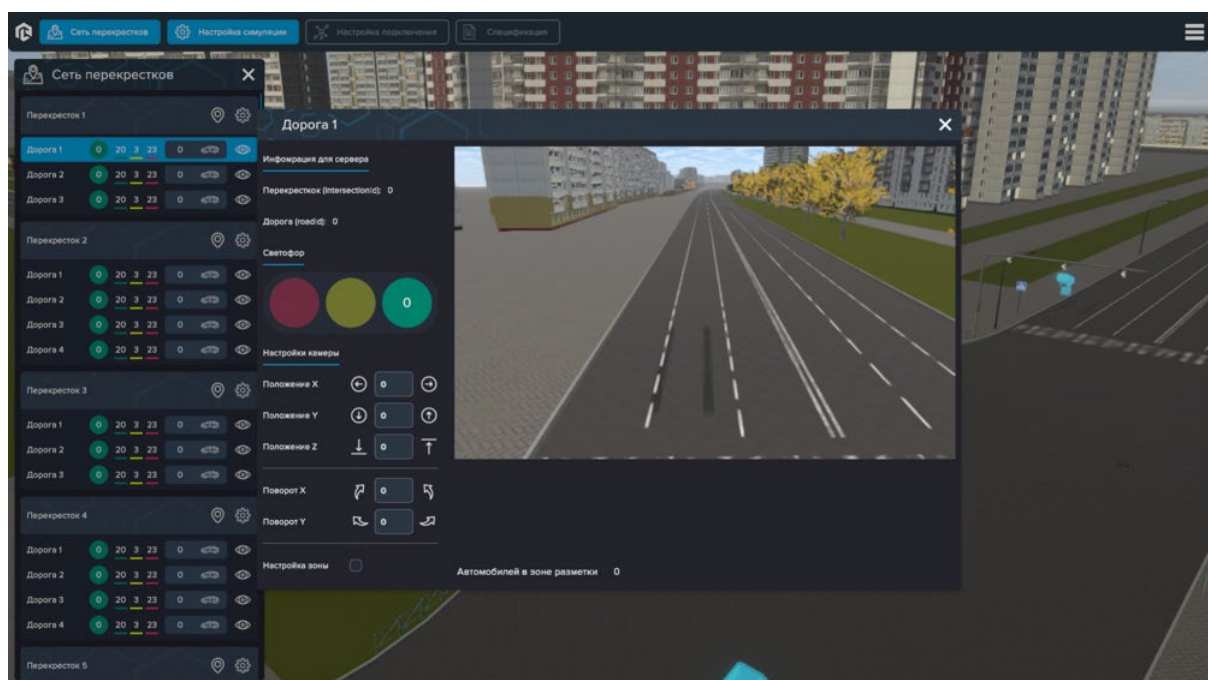
1. Кнопка перехода к местоположению перекрестка.
2. Кнопка открытия окна настроек перекрестка.
3. Кнопка открытия окна настроек светофора.

При нажатии на кнопку открытия окна настроек перекрестка откроется окно настроек перекрестка, в котором можно настроить время горения зеленого света для группы светофоров, изменить группы светофоров, а также можно увидеть в каком сейчас цикле работает каждый из светофоров и увидеть изображение, которое снимает камера на светофоре.



Рисование символов

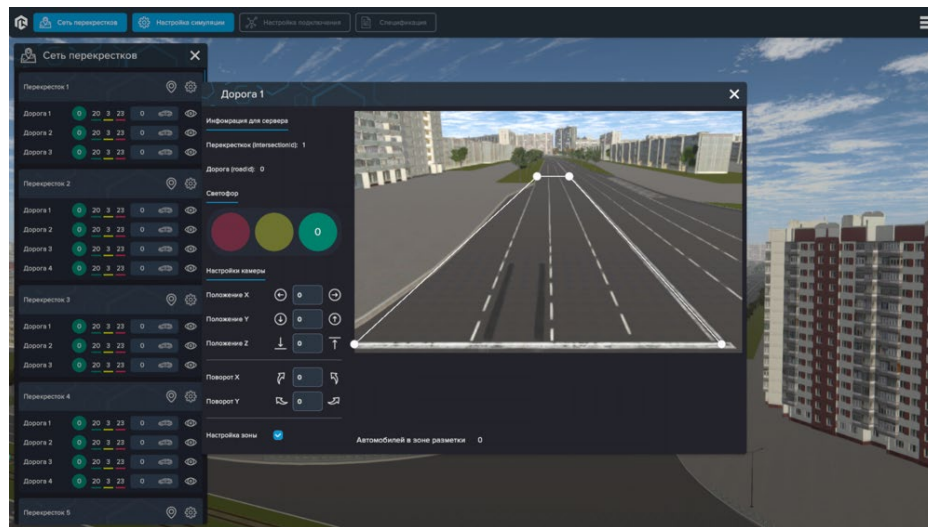
При нажатии на кнопку настроек светофора открывается окно настроек светофора, в котором располагается информация о переменных перекрестка и камеры для сервера, режим работы светофора, настройки камеры, а именно, положение камеры по вертикали, горизонтали, высоте а так же поворот камеры по горизонтали и по вертикали. Кнопка настройка зоны включает отображение и настройку маски для распознавания объектов.



Окно настройки светофора

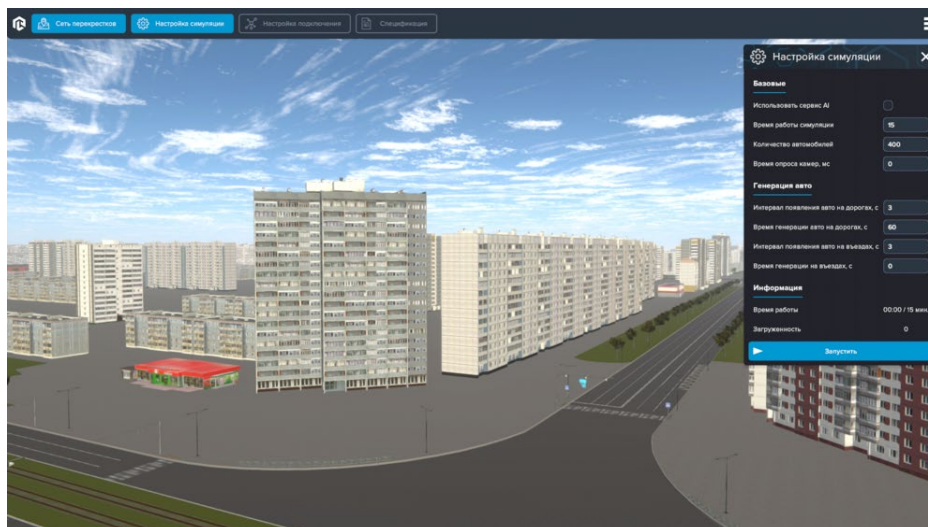
При включении отображения и настройки маски распознавания объектов, появляется настроенная зона распознавания, размер и форму маски можно

перенастроить в данном окне, при изменении настроек положения или поворота камеры необходимо изменить маску распознавания.



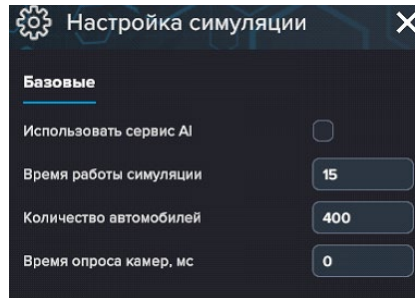
Настройка зоны распознавания

Вкладка настроек симуляции содержит основные настройки симуляции работы виртуального города. Вкладку настроек симуляции можно разделить на 3 основных блока настроек.



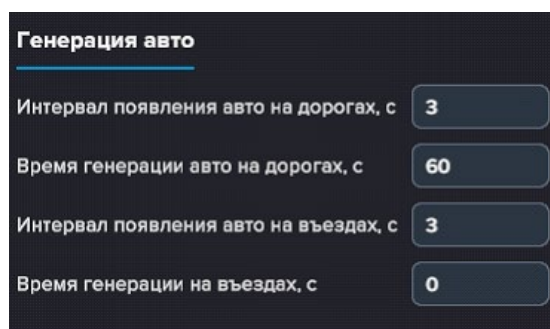
Настройка симуляции

Базовые настройки включают в себя кнопку использовать нейросеть для распознавания, поле настроек времени симуляции, измеряемое в минутах, поле общего количество автомобилей, поле настройки времени опроса камер, данное поле влияет на частоту отправки информации с камер на сервер распознавания.



Базовые настройки

Генерация авто содержат поля настройки интервалов появления автомобилей и времени появления заданного количества автомобилей. Виртуальный город разделен на блоки, где появляются автомобили, автомобили могут появиться в самом городе или на въездах в город.



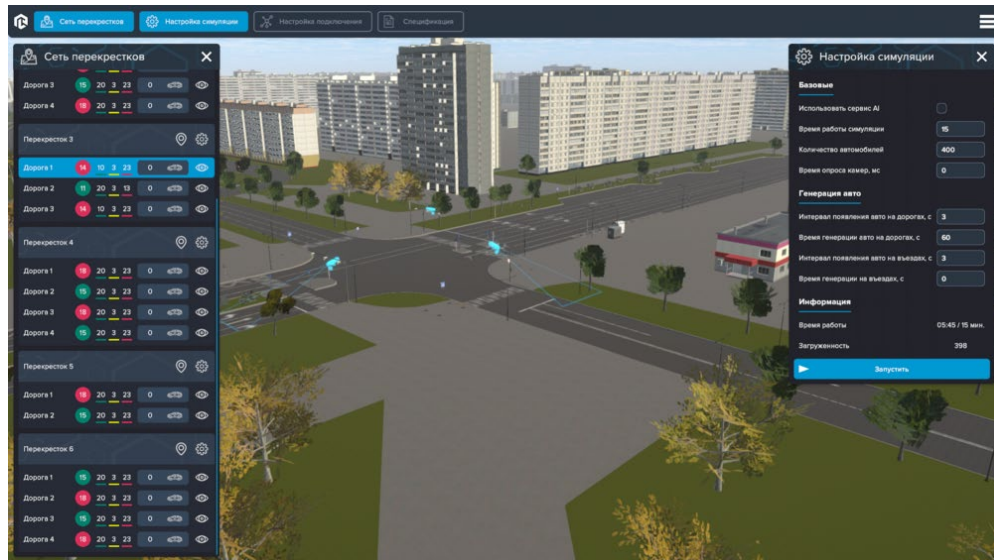
Генерация авто

Информация содержит время работы симуляции и общее количество автомобилей на дорогах.



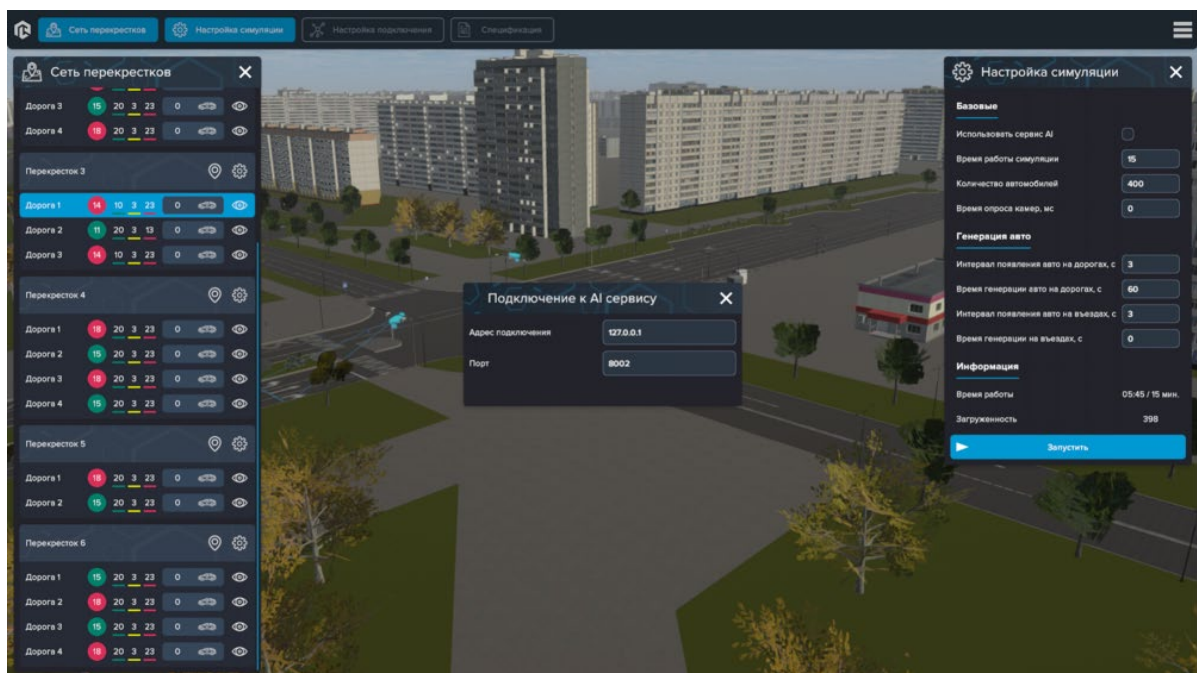
Информация

При нажатии на кнопку запустить симуляция начинает работу, в случае если не подключено к сервисам AI, то время работы светофоров не будет меняться в автоматическом режиме, также не будет подсчитываться количество автомобилей на светофорах.



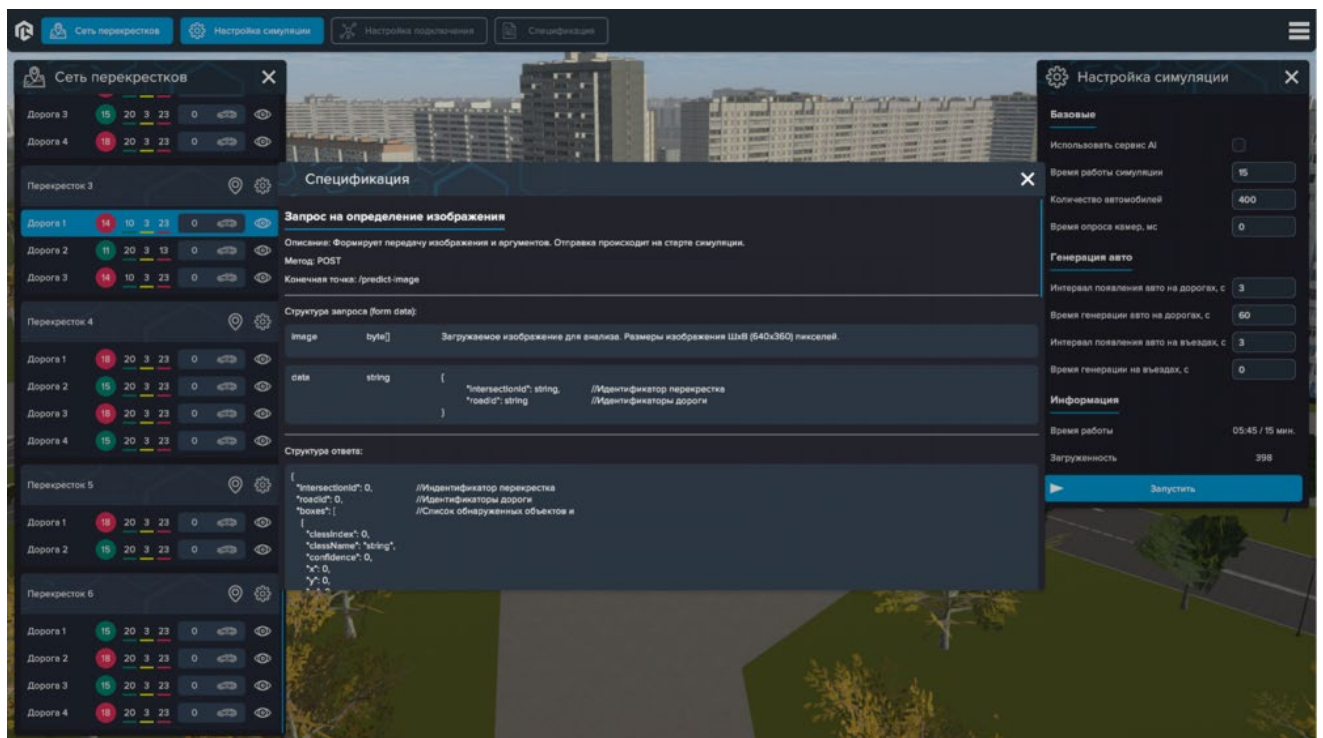
Запуск симуляции без сервисов AI

Для подключения к серверу распознавания нажмите на кнопку Настройка подключения, в открытом окне введите адрес сервера и порт подключения.



Настройка подключения

При нажатии на кнопку спецификация откроется окно с спецификацией отправляемых и получаемых запросов.



Спецификация

При включении симуляции с сервисами AI нейросеть будет определять загруженность светофоров и управлять временем работы светофоров а также выводить информацию о загруженности во вкладку сети перекрестков.



Симуляция с AI сервисами

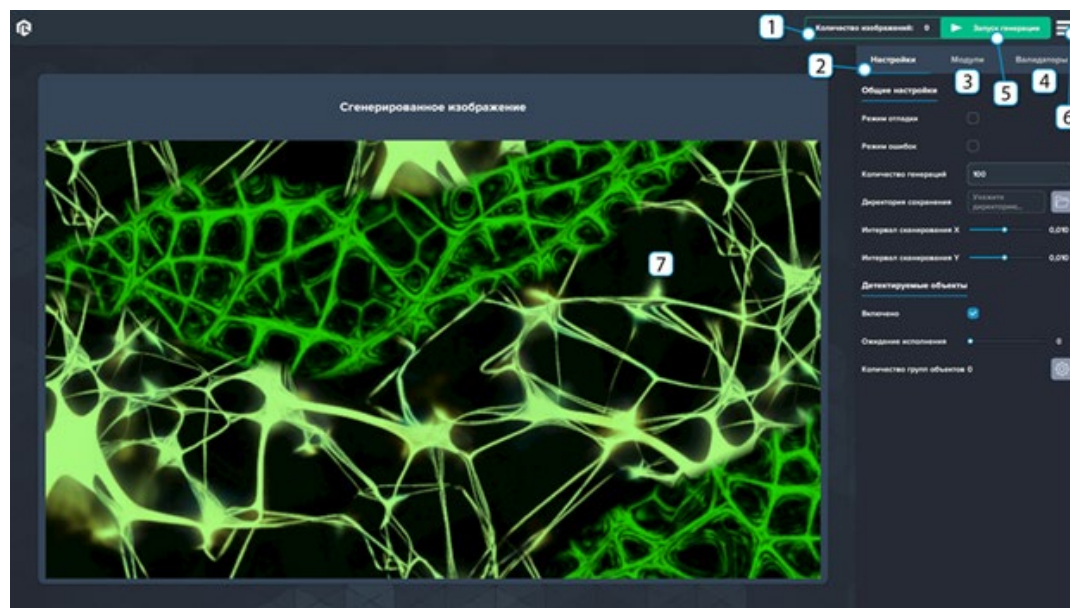
Подробнее с нагрузкой светофора можно ознакомиться в окне настроек светофора, в окне настроек светофора можно увидеть, как нейросеть определяет автомобили и к каким группам они относятся.



Группы автомобилей

Создание датасета

После запуска данного режима откроется следующее окно:

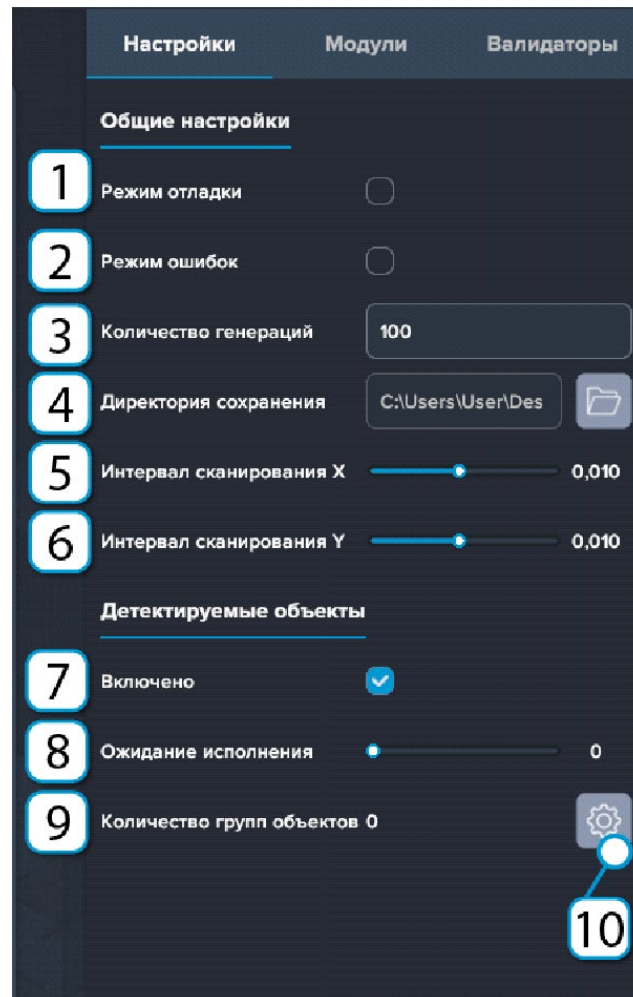


Создание датасетов

- 1 – Счетчик количества сгенерированных изображений
- 2 – Настройки генерации
- 3 – Модули
- 4 – Валидаторы
- 5 – Запуск генерации
- 6 – Меню
- 7 – Генерируемые изображения

Данный модуль позволяет сгенерировать любое количество изображения для синтетического датасета с любыми wybranными моделями. Объекты случайным образом располагаются на случайно сгенерированном фоне – это позволяет с большой долей точности обучить в будущем нейронную сеть. Так же предусмотрена возможность тонкой настройки различных параметров для достижения наилучшего результат при обучении.

Во вкладке **Настройки** представлены настройки генерации:



Настройки

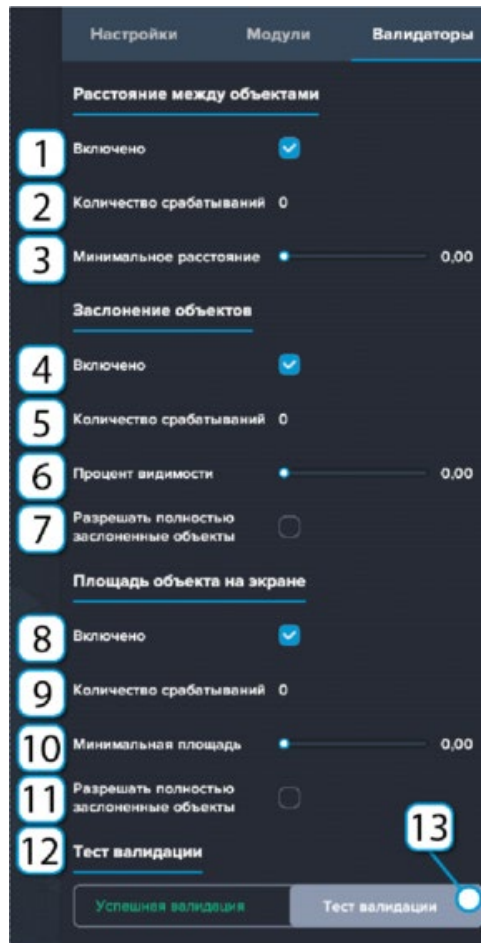
- 1 – Режим отладки (добавляет дополнительную информацию к каждому сохраненному изображению)
 - 2 – Режим ошибок (сохраняются лишь те изображения, что являются ошибочными, нужен для режима отладки)
 - 3 – Индикатор количества генераций
 - 4 – Директория сохранения
 - 5 – Интервал сканирования X
 - 6 – Интервал сканирования Y
 - 7 – Включение/выключение детектируемых объектов
 - 8 – Регулировка времени обновления для объекта (выше число - реже обновляется)
 - 9 – Индикатор количества групп
 - 10 – Настройка групп
- Во вкладке **Модули** представлены настройки модуля:



Вкладка модули

- 1 – Включение/отключение освещения
- 2 – Регулировка времени обновления для света (выше число - реже обновляется)
- 3 – Включение/отключение случайного цвета
- 4 – Включение/отключение случайного направления света
- 5 – Регулировка коэффициента интенсивности света
- 6 – Регулировка коэффициента насыщенности света
- 7 – Включение/отключение фона
- 8 – Регулировка времени обновления для фона (выше число - реже обновляется)
- 9 – Регулировка коэффициента размера шума
- 10 – Регулировка коэффициента распределения текстур
- 11 – Регулировка коэффициента плавности перехода текстур
- 12 – Регулировка коэффициента размера текстур
- 13 – Регулировка коэффициента поворота текстур

Во вкладке **Валидация** представлены настройки валидации:



Валидация

- 1 – Включение/отключение освещения
- 2 – Регулировка времени обновления для света (выше число - реже обновляется)
- 3 – Включение/отключение случайного цвета
- 4 – Включение/отключение случайного направления света
- 5 – Регулировка коэффициента интенсивности света
- 6 – Регулировка коэффициента насыщенности света
- 7 – Включение/отключение фона
- 8 – Регулировка времени обновления для фона (выше число - реже обновляется)
- 9 – Регулировка коэффициента размера шума
- 10 – Регулировка коэффициента распределения текстур
- 11 – Регулировка коэффициента плавности перехода текстур
- 12 – Регулировка коэффициента размера текстур
- 13 – Регулировка коэффициента поворота текстур

Во вкладке **Валидация** представлены настройки условий, определяющие будет ли сохранено изображение.

Если изображение не подходит, появится уведомление:

Ошибка валидации

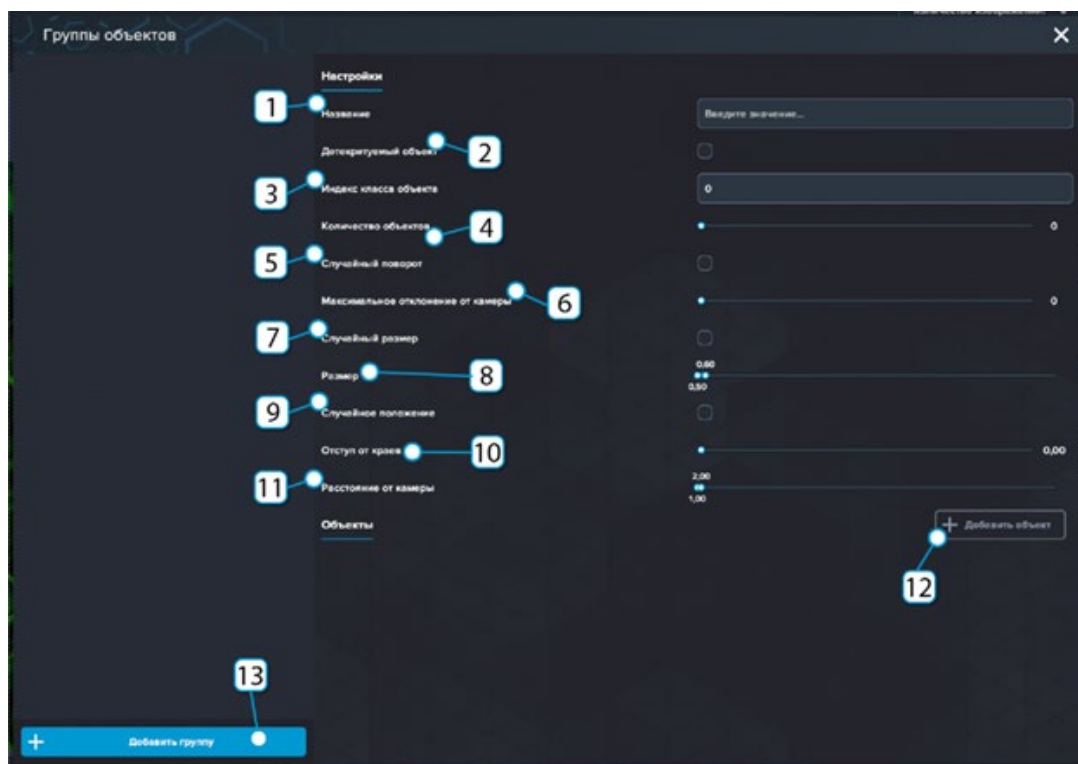
Ошибка валидации

Если изображение подходит, в строке уведомления будет:

Успешная валидация

Успешная валидация

Для того что бы начать генерацию датасета нужно создать и настроить группу объектов. Для этого нажмите на шестеренку во вкладке **Настройки**. Откроется окно выбора групп объектов:

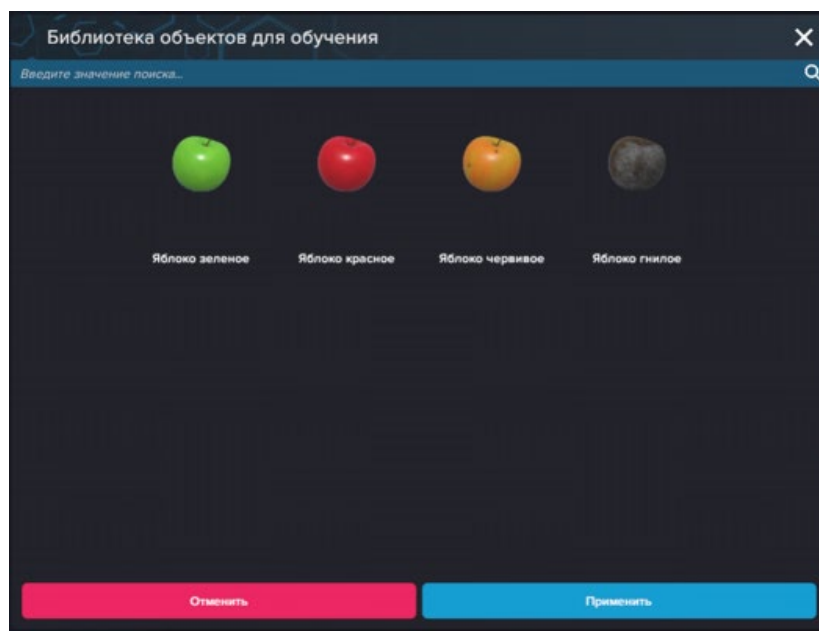


Настройка групп объектов

- 1 – Поле для ввода названия группы
- 2 – Указание детектируемости объекта
- 3 – Поле присвоения индекса класса объекта
- 4 – Регулировка количества объектов
- 5 – Включение/отключение случайного поворота для объектов
- 6 – Регулировка максимального отклонения от камеры
- 7 – Включение/отключение случайного размера объекта

- 8 – Регулировка интервалов коэффициента размера объекта
- 9 – Включение/отключение случайного положения объекта
- 10 – Регулировка отступов от краев
- 11 – Регулировка интервала коэффициента расстояния от камеры
- 12 – Добавление объекта
- 13 – Добавление группы

Для добавления группы нажмите **Добавить группу (13)**. В списке групп слева появится только что созданная группа. Далее нажмите **Добавить объект (12)** для добавления необходимых объектов в созданную группу, откроется следующее окно



Библиотека объектов для обучения

Выберите объекты, согласно сущности вашей группы – если группа относится к дефектным моделям, выберите их и нажмите **Применить**. При желании можно удалить выбранные объекты.

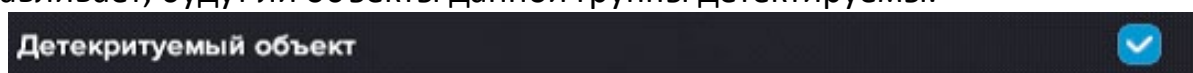
Для примера, создадим две группы с двумя хорошими яблоками в первой группе и двумя плохими во второй.

Рассмотрим группу хороших яблок, в названии которой будет указано «Хорошие».



Название

ВАЖНО: в графе (2) проверьте, что установлена галочка. Данная настройка устанавливает, будут ли объекты данной группы детектируемы.



Деактивируемость объекта

Укажите индекс класса объекта. **ВАЖНО:** для каждой группы должен быть свой индекс класса объекта.

Интерфейс с полем ввода "Индекс класса объекта" и значением "0".

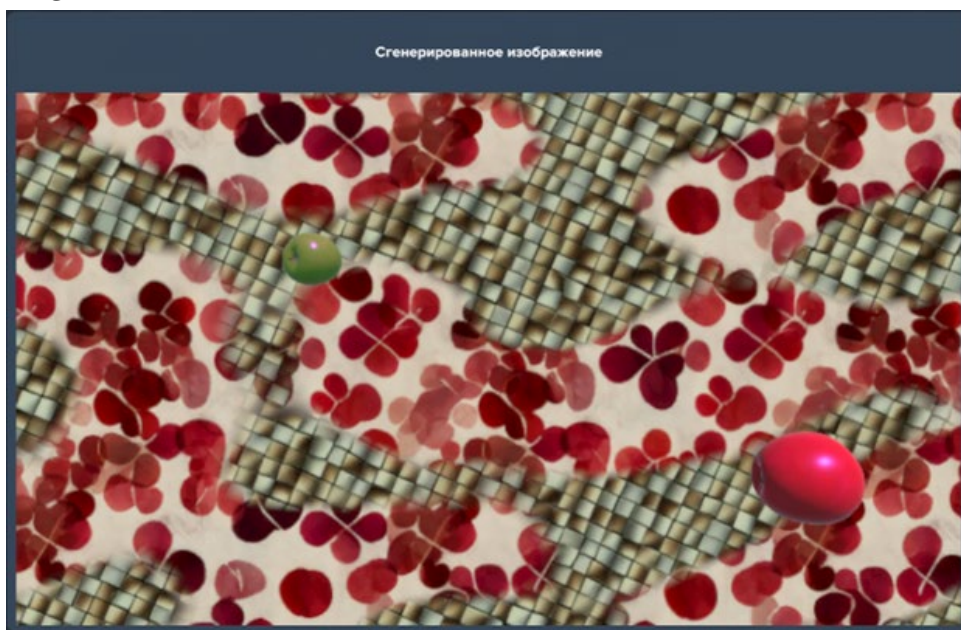
Индекс класса объекта

Установите, сколько объектов будет детектироваться (2), так как выбрано 2 объекта, то выставим 2.

Интерфейс с ползунком "Количество объектов" и значением "2".

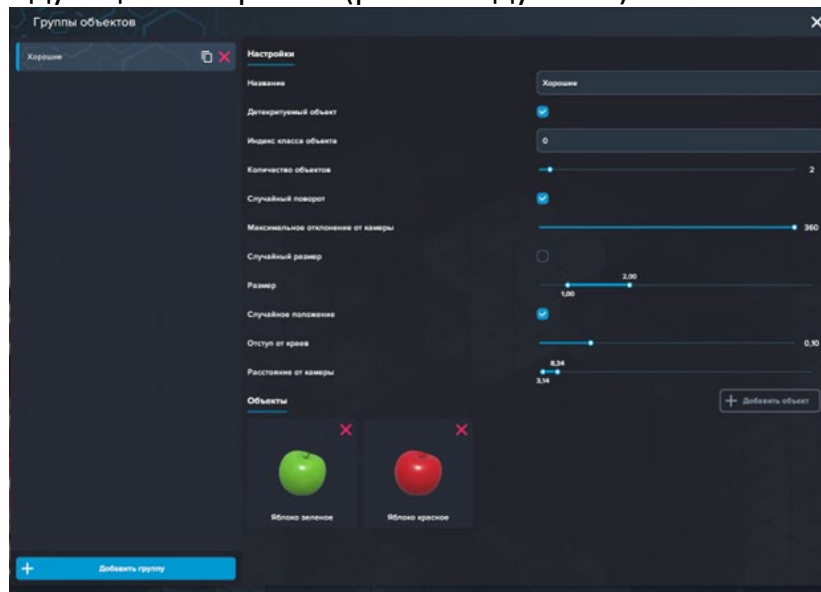
Количество объектов

Остальные настройки подгоняются для объектов индивидуально. Для проверки, во вкладке **Валидаторы**, нажмите **Тест валидации**. Если объект слишком маленький, или слишком большой, некорректно смещен и т.д. настройте группу объектов и другие настройки, так, чтобы результат был примерно как на скриншоте ниже:

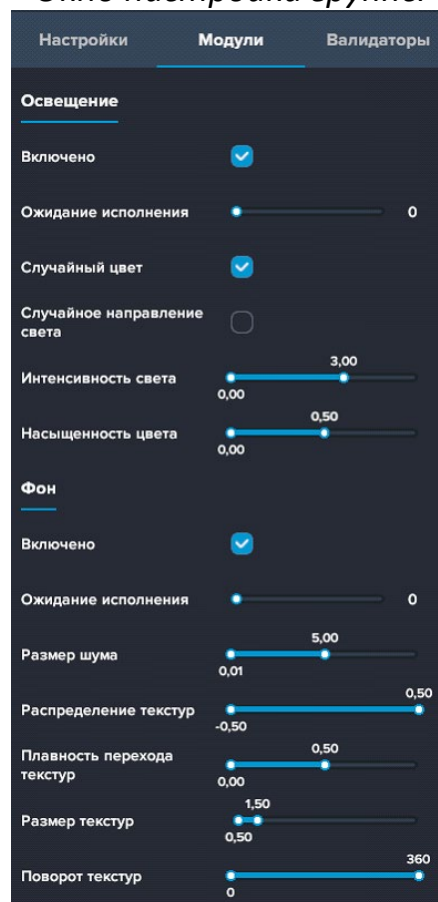


Сгенерированное изображение

Были взяты следующие настройки (рекомендуемые):

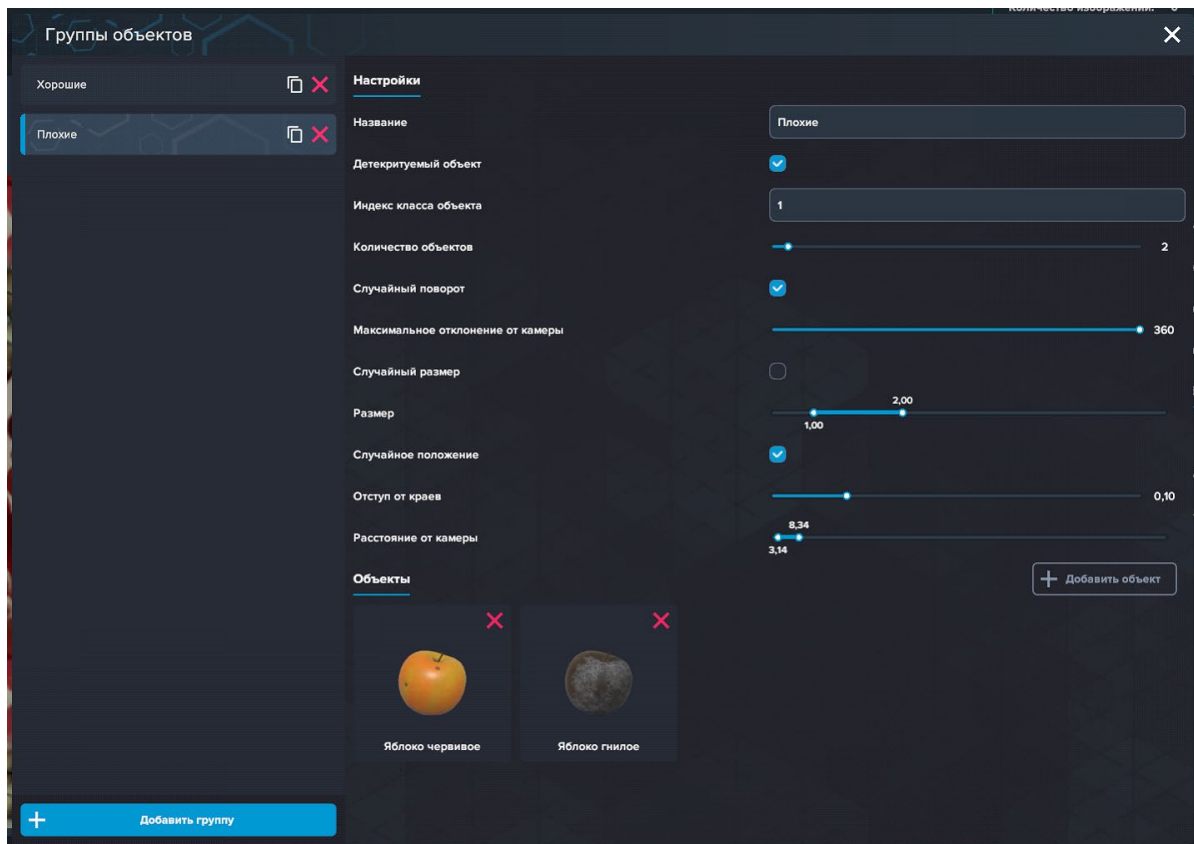


Окно настройки группы



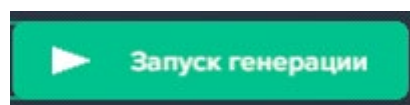
Модули

После настройки одной группы, добавим еще одну группу детектируемых объектов, в нашем случае, «Плохие», **обязательно добавив иной индекс класса объекта, чем у первой группы:**



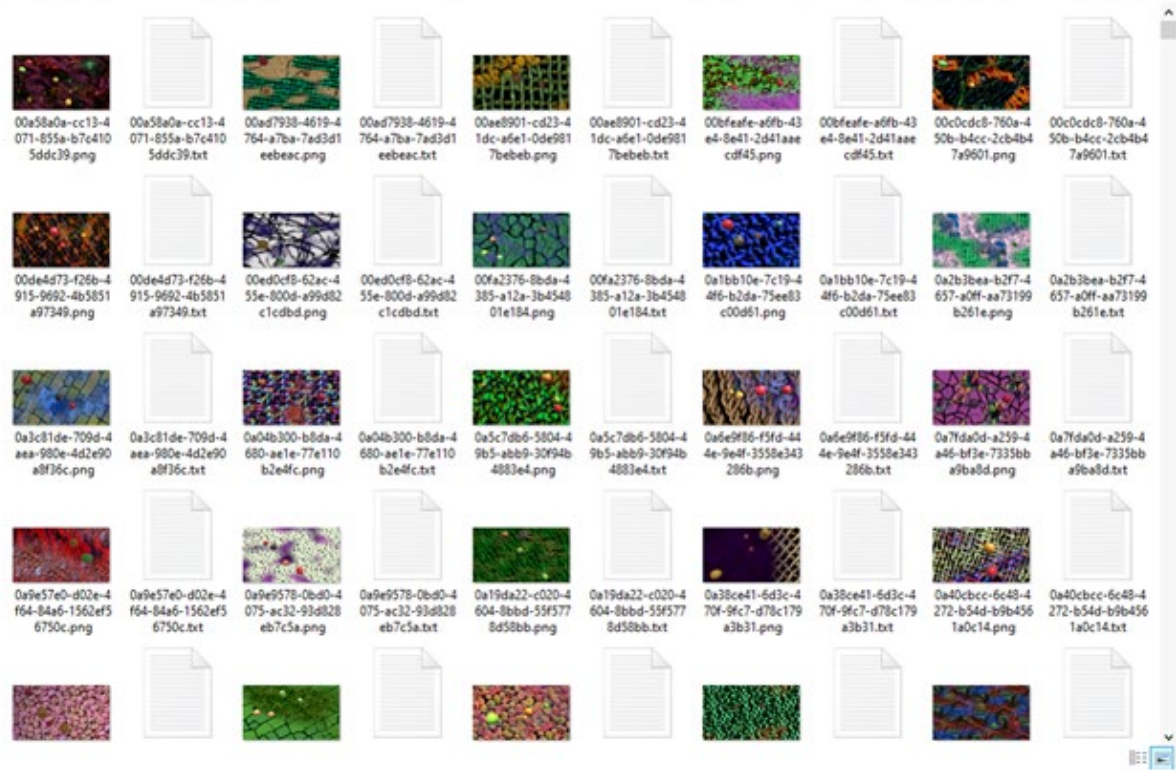
Окно настройки группы

Для запуска генерации, во вкладке **Настройки** в графе **Директория сохранения** укажите путь и нажмите **Запуск генерации**. Затем в графе **Количество генераций** укажите количество итераций для генерации. Рекомендуется брать от 10000. Нажмите запустить генерацию:



Запуск

В выбранной директории появятся все сгенерированные изображения:

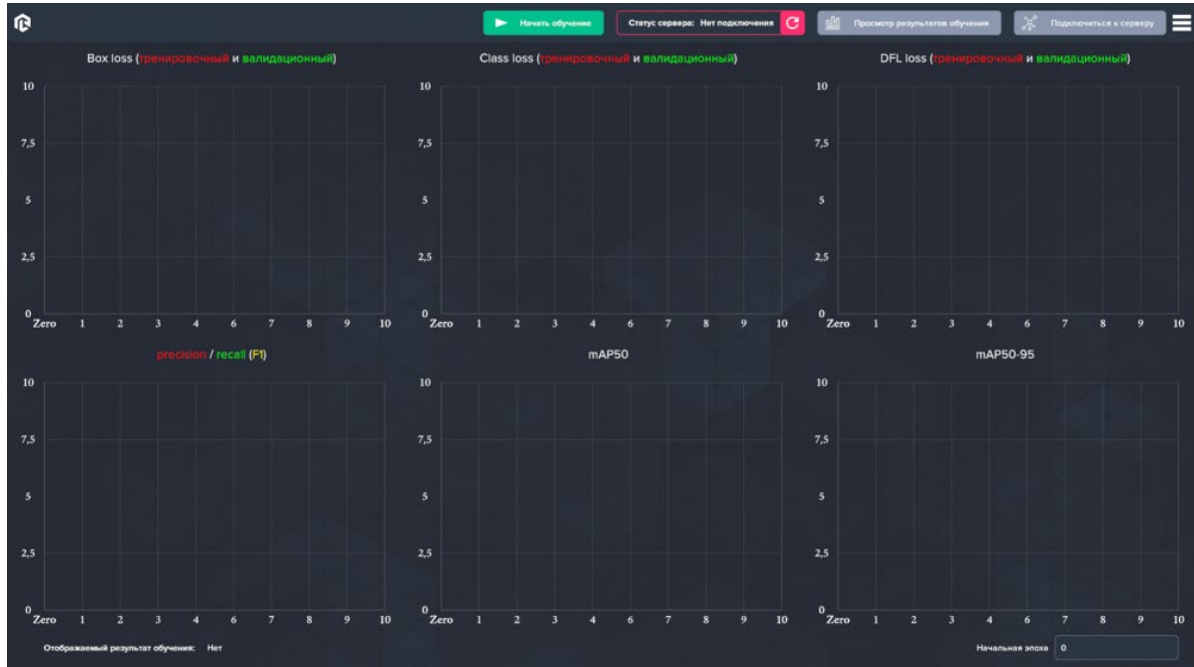


Изображения в директории

Первый файл – само изображение, второй – техническая информация к каждому изображению.

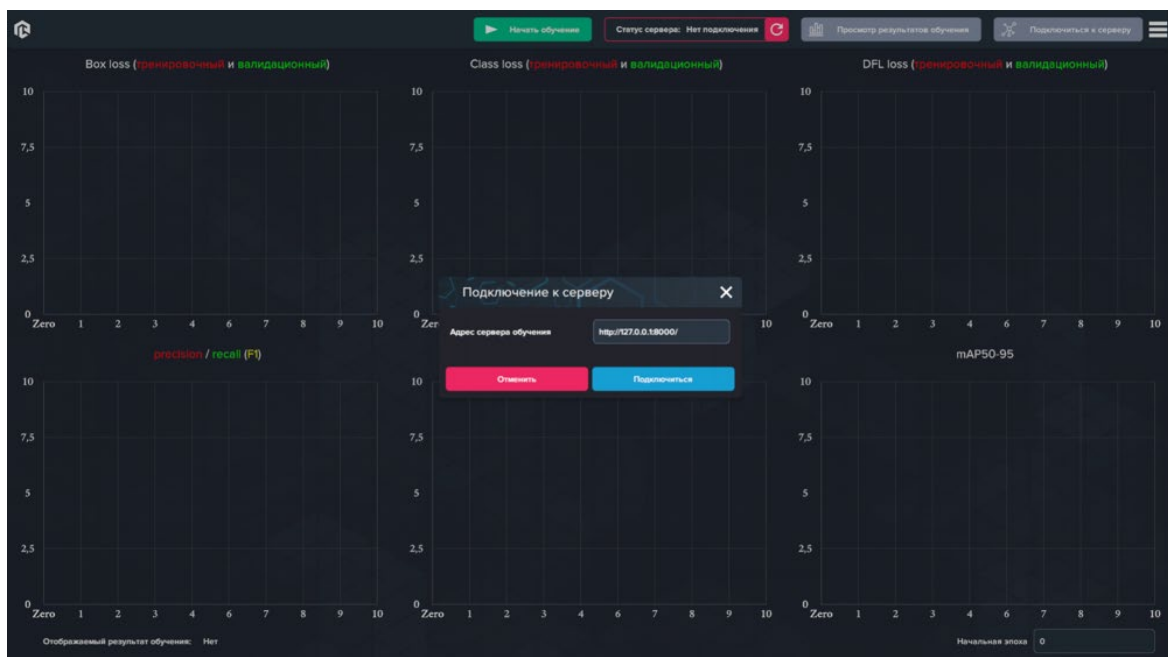
Обучение на созданном датасете

Для обучения на созданном датасете подключите оба сервера нейросетей, после чего перейдите в модуль обучение по датасету. В данном модуле визуально в виде графиков представлен процесс обучения, а также информация о подключённом сервере и кнопка запуска обучения.



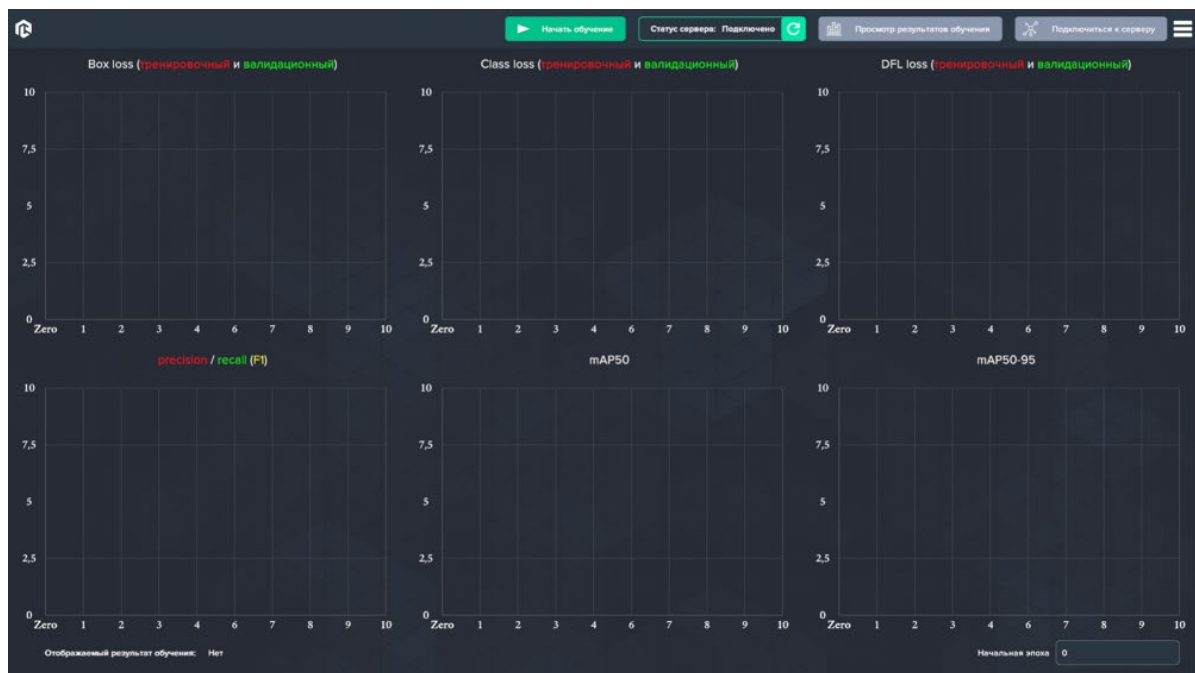
Главный экран

При запуске сервер не подключен, нажмите на кнопку подключиться к серверу и в открывшемся окне введите адрес вашего сервера и порт, после чего нажмите подключиться.



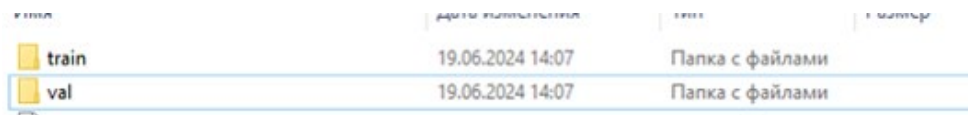
Подключение к серверу

При удачном подключении информация о статусе сервера сменится на Подключено.



Успешное подключение к серверу

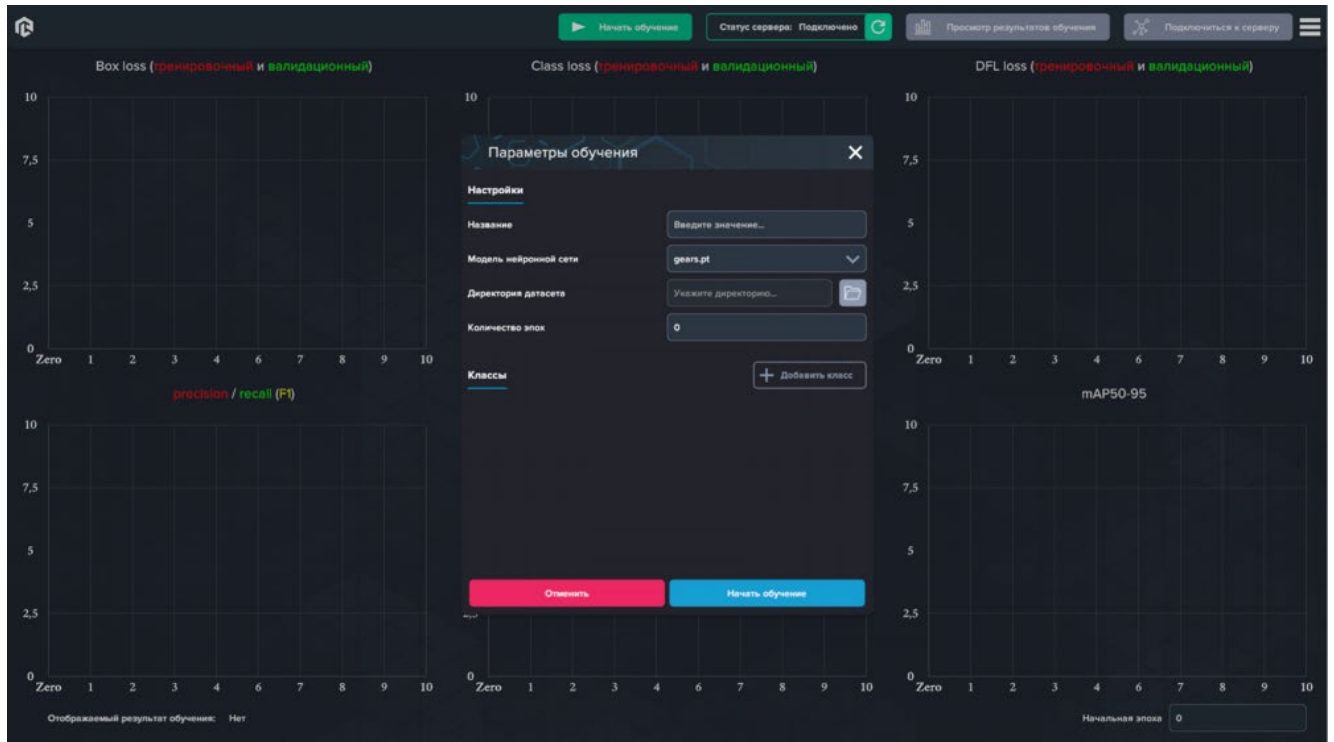
Перед началом обучения перейдите в папку местоположения датасета и создайте внутри 2 папки *train* и *val*.



Создание папок для обучения

После чего перенесите 80% созданных файлов в папку *train* и оставшиеся в папку *val*.

Для начала обучения нажмите на кнопку начать обучение, в открывшемся окне будет информация о параметрах обучения.



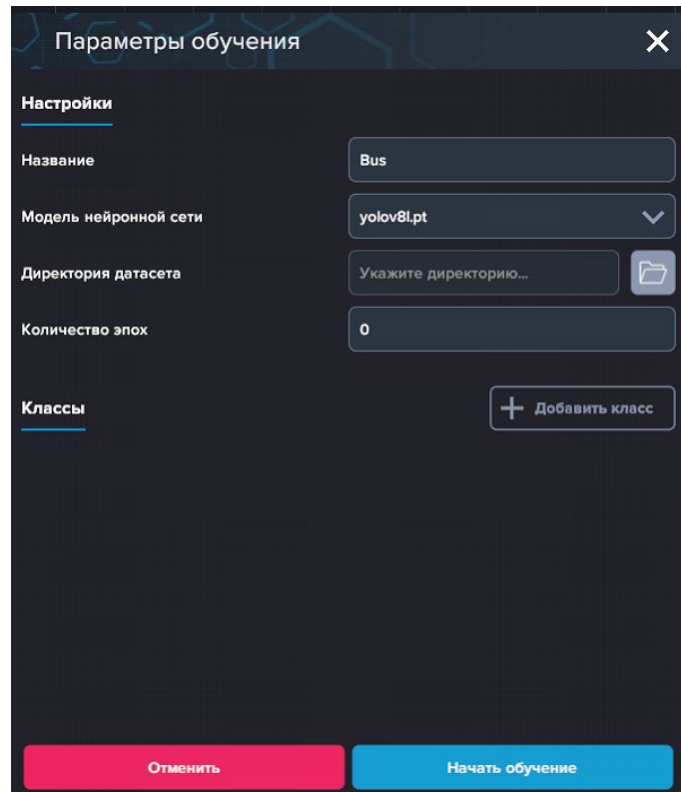
Окно параметров обучения

Для старта обучения необходимо ввести название модели.

This is a close-up of the 'Параметры обучения' dialog box. The 'Настройки' section is expanded, and the 'Название' (Name) field contains the text 'Bus'. Other fields remain the same as in the previous screenshot: 'Модель нейронной сети' is 'yolov8.pt', 'Директория датасета' is empty, and 'Количество эпох' is '0'. The 'Классы' section has a '+ Добавить класс' button. At the bottom are 'Отменить' and 'Начать обучение' buttons.

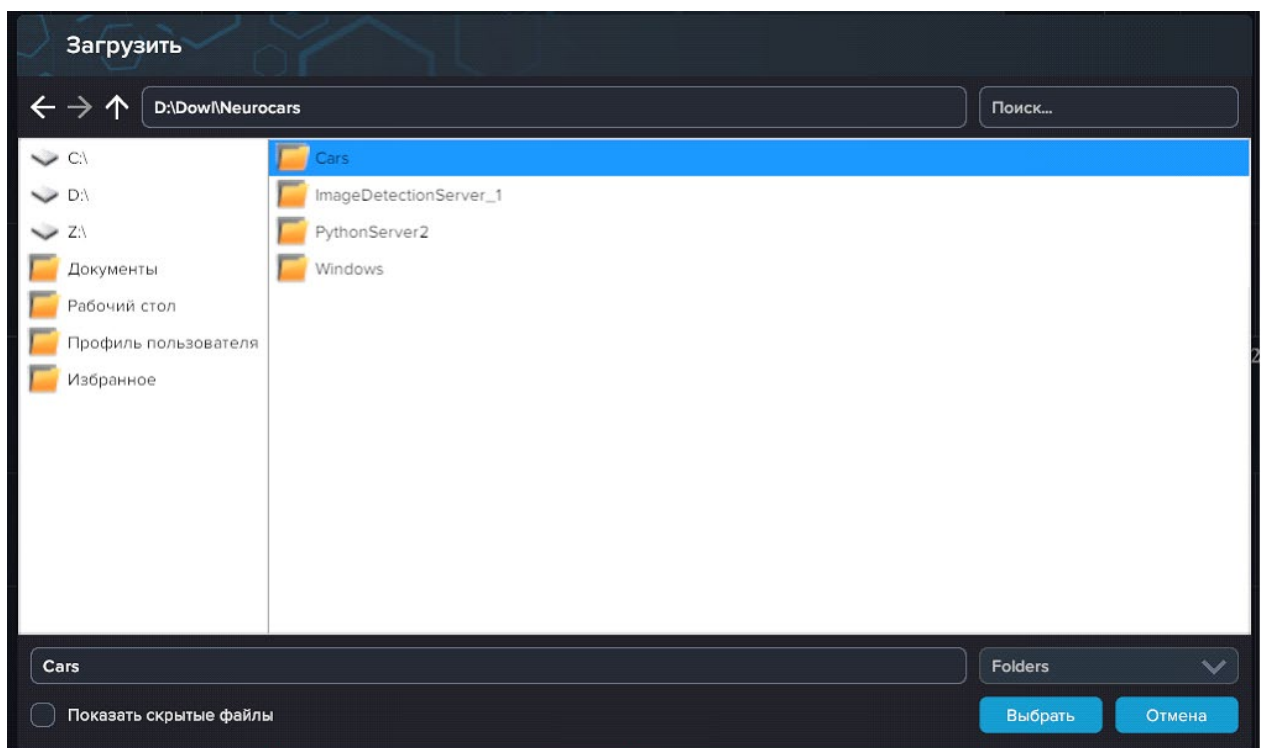
Ввод названия

Выберите из списка модель используемой нейронной сети.



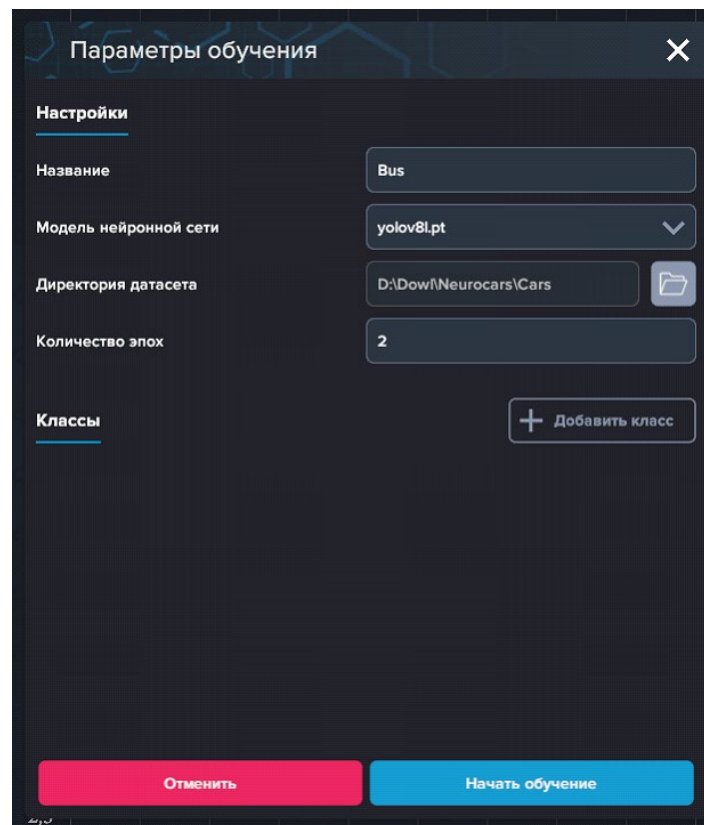
Выбор модели

Укажите местоположение созданного ранее датасета



Выбор датасета

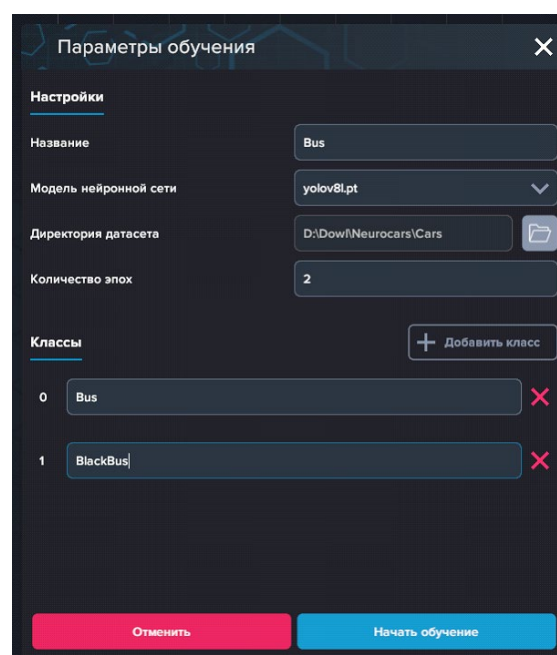
Укажите количество эпох обучения нейросети, количество эпох будет влиять на точность распознавания и на скорость обучения нейросети.



The screenshot shows a dark-themed dialog box titled "Параметры обучения" (Training Parameters). Under the "Настройки" (Settings) section, the "Количество эпох" (Number of epochs) field is set to the value "2". Other fields include "Название" (Name) set to "Bus", "Модель нейронной сети" (Neural network model) set to "yolov8L.pt", and "Директория датасета" (Dataset directory) set to "D:\Dowl\Neurocars\Cars". At the bottom, there are two buttons: "Отменить" (Cancel) in red and "Начать обучение" (Start training) in blue.

Указание количества эпох

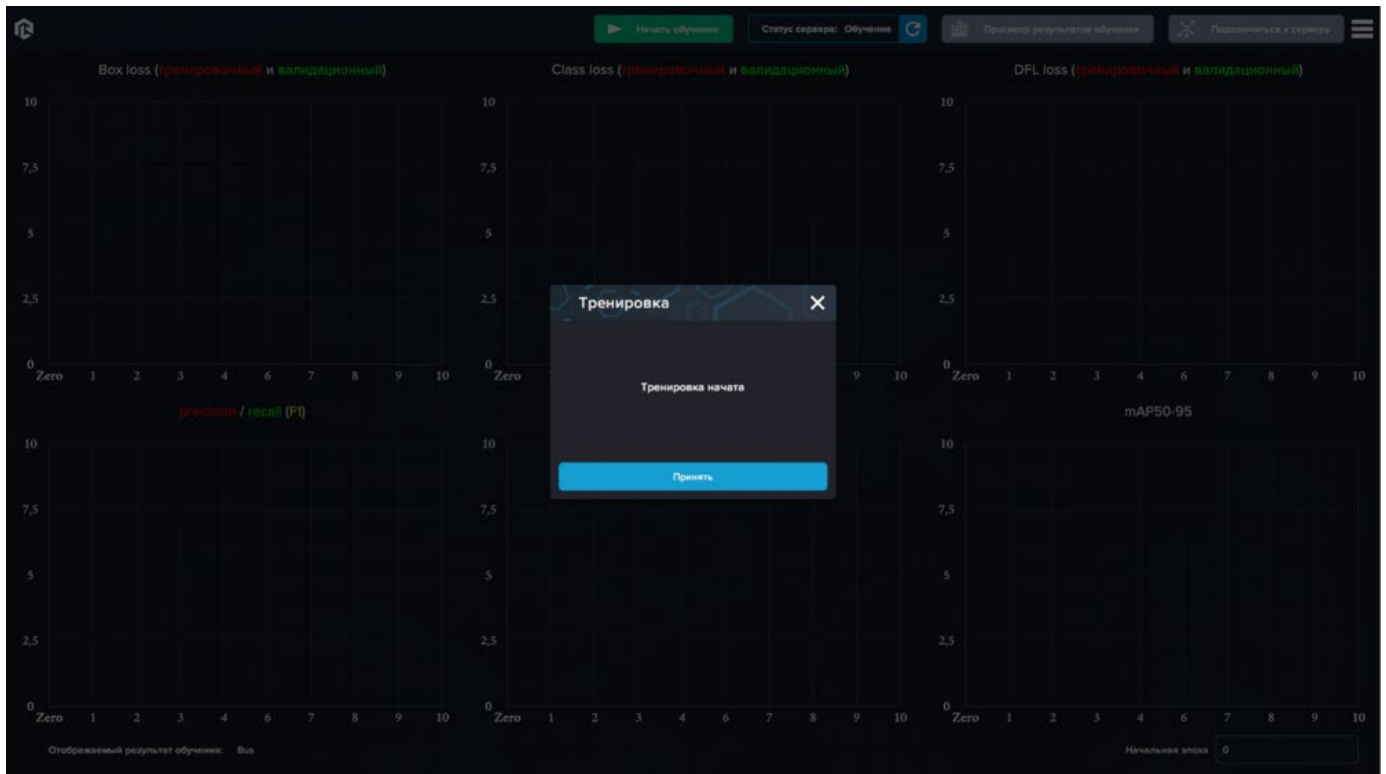
Укажите количество классов для распознавания классы должны соотноситься с индексом класса в созданном датасете.



This screenshot shows the same "Параметры обучения" dialog box, but now with the "Классы" (Classes) section expanded. A "+ Добавить класс" (Add class) button is visible. Below it, two class entries are listed: index "0" with the name "Bus" and index "1" with the name "BlackBus". Each entry has a red "X" icon to its right. The "Отменить" (Cancel) and "Начать обучение" (Start training) buttons remain at the bottom.

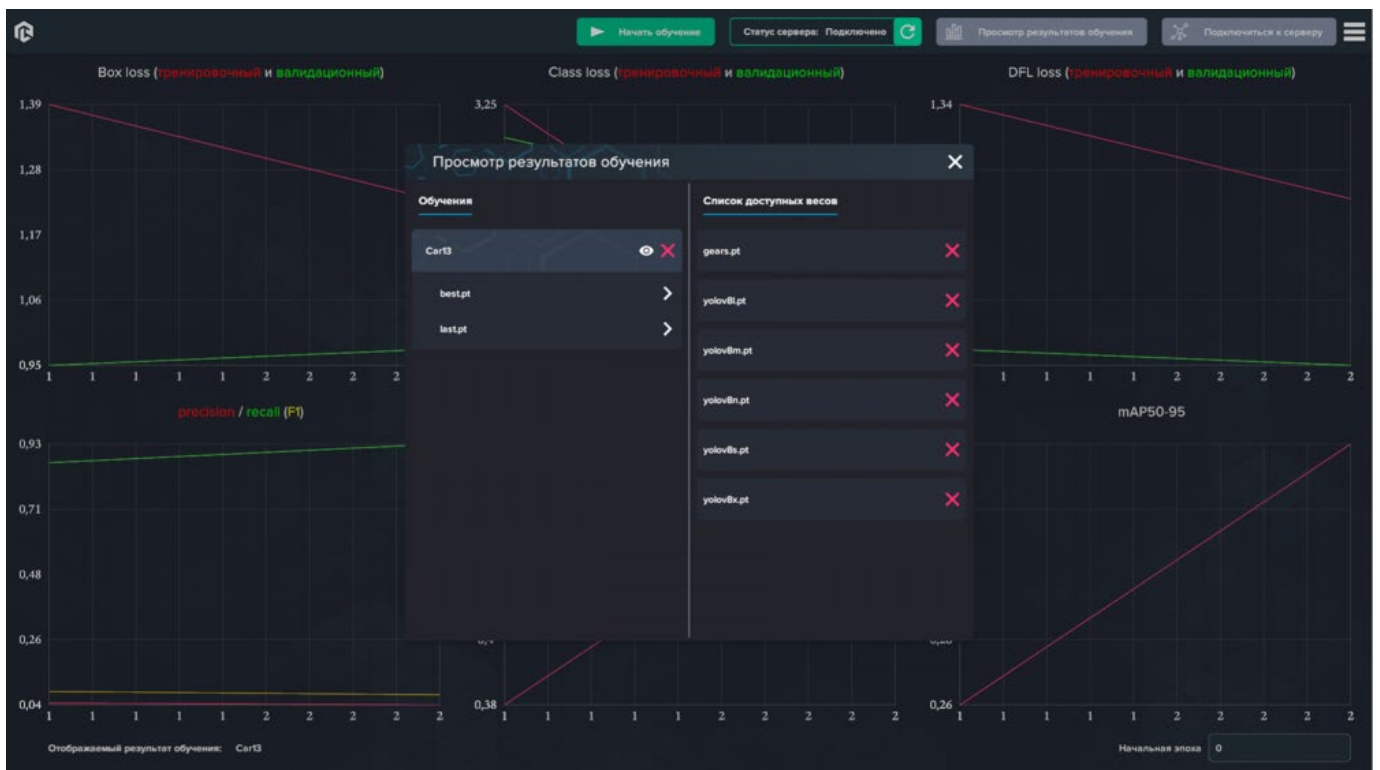
Введение классов

Нажмите на кнопку начать обучение.



Окно информации о запуске обучения

После успешного начала обучения статус сервера сменится, а так же появится окно с информацией о начале обучения.



Просмотр результатов

Для ознакомления с метриками обучения, нажмите на кнопку просмотра результатов, в открывшемся окне выберите необходимое вам обучение и нажмите на иконку глаза.



Ознакомление с метриками

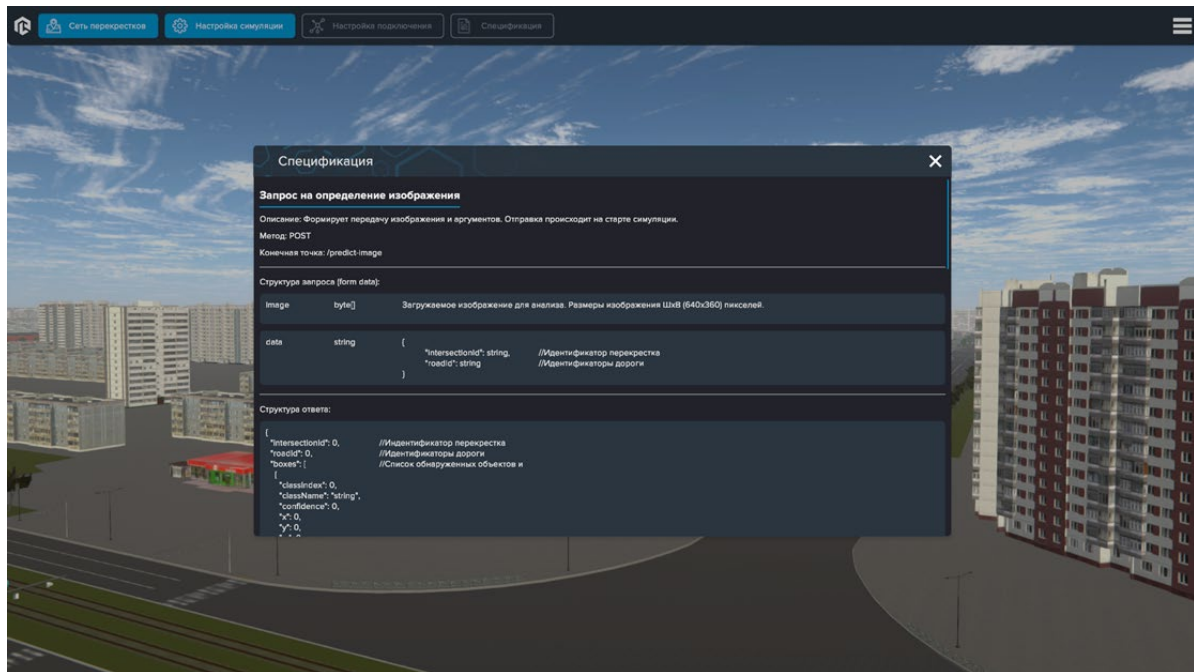
В метриках обучения можно ознакомиться с графиками процесса обучения по количеству эпох, так же можно обрезать количество эпох задав начальную эпоху.

Спецификация API

Спецификация сервера управления городом

В приложении вы можете ознакомиться со спецификацией отправки и получения запросов приложения от приложения к серверу.

При нажатии на вкладку спецификации откроется окно с основными запросами и ответами от приложения к серверу.



Спецификация

Перезапустить сервер

Команда **Перезапустить сервер**, приложение посылает запрос к конечной точке **/restart**, адресованный к серверу, сервер перезапускается и отправляет обратно информацию о перезапуске в виде json ответа.

Пример JSON ответа:

1	{	
2	string status	//Окончание перезапуска
3	}	

Пинг сервера

Команда **Пинг сервера**, приложение посылает запрос к конечной точке **/ping**, адресованный к серверу, сервер отправляет информацию о своем состоянии в виде json ответа.

Пример JSON ответа:

```
1 {  
2   "status": "string"           //Ответ работает ли сервер  
3 }
```

Обнаружение объектов

Команда **Обнаружение объектов на изображении** приложение посылает запрос к конечной точке **/predict-image** в виде изображения и параметров, адресованный к серверу в виде json запроса.

Пример JSON запроса:

```
1   Загружаемое изображение для анализа. Размеры изображения ШхВ (640x360)  
2   пикселей.  
3   {  
4     "intersectionId": string,   //Идентификатор перекрестка  
5     "roadId": string           //Идентификаторы дороги  
   }
```

Сервер получает изображение, отправляет изображение к нейросети и отправляет результат распознавания в виде json ответа.

Пример JSON запроса:

```
1 {
2   "intersectionId": 0,    //Идентификатор перекрестка
3   "roadId": 0,          //Идентификаторы дороги
4   "boxes": [           //Список обнаруженных объектов и
5     {
6       "classIndex": 0,
7       "className": "string",
8       "confidence": 0,
9       "x": 0,
10      "y": 0,
11      "w": 0,
12      "h": 0,
13      "polygonIndexes": [
14        0
15      ]
16    }
17  ]
}
```

В случае ошибки ответ сервера будет выглядеть как следующий json ответ.

Пример сообщения JSON об ошибке:

```
1 {
2   "detail": [
3     {
4       "loc": [
5         "string",
6         0
7       ],
8       "msg": "string",
9       "type": "string"
10    }
11  ]
12 }
```

Объявление зон детектирования

Команда **Объявление зон детектирования** приложение посылает запрос к конечной точке */set-mask-profiles*, адресованный к серверу, в виде json запроса.

Пример JSON ответа:

1	{
2	"masks": [//Область детектирования
3	{
4	"intersectionId": 0, //Идентификатор перекрестка
5	"roadId": 0, //Идентификатор дороги
6	"polygons": [//Многоугольники области детектирования
7	{
8	"vectors": [
9	{
10	"x": 0, //Координата x относительно размера изображения
11	"y": 0 //Координата y относительно размера изображения
12	}
13]
14	}
15]
16	}
17	}

Получив от приложения запрос, сервер отошлет следующий json ответ.

Пример JSON ответа:

1	"string"
---	----------

В случае ошибки ответ сервера будет выглядеть как следующий json ответ.

Пример сообщения JSON об ошибке:

```

1   {
2     "detail": [
3       {
4         "loc": [
5           "string",
6           0
7         ],
8         "msg": "string",
9         "type": "string"
10      }
11     ]
12  }

```

Объявление городской структуры

Команда **Объявление городской структуры** приложение посылает запрос к конечной точке **/set-city-structure**, адресованный к серверу, в виде json запроса.

Пример JSON запроса:

```

1   {
2     "city": {           //Объект параметров города
3       "intersections": [ //Перекрестки города
4         {
5           "id": 0,      //Идентификатор перекрестка
6           "roadIds": [ //Идентификаторы дорог
7             0
8           ],
9           "directions": [ //Направление движения
10          {
11            "groupId": 0, //Идентификаторы направления(объединяющие дороги)
12            "roadIds": [ //Идентификаторы дорог
13              0
14            ],
15            "greenTime": 0 //Установленное по умолчанию время разрешенного
16            движения
17          }
18        }
19      ]
20    }

```

Пример JSON ответа:

```

1   "string"

```

В случае ошибки ответ сервера будет выглядеть как следующий json ответ.

Пример сообщения JSON об ошибке:

```
1 {
2   "detail": [
3     {
4       "loc": [
5         "string",
6         0
7       ],
8       "msg": "string",
9       "type": "string"
10    }
11  ]
12 }
```


Запрос структуры состояния светофоров

Команда **Запрос структуры состояния светофоров** приложение посылает запрос к конечной точке **/get-traffic-lights**, адресованный к серверу, сервер получает запрос, информацию о смене времени на светофоре в виде json ответа.

Пример JSON ответа:

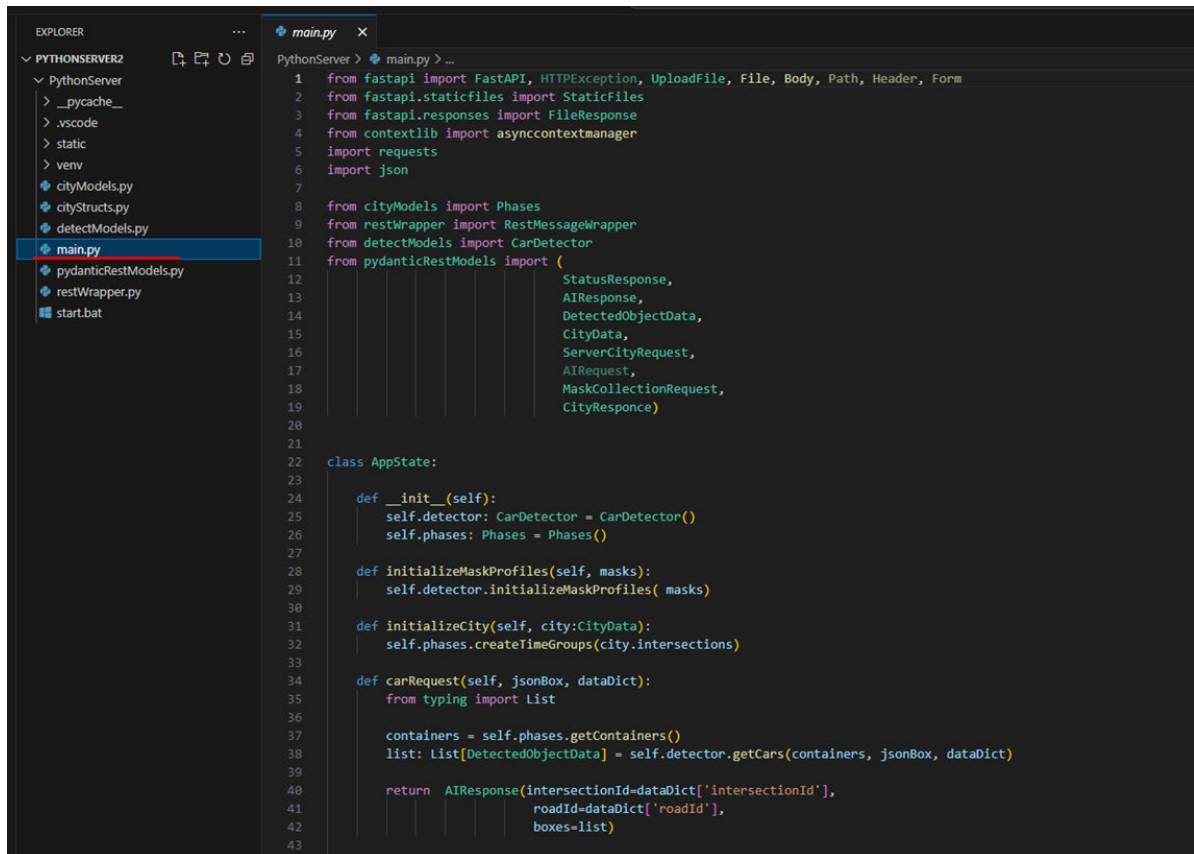
```
1      {
2      "groups": [          //Информация о перекрестках города
3      {
4      "id": 0,           //Идентификатор перекрестка
5      "roadIds": [      //Идентификаторы дорог
6      0
7      ],
8      "directions": [   //Направление движения
9      {
10     "groupId": 0,     //Идентификаторы направления(объединяющие дороги)
11     "roadIds": [     //Идентификаторы дорог
12     0
13     ],
14     "greenTime": 0    //Установленное по умолчанию время разрешенного
15     движения
16     }
17     ]
18     }
19     ]
20     }
```

Описание лабораторной работы

В данном разделе описывается основной код лабораторной работы, все лабораторные работы написаны на языке программирования Python.

Описание сервера управления городом

Сперва рассмотрим код сервера управления городом, а именно файл *main.py*.



```

1  from fastapi import FastAPI, HTTPException, UploadFile, File, Body, Path, Header, Form
2  from fastapi.staticfiles import StaticFiles
3  from fastapi.responses import FileResponse
4  from contextlib import asynccontextmanager
5  import requests
6  import json
7
8  from cityModels import Phases
9  from restWrapper import RestMessageWrapper
10 from detectModels import CarDetector
11 from pydanticRestModels import (
12     StatusResponse,
13     AIResponse,
14     DetectedObjectData,
15     CityData,
16     ServerCityRequest,
17     AIRequest,
18     MaskCollectionRequest,
19     CityResponse)
20
21
22 class AppState:
23
24     def __init__(self):
25         self.detector: CarDetector = CarDetector()
26         self.phases: Phases = Phases()
27
28     def initializeMaskProfiles(self, masks):
29         self.detector.initializeMaskProfiles( masks)
30
31     def initializeCity(self, city:CityData):
32         self.phases.createTimeGroups(city.intersections)
33
34     def carRequest(self, jsonBox, dataDict):
35         from typing import List
36
37         containers = self.phases.getContainers()
38         list: List[DetectedObjectData] = self.detector.getCars(containers, jsonBox, dataDict)
39
40         return AIResponse(intersectionId=dataDict['intersectionId'],
41                           roadId=dataDict['roadId'],
42                           boxes=list)
43

```

Сервер Управления городом

Инициализация переменных используемых в лабораторной работе:

1	class AppState:
2	
3	def __init__(self):
4	self.detector: CarDetector = CarDetector()
5	self.phases: Phases = Phases()

Инициализация маски распознавания в лабораторной работе:

1	def initializeMaskProfiles(self, masks):
2	self.detector.initializeMaskProfiles(masks)

Инициализация групп данных города в лабораторной работе:

```
1 def initializeCity(self, city:CityData):
2     self.phases.createTimeGroups(city.intersections)
```

Метод запросов данных о авто в лабораторной работе:

```
1 def carRequest(self, jsonBox, dataDict):
2     from typing import List
3
4     containers = self.phases.getContainers()
5     list: List[DetectedObjectData] = self.detector.getCars(containers, jsonBox,
6 dataDict)
7
8     return AIResponse(intersectionId=dataDict['intersectionId'],
9                       roadId=dataDict['roadId'],
10                      boxes=list)
```

Инициализация метод передачи информации о времени светофорах:

```
1 def getTrafficLightState(self):
2
3     self.phases.makeAnalysis()
4
5     containers = self.phases.getContainers()
6
7     print(f"containers.count={len(containers)}")
8     wrapper = RestMessageWrapper(containers)
9
10    return wrapper.getConvertedResponse()
```

Метод жизненного цикла сервера:

```
1 @asynccontextmanager
2 async def lifespan(app: FastAPI):
3     requests.post(f"http://{PREDICT_IMAGE_HOSTADDRESS}/set-model",
4 json={"name": MODEL_NAME})
5     yield
```

Инициализация и запуск сервера:

```
1 app = FastAPI(  
2     title="PLNeuro",  
3     description="API для нейронных сетей",  
4     version="1.0.0",  
5     lifespan=lifespan  
6 )
```

Метод перезапуска сервера, при получении запроса о перезапуске перезапускает сервер:

```
1 app.post("/restart", summary="Перезапустить сервер")  
2 def restart():  
3     # os.kill(os.getpid(), signal.SIGINT)  
4     return {"Main": "Page"}
```

Метод пинга сервера, при получении запроса отправляет информацию о состоянии:

```
1 @app.get("/ping", summary="Пинг сервера")  
2 def ping():  
3     return StatusResponse(status="pong")
```

Функция вызова метода обнаружения объектов на изображении:

```
1 @app.post("/predict-image", response_model=AIResponse,  
2 summary="Обнаружение объектов на изображении")  
3 async def predictImage(image: UploadFile = File(..., description="Загружаемое  
4 изображение для анализа"),  
5 data: str = Body(..., description="Параметры точки запроса")):  
6  
7     aiResponse =  
8 requests.post(f"http://{PREDICT_IMAGE_HOSTADDRESS}/predict-to-data",  
9 files={"image": image.file})  
10  
11 return app_state.carRequest(aiResponse.json(), json.loads(data))
```

Функция вызова метода объявления зон детектирования:

```
1 @app.post("/set-mask-profiles", summary="Объявление зон детектирования")
2 async def initMaskProfiles(request: MaskCollectionRequest = Body(...,
3 description="Json границ зоны детектирования")):
4
5     app_state.initializeMaskProfiles(request.masks)
6     return {"Main": "Page"}
```

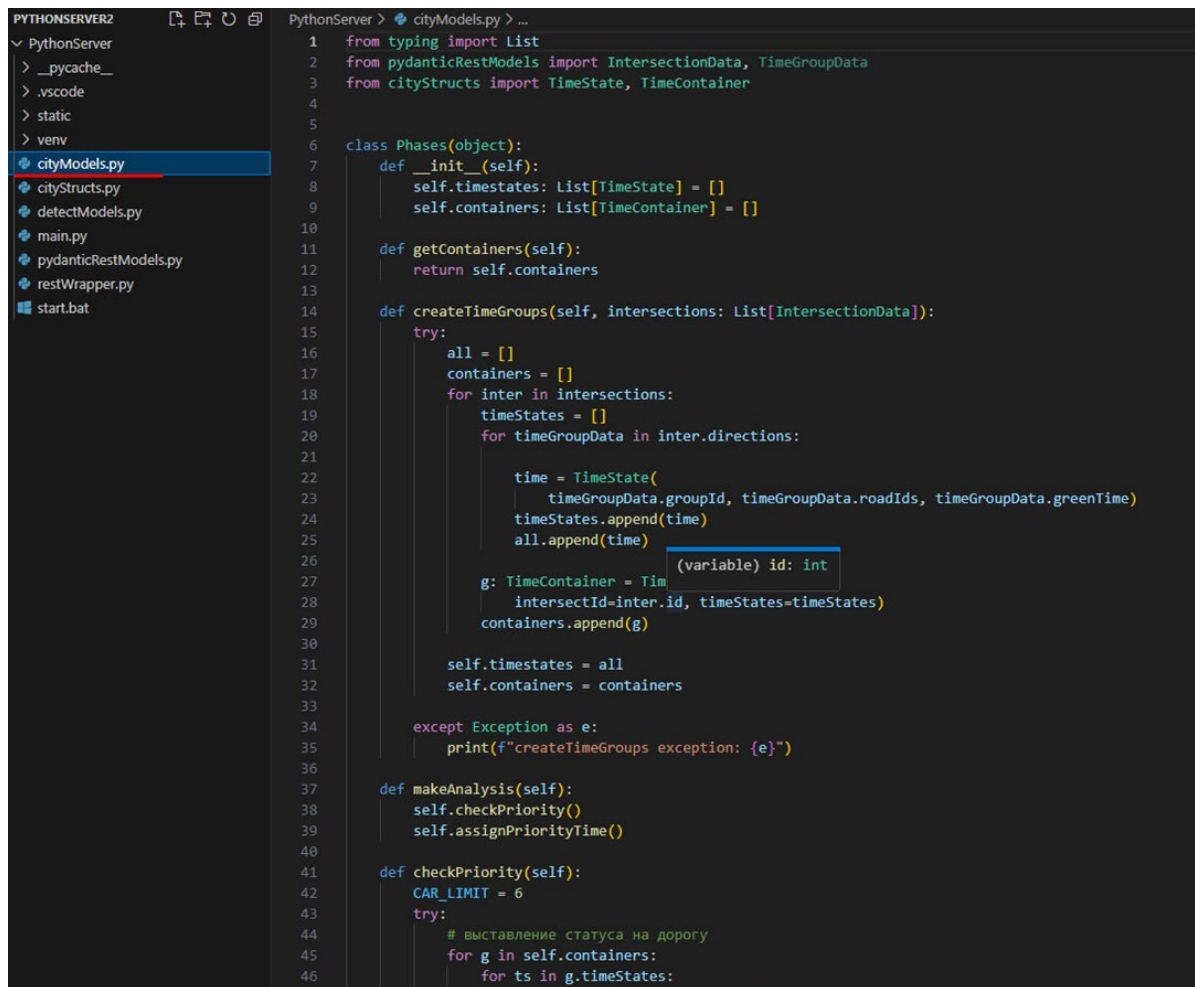
Функция вызова метода объявления городской структуры:

```
1 @app.post("/set-city-structure", summary="Объявление городской
2 структуры")
3 async def initCityStructure(request: ServerCityRequest = Body(...,
4 description="Json структурных связей объектов")):
5
6     app_state.initializeCity(request.city)
7     return {"Main": "Page"}
```

Функция вызова метода запроса структуры состояния светофоров:

```
1 @app.get("/get-traffic-lights",
2 response_model=CityResponse, summary="Запрос структуры состояния
3 светофоров")
4 async def getTrafficLights():
5     return app_state.getTrafficLightState()
```

Рассмотрим код управления городом данный класс создает группы светофоров после чего задает время на светофорах, регулировании времени на светофорах в зависимости от загруженности дорог.



```

1  from typing import List
2  from pydanticRestModels import IntersectionData, TimeGroupData
3  from cityStructs import TimeState, TimeContainer
4
5
6  class Phases(object):
7      def __init__(self):
8          self.timestates: List[TimeState] = []
9          self.containers: List[TimeContainer] = []
10
11     def getContainers(self):
12         return self.containers
13
14     def createTimeGroups(self, intersections: List[IntersectionData]):
15         try:
16             all = []
17             containers = []
18             for inter in intersections:
19                 timeStates = []
20                 for timeGroupData in inter.directions:
21
22                     time = TimeState(
23                         timeGroupData.groupId, timeGroupData.roadIds, timeGroupData.greenTime)
24                     timeStates.append(time)
25                     all.append(time)
26
27                     g: TimeContainer = TimeContainer(
28                         intersectId=inter.id, timeStates=timeStates)
29                     containers.append(g)
30
31             self.timestates = all
32             self.containers = containers
33
34         except Exception as e:
35             print(f"createTimeGroups exception: {e}")
36
37     def makeAnalysis(self):
38         self.checkPriority()
39         self.assignPriorityTime()
40
41     def checkPriority(self):
42         CAR_LIMIT = 6
43         try:
44             # выставление статуса на дорогу
45             for g in self.containers:
46                 for ts in g.timeStates:

```

Класс управления светофорами

Метод создания временных групп светофоров и дорог :

```
1 def createTimeGroups(self, intersections: List[IntersectionData]):
2     try:
3         all = []
4         containers = []
5         for inter in intersections:
6             timeStates = []
7             for timeGroupData in inter.directions:
8
9                 time = TimeState(
10                    timeGroupData.groupId, timeGroupData.roadIds,
11                    timeGroupData.greenTime)
12                 timeStates.append(time)
13                 all.append(time)
14
15                 g: TimeContainer = TimeContainer(
16                    intersectId=inter.id, timeStates=timeStates)
17                 containers.append(g)
18
19                 self.timestates = all
20                 self.containers = containers
21
22     except Exception as e:
23         print(f"createTimeGroups exception: {e}")
```

Метод анализа данных загруженности дорог:

```
1 def makeAnalysis(self):
2     self.checkPriority()
3     self.assignPriorityTime()
```

Инициализация переменных используемых в лабораторной работе:

```
1 class AppState:
2
3     def __init__(self):
4         self.detector: CarDetector = CarDetector()
5         self.phases: Phases = Phases()
```

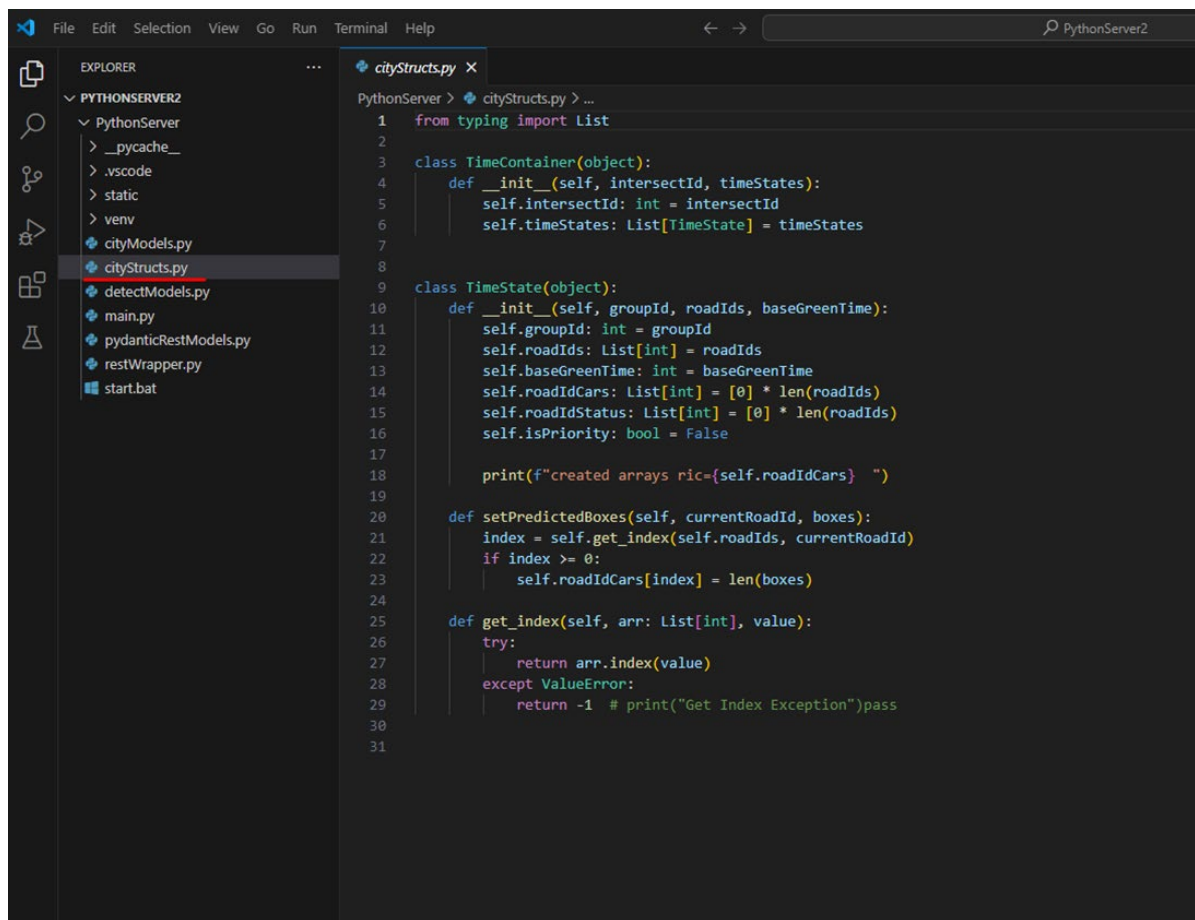

Метод проверки загруженности дорог и выставления статуса о загруженности:

```
1 def checkPriority(self):
2     CAR_LIMIT = 6
3     try:
4         # выставление статуса на дорогу
5         for g in self.containers:
6             for ts in g.timeStates:
7                 for i in range(len(ts.roadIdCars)):
8                     if ts.roadIdCars[i] >= CAR_LIMIT:
9                         ts.roadIdStatus[i] = 1
10                    else:
11                        ts.roadIdStatus[i] = 0
12
13                # назначение приоритета направления в зависимости от статуса
14                for g in self.containers:
15                    for tss in g.timeStates:
16                        if (self.common_member([1], tss.roadIdStatus)):
17                            tss.isPriority = True
18                            print(f"st={tss.roadIdStatus} is ={tss.isPriority}")
19
20            except Exception as e:
21                print(f"checkPriority exception: {e}")
```

Метод изменения времени на светофоре в зависимости от загруженности дороги:

```
1 def assignPriorityTime(self):
2     CAR_AVERAGE_TIME = 25
3     CAR_MAX_PRIORITY_TIME = 45
4     try:
5         for container in self.containers:
6             bools = []
7             for obj in container.timeStates:
8                 # конвертация статусов в массив с булевыми
9                 bools.append(any(obj.roadIdStatus))
10
11             if (all(bools)):
12                 # all true (все направления имеют много машин)
13                 for fl in container.timeStates:
14                     fl.greenTime = CAR_AVERAGE_TIME
15             else:
16                 for flw in container.timeStates:
17                     arr = flw.roadIdStatus
18                     if (not any(arr)): # all false
19                         flw.greenTime = flw.baseGreenTime
20                     elif (any(arr)): # any true (на одном напрвлении много машин)
21                         flw.greenTime = CAR_MAX_PRIORITY_TIME
22
23     except Exception as e:
24         print(f"GetCityAnalysis Exception: {e}")
```

Рассмотрим код структур хранения временных настроек для работы светофоров, индексов дорог, приоритетов и перекрестков.



```

1  from typing import List
2
3  class TimeContainer(object):
4      def __init__(self, intersectId, timeStates):
5          self.intersectId: int = intersectId
6          self.timeStates: List[TimeState] = timeStates
7
8
9  class TimeState(object):
10     def __init__(self, groupId, roadIds, baseGreenTime):
11         self.groupId: int = groupId
12         self.roadIds: List[int] = roadIds
13         self.baseGreenTime: int = baseGreenTime
14         self.roadIdCars: List[int] = [0] * len(roadIds)
15         self.roadIdStatus: List[int] = [0] * len(roadIds)
16         self.isPriority: bool = False
17
18         print(f"created arrays ric={self.roadIdCars} ")
19
20     def setPredictedBoxes(self, currentRoadId, boxes):
21         index = self.get_index(self.roadIds, currentRoadId)
22         if index >= 0:
23             self.roadIdCars[index] = len(boxes)
24
25     def get_index(self, arr: List[int], value):
26         try:
27             return arr.index(value)
28         except ValueError:
29             return -1 # print("Get Index Exception")pass
30
31

```

Класс хранения данных о городе

Метод обработки и хранения данных полученных от города:

1	def __init__(self, groupId, roadIds, baseGreenTime):
2	self.groupId: int = groupId
3	self.roadIds: List[int] = roadIds
4	self.baseGreenTime: int = baseGreenTime
5	self.roadIdCars: List[int] = [0] * len(roadIds)
6	self.roadIdStatus: List[int] = [0] * len(roadIds)
7	self.isPriority: bool = False
8	
9	print(f"created arrays ric={self.roadIdCars} ")

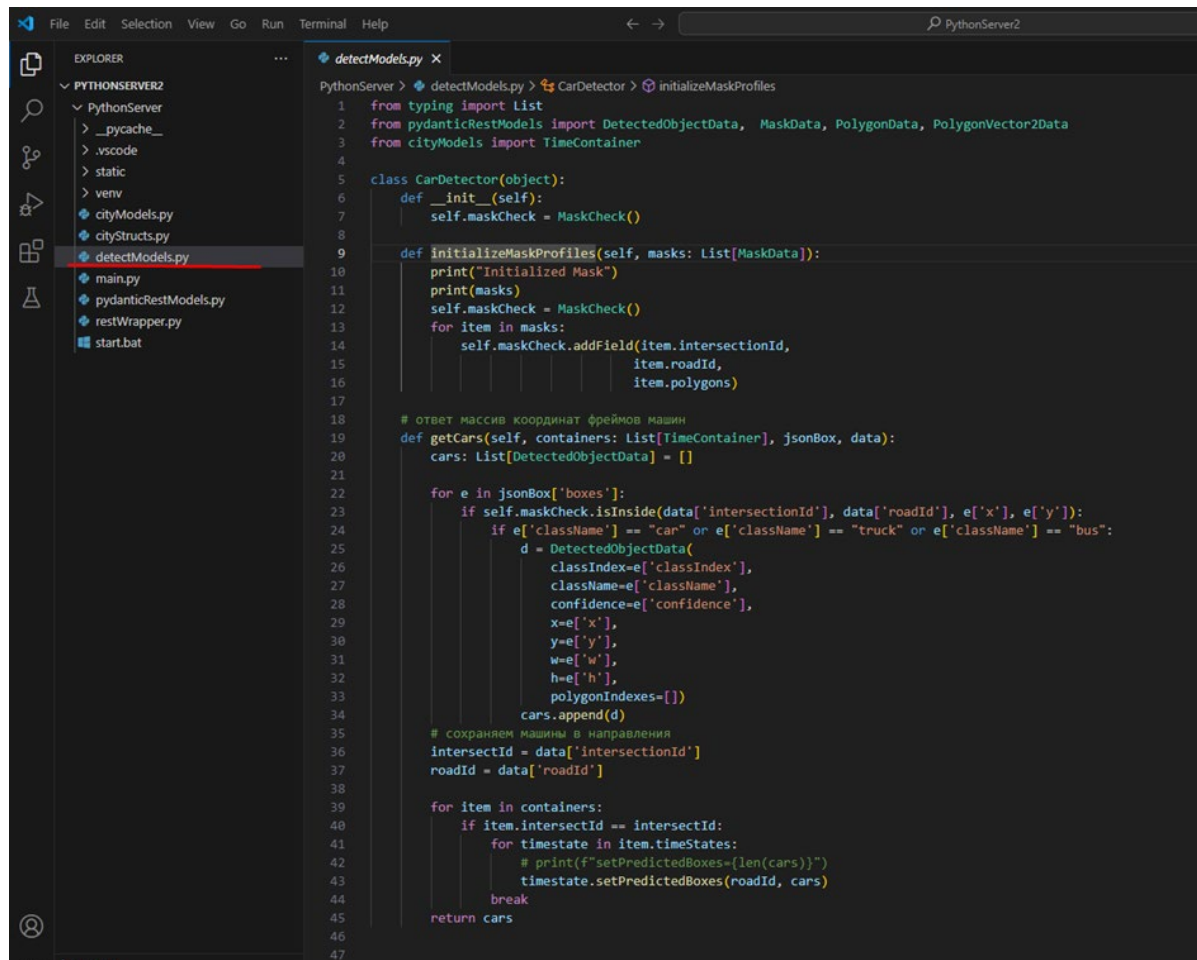
Метод обработки данных о состоянии дорог:

```

1 def setPredictedBoxes(self, currentRoadId, boxes):
2     index = self.get_index(self.roadIds, currentRoadId)
3     if index >= 0:
4         self.roadIdCars[index] = len(boxes)

```

Рассмотрим код создания масок для определения автомобилей, просчет попадания автомобилей в зону маски и получение данных от нейросети для классификации данных.



```

1 from typing import List
2 from pydanticRestModels import DetectedObjectData, MaskData, PolygonData, PolygonVector2Data
3 from cityModels import TimeContainer
4
5 class CarDetector(object):
6     def __init__(self):
7         self.maskCheck = MaskCheck()
8
9     def initializeMaskProfiles(self, masks: List[MaskData]):
10        print("Initialized Mask")
11        print(masks)
12        self.maskCheck = MaskCheck()
13        for item in masks:
14            self.maskCheck.addField(item.intersectionId,
15                                   item.roadId,
16                                   item.polygons)
17
18 # ответ массив координат фреймов машин
19 def getCars(self, containers: List[TimeContainer], jsonBox, data):
20     cars: List[DetectedObjectData] = []
21
22     for e in jsonBox['boxes']:
23         if self.maskCheck.isInside(data['intersectionId'], data['roadId'], e['x'], e['y']):
24             if e['className'] == "car" or e['className'] == "truck" or e['className'] == "bus":
25                 d = DetectedObjectData(
26                     className=e['className'],
27                     confidence=e['confidence'],
28                     x=e['x'],
29                     y=e['y'],
30                     w=e['w'],
31                     h=e['h'],
32                     polygonIndexes=[]
33                 )
34                 cars.append(d)
35
36 # сохраняем машины в направления
37 intersectId = data['intersectionId']
38 roadId = data['roadId']
39
40 for item in containers:
41     if item.intersectId == intersectId:
42         for timestep in item.timeStates:
43             # print(f"setPredictedBoxes={len(cars)}")
44             timestep.setPredictedBoxes(roadId, cars)
45             break
46
47     return cars

```

Класс работы с масками

Инициализация переменных используемых в лабораторной работе:

```

1 def __init__(self):
2     self.maskCheck = MaskCheck()

```

Метод инициализации маски для классификации объекта:

```
1 def initializeMaskProfiles(self, masks: List[MaskData]):
2     print("Initialized Mask")
3     print(masks)
4     self.maskCheck = MaskCheck()
5     for item in masks:
6         self.maskCheck.addField(item.intersectionId,
7                                 item.roadId,
8                                 item.polygons)
```

Метод обработки информации о распознаном автомобиле:

```
1 def getCars(self, containers: List[TimeContainer], jsonBox, data):
2     cars: List[DetectedObjectData] = []
3
4     for e in jsonBox['boxes']:
5         if self.maskCheck.isInside(data['intersectionId'], data['roadId'], e['x'],
6 e['y']):
7             if e['className'] == "car" or e['className'] == "truck" or e['className']
8 == "bus":
9                 d = DetectedObjectData(
10                    classIndex=e['classIndex'],
11                    className=e['className'],
12                    confidence=e['confidence'],
13                    x=e['x'],
14                    y=e['y'],
15                    w=e['w'],
16                    h=e['h'],
17                    polygonIndexes=[])
18                 cars.append(d)
19                 # сохраняем машины в направления
20                 intersectId = data['intersectionId']
21                 roadId = data['roadId']
22
23                 for item in containers:
24                     if item.intersectId == intersectId:
25                         for timestate in item.timeStates:
26                             # print(f"setPredictedBoxes={len(cars)}")
27                             timestate.setPredictedBoxes(roadId, cars)
28                         break
29                 return cars
```

Класс обработки информации данных присланных нейросетью о определении автомобиля в маске распознавания:

```
1 class MaskCheck:
2     def __init__(self):
3         self.fields: List[MaskField] = []
4
5     def addField(self, interId, roadId, polygons):
6         self.fields.append(MaskField(interId, roadId, polygons))
7
8     def isInside(self, interId, roadId, xp, yp):
9         yp = (1 - yp) # инверсия(коррекция вывода с нейронки)
10        count: int = 0
11        for field in self.fields:
12            if field.interId == interId and field.roadId == roadId:
13                for edge in field.edges:
14                    x1 = edge.v1.x
15                    y1 = edge.v1.y
16                    x2 = edge.v2.x
17                    y2 = edge.v2.y
18                    if ((yp < y1) != (yp < y2)) and (xp < x1 + ((yp - y1) / (y2 - y1)) * (x2 -
19 x1)):
20                        count += 1
21                        break
22                else:
23                    continue
24        return count % 2 == 1
```

Класс определения попадания объекта в маску:

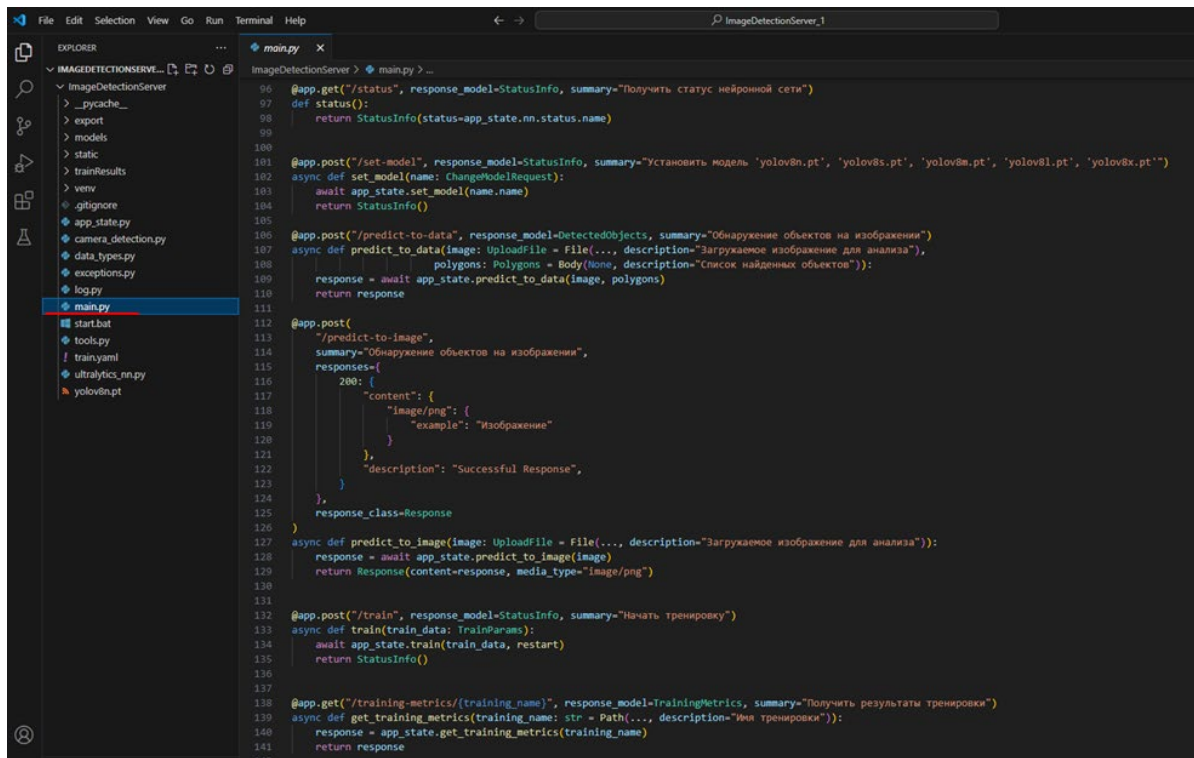
```
1 class MaskField:
2     def __init__(self, interId, roadId, polygons: List[PolygonData]):
3         self.interId: int = interId
4         self.roadId: int = roadId
5         self.edges: List[Edge] = []
6
7         vectors: List[PolygonVector2Data] = []
8         for poly in polygons:
9             for vectorData in poly.vectors:
10                v = PolygonVector2Data(x=vectorData.x, y=vectorData.y)
11                vectors.append(v)
12
13            for i in range(len(vectors)):
14                if i == (len(vectors) - 1):
15                    self.edges.append(Edge(vectors[i], vectors[0]))
16                else:
17                    self.edges.append(Edge(vectors[i], vectors[i + 1]))
```

Класс визуализатор структуры маски для распознавания:

```
1 class Edge(object):
2     def __init__(self, v1:PolygonVector2Data, v2:PolygonVector2Data):
3         self.v1 = v1
4         self.v2 = v2
```

Описание сервера нейросети распознавание объектов

Сперва рассмотрим код сервера нейросети распознавания объектов, а именно файл *main.py*.



```

96 @app.get("/status", response_model=StatusInfo, summary="Получить статус нейронной сети")
97 def status():
98     return StatusInfo(status=app_state.nn.status.name)
99
100
101 @app.post("/set-model", response_model=StatusInfo, summary="Установить модель 'yolov8n.pt', 'yolov8s.pt', 'yolov8m.pt', 'yolov8l.pt', 'yolov8x.pt'")
102 async def set_model(name: ChangeModelRequest):
103     await app_state.set_model(name.name)
104     return StatusInfo()
105
106 @app.post("/predict-to-data", response_model=DetectedObjects, summary="Обнаружение объектов на изображении")
107 async def predict_to_data(image: UploadFile = File(..., description="Загружаемое изображение для анализа"),
108     polygons: Polygons = Body(None, description="Список найденных объектов")):
109     response = await app_state.predict_to_data(image, polygons)
110     return response
111
112 @app.post(
113     "/predict-to-image",
114     summary="Обнаружение объектов на изображении",
115     responses={
116         200: {
117             "content": {
118                 "image/png": {
119                     "example": "Изображение"
120                 }
121             },
122             "description": "Successful Response",
123         }
124     },
125     response_class=Response
126 )
127 async def predict_to_image(image: UploadFile = File(..., description="Загружаемое изображение для анализа")):
128     response = await app_state.predict_to_image(image)
129     return Response(content=response, media_type="image/png")
130
131
132 @app.post("/train", response_model=StatusInfo, summary="Начать тренировку")
133 async def train(train_data: TrainParams):
134     await app_state.train(train_data, restart)
135     return StatusInfo()
136
137
138 @app.get("/training-metrics/{training_name}", response_model=TrainingMetrics, summary="Получить результаты тренировки")
139 async def get_training_metrics(training_name: str = Path(..., description="Имя тренировки")):
140     response = app_state.get_training_metrics(training_name)
141     return response
142
143

```

Сервер распознавания

Метод жизненного цикла сервера:

1	@asynccontextmanager
2	async def lifespan(app: FastAPI):
3	setup_rich_logger()
4	await app_state.set_model(app_state.default_models[0])
5	yield

Инициализация и запуск сервера:

```
1 app = FastAPI(  
2     title="PLNeuro",  
3     description="API для нейронных сетей",  
4     version="1.0.0",  
5     lifespan=lifespan,  
6     docs_url=None,  
7     redoc_url=None  
8 )
```

Метод перезапуска сервера, при получении запроса о перезапуске перезапускает сервер:

```
1 @app.post("/restart", summary="Перезапустить сервер")  
2 def restart():  
3     os.kill(os.getpid(), signal.SIGINT)  
4     return StatusInfo()
```

Метод пинга сервера, при получении запроса отправляет информацию о состоянии:

```
1 app.get("/ping", response_model=StatusInfo, summary="Пинг сервера")  
2 def ping():  
3     return StatusInfo()
```

Функция вызова метода установки используемой модели нейросети:

```
1 app.post("/set-model", response_model=StatusInfo, summary="Установить  
2 модель 'yolov8n.pt', 'yolov8s.pt', 'yolov8m.pt', 'yolov8l.pt', 'yolov8x.pt'")  
3 async def set_model(name: ChangeModelRequest):  
4     await app_state.set_model(name.name)  
     return StatusInfo()
```

Функция вызова метода обнаружения объектов на изображении:

```
1 @app.post("/predict-to-data", response_model=DetectedObjects,
2 summary="Обнаружение объектов на изображении")
3 async def predict_to_data(image: UploadFile = File(...,
4 description="Загружаемое изображение для анализа"),
5 polygons: Polygons = Body(None, description="Список найденных
6 объектов")):
7     response = await app_state.predict_to_data(image, polygons)
8     return response
9
10 @app.post(
11     "/predict-to-image",
12     summary="Обнаружение объектов на изображении",
13     responses={
14         200: {
15             "content": {
16                 "image/png": {
17                     "example": "Изображение"
18                 }
19             },
20             "description": "Successful Response",
21         }
22     },
23     response_class=Response
24 )
25 async def predict_to_image(image: UploadFile = File(...,
26 description="Загружаемое изображение для анализа")):
27     response = await app_state.predict_to_image(image)
28     return Response(content=response, media_type="image/png")
```

Функция вызова метода обучения нейросети:

```
@app.post("/train", response_model=StatusInfo, summary="Начать
тренировку")
async def train(train_data: TrainParams):
    await app_state.train(train_data, restart)
    return StatusInfo()
```

Функция вызова метода получения метрик обучения нейросети:

```
1 @app.get("/training-metrics/{training_name}",
2 response_model=TrainingMetrics, summary="Получить результаты тренировки")
3 async def get_training_metrics(training_name: str = Path(..., description="Имя
4 тренировки")):
5     response = app_state.get_training_metrics(training_name)
6     return response
```

Функция вызова метода получения информации о выбранной модели:

```
1 app.get("/model", response_model=ModelInfo, summary="Получить
2 информацию об активированной модели")
3 async def get_model():
4     response = app_state.get_model_info()
5     return response
```

Функция вызова метода получения информации о результатах тренировок:

```
1 @app.get("/training-results", response_model=TrainingResults,
2 summary="Получить список результатов тренировок")
3 async def training_results():
4     response = app_state.get_training_results()
5     return response
```

Функция вызова метода удаления результатов тренировки:

```
1 @app.delete("/training-results/{training_name}", response_model=StatusInfo,
2 summary="Удалить результат тренировки")
3 async def delete_training_result(training_name: str = Path(..., description="Имя
4 тренировки")):
5     await app_state.delete_training_result(training_name)
6     return StatusInfo()
```

Функция вызова метода получения списка моделей:

```
1 @app.get("/models", response_model=ModelsNames, summary="Получить
2 список существующих моделей")
3 async def get_models():
4     return ModelsNames(models=[
5         ModelNameInfo(modelName=file_name)
6         for file_name in app_state.get_models()
7     ])
```

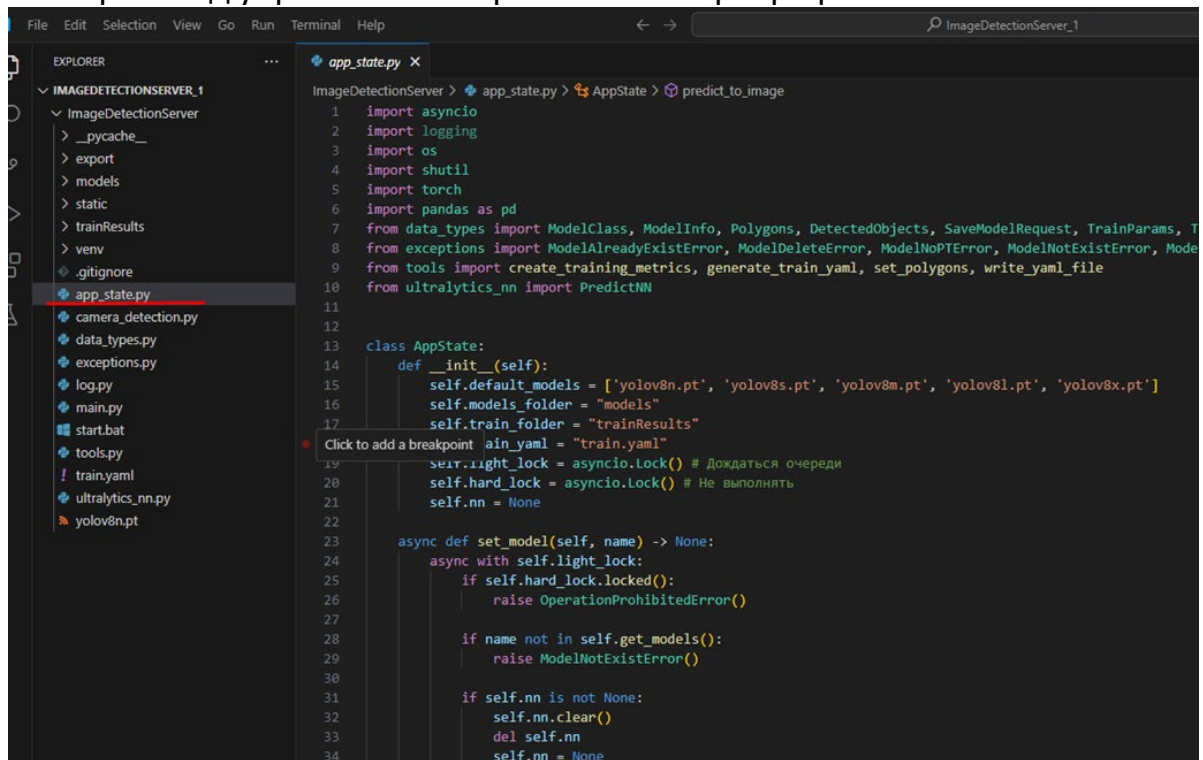
Функция вызова метода удаления модели:

```
1 @app.delete("/models/{model_name}", response_model=StatusInfo,
2 summary="Удалить модель")
3 async def delete_model(model_name: str = Path(..., description="Имя
4 модели")):
5     await app_state.delete_model(model_name)
6     return StatusInfo()
```

Функция вызова метода сохранения модели по результатам тренировки:

```
1 @app.post("/save-training-model", response_model=StatusInfo,
2 summary="Сохранить модель из результата тренировки")
3 async def save_training_model(request: SaveModelRequest):
4     await app_state.save_training_model(request)
5     return StatusInfo()
6
```

Рассмотрим код управления нейросетью на сервере распознавания.



```

1  import asyncio
2  import logging
3  import os
4  import shutil
5  import torch
6  import pandas as pd
7  from data_types import ModelClass, ModelInfo, Polygons, DetectedObjects, SaveModelRequest, TrainParams, T
8  from exceptions import ModelAlreadyExistError, ModelDeleteError, ModelNoModelError, ModelNotExistError, Mode
9  from tools import create_training_metrics, generate_train_yaml, set_polygons, write_yaml_file
10 from ultralytics_nn import PredictNN
11
12
13 class AppState:
14     def __init__(self):
15         self.default_models = ['yolov8n.pt', 'yolov8s.pt', 'yolov8m.pt', 'yolov8l.pt', 'yolov8x.pt']
16         self.models_folder = "models"
17         self.train_folder = "trainResults"
18         self.train_yaml = "train.yaml"
19         self.light_lock = asyncio.Lock() # Дождаться очереди
20         self.hard_lock = asyncio.Lock() # Не выполнять
21         self.nn = None
22
23     async def set_model(self, name) -> None:
24         async with self.light_lock:
25             if self.hard_lock.locked():
26                 raise OperationProhibitedError()
27
28             if name not in self.get_models():
29                 raise ModelNotExistError()
30
31             if self.nn is not None:
32                 self.nn.clear()
33                 del self.nn
34                 self.nn = None

```

Управление нейросетью

Инициализация переменных используемых в лабораторной работе:

1	def __init__(self):
2	self.default_models = ['yolov8n.pt', 'yolov8s.pt', 'yolov8m.pt', 'yolov8l.pt',
3	'yolov8x.pt']
4	self.models_folder = "models"
5	self.train_folder = "trainResults"
6	self.train_yaml = "train.yaml"
7	self.light_lock = asyncio.Lock() # Дождаться очереди
8	self.hard_lock = asyncio.Lock() # Не выполнять
	self.nn = None

Метод установки модели:

```
1     async def set_model(self, name) -> None:
2         async with self.light_lock:
3             if self.hard_lock.locked():
4                 raise OperationProhibitedError()
5
6             if name not in self.get_models():
7                 raise ModelNotExistError()
8
9             if self.nn is not None:
10                self.nn.clear()
11                del self.nn
12                self.nn = None
13                torch.cuda.empty_cache()
14
15                self.nn = PredictNN(f"{self.models_folder}/{name}")
16                return
```

Метод загрузки изображения для определения:

```
1     async def predict_to_image(self, image) -> bytes:
2         async with self.light_lock:
3             if self.hard_lock.locked():
4                 raise OperationProhibitedError()
5
6             results_image = await self.nn.predictToImage(image)
7             return results_image
```

Метод обучения нейросети:

```
1     async def train(self, train_data: TrainParams, on_train_end) -> None:
2         async with self.light_lock:
3             if self.hard_lock.locked():
4                 raise OperationProhibitedError()
5             if train_data.trainingName in self.get_training_results():
6                 raise TrainingResultNotExistError()
7             if train_data.modelName not in self.get_models():
8                 raise ModelNotExistError()
9
10
11                 write_yaml_file(self.train_yaml,
12 generate_train_yaml(train_data.datasetPath, "train", "val", train_data.classes))
13
14                 self.set_model(train_data.modelName)
15                 asyncio.create_task(self.nn.train(train_data, self.train_yaml,
self.train_folder, self.hard_lock, on_train_end))
16
17                 return
```

Метод получения метрик обучения:

```
1 def get_training_metrics(self, training_name: str) -> TrainingMetrics:
2     folder_names = [
3         folder_name
4         for folder_name in os.listdir(self.train_folder)
5         if os.path.isdir(os.path.join(self.train_folder, folder_name))
6     ]
7
8     if training_name not in folder_names:
9         raise TrainingResultNotExistError()
10
11     try:
12         results_dir = os.path.join(self.train_folder, training_name,
13 "results.csv")
14
15         if os.path.exists(results_dir):
16             df = pd.read_csv(results_dir, skipinitialspace=True)
17         else:
18             df = pd.DataFrame()
19
20         metrics = create_training_metrics(df)
21         return metrics
22     except Exception as e:
23         raise TrainingMetricsNotExistError()
```

Метод получения информации о результатах тренировки:

```
1 async def delete_training_result(self, training_name: str) -> None:
2     async with self.light_lock:
3         if self.hard_lock.locked():
4             raise OperationProhibitedError()
5         train_results_dir = self.train_folder
6         folder_path = os.path.join(train_results_dir, training_name)
7         if os.path.exists(folder_path):
8             try:
9                 shutil.rmtree(folder_path)
10                return
11            except:
12                raise TrainingResultDeleteError()
13        else:
14            raise TrainingResultNotExistError()
```


Метод удаления модели:

```
1     async def delete_model(self, model_name: str) -> None:
2         async with self.light_lock:
3             if self.hard_lock.locked():
4                 raise OperationProhibitedError()
5             if model_name not in self.get_models():
6                 raise ModelNotExistError()
7
8             if model_name in self.default_models:
9                 raise UnableDeleteDefaultModelError()
10
11             model_path = os.path.join(self.models_folder, model_name)
12
13             if os.path.exists(model_path):
14                 try:
15                     os.remove(model_path)
16                 except:
17                     raise ModelDeleteError()
18                 return
19             else:
20                 raise ModelNotExistError()
```

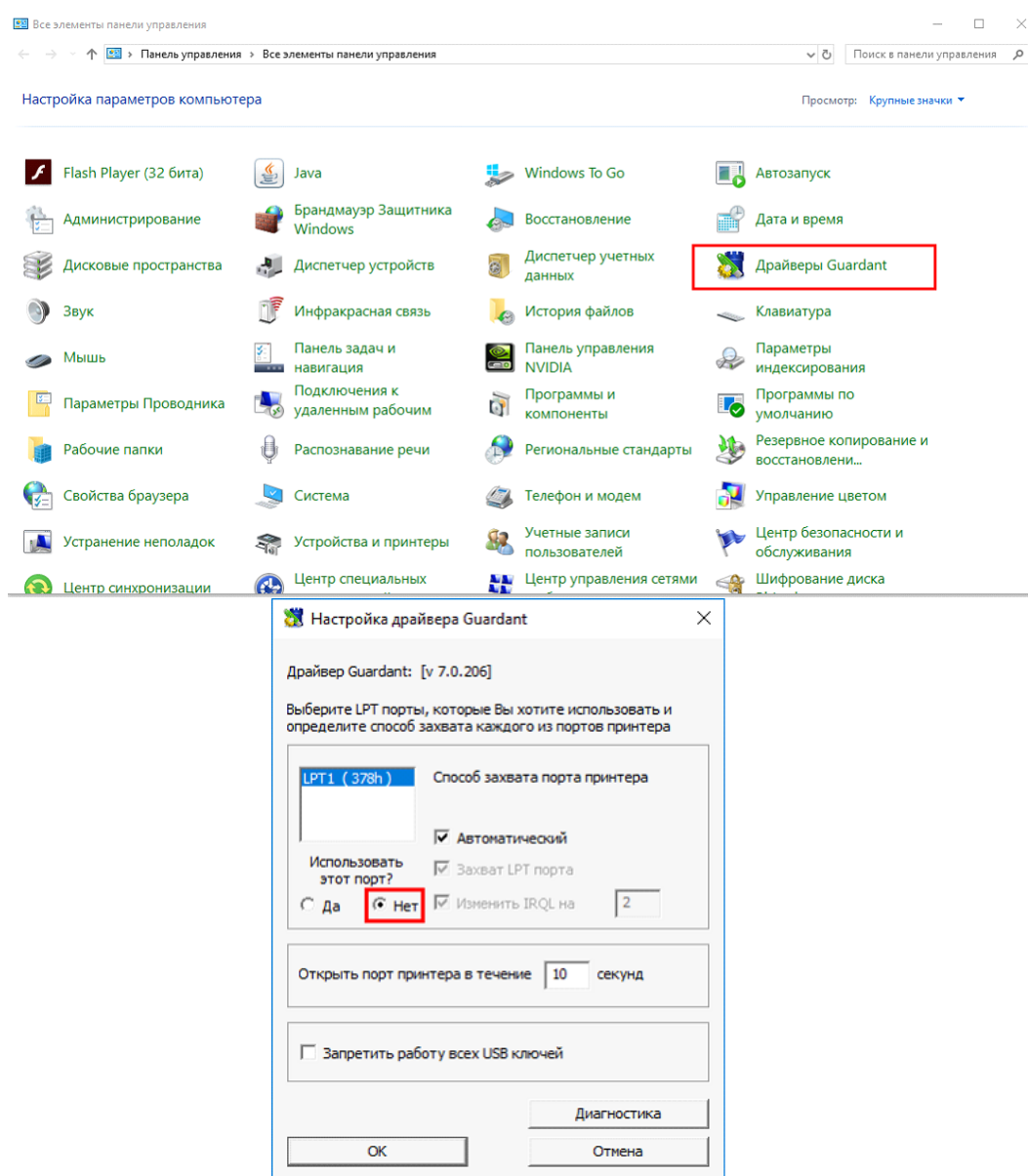
Метод удаления модели:

```
1     async def save_training_model(self, request: SaveModelRequest) -> None:
2         async with self.light_lock:
3             if self.hard_lock.locked():
4                 raise OperationProhibitedError()
5                 training_result_path = os.path.join(self.train_folder,
6 request.trainingResultName)
7                 if not os.path.exists(training_result_path):
8                     raise TrainingResultNotExistError()
9
10                weight_file_path = os.path.join(training_result_path, "weights",
11 request.weightsName)
12                if not os.path.exists(weight_file_path):
13                    raise WeightsNotExistError()
14
15                new_model_path = os.path.join(self.models_folder,
16 request.newModelName)
17                if os.path.exists(new_model_path):
18                    raise ModelAlreadyExistError()
19
20                if not request.newModelName.endswith(".pt"):
21                    raise ModelNoPTError()
22                try:
23                    shutil.copy(weight_file_path, new_model_path)
24                except:
25                    raise ModelSaveError()
26
27                return
```

Устранение проблем и ошибок

Проблема совместимости и её устранение

В случае если курсор мыши перемещается рывками во время работы с модулем запуска учебных комплексов, перейдите в панель управления и выберите подпункт **«Драйверы Guardant»**. Затем поставьте **«Нет»** в **«Использовать этот порт?»** и нажмите **«Ок»**.



Устранение проблем и ошибок дистанционно с помощью специалистов компании «Програмлаб»

При возникновении ошибок в работе с программным обеспечением свяжитесь со специалистом поддержки «Програмлаб». Для этого опишите вашу проблему в письме на почту support@pl-llc.ru либо позвоните по телефону 8 800 550 89 72.

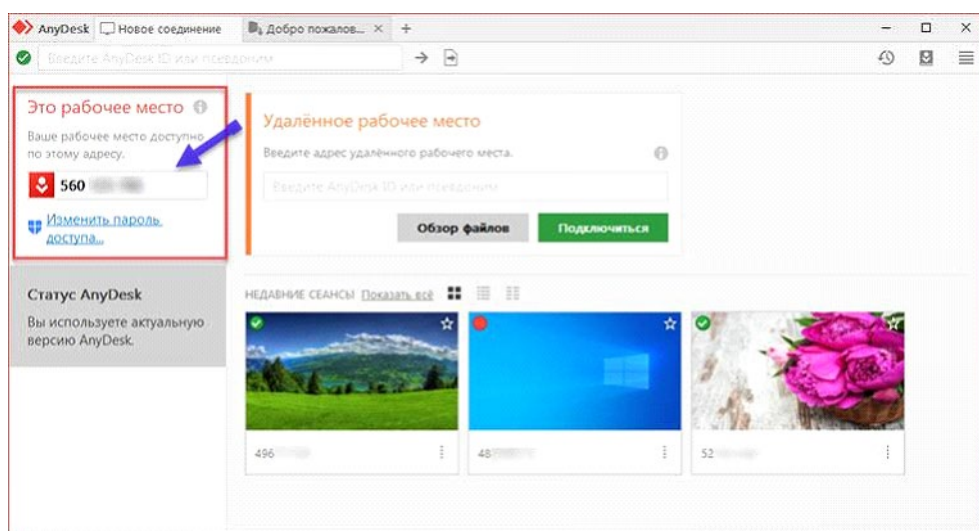
Для того чтобы специалист смог подключиться к вашему ПК и устранить проблемы вам необходимо запустить ПО для дистанционного управления ПК Anydesk и сообщить данные для доступа.

Приложение Anydesk можно найти на USB-носителе с дистрибутивом. Вставьте USB-носитель в ПК и запустите файл с названием Anydesk.exe

После того как приложение скачано нужно запустить его. Необходимый файл называется **AnyDesk.exe** и лежит папке «**Загрузки**».

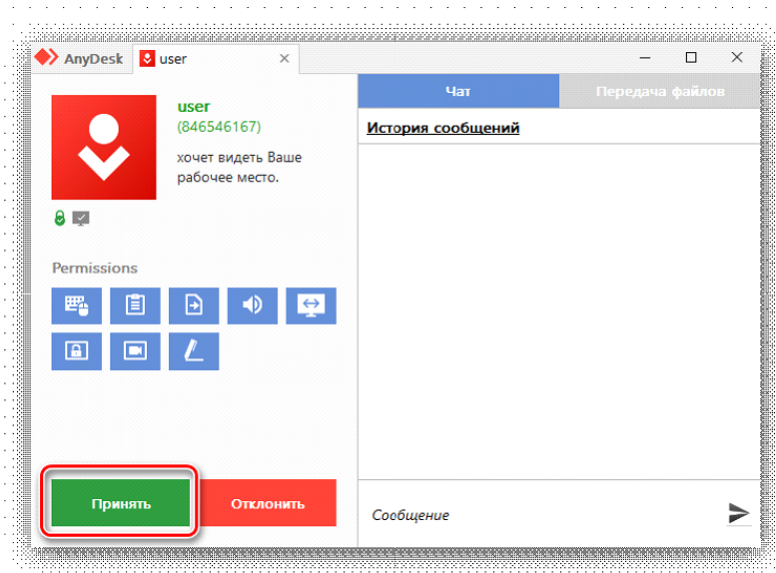
При первом запуске может возникнуть окно с требованием предоставить разрешение. Необходимо нажать на кнопку **Разрешить доступ**.

Для того, чтобы к вашему компьютеру мог подключиться другой пользователь, необходимо ему передать специальный адрес, который называется «Это рабочее место». Сообщите этот адрес специалисту.



Окно Anydesk с адресом

После того как специалист введет переданный вами адрес вам нужно будет подтвердить разрешение на доступ к вашему ПК. Откроется табличка с вопросом «Принять» или «Отклонить» удаленное соединение. Нажмите «Принять».



Окно Anydesk Принять/Отклонить

На этом настройка удаленного соединения завершена: специалист получил доступ к вашему ПК. В случае необходимости продолжайте следовать инструкциям специалиста.



Sk
Resident

**ВИРТУАЛЬНЫЕ ЛАБОРАТОРИИ
ТРЕНАЖЕРЫ - СИМУЛЯТОРЫ
ИНТЕРАКТИВНЫЕ МАКЕТЫ
ЛАБОРАТОРНЫЕ СТЕНДЫ
ЦИФРОВЫЕ ДВОЙНИКИ
VR И AR КОМПЛЕКСЫ**

