



PROGRAMLAB
INNOVATIVE DIGITAL SYSTEMS

РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

**ВИРТУАЛЬНЫЙ УЧЕБНЫЙ КОМПЛЕКС
«РАСПОЗНАВАНИЕ И КЛАССИФИКАЦИЯ
ОБЪЕКТОВ, КОНТРОЛЬ КАЧЕСТВА ПРОДУКЦИИ
С ПОМОЩЬЮ ТЕХНОЛОГИЙ МАШИННОГО
ОБУЧЕНИЯ»**

ОГЛАВЛЕНИЕ

Инструкция по установке и запуску проекта.....	3
Запуск и управление в программе	5
Устранение проблем и ошибок	7
Введение в нейронные сети.....	9
Установка и настройка сервера	25
Работа в программе.....	34
Запуск физического стенда	60
Спецификация API сервера нейронной сети	63
API RabbitMQ	68
Описание лабораторной работы.....	72

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.

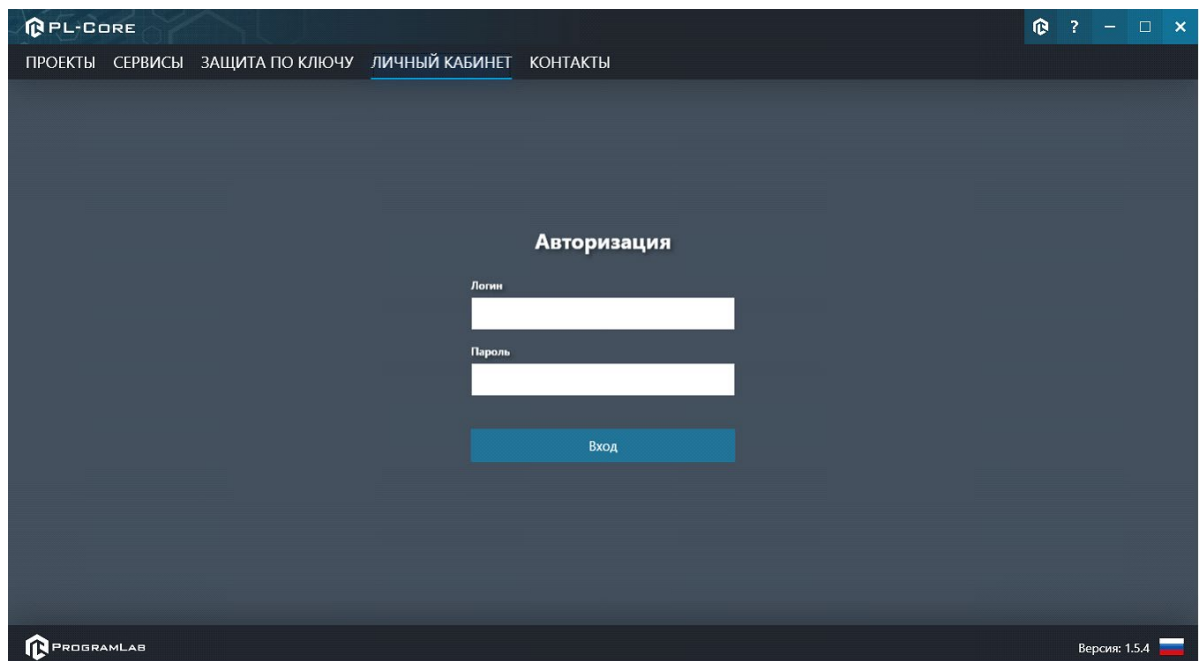
В случае поставки программного комплекса без ПК вам необходимо установить программное обеспечение с USB-носителя.

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».

ТУТ ПОНАДОБИТСЯ ЛОГИН И ПАРОЛЬ ИЗ КОНВЕРТА.

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. Установите «**PLStudy**» – Администрирование сервера данных учебных модулей.

Если ваш стенд предполагает автоматическую отправку результатов, а также систему ролей пользователей для работы группы, то вам понадобится программный модуль «Администрирование сервера данных учебных модулей». Модуль позволяет управлять базой данных студентов и их результатов для всех комплексов нашей компании сразу.

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

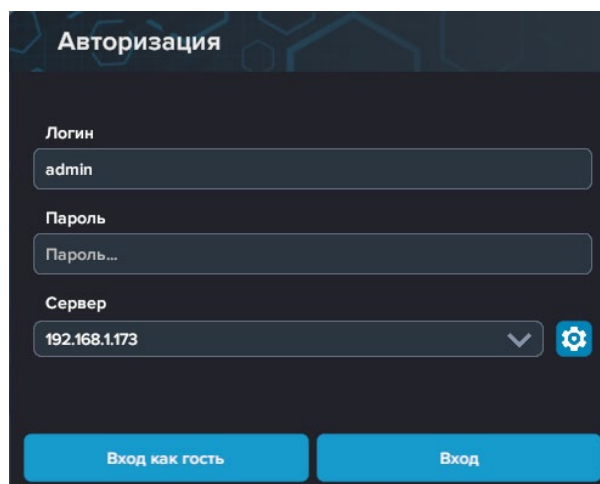
По умолчанию в системе создается пользователь с именем Администратор и ролью Администратор. Этот пользователь не может быть удален, но его параметры могут быть изменены.

По умолчанию логин пользователя: admin; Пароль: admin.

6. 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 сервер данных учебных модулей.

Чтобы изменить IP-адрес см. пункт «Запуск и управление в модуле» в руководстве пользователя «**PLStudy**».



Окно авторизации

Запуск и управление в программе



— Левая кнопка мыши – действие, выбор объекта;



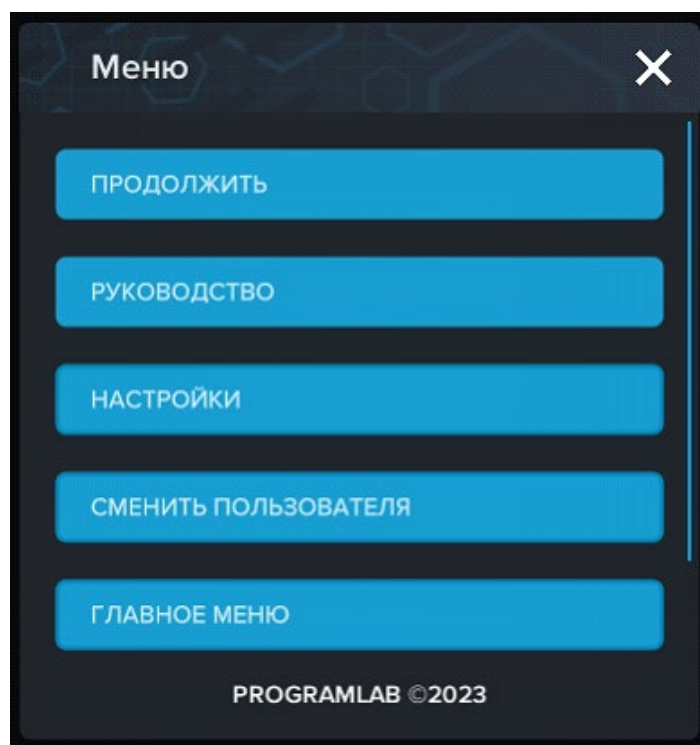
— Правая кнопка мыши – вращение камеры;



— Вращение колеса мыши – приближение\отдаление камеры;



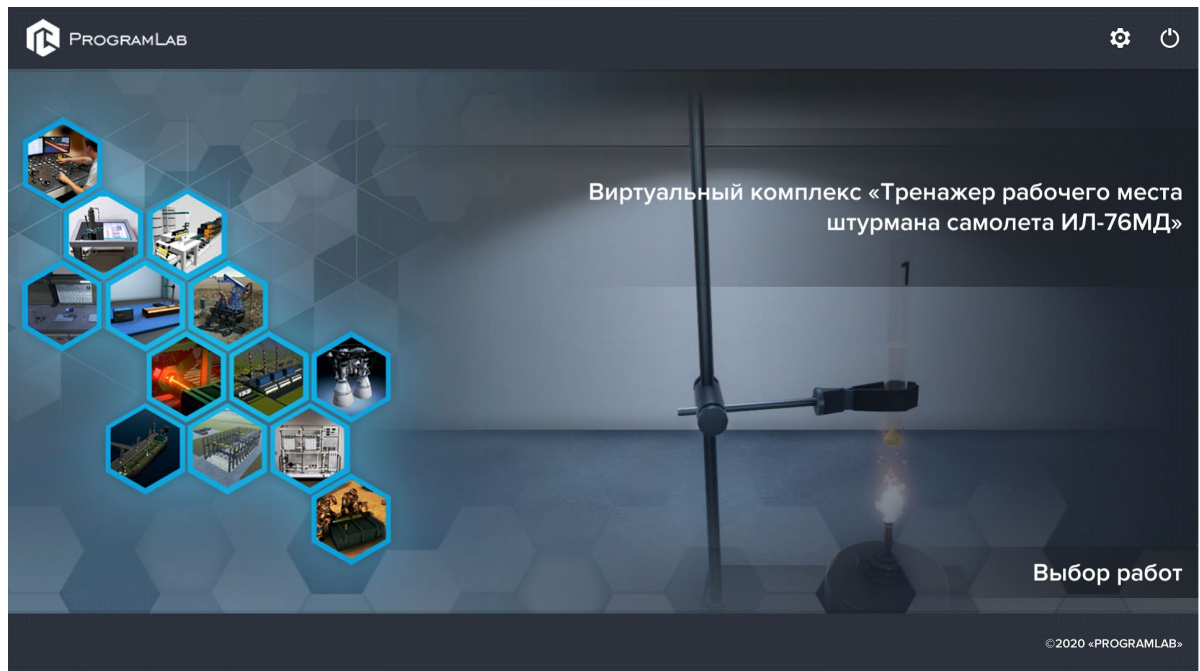
— Вызов меню программы.



Меню программы

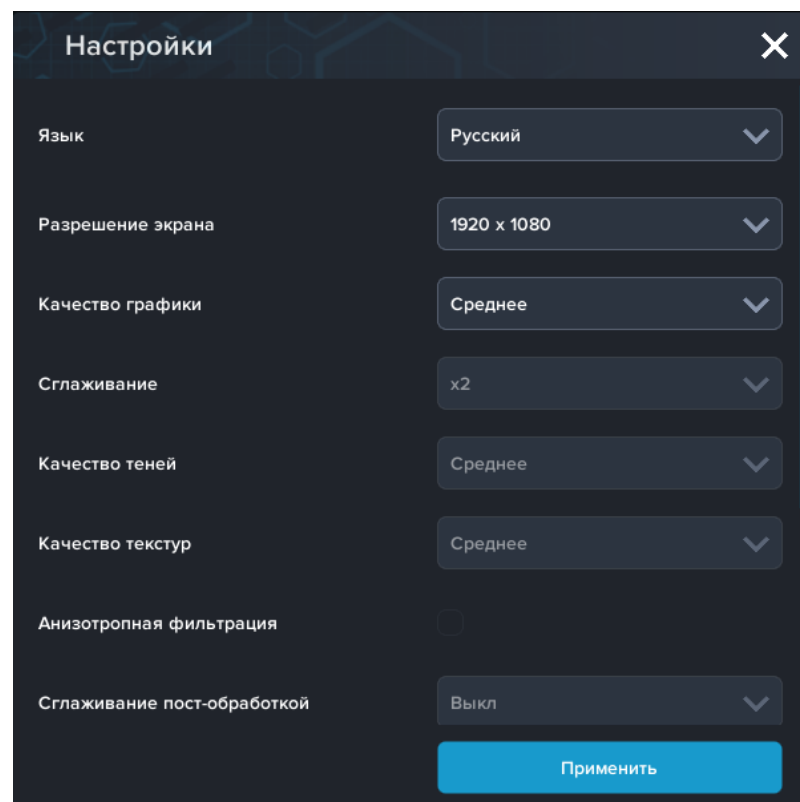
- «Продолжить» – вернуться в программу;
- «Руководство» – вызвать руководство пользователя;
- «Настройки» – настройки параметров графики;
- «Сменить пользователя» – пройти авторизацию повторно;
- «Главное меню» – выход в главное меню;
- «Выход» – выход из программы.

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 .

Устранение проблем и ошибок

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.

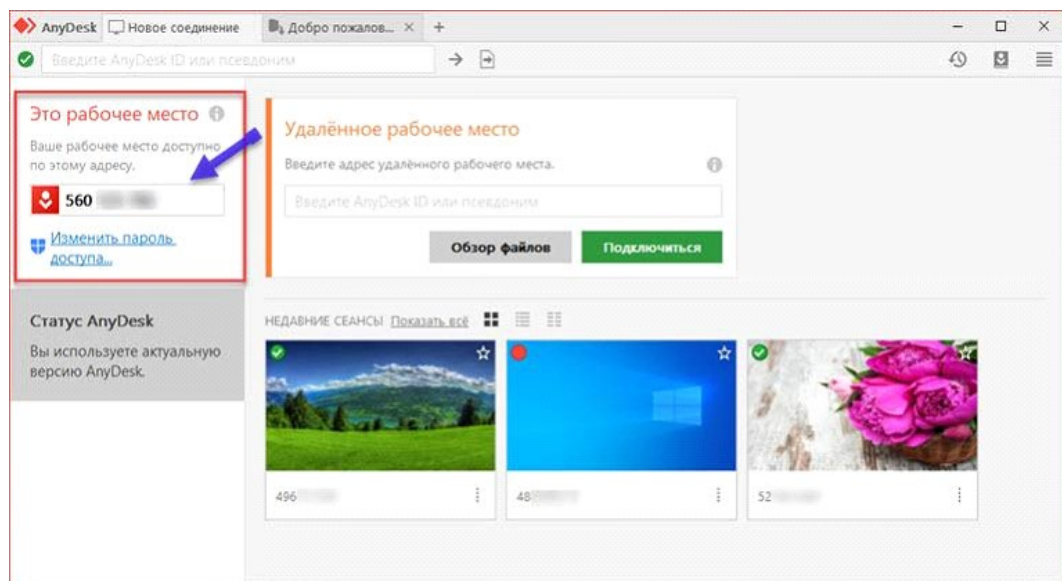
Для того чтобы специалист смог подключиться к вашему ПК и устранить проблемы вам необходимо запустить ПО для дистанционного управления ПК Anydesk и сообщить данные для доступа.

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».

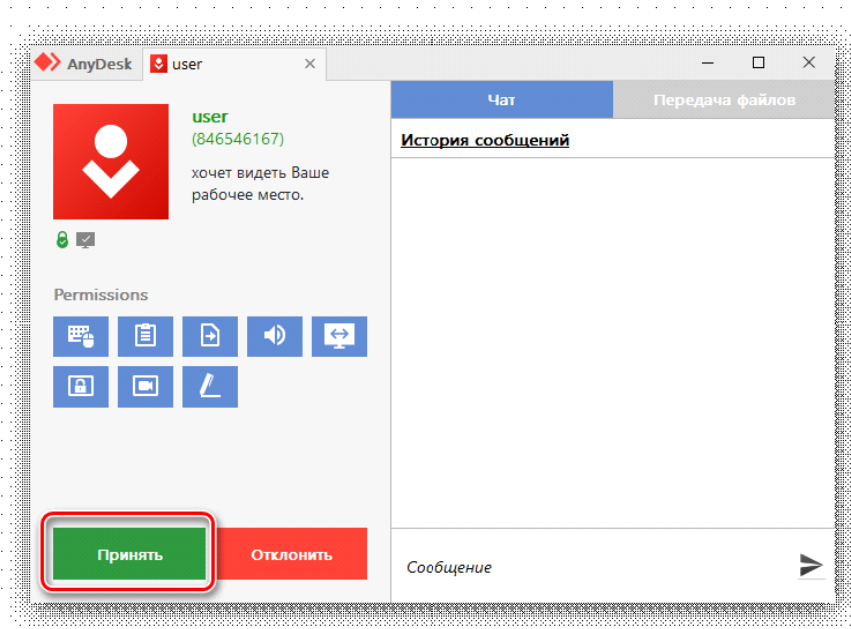
При первом запуске может возникнуть окно с требованием предоставить разрешение. Необходимо нажать на кнопку **Разрешить доступ**.

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.

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.

Первую 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.

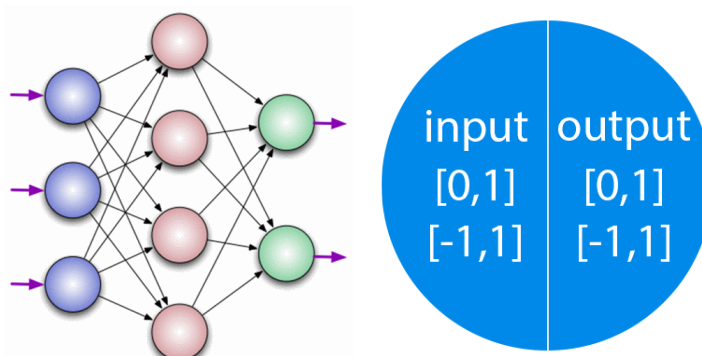
Современные GPU позволили развивать «глубокое обучение» — повышать глубину слоев нейросети. Именно благодаря ему появились самообучаемые нейросети, которые не требуют специальной настройки, а самостоятельно обрабатывают входящую информацию.

Структура нейросети



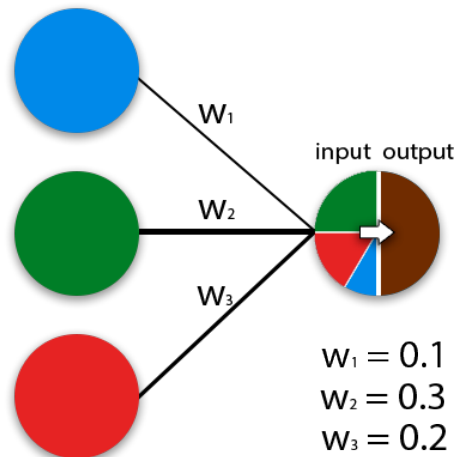
Главное отличие нейросетевых моделей от классических заключается в их структуре. Основные элементы, из которых он состоит – искусственные нейроны и связи между ними.

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.

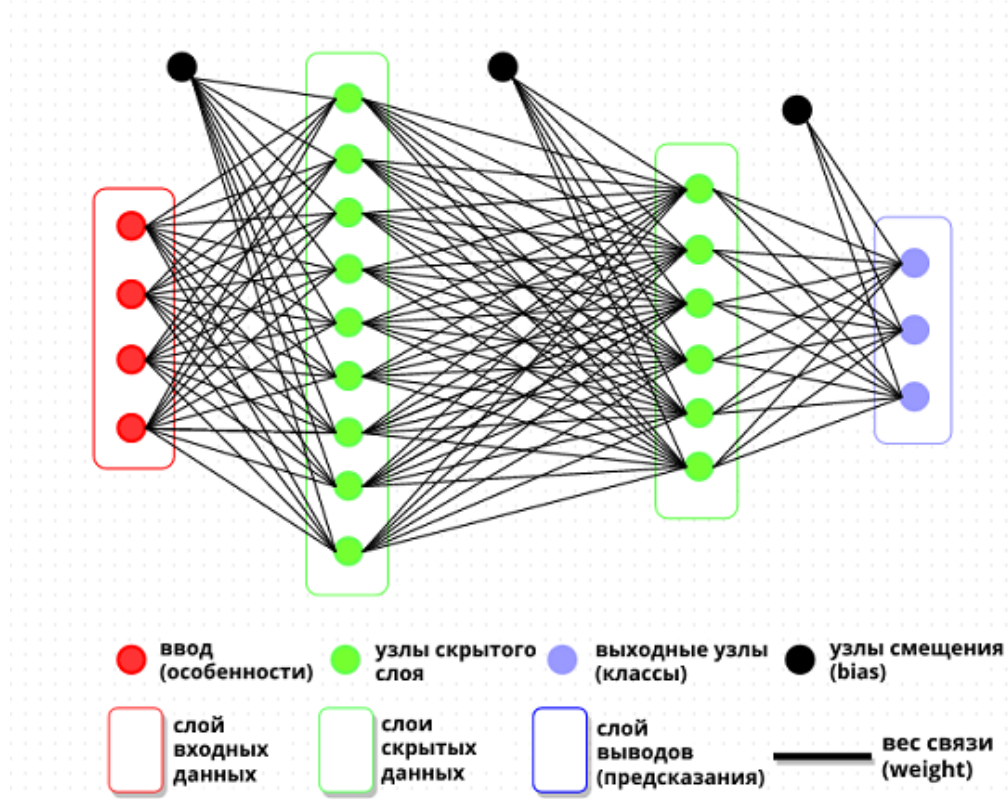
Искусственный интеллект — понятие более широкое. Оно включает в себя не только нейронные сети, но и другие методы обработки информации, в том числе экспертные и логические программы. Нейронные сети — один из видов искусственного интеллекта. Их отличительная особенность — обучение и адаптация в основе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

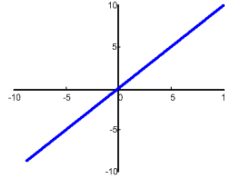
Чем точнее и подробнее запрос, тем быстрее и качественнее получится результат.

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

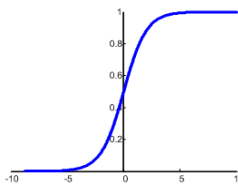
$$f(x) = x$$



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

Сигмоид

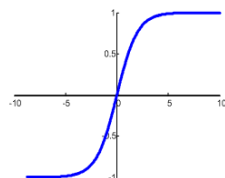
$$f(x) = \frac{1}{1 + e^{-x}}$$



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. Эпоха увеличивается каждый раз, когда мы проходим весь набор тренировочных сетов.

 `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:

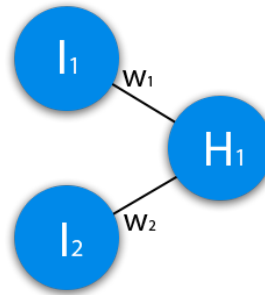
$$\sqrt{\frac{(i_1 - a_1)^2 + (i_2 - a_2)^2 + \dots + (i_n - a_n)^2}{n}}$$

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.

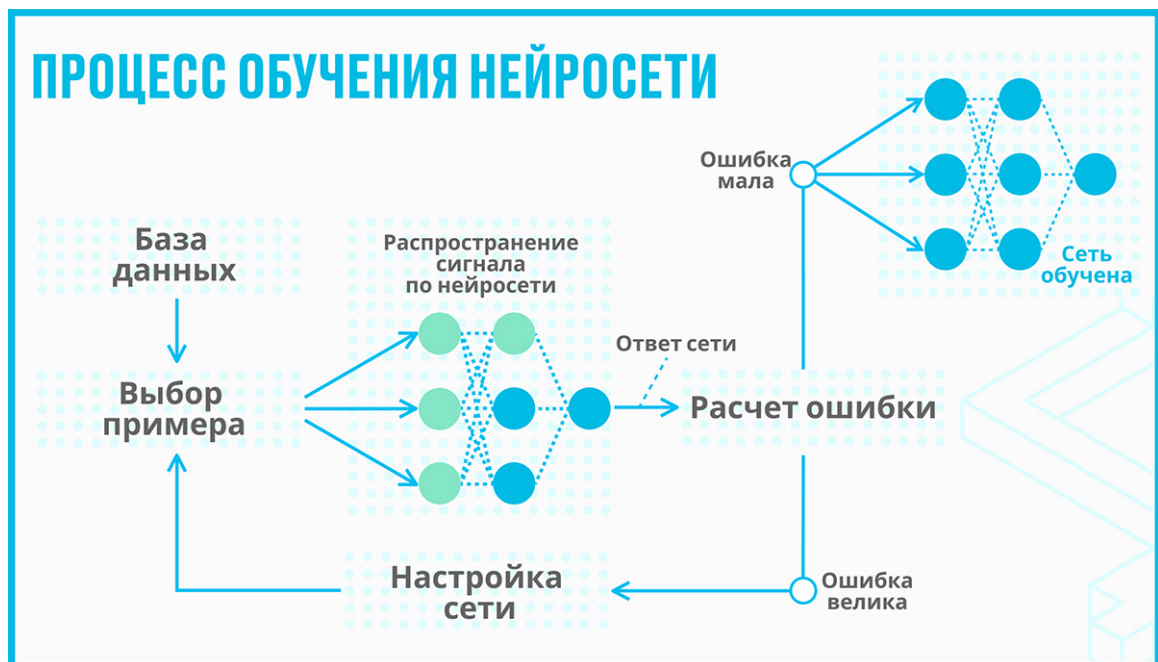
Существуют следующие методы обучения:



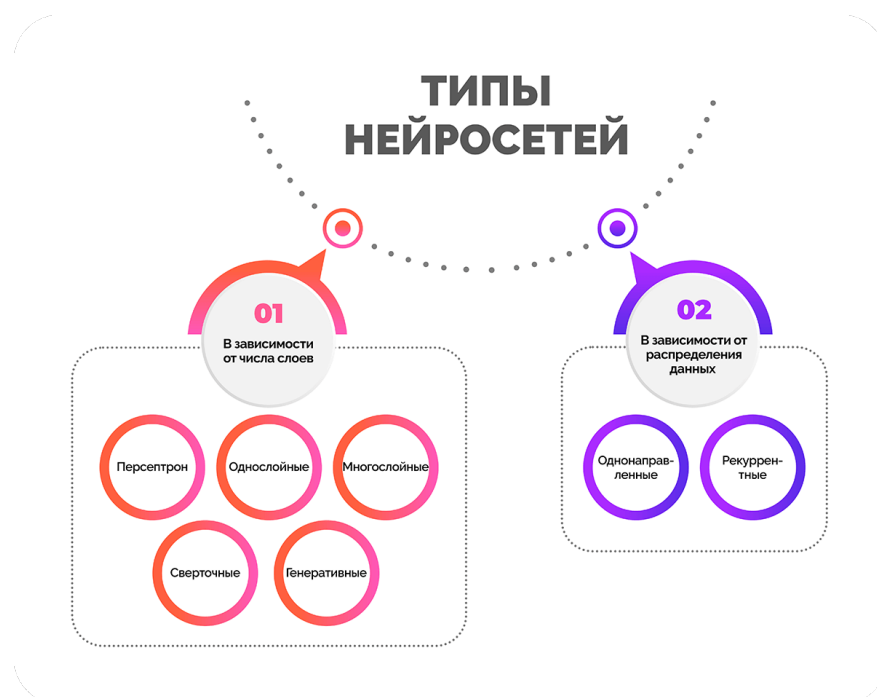
- **С учителем.** Пользователь дает сигнал на вход, получает на выходе ответ нейросети, затем сравнивает его с уже известным правильным. После этого с помощью специальных алгоритмов меняются веса связей и снова задается входной сигнал. Процесс продолжается до тех пор, пока нейросеть не начнет отвечать 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.
- **Рекуррентные** (с обратными связями). Рекуррентные 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:

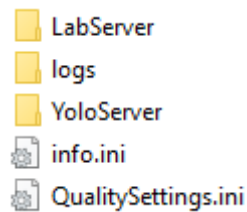
1. **Медицина.** Искусственный интеллект помогает обрабатывать снимки и другие данные исследований и тем самым позволяет врачам устанавливать точный диагноз, при этом тратить меньше времени.
2. **Образование.** Преподаватели с помощью искусственных сетей имеют возможность быстрее проверять домашние задания, 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 маркетплейсах, фотографии для социальных сетей.

Установка и настройка сервера

После установки ПО в директории по умолчанию C:\Users\Public\Documents\ProgramLab\ObjectRecognitionClassification\Base_M будут располагаться папки **LabServer** и **YoloServer**.



Содержимое папки Base_M

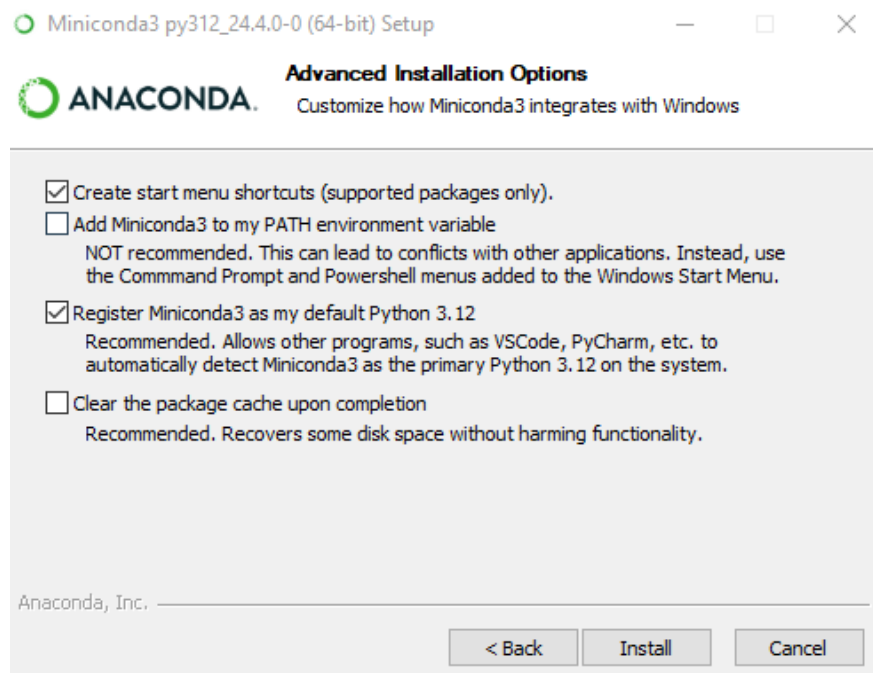
В папке **YoloServer** – сервер нейронной сети.

В папке **LabServer** – сервер лабораторной работы.

Установка и настройка сервера нейронной сети

Перед установкой сервера нейронной сети установите программу Miniconda, скачать можно по ссылке: <https://docs.anaconda.com/miniconda/>.

Запустите скачанный файл и следуйте подсказкам по установке. В следующем окне проверьте, что выбран рекомендуемый вариант установки:



Окно установки Miniconda

ВАЖНО: установите галочку во втором пункте данного окна:

Miniconda3 py312_24.4.0-0 (64-bit) Setup

ANACONDA. **Advanced Installation Options**
 Customize how Miniconda3 integrates with Windows

Create start menu shortcuts (supported packages only).

Add Miniconda3 to my PATH environment variable
 NOT recommended. This can lead to conflicts with other applications. Instead, use the Command Prompt and Powershell menus added to the Windows Start Menu.

Register Miniconda3 as my default Python 3.12
 Recommended. Allows other programs, such as VSCode, PyCharm, etc. to automatically detect Miniconda3 as the primary Python 3.12 on the system.


Clear the package cache upon completion
 Recommended. Recovers some disk space without harming functionality.


Anaconda, Inc.

< Back Install Cancel

Окно установки Miniconda

Откройте окно **Выполнить**, нажав комбинацию **Win+R**. Откроется следующее окно:

 **Выполнить** X

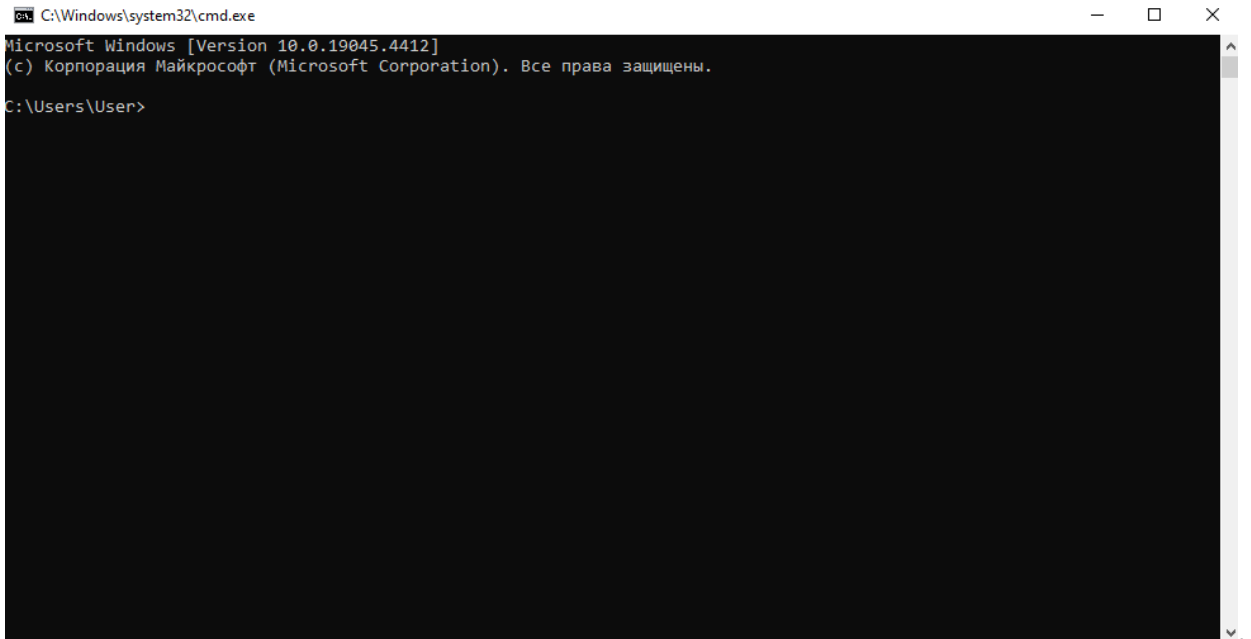
 Введите имя программы, папки, документа или ресурса Интернета, которые требуется открыть.

Открыть:

OK Отмена Обзор...

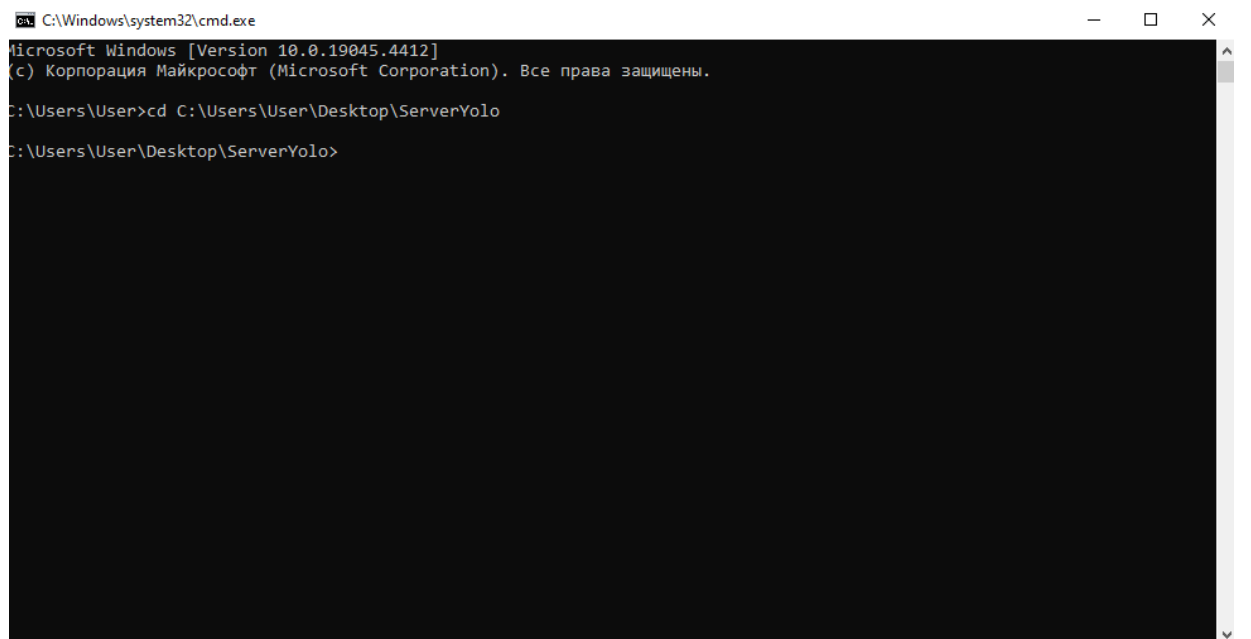
Окно Выполнить

В появившемся окне введите **cmd** для того, чтобы запустить командную строку:



Командная строка

Ведите команду **cd**, указав путь до папки сервера нейронной сети:



Командная строка

Введите команду **conda env create -p ./venv --file=./export/environmentBase.yml** и нажмите **Enter**.

Начнется загрузка необходимых библиотек, дождитесь окончания загрузки:

```

C:\Windows\system32\cmd.exe - conda env create -p ./venv --file=./export/environmentBase.yml

Downloading and Extracting Packages:
pytorch-2.3.1 | 1.21 GB | #####9 | 8%
libcublas-dev-12.1.0 | 348.3 MB | #####8 | 6%
libcuspars...-dev-12.0 | 162.5 MB | #####1 | 48%
libnpp-dev-12.0.2.50 | 135.6 MB | #####7 | 66%
mk1-2021.4.0 | 114.9 MB | ##### | 100%
libcufft-dev-11.0.2. | 102.6 MB | #####2 | 95%
libcusolver-dev-11.4 | 95.7 MB | | 0%
cuda-nvrtc-12.1.105 | 73.2 MB | | 0%
libnvjitlink-12.1.10 | 67.3 MB | | 0%
qt-main-5.15.2 | 59.4 MB | | 0%
qt-webengine-5.15.9 | 58.1 MB | | 0%
libcurand-10.3.6.39 | 49.3 MB | | 0%
opencv-4.6.0 | 29.9 MB | | 0%
libclang13-14.0.6 | 22.5 MB | | 0%
scipy-1.10.1 | 18.8 MB | | 0%
ffmpeg-4.2.2 | 17.6 MB | | 0%
cuda-nvrtc-dev-12.1. | 16.5 MB | | 0%
python-3.10.14 | 15.9 MB | | 0%
libnvjitlink-dev-12. | 13.8 MB | | 0%
pandas-2.2.2 | 12.0 MB | | 0%
hdf5-1.12.1 | 11.9 MB | | 0%
cuda-cupti-12.1.105 | 11.6 MB | | 0%
qtwebkit-5.212 | 11.6 MB | | 0%
sympy-1.12 | 10.5 MB | | 0%
icu-58.2 | 9.4 MB | | 0%
matplotlib-base-3.8. | 8.7 MB | | 0%
openssl-3.0.14 | 7.8 MB | | 0%
torchvision-0.18.1 | 7.6 MB | | 0%
... (more hidden) ...

```

Загрузка пакетов данных

При успешной загрузке, в консоли отобразится следующая информация:

```

C:\Windows\system32\cmd.exe

Downloading and Extracting Packages:

Preparing transaction: done
Verifying transaction: done
Executing transaction: done
Installing pip dependencies: \ Ran pip subprocess with arguments:
['C:\Users\User\Desktop\ServerYolo\venv\python.exe', '-m', 'pip', 'install', '-U', '-r', 'C:\Users\User\Desktop\ServerYolo\export\condaenv.z46hz61g.requirements.txt', '--exists-action=b']
Pip subprocess output:
Collecting opencv-python (from -r C:\Users\User\Desktop\ServerYolo\export\condaenv.z46hz61g.requirements.txt (line 1))
  Downloading opencv_python-4.10.0.84-cp37-abi3-win_amd64.whl.metadata (20 kB)
Requirement already satisfied: numpy>=1.21.2 in c:\users\user\desktop\serveryolo\venv\lib\site-packages (from opencv-python->-r C:\Users\User\Desktop\ServerYolo\export\condaenv.z46hz61g.requirements.txt (line 1)) (1.24.3)
Downloading opencv_python-4.10.0.84-cp37-abi3-win_amd64.whl (38.8 MB)
----- 38.8/38.8 MB 19.2 MB/s eta 0:00:00
Installing collected packages: opencv-python
Successfully installed opencv-python-4.10.0.84

done
#
# To activate this environment, use
#
#   $ conda activate C:\Users\User\Desktop\ServerYolo\venv
#
# To deactivate an active environment, use
#
#   $ conda deactivate
#
C:\Users\User\Desktop\ServerYolo>

```

Загрузка пакетов данных завершена

Запуск сервера нейронной сети

Для запуска сервера необходимо открыть файл **start.bat** в папке **YoloServer** – **данный .bat-файл запускает сервер нейронной сети**. В командной строке отобразится следующая информация о запущенном сервере:

```
C:\Windows\system32\cmd.exe
Activating conda environment...
Starting the FastAPI server...
INFO: Started server process [3832]
INFO: Waiting for application startup.
[15:52:17] INFO Application startup complete.
INFO Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
```

Запущенный сервер нейронной сети

Установка и настройка сервера лабораторной работы

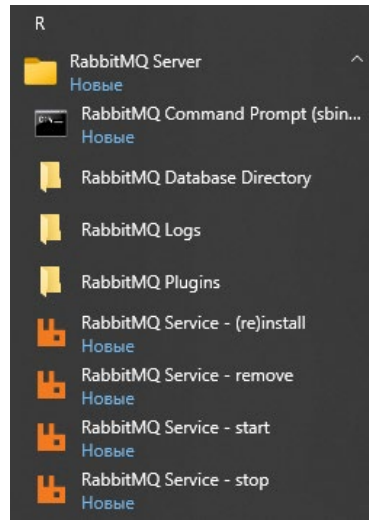
Перед установкой сервера лабораторной работы скачайте и установите следующие программы, следуя подсказкам (используйте стандартные настройки):

- **Erlang** (можно скачать по ссылке: <https://www.erlang.org/downloads>).
- **RabbitMQ** (можно скачать по ссылке: <https://www.rabbitmq.com/>).

Erlang устанавливается вначале, так как необходим для установки RabbitMQ. RabbitMQ - это программное обеспечение, которое помогает приложениям обмениваться сообщениями, организовывать обмен данными между различными частями системы (далее – **шина**).

Если при установке откроется окно оповещения брандмауэра Windows, разрешите программам связь в частных сетях, отметив галочкой соответствующий пункт.

После установки, у Вас в папке с установленной RabbitMQ и в меню **пуск** появятся следующие программы:



Файлы RabbitMQ

Затем откройте **командную строку RabbitMQ (RabbitMQ Command Prompt)**, расположенную в папке с программой, либо в панели **пуск** и введите команду ***rabbitmq-plugins enable rabbitmq_management***.

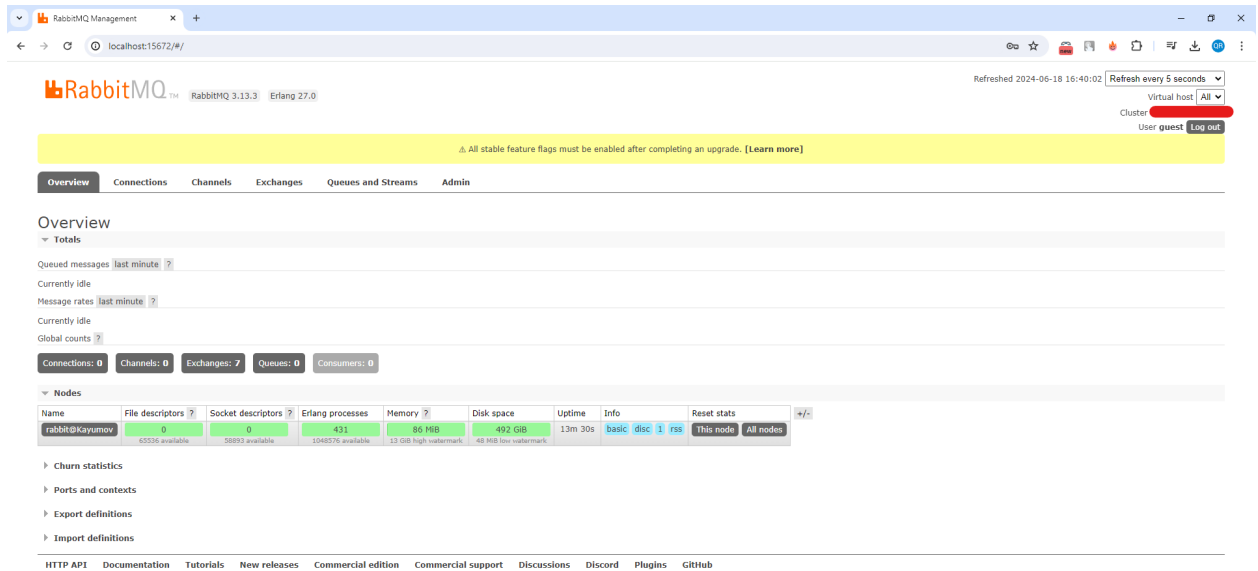
Далее перезагрузите RabbitMQ, для этого запустите в папке с программой, либо через меню **пуск RabbitMQ Service – stop**, затем, когда окно закроется, запустите **RabbitMQ Service – start**, так же, ожидая закрытия окна, это нужно будет сделать 1 раз.

Далее с помощью браузера перейдите по адресу <http://localhost:15672> у Вас откроется окно:



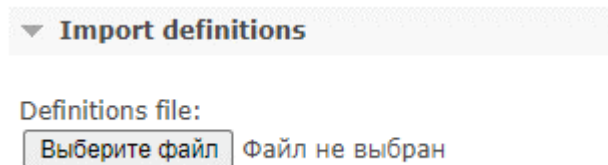
Окно веб-интерфейса RabbitMQ

Войдите в настройку шины, используя ***quest/quest***:

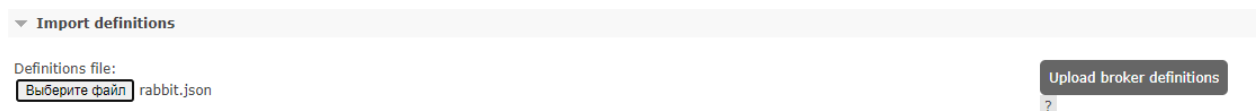


Окно веб-интерфейса RabbitMQ

Далее необходимо импортировать настройки для шины. Для этого в пункте **Import definitions** выберите файл из поставляемого USB-носителя и нажмите **Upload broker definitions**.



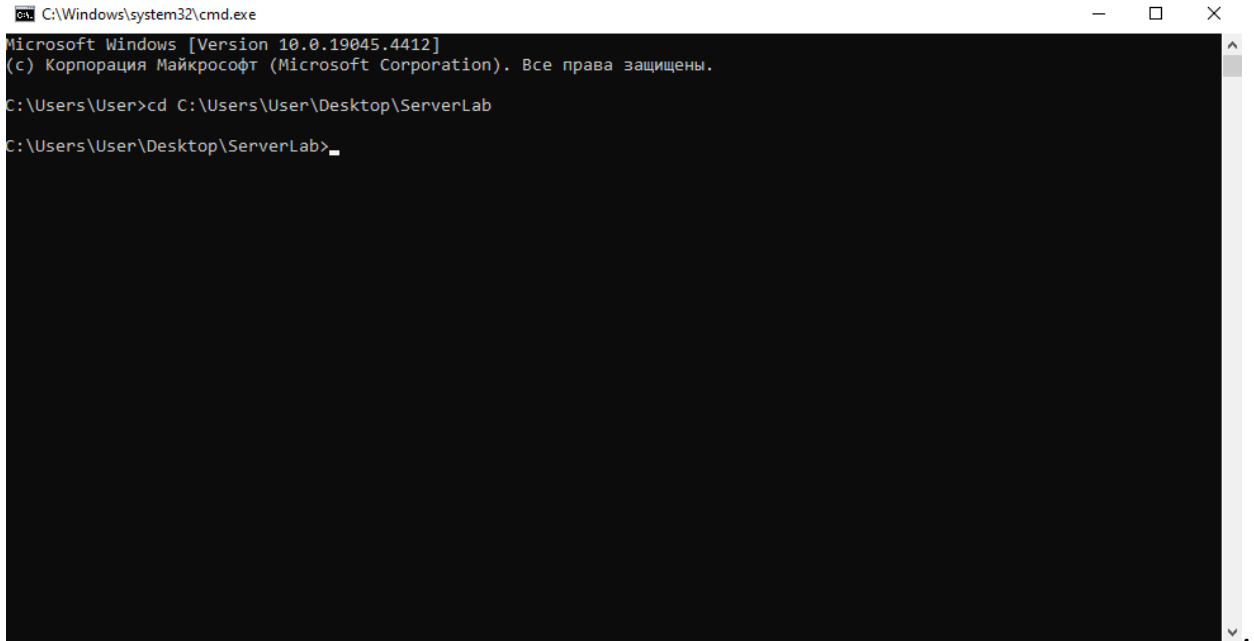
Окно Import definitions



Окно Import definitions с выбранным файлом

На данном этапе Вы успешно установили и настроили шину.

Откройте командную строку через команду **cmd**, введите команду **cd**, затем введите путь до папки с сервером лабораторной работы **ServerLab**:



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 10.0.19045.4412]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.
C:\Users\User>cd C:\Users\User\Desktop\ServerLab
C:\Users\User\Desktop\ServerLab>_
```

Командная строка

Введите команду ***conda env create -p ./venv --file=./export/environmentBase.yml*** и нажмите ***Enter***.

Начнется загрузка необходимых библиотек, дождитесь окончания загрузки.

При успешной загрузке, в консоли отобразится следующая информация:

```

C:\Windows\system32\cmd.exe
Microsoft Windows [Version 10.0.19045.4412]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\User>cd C:\Users\User\Desktop\ServerLab

C:\Users\User\Desktop\ServerLab>conda env create -p ./venv --file=./export/environmentBase.yml
Channels:
 - defaults
 - conda-forge
Platform: win-64
Collecting package metadata (repodata.json): done
Solving environment: done

Downloading and Extracting Packages:

Preparing transaction: done
Verifying transaction: done
Executing transaction: done

# To activate this environment, use
#
#   $ conda activate C:\Users\User\Desktop\ServerLab\venv
#
# To deactivate an active environment, use
#
#   $ conda deactivate
#

C:\Users\User\Desktop\ServerLab>
  
```

Загрузка пакетов данных завершена

Запуск сервера лабораторной работы

Для запуска сервера и необходимо открыть файл **start.bat** в папке **LabServer** – **данный .bat-файл запускает сервер лабораторной работы**. В командной строке отобразится следующая информация о запущенном сервере:

```

C:\Windows\system32\cmd.exe
Activating conda environment...
Starting the server...
[17:09:33] INFO    Listening on cameras_exchange #
INFO    Listening on lab_exchange lab.get_models
INFO    Listening on lab_exchange lab.get_model
INFO    Listening on lab_exchange lab.set_model
INFO    Listening on lab_exchange lab.set_enabled
INFO    Subscribed to cameras_detections_exchange #
rabbit.py:72
rabbit.py:45
rabbit.py:45
rabbit.py:45
rabbit.py:45
rabbit.py:12
  
```

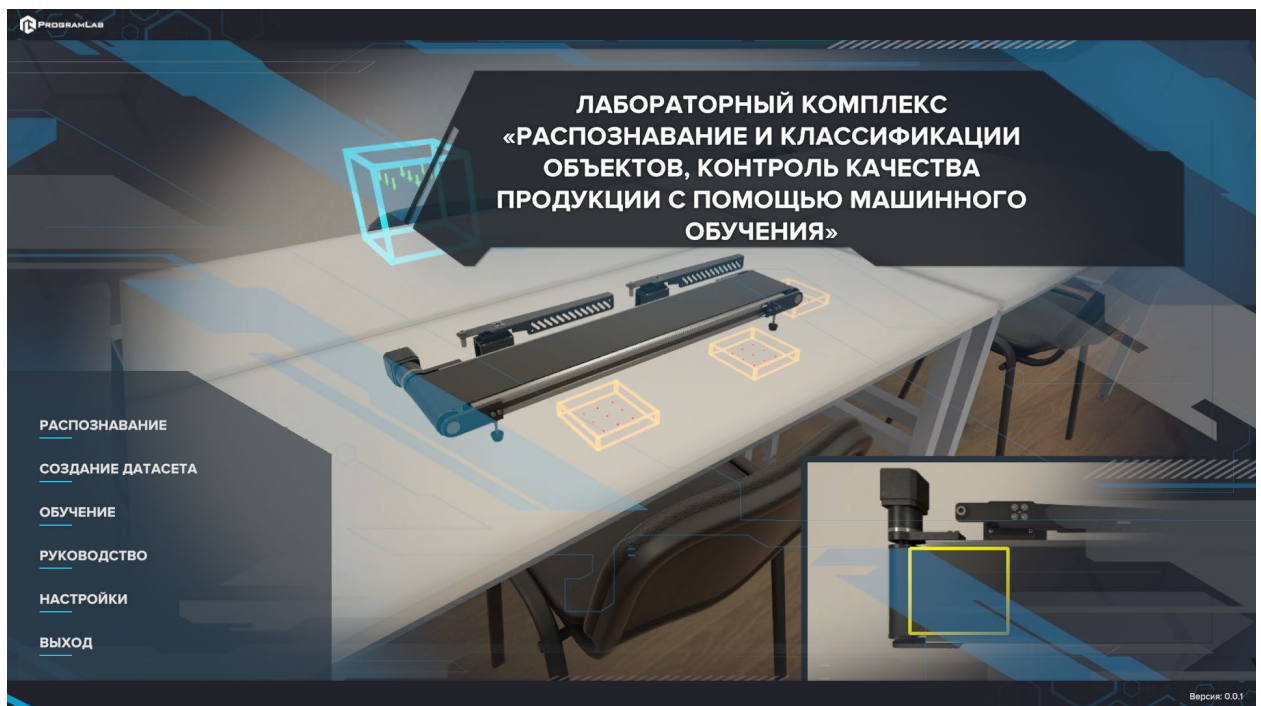
Запущенный сервер лабораторной работы

Запуск серверов

Если Вы все правильно установили и настроили, то для дальнейшего запуска серверов достаточно запустить **start.bat** сначала в папке **YoloServer**, затем **start.bat** в папке **LabServer**.

Работа в программе

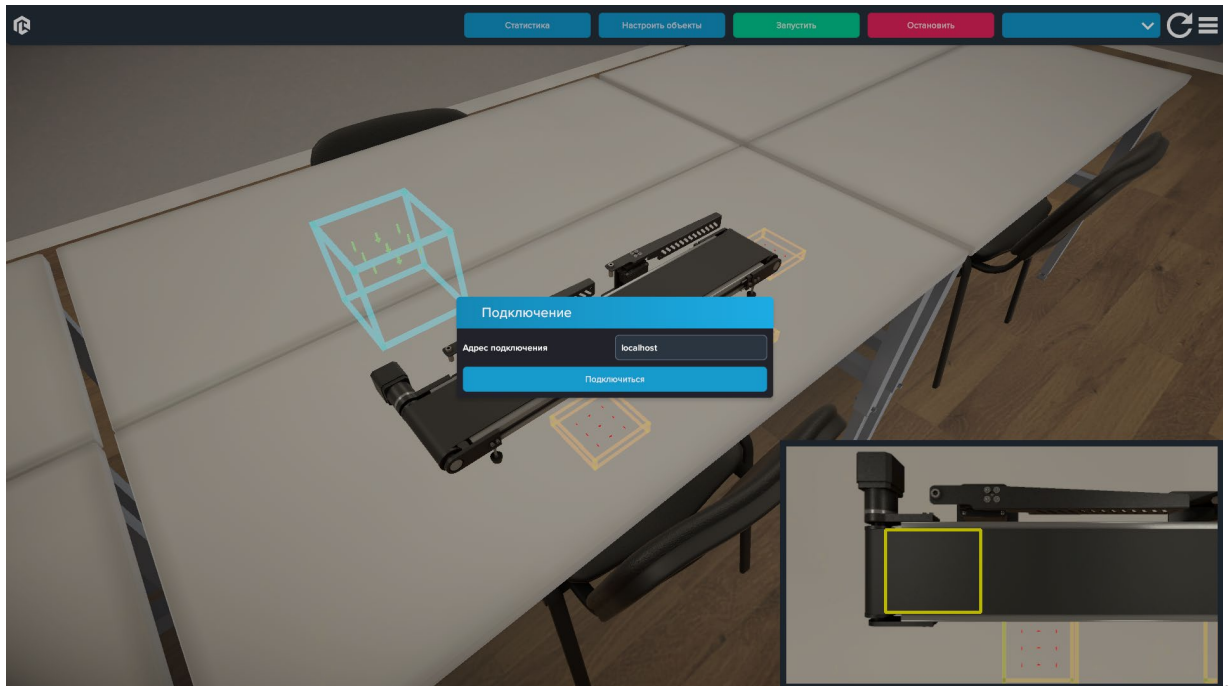
После запуска программы откроется главный экран. На главном экране представлен интерфейс программы, для управления используйте мышь. При первом запуске проверьте, что запущены и настроены сервер нейронной сети и сервер лабораторной работы, иначе многие функции могут быть неактивны.



Главное меню

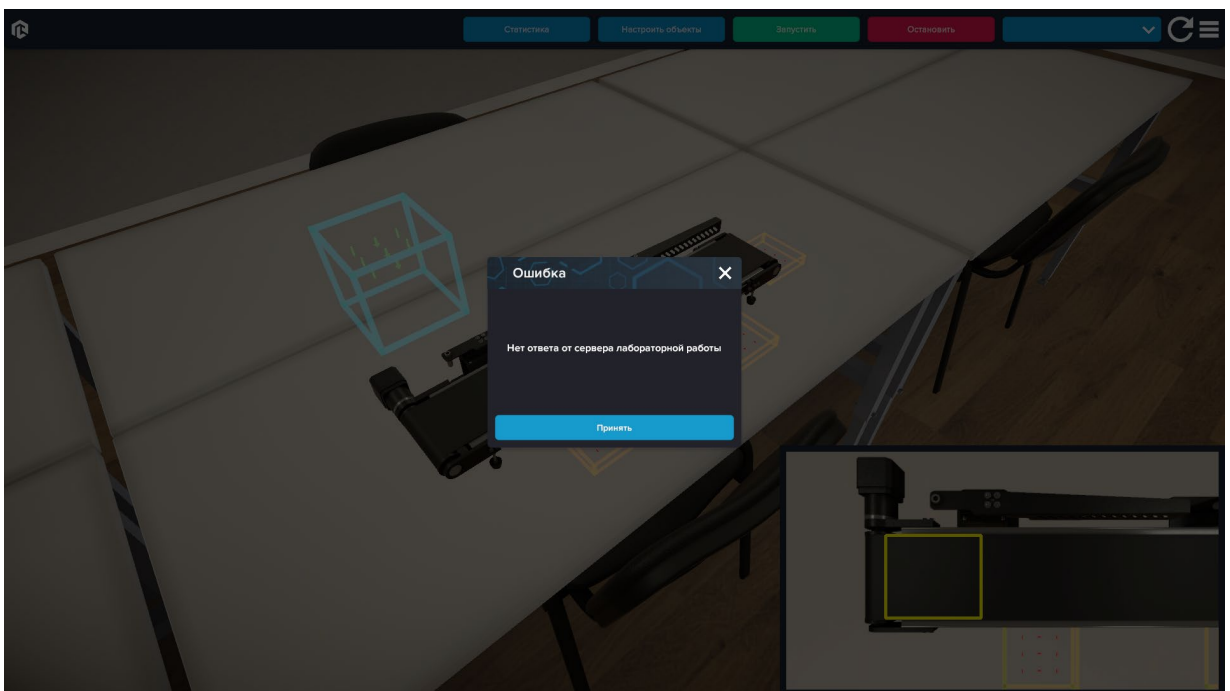
Распознавание

После запуска данного режима появится меню подключения. Проверьте, что все необходимые серверы запущены. Нажмите нажмите **Подключить** в появившемся окне:



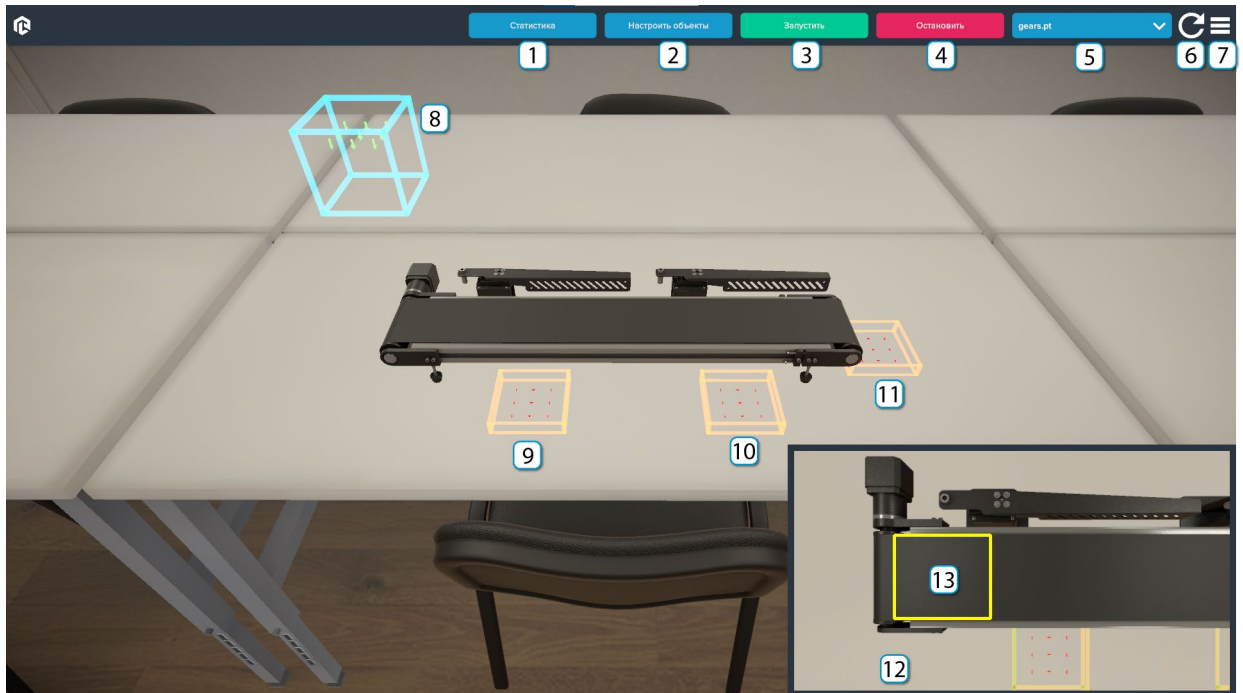
Интерфейс программы

Если программа не получит ответа от сервера, то выдаст ошибку:



Нет ответа от сервера

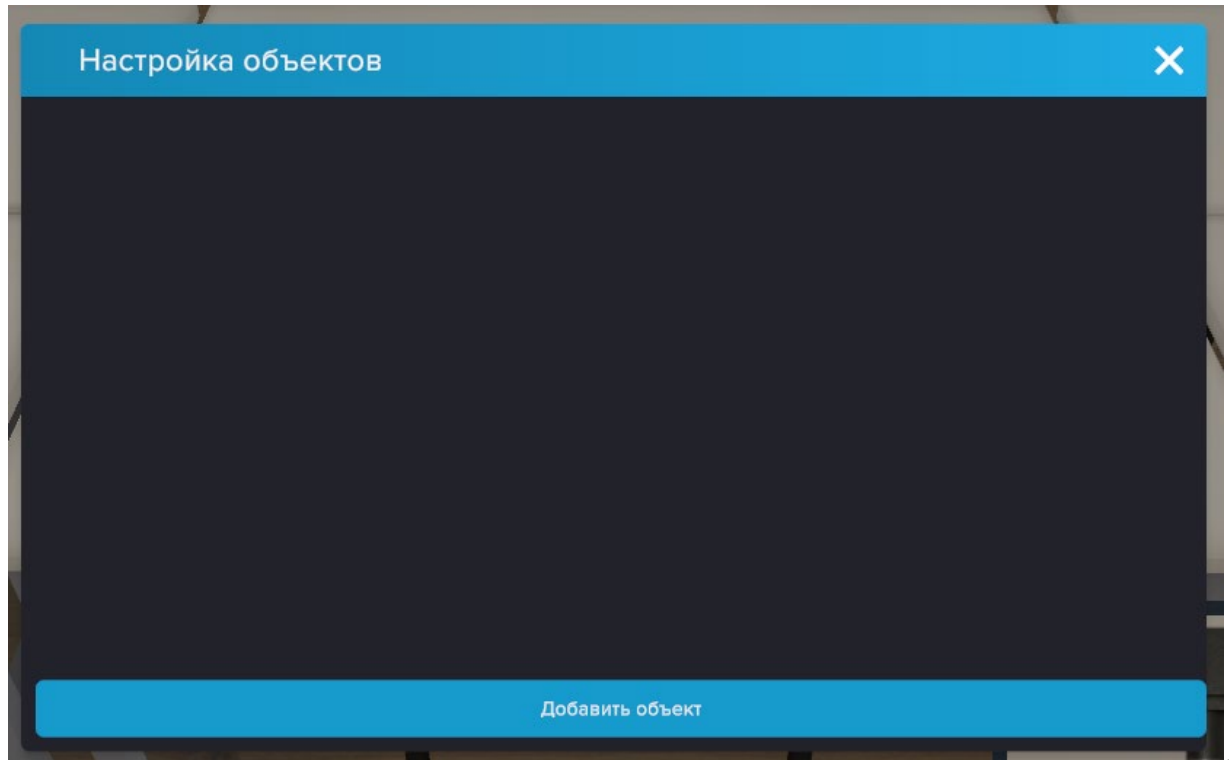
В таком случае, включите серверы, и нажмите **Принять** и **Обновить (6)**. После успешного подключения откроется основной интерфейс:



Интерфейс программы

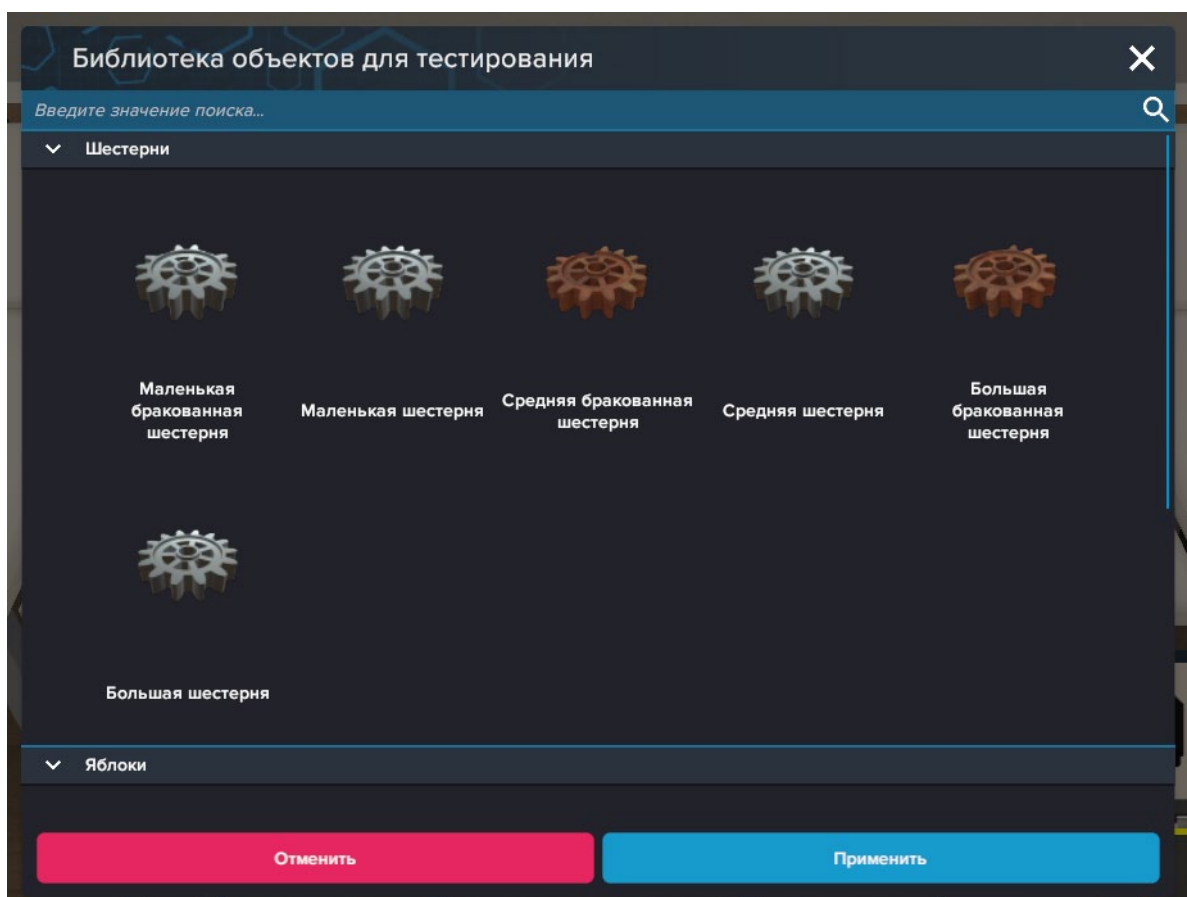
- 1 – Статистика по распознанным объектам
- 2 – Добавление/удаление объектов
- 3 – Запуск работы конвейера и режима распознавания
- 4 – Остановка работы конвейера и режима распознавания
- 5 – Выбор обученной модели
- 6 – Обновление
- 7 – Меню
- 8 – Спавнер объектов на конвейер
- 9 – Зона отсортированных объектов 1
- 10 – Зона отсортированных объектов 2
- 11 – Зона отсортированных объектов 3
- 12 – Вид с камеры
- 13 – Область обнаружения

Перед запуском режима распознавания необходимо выбрать объекты, которые будет распознавать нейронная сеть. Для этого во вкладке **Настроить объекты (2)** нажмите **Добавить объекты**:



Окно настройки объектов

Откроется окно библиотеки объектов для тестирования. Выберите интересующий Вас объект, кликнув левой кнопкой мыши, затем нажмите **Применить**. Вы можете выбрать любое количество объектов.



Библиотека объектов для тестирования








Обратите внимание на выбор обученной модели (5). Нейронная сеть будет распознавать только те объекты, на которых она обучалась, то есть, если Вы хотите распознавать **шестерни**, в списке (5) нужно выбрать **gears.pt**.

Кнопка **Запустить (3)** запустит конвейер и режим распознавания:



Запущен режим распознавания

Во вкладке **Статистика (1)** Вы можете проверить правильность распознавания. В данной вкладка представлен тип объекта, его количество и в какую зону он был отсортирован:

Статистика				Зона 1				Зона 2				Зона 3			
	Средняя бракованная шестерня	16			Маленькая шестерня	16			Средняя шестерня	1					
	Маленькая бракованная шестерня	7			Средняя шестерня	9									
	Большая бракованная шестерня	7			Большая шестерня	6									

Очистить статистику

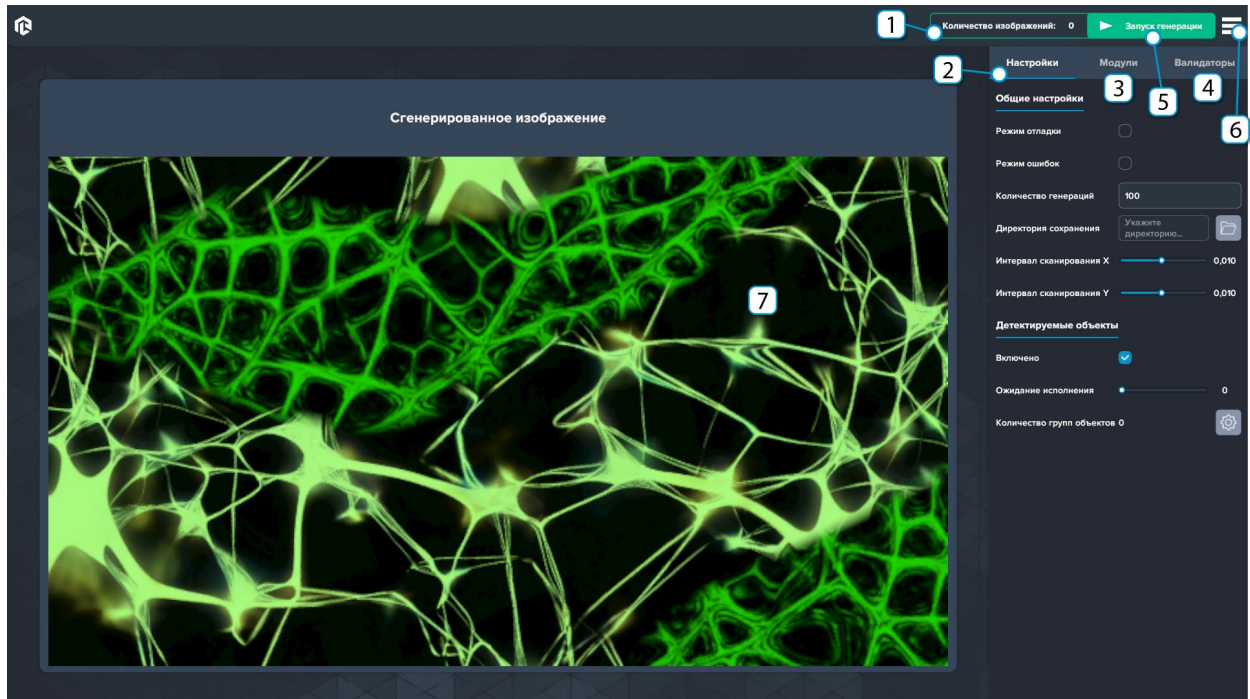
Статистика по распознанным объектам

Примечание: на данном скриншоте в зону 3 попала целая шестерня, так как, во время того, как эта модель появилась на конвейере, была выбрана другая модель, соответственно, нейронная сеть «не видела», эту шестерню.

Нажмите **Очистить статистику**, чтобы сбросить собранную статистику по объектам.

Создание датасета

После запуска данного режима откроется следующее окно:

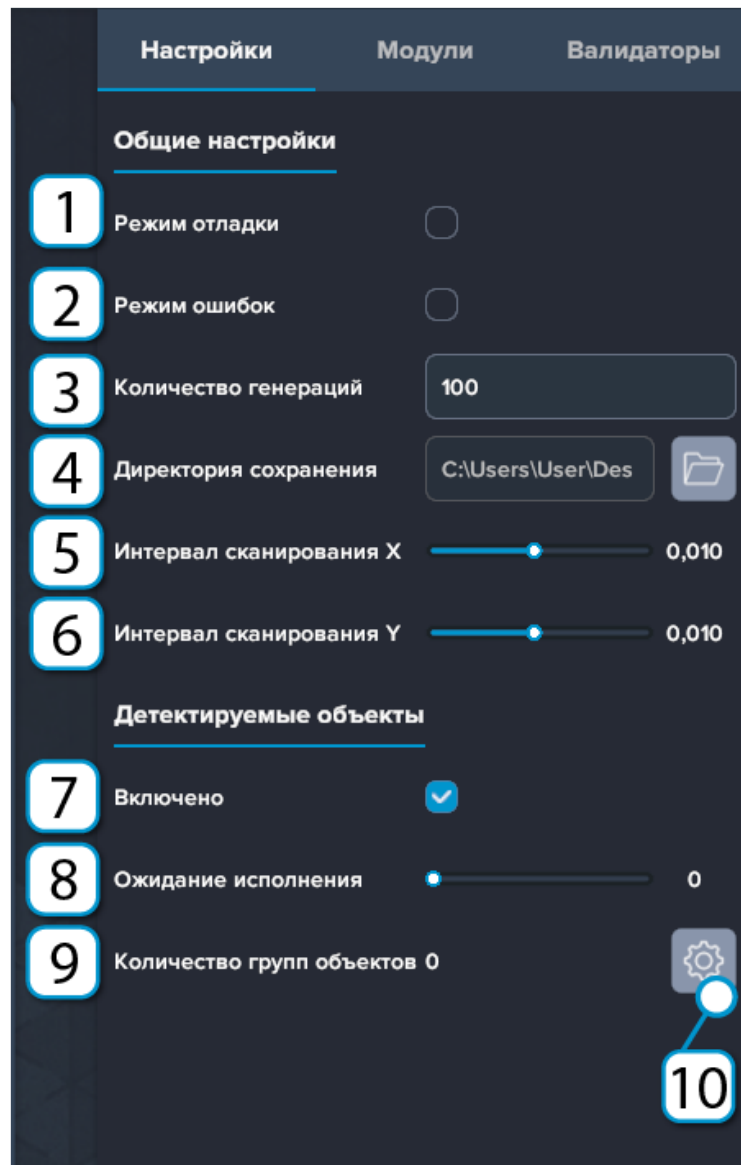


Создание датасетов

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

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

Во вкладке **Настройки** представлены настройки генерации:



Настройки

1 – Режим отладки (добавляет в файл разметки дополнительную информацию к каждому сохраненному изображению) **ВАЖНО**: на отладочных данных нельзя обучить нейронную сеть.

2 – Режим ошибок (сохраняются лишь те изображения, что являются ошибочными, нужен для отладки)

3 – Индикатор количества генераций

4 – Выбор директории 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

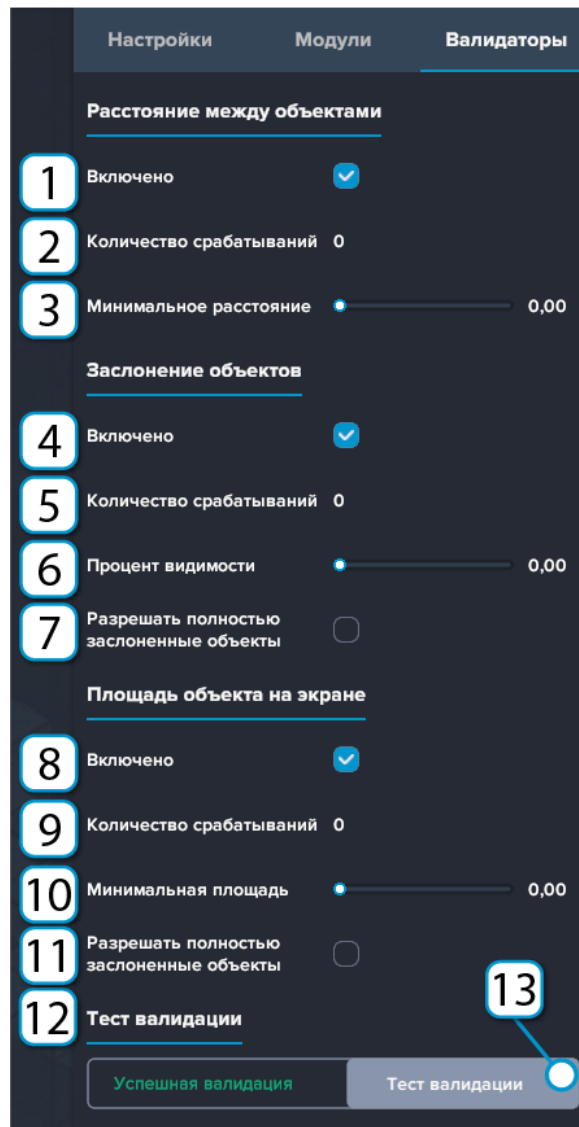
Во вкладке **Модули** представлены настройки модуля:



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 генерации 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:

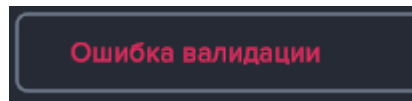


Валидация

- 1 – Включение/отключение валидатора расстояния между объектами
- 2 – Количество изображений, отсеянное валидатором
- 3 – Регулировка минимального расстояния между объектами
- 4 – Включение/отключение валидатора заслонения объекта (на сколько процентов объект может быть виден)
- 5 – Количество изображений, отсеянное валидатором
- 6 – Регулировка процента видимости объекта
- 7 – Разрешать полностью заслоненные объекты
- 8 – Включение/отключение валидатора площади объекта на экране
- 9 – Количество изображений, отсеянное валидатором
- 10 – Регулировка коэффициента распределения текстур
- 11 – Регулировка коэффициента плавности перехода текстур
- 12 – Регулировка коэффициента размера текстур
- 13 – Регулировка коэффициента поворота текстур

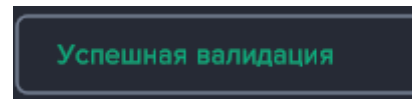
Во вкладке **Валидация** представлены настройки условий, определяющие будет ли сохранено изображение.

Если изображение не подходит, появится уведомление:



Ошибка валидации

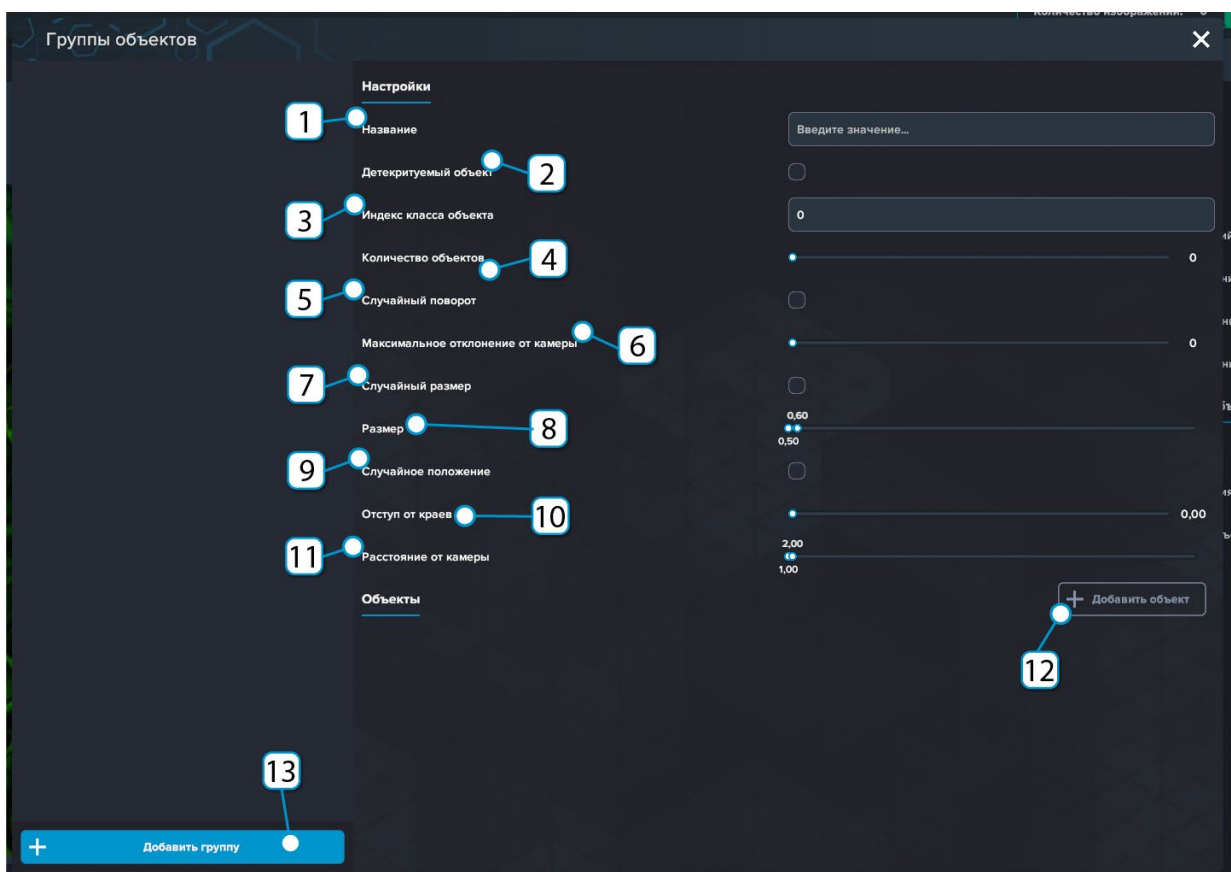
Если изображение подходит, в строке уведомления будет:



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:

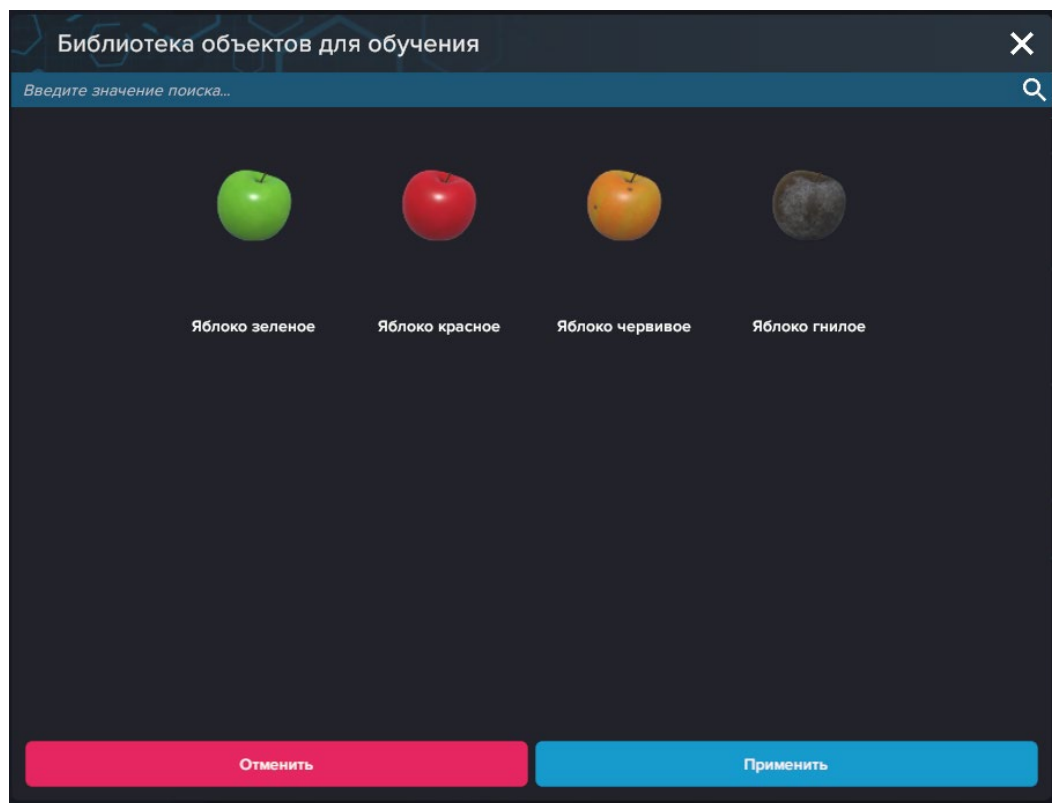


Настройки групп объектов

- 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, в интервале от 0 до 360
- 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 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

Хорошие

Название

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

Детектируемый объект



Детектируемость объекта

Укажите индекс класса объекта. **ВАЖНО:** индекс класса объекта должен быть однозначным у тех групп, которые в конечном итоге должны быть распознаны нейронной сетью.

Индекс класса объекта

0

Индекс класса объекта

Установите количество объектов, которые будут на экране. Не зависит от количества выбранных в группе объектов.

Количество объектов

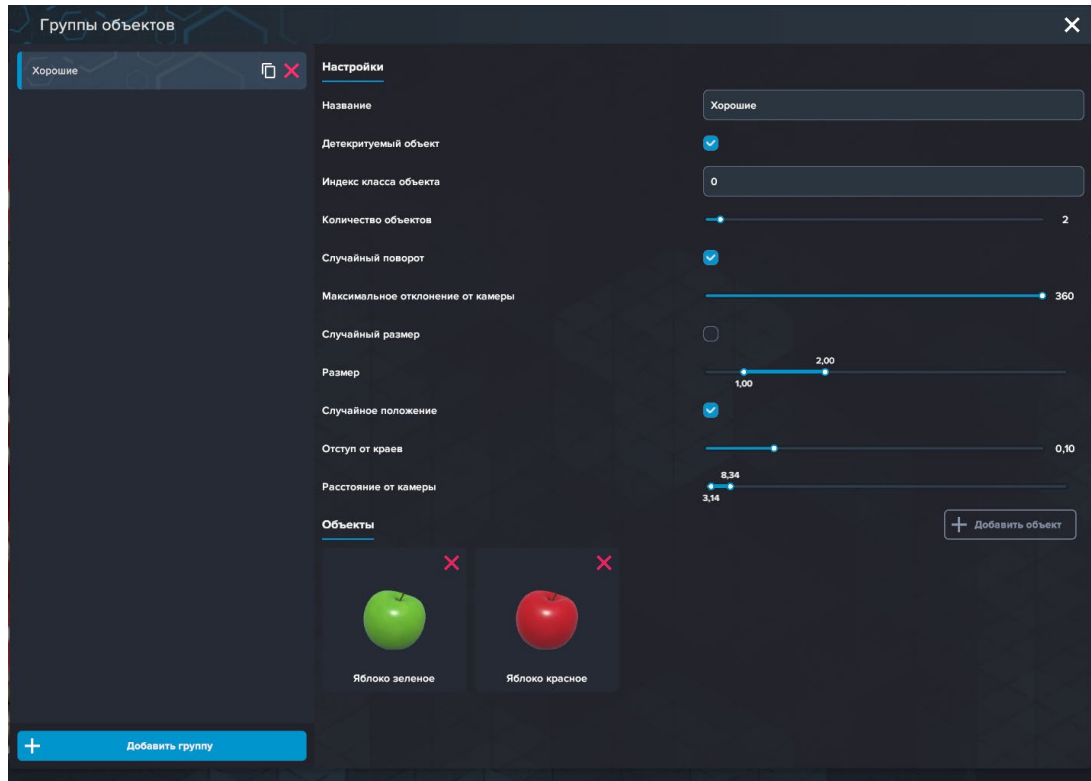


2

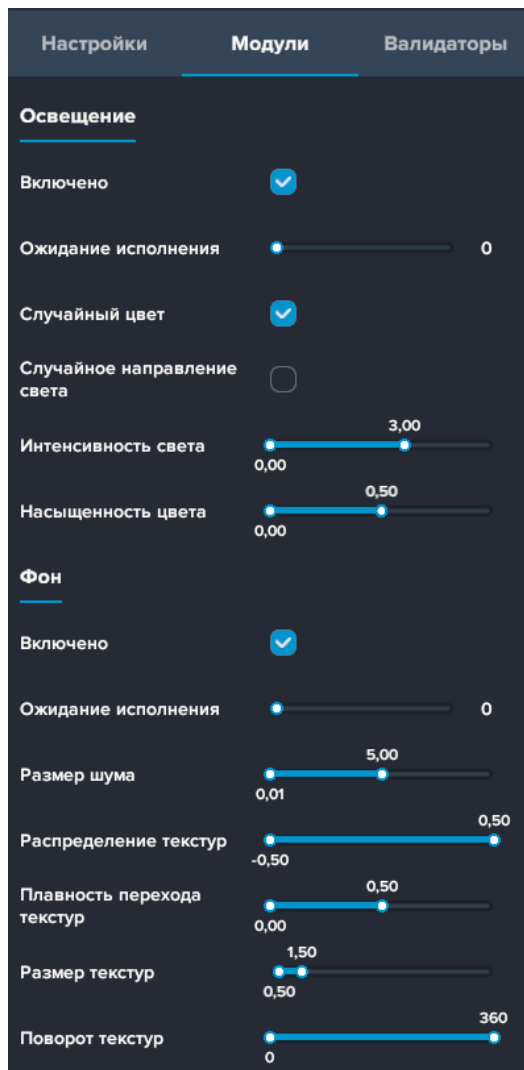
Количество объектов

Остальные настройки подгоняются для групп объектов индивидуально. Для проверки, во вкладке **Валидаторы**, нажмите **Тест валидации**. Регулируйте настройки до тех пор, пока не достигните качественной обучающей выборки.

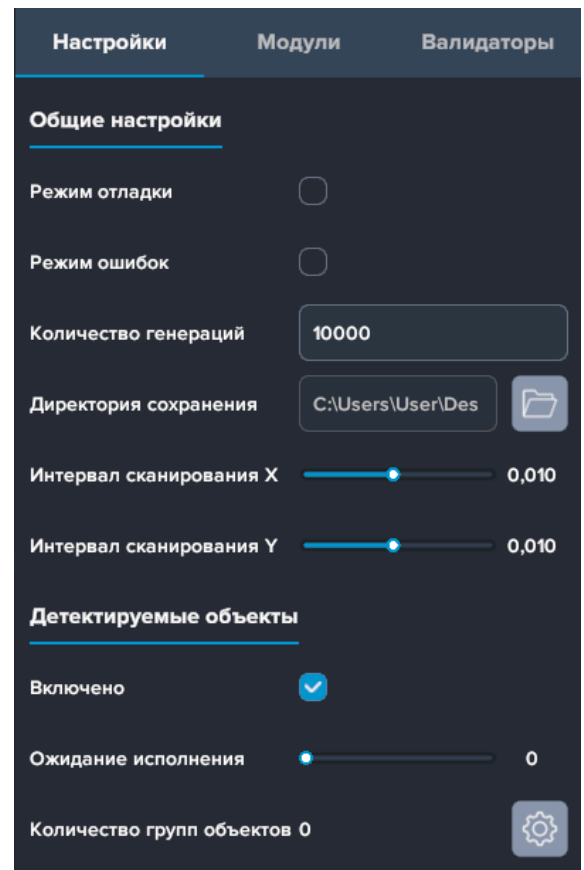
Были взяты следующие настройки (рекомендуемые) для объектов типа яблоки:



Окно настройки группы

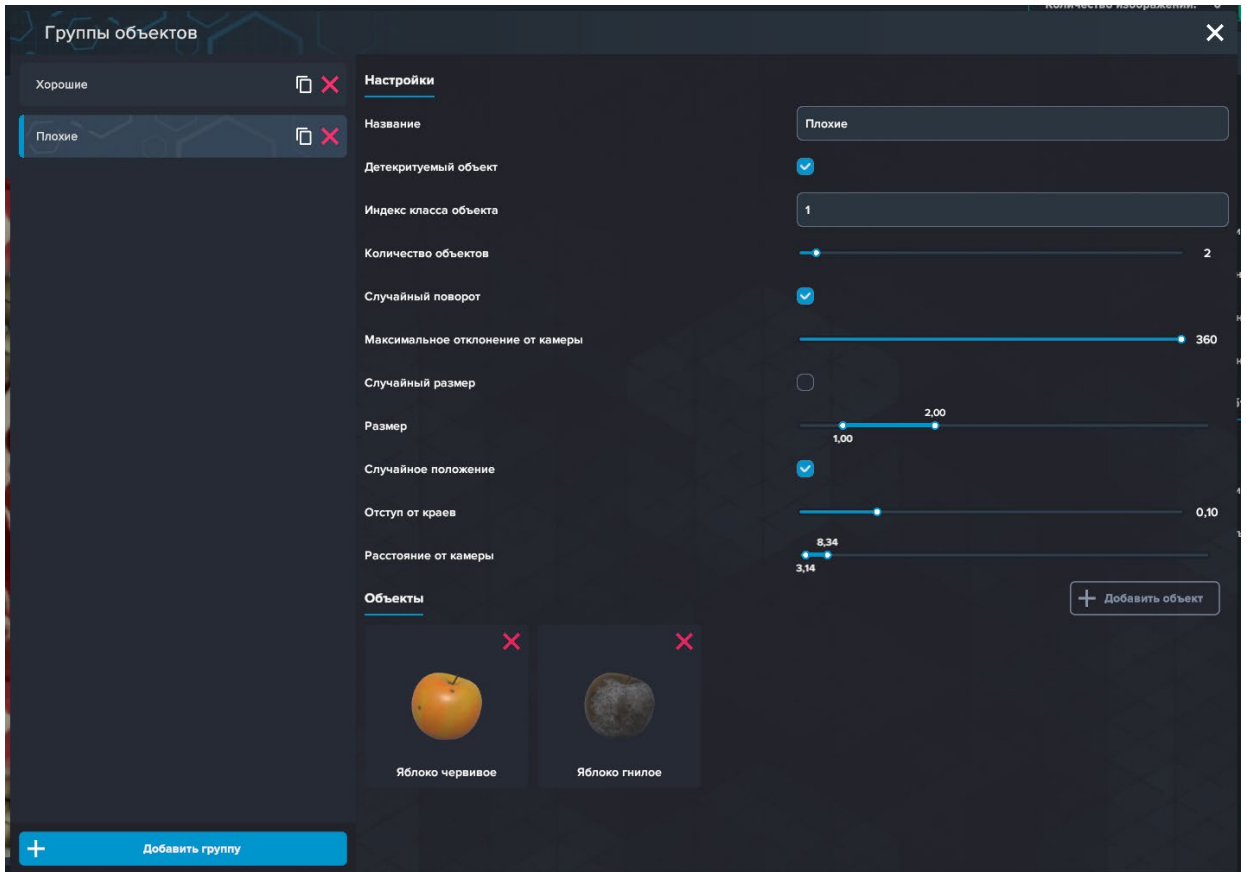


Модули



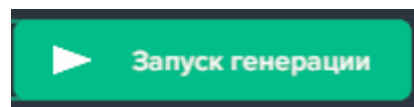
Настройки

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



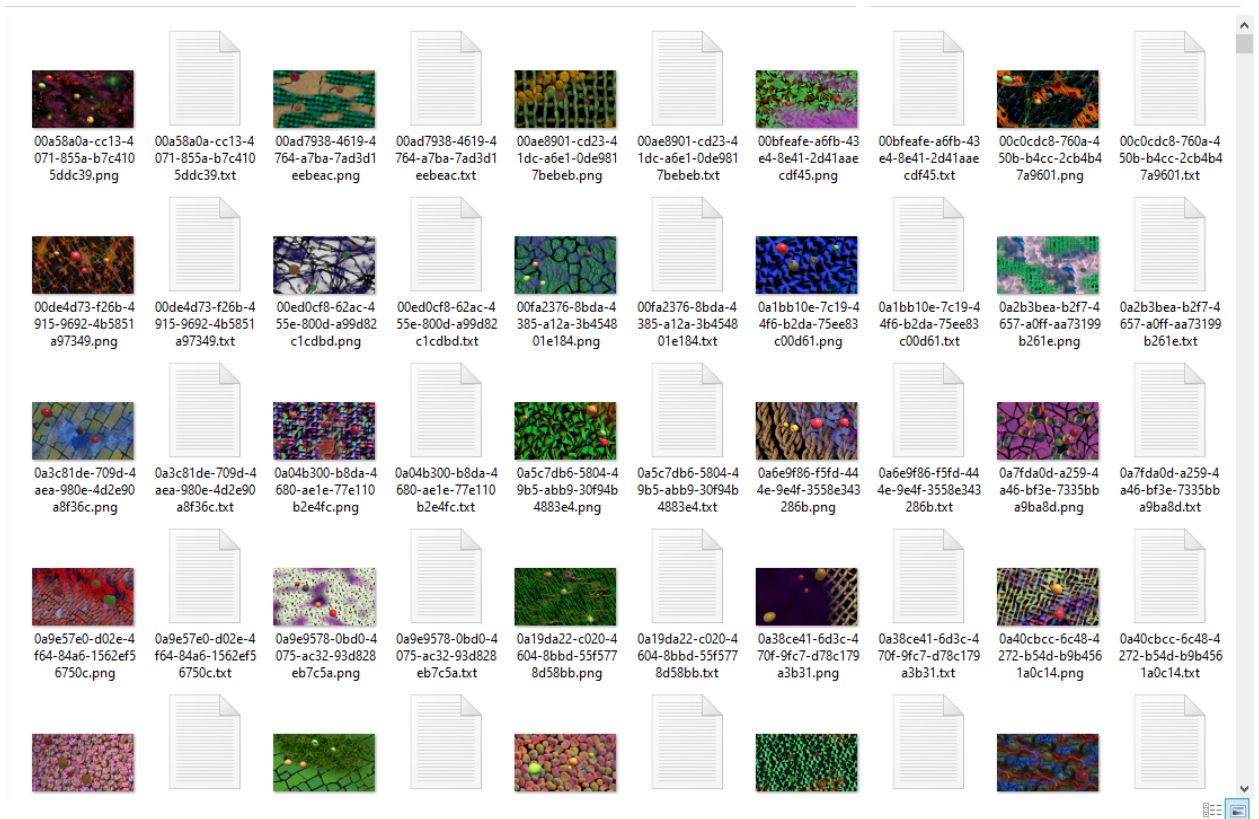
Окно настройки группы

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:

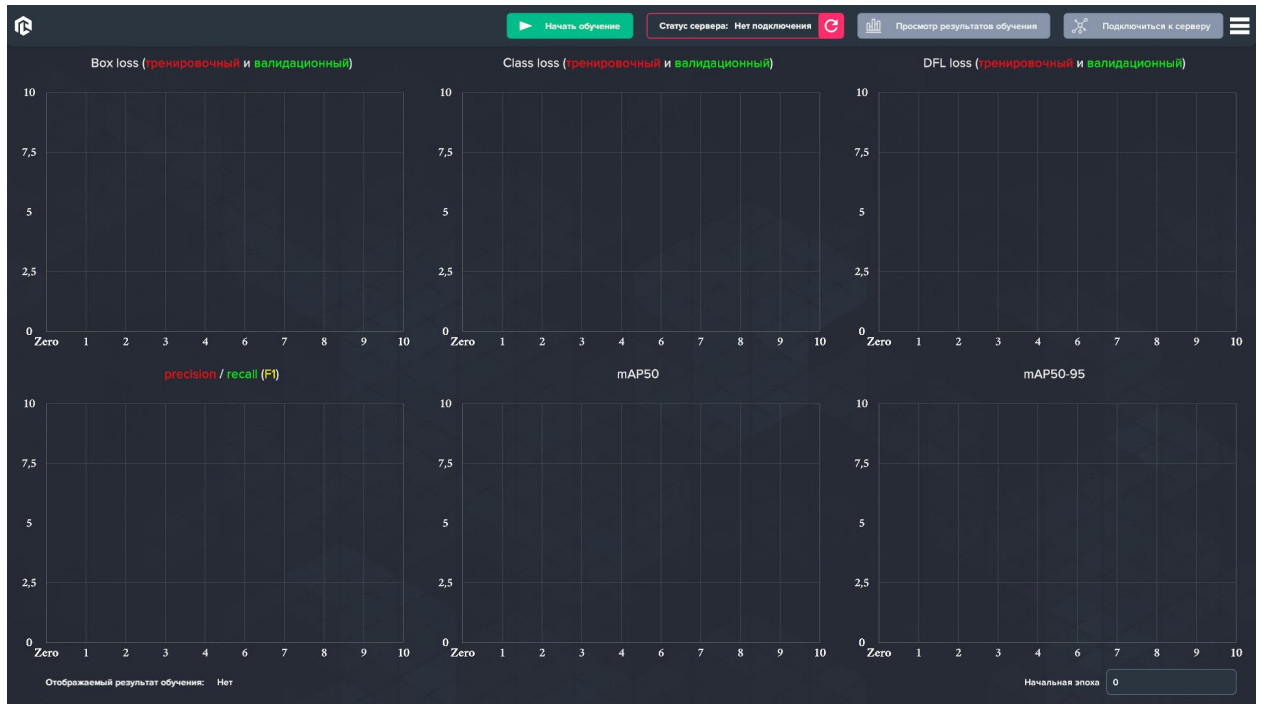


Изображения в директории

Первый файл – само изображение, второй – файл разметки к каждому изображению.

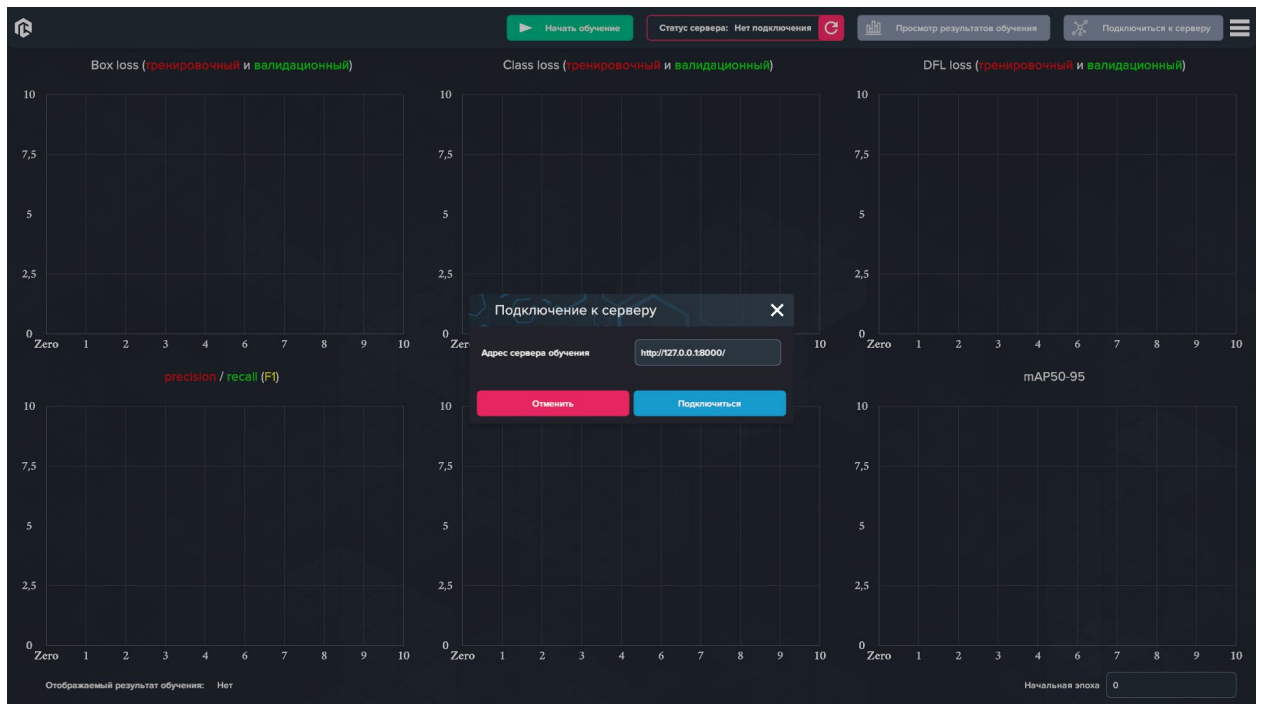
Обучение на созданном датасете

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



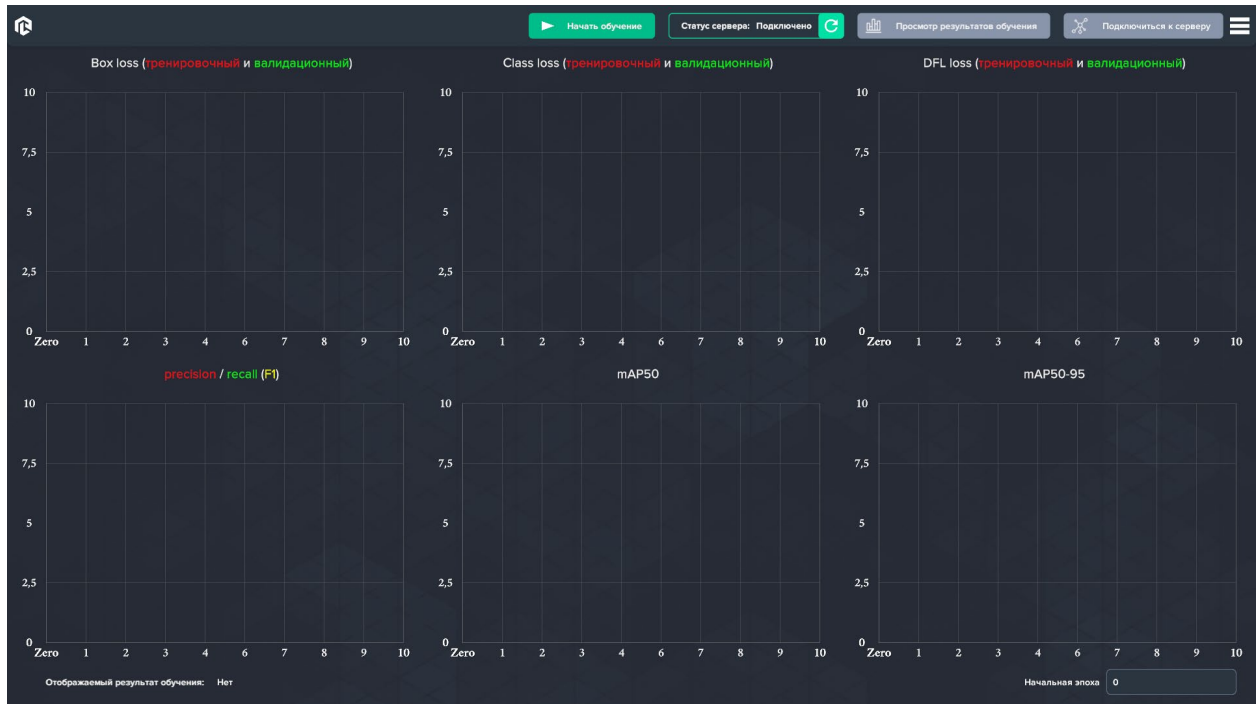
Главный экран

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

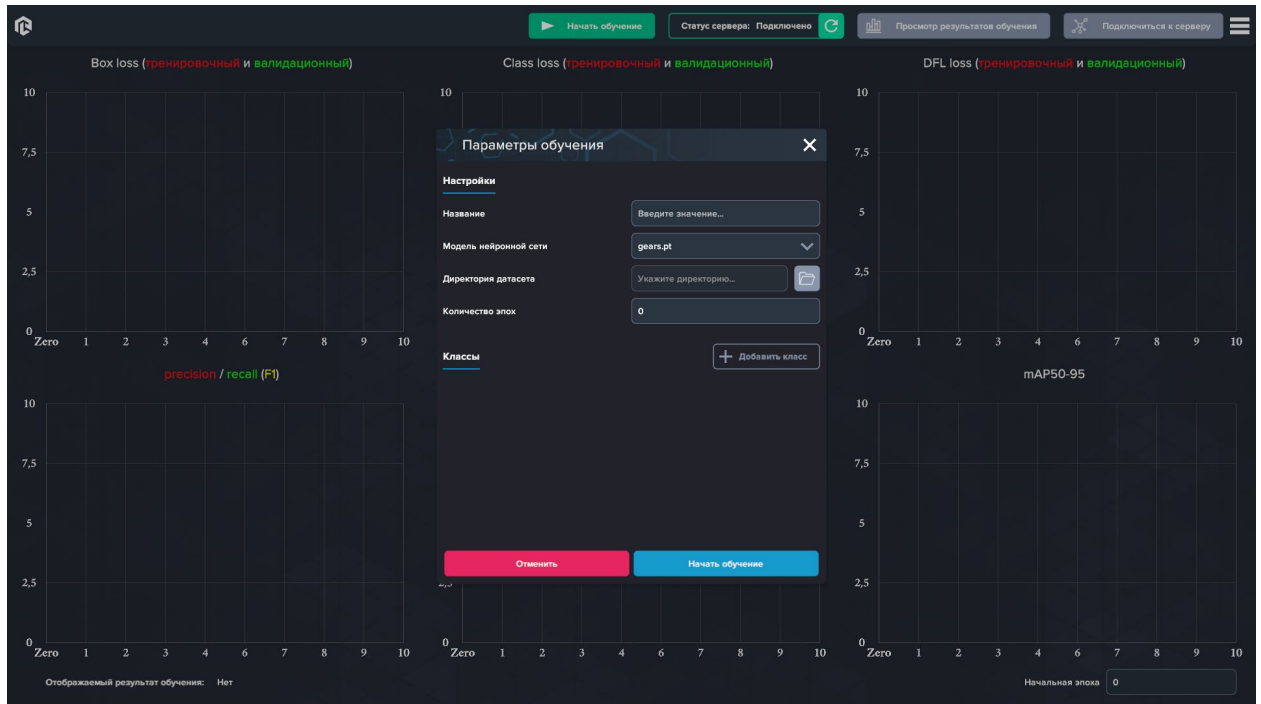
Перед началом обучения перейдите в папку местоположения датасета и создайте внутри 2 папки *train* и *val*.

Имя	Дата изменения	Тип
train	19.06.2024 14:07	Папка с файлами
val	19.06.2024 14:07	Папка с файлами

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.

Параметры обучения ✕

Настройки

Название

Модель нейронной сети

Директория датасета

Количество эпох

Классы

+ Добавить класс

Отменить

Начать обучение

Ввод названия


Выберите из списка модель используемой нейронной сети.

Параметры обучения ✕

Настройки

Название

Модель нейронной сети

Директория датасета 

Количество эпох

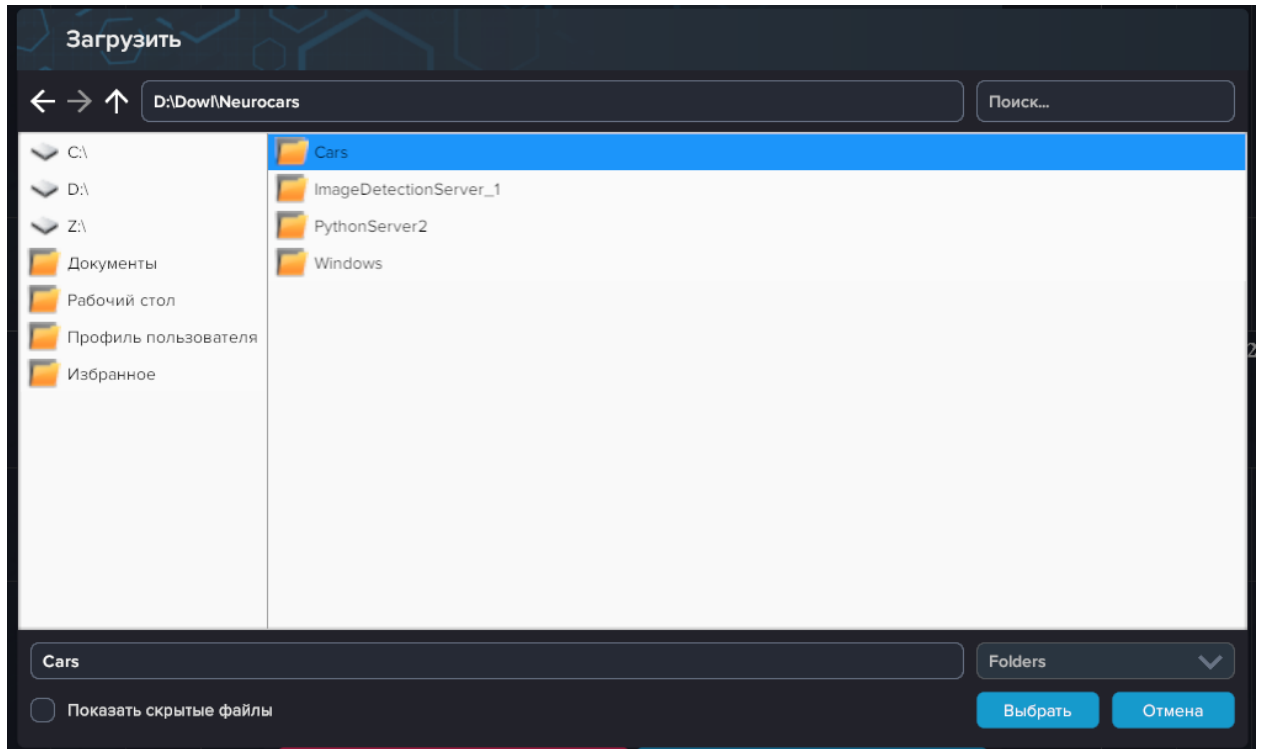
Классы

[+ Добавить класс](#)

[Отменить](#) [Начать обучение](#)

Выбор модели

Укажите местоположение созданного ранее датасета



Выбор датасета

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

Параметры обучения ✕

Настройки

Название

Модель нейронной сети ▼

Директория датасета 📁

Количество эпох

Классы

+ Добавить класс

Отменить

Начать обучение

указание количества эпох

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

Параметры обучения ✕

Настройки

Название

Модель нейронной сети

Директория датасета

Количество эпох

Классы + Добавить класс

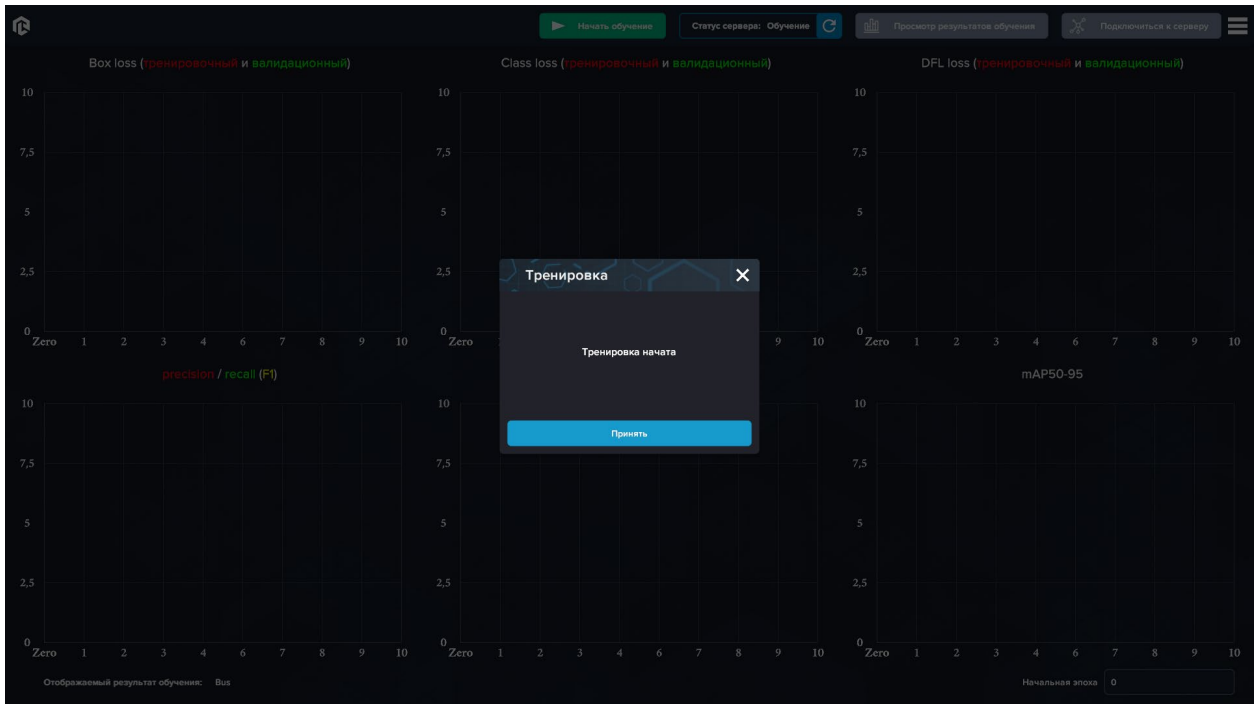
0	<input style="width: 100%;" type="text" value="Bus"/>	✕
1	<input style="width: 100%;" type="text" value="BlackBus"/>	✕

Отменить

Начать обучение

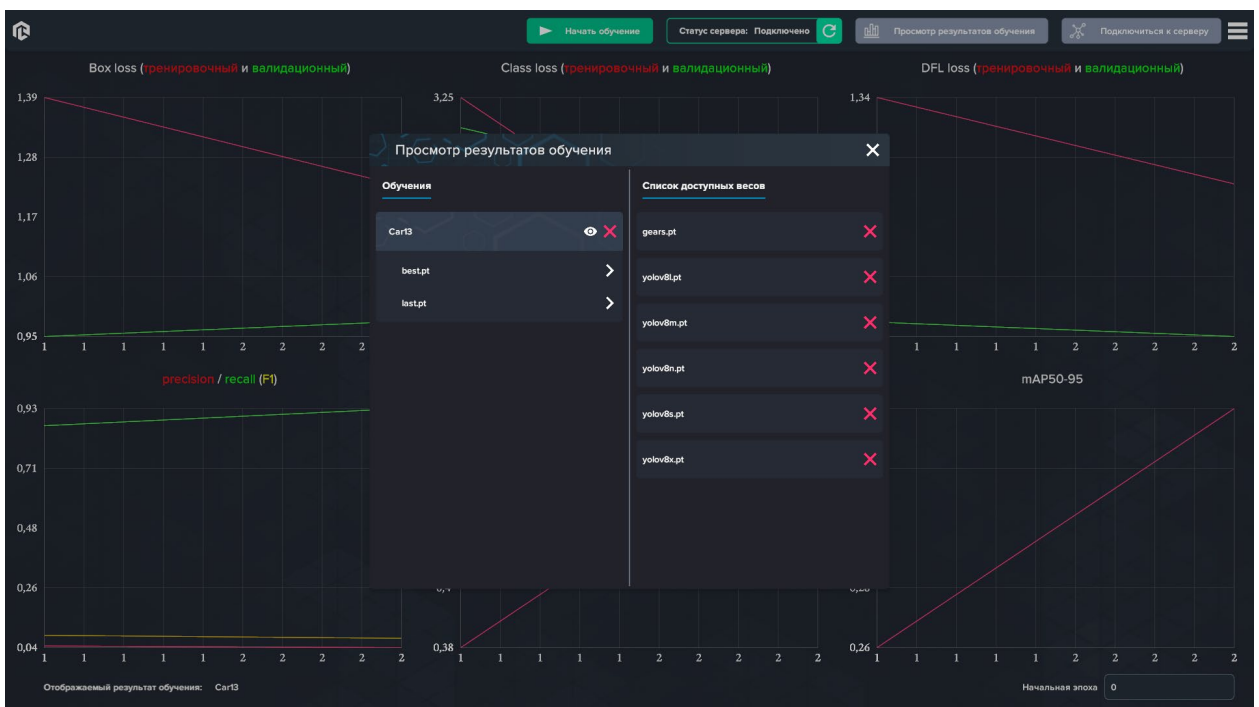
Введение классов

Нажмите на кнопку начать обучение.



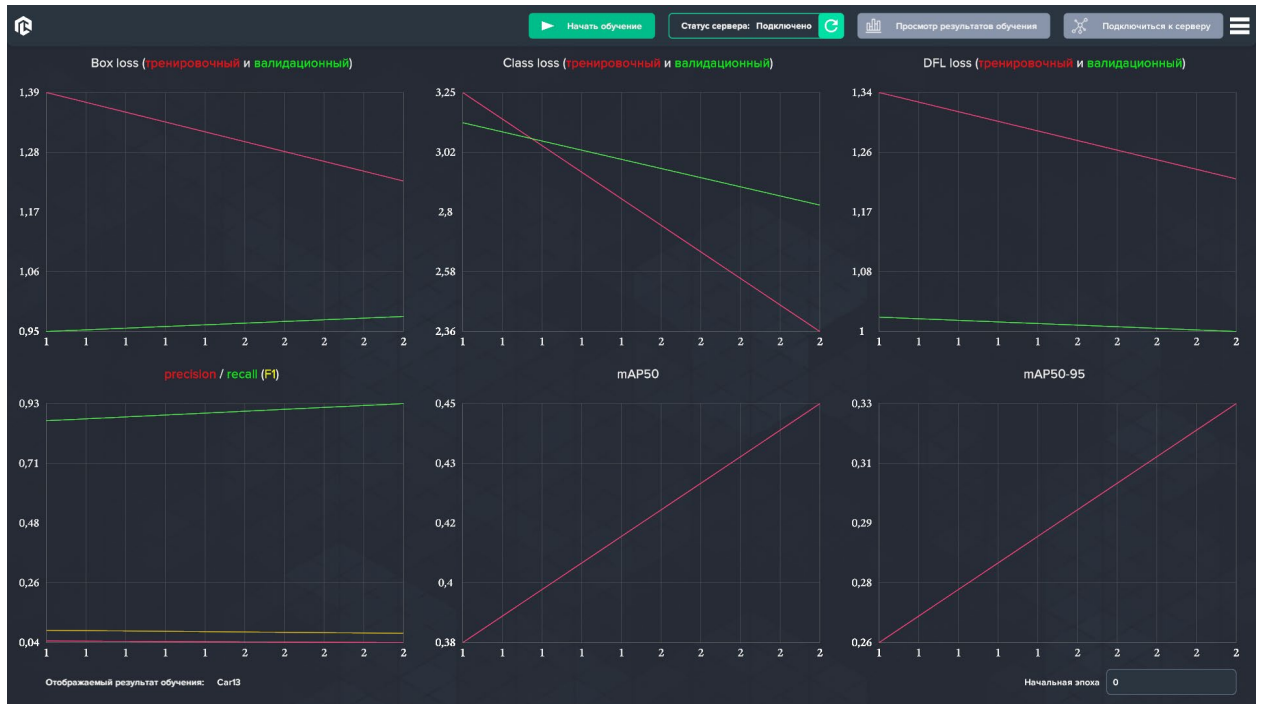
Окно информации о запуске обучения

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.



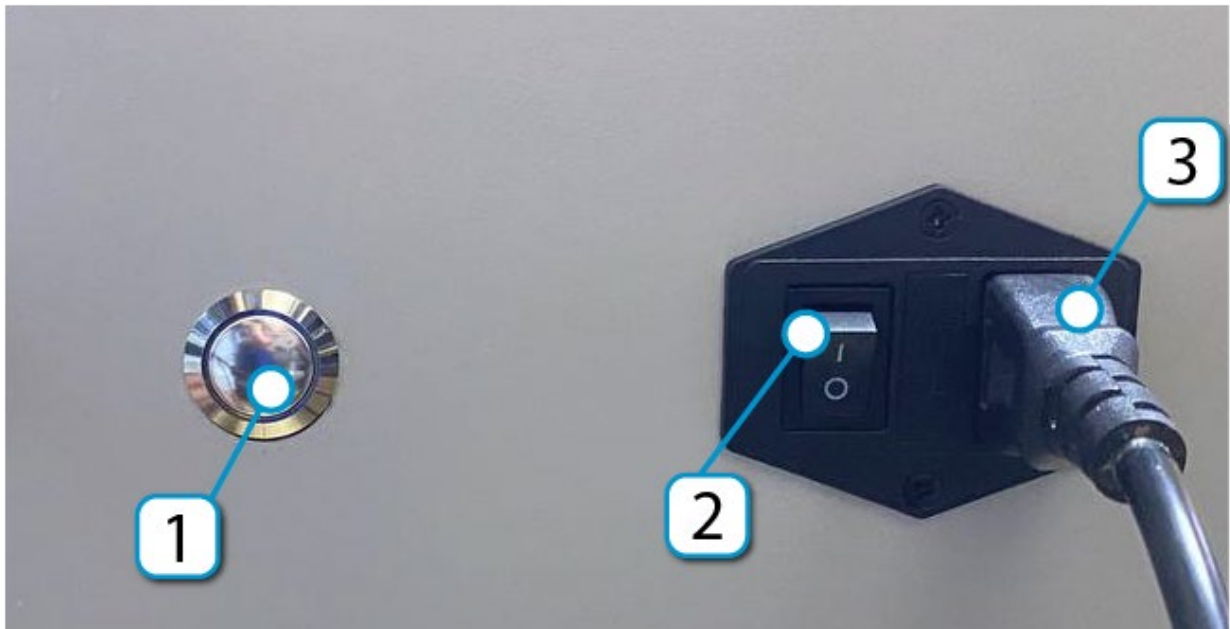
Ознакомление с метриками

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

Запуск физического стенда

ВАЖНО: перед использованием проверьте, что стенд подключен к источнику питания и к компьютеру, на котором будут запущены сервера, а также установлена, закреплена и подключена камера.

Запустите сервер нейронной сети и сервер лабораторной работы. Затем переведите тумблер на задней панели стенда в режим вкл.:



Выключатели на задней панели стенда

- 1 – Кнопка включения устройства управления конвейером
- 2 – Тумблер питания
- 3 – Кабель питания

На лицевой панели стенда включатся цифровые индикаторы зон сортировки объектов, а также индикатор питания:



Цифровые индикаторы зон сортировки объектов

Далее нажмите кнопку включения устройства управления конвейером (1) и подождите инициализации – на цифровых индикаторах должны появиться

нули. Нажмите **кнопку запуска (2)** для включения конвейера и режима распознавания:



Лицевая сторона стнда

1 – Индикатор питания

2 – Кнопка запуска

3 – Кабель питания

По умолчанию установлена модель для распознавания шестерен.

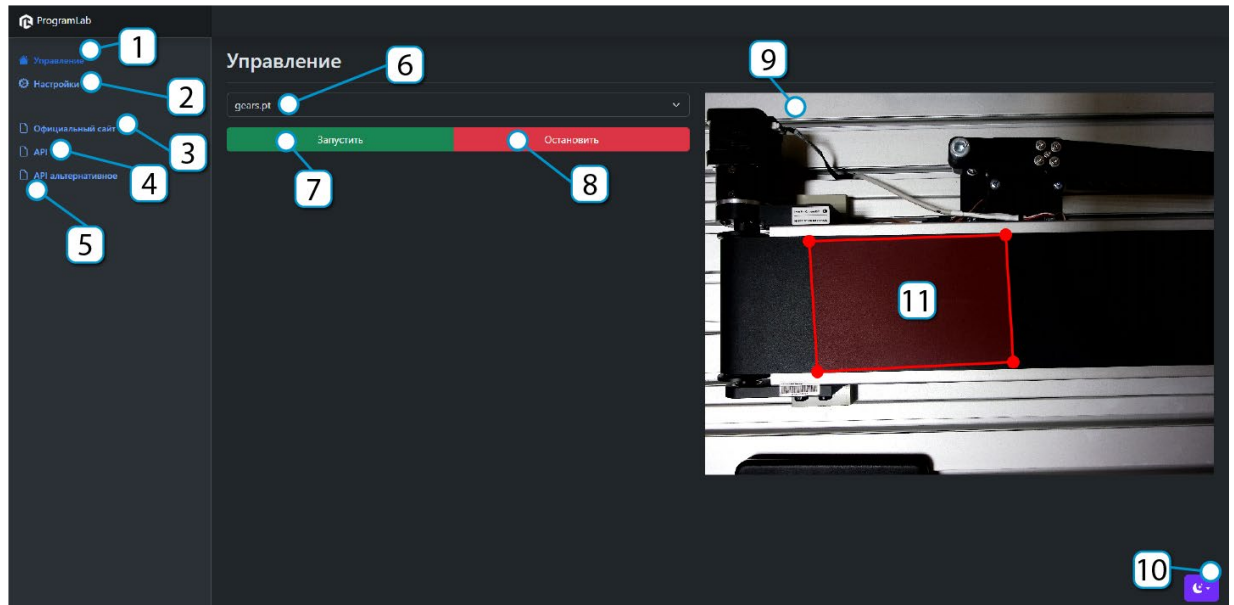
Положите исследуемый объект на начало движения конвейера. Стенд оснащен механизмами сбрасывания **ВАЖНО**: не препятствуйте срабатыванию механизмов сбрасывания руками или иными предметами – это может привести к поломке!

Чтобы остановить конвейер и режим распознавания нажмите **кнопку стоп (3)**.

ВАЖНО: цифровые индикаторы зон сортировки сбрасываются после нового запуска.

Веб-интерфейс

Данный стенд предусматривает управление и взаимодействие с некоторыми настройками конвейера через веб-интерфейс. Чтобы подключиться используйте ip стенда и порт 8000 в таком виде: «<https://192.168.99.51:8000>». Чтобы узнать ip стенда можно воспользоваться бесплатной программой Advanced IP Scanner. С помощью нее Вы сможете увидеть все устройства, подключенные к данной локальной сети. Нужный Вам ip адрес будет у устройства с именем, начинающимся с «DESKTOP...» и в колонке Производитель «Shenzhen...».



Веб-интерфейс

- 1 – Вкладка управление
- 2 – Вкладка настройки
- 3 – Официальный сайт ProgramLab
- 4 – API
- 5 – API в измененном дизайне
- 6 – Выбранная модель для распознавания
- 7 – Запуск конвейера и режима распознавания
- 8 – Остановка конвейера и режима распознавания
- 9 – Изображение, получаемое с камеры
- 10 – Выбор темы
- 11 – Область распознавания камеры

Во вкладке управление (1) Вы можете поменять модель для распознавания (6), а также запустить и остановить конвейер и режим распознавания.

Так же справа расположена трансляция изображения, получаемая с камеры (9). Вы можете настроить область распознавания (11), передвигая 4 угла мышью, таким образом Вы можете корректировать время срабатывания сбрасывателей. Справа снизу Вы можете выбрать светлую или темную тему (10).

Во вкладке настройки Вы можете изменить ip шины (ip компьютера, на котором запущена шина), а также изменить com port (не рекомендуется менять). Изменения вступают в силу после перезагрузки.

Спецификация API сервера нейронной сети

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": "success" //Ответ работает ли сервер (тип данных - string)
3	}

Статус нейронной сети

Команда **Получить статус нейронной сети**, приложение посылает запрос к конечной точке **/status**, адресованный к серверу, сервер отправляет информацию о своем состоянии в виде json ответа.

Пример JSON ответа:

1	{
2	"status": "TRAINING» //Ответ работает ли сервер (тип данных – string)
3	}

Установить модель

Команда **Установить модель**, приложение посылает запрос к конечной точке **/set-model**, адресованный к серверу, сервер отправляет информацию о своем состоянии в виде json ответа.

Пример JSON запроса:

1	{
2	"name": "model.pt"
3	}

Пример JSON ответа:

```

1 {
2   "status": "success"
3 }

```

Обнаружение объектов

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

Form-data запрос:

image – Загружаемое изображение для анализа в бинарном виде

polygons

```

1 {
2   "polygons":[ // Список полигонов
3     {
4       "points":[ // Список точек
5         {
6           "x": 0, // Координата X
7           "y": 0 // Координата Y
8         },
9         {
10          "x": 0,
11          "y": 1
12        },
13        {
14          "x": 1,
15          "y": 1
16        },
17        {
18          "x": 1,
19          "y": 0
20        }
21      ]
22    }

```

Пример JSON ответа:

```

1 {
2   "boxes": [
3     {

```



```
4     "classIndex": 0,  
5     "className": "string",  
6     "confidence": 0,  
7     "x": 0,  
8     "y": 0,  
9     "w": 0,  
10    "h": 0,  
11    "polygonIndexes": [  
12      0  
13    ]  
14  }  
15 ]  
16 }
```

Начать обучение

Команда **Начать обучение**, приложение посылает запрос к конечной точке **/train**, адресованный к серверу, сервер отсылает информацию о своем состоянии в виде json ответа.

Пример JSON запроса:

```
1 {  
2   "modelName": "string",  
3   "trainingName": "string",  
4   "datasetPath": "string",  
5   "classes": [  
6     "string"  
7   ],  
8   "epochs": 0  
9 }
```

Пример JSON ответа:

```
1 {  
2   "status": "success"  
3 }
```

Получить результаты обучения

Команда **Получить результаты обучения**, приложение посылает запрос к конечной точке **/training-metrics/{training_name}**, где {training_name} – имя обучения, адресованный к серверу, сервер отсылает информацию о своем состоянии в виде json ответа.

Пример JSON ответа:

```
1 {
2   "metrics": {
3     "additionalProp1": [
4       0
5     ],
6     "additionalProp2": [
7       0
8     ],
9     "additionalProp3": [
10      0
11    ]
12  }
13 }
```

Получить информацию об активированной модели

Команда *Получить информацию об активированной модели*, приложение посылает запрос к конечной точке */model*, адресованный к серверу, сервер отправляет информацию о своем состоянии в виде json ответа.

Пример JSON ответа:

```
1 {
2   "name": "string",
3   "classes": [
4     {
5       "classIndex": 0,
6       "className": "string"
7     }
8   ]
9 }
```

Получить список результатов обучений

Команда *Получить список результатов обучений*, приложение посылает запрос к конечной точке */training-results*, адресованный к серверу, сервер отправляет информацию о своем состоянии в виде json ответа.

Пример JSON ответа:

```
1 {
2   "trainingResults": [
3     {
4       "trainingResultName": "string",
5       "weights": [
6         {
```

```
7     "name": "string"
8     }
9   ]
10  }
11 ]
12 }
```

Удалить результаты обучения

Команда **Удалить результаты обучения**, приложение посылает запрос к конечной точке **/training-results/{training_name}**, где {training_name} – имя обучения, адресованный к серверу, сервер отправляет информацию о своем состоянии в виде json ответа.

Пример JSON ответа:

```
1 {
2   "status": "success"
3 }
```

Получить список существующих моделей

Команда **Получить список результатов обучений**, приложение посылает запрос к конечной точке **/models**, адресованный к серверу, сервер отправляет информацию о своем состоянии в виде json ответа.

Пример JSON ответа:

```
1 {
2   "models": [
3     {
4       "modelName": "string"
5     }
6   ]
7 }
```

Удалить модель

Команда **Удалить результаты обучения**, приложение посылает запрос к конечной точке **/models/{model_name}**, где {model_name} – имя обучения, адресованный к серверу, сервер отправляет информацию о своем состоянии в виде json ответа.

Пример JSON ответа:

```
1 {
2   "status": "success"
3 }
```

Сохранить модель из результата обучения

Команда **Начать обучение**, приложение посылает запрос к конечной точке **/save-training-model**, адресованный к серверу, сервер отправляет информацию о своем состоянии в виде json ответа.

Пример JSON запроса:

```

1 {
2   "trainingResultName": "string",
3   "weightsName": "string",
4   "newModelName": "string"
5 }
```

Пример JSON ответа:

```

1 {
2   "status": "success"
3 }
```

API RabbitMQ

cameras_exchange

Данный exchange предназначен для изображений с камеры. Камера с физического или виртуального стенда посылает изображение в cameras_exchange. Сервер лабораторной работы получает это изображение с заголовком.

camera.1

Топик для изображений с первой и единственной в данной лабораторной работе камеры. Содержит изображения и полигоны, относительно которых происходит распознавание. В данной лабораторной работе используется один полигон, для обозначения границ распознавания.

Тело сообщения:

```

1 | Изображение в виде байт
```

Заголовки:

```

1 | {
2 |   "polygons": // Список полигонов, которые обозначают границы областей
3 |   распознавания
4 |   [
5 |     {
6 |       "points": // Список точек для каждого полигона
7 |       [
8 |         {
9 |           "x": 0, // Координата X
```

```

10         "y": 1 // Координата Y
11     },
12     {
13         "x": 0, // Координата X
14         "y": 1 // Координата Y
15     },
16     {
17         "x": 0, // Координата X
18         "y": 1 // Координата Y
19     },
20     {
21         "x": 0, // Координата X
22         "y": 1 // Координата Y
23     }
24 ]
25 }

```

cameras_detections_exchange

Данный exchange предназначен для результатов распознавания. Сервер лабораторной работы возвращает в данный exchange результат распознавания изображения с камеры. Виртуальный стенд забирает эти данные.

camera.1

Топик для результатов распознавания изображений.

Тело сообщения:

```

1  {
2    "boxes":
3    [
4      {
5        "classIndex": 0, // Класс объекта
6        "className": "string", // Имя объекта
7        "confidence": 0, // Уверенность (на сколько вероятно объект является данным
8        классом)
9        "x": 0, // Координата X
10       "y": 0, // Координата X
11       "w": 0, // Высота
12       "h": 0, // Ширина
13       "polygonIndexes": // Индексы полигонов, в которых находится данный объект
14       [
15         0
16       ]
17     }
18   ]
19 }

```

lab_exchange

Данный exchange предназначен для запросов к серверу лабораторной работы. Виртуальный и физический стенд посылают запросы в данный exchange для сервера лабораторной работы. Сервер лабораторной работы принимает эти данные.

Тело сообщения:

1	{
2	"isEnabled": true // Состояние, в которое необходимо установить лабораторную
3	работу
	}

Тело сообщения ответа из "Direct Reply-to":

1	{
2	"status": "success"
3	}

lab.get_models

Топик для получения моделей, которые присутствуют на сервере нейронной сети.

Тело сообщения отсутствует.

Тело сообщения ответа из "Direct Reply-to":

1	{
2	"models": // Список моделей
3	[
4	{
5	"modelName": "model1" // Название модели
6	},
7	{
8	"modelName": "model2" // Название модели
9	}
10]

Тело сообщения ответа из "Direct Reply-to" в случае ошибки:

1	{
2	"error": "Нет подключения к серверу нейронной сети"
3	}

lab.get_model

Топик для получения текущей активированной модели.

Тело сообщения отсутствует.

Тело сообщения ответа из "Direct Reply-to":

```
1 {
2   "name": "model", // Имя модели
3   "classes": // Список классов модели
4   [
5     {
6       "classIndex": 0, // Индекс класса модели
7       "className": "string" // Имя класса модели
8     }
9   ]
10 }
```

Тело сообщения ответа из "Direct Reply-to" в случае ошибки:

```
1 {
2   "error": "Нет подключения к серверу нейронной сети"
3 }
```

lab.set_model

Топик для установки модели.

Тело сообщения:

```
1 {
2   "name": "model" // Имя модели
3 }
```

Тело сообщения ответа из "Direct Reply-to":

```
1 {
2   "status": "success"
3 }
```

Тело сообщения ответа из "Direct Reply-to" в случае ошибки:

```
1 {
2   "error": "Нет подключения к серверу нейронной сети"
3 }
```

commands_exchange

Данный exchange предназначен для команд виртуальному и физическому стенду. Лабораторная работа отправляет команды для виртуального и физического стенда. Они принимают эти команды и исполняют.

command.set_state

Топик команд изменения состояний объектов.

Тело сообщения:

```
1 {
2   "target": "rejector1", // Имя объекта (rejector1, rejector2, belt)
```

```
2 "state": true // Необходимое состояние объекта
3 }
4
```

command.set_value

Топик команд изменения значений параметров объектов.

Тело сообщения:

```
1 {
2 "target": "display1", // Имя объекта (display1, display2, display3)
3 "value": 0 // Необходимое значение для объекта
4 }
```

Описание лабораторной работы

Описание сервера нейросети распознавания объектов

main.py

Основной файл, с которого начинается запуск сервера. В нем находится rest api сервера и инициализация приложения.

app_state.py

Класс AppState, которые отвечает за состояние сервера и основную логику взаимодействия. Методы этого класса вызываются из api в main.py.

data_types.py

Pydantic классы данных, которые используются в rest api и не только.

ultralytics_nn.py

Класс PredictNN, который отвечает за взаимодействие с нейронной сетью: распознавание и обучение.

exceptions.py

Исключения, которые возникают при работе с нейросетью. Все исключения унаследованы от NNErrror. Также логика конвертации исключений нейронной сети в http исключения fastapi.

tools.py

Различные полезные функции, такие как вычисление нахождения точек в координатах или конвертации обучающих метрик.

log.py

Инициализация логов. Необходима для того, чтобы настроить конфигурацию rich и перезаписать логи fastapi и uvicorn.

camera_detection.py

Скрипт позволяющий запускать распознавание с камеры, подключенной к компьютеру. Не требует сервера для запуска.

train.yaml

Конфигурация для обучения нейронной сети. При использовании приложения для обучения настраивается автоматически.

start.bat

Файл для запуска сервера. Также в нем можно указать порт и ip для сервера.

docs.bat

Файл для генерации документации. Может быть необходимо при изменении исходного кода.

venv

Виртуальное окружение Python, в котором содержатся все библиотеки необходимые для работы сервера.

trainResults

Результаты обучения нейросети.

static

Статические файлы сервера. Содержат файлы необходимые для отображения веб страниц документации.

models

Веса нейронной сети. В директории сразу содержится 5 предобученных моделей разных размеров. В скобках указаны размеры в миллионах параметров:

yolov8n.pt (3.2)

yolov8s.pt (11.2)

yolov8m.pt (25.9)

yolov8l.pt (43.7)

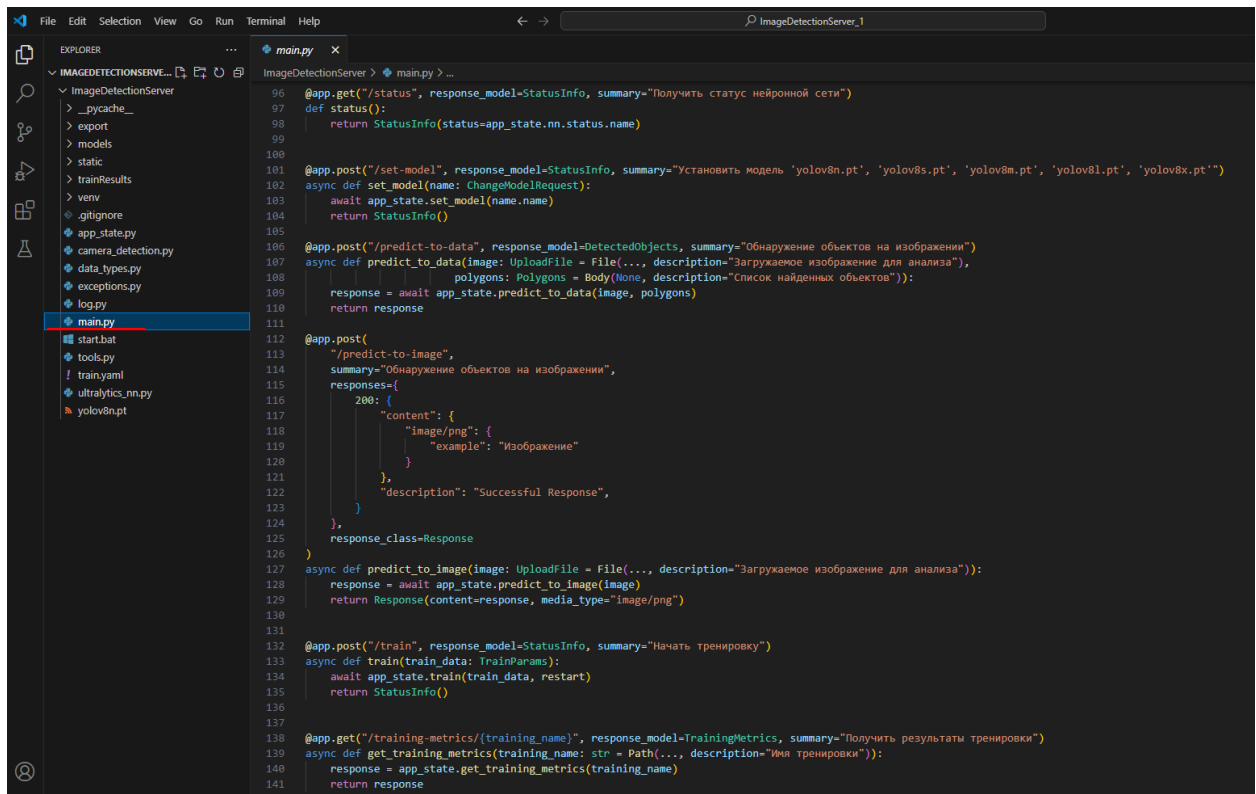
yolov8x.pt (68.2)

Также сюда можно переместить уже обученные веса из trainResults.

export

Файлы для создания виртуального окружения Python.

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
107 @app.post("/predict-to-data", response_model=DetectedObjects, summary="Обнаружение объектов на изображении")
108 async def predict_to_data(image: UploadFile = File(..., description="Загружаемое изображение для анализа"),
109     polygons: Polygons = Body(None, description="Список найденных объектов")):
110     response = await app_state.predict_to_data(image, polygons)
111     return response
112
113
114 @app.post(
115     "/predict-to-image",
116     summary="Обнаружение объектов на изображении",
117     responses={
118         200: {
119             "content": {
120                 "image/png": {
121                     "example": "Изображение"
122                 }
123             },
124             "description": "Successful Response",
125         }
126     },
127     response_class=Response
128 )
129 async def predict_to_image(image: UploadFile = File(..., description="Загружаемое изображение для анализа")):
130     response = await app_state.predict_to_image(image)
131     return Response(content=response, media_type="image/png")
132
133
134 @app.post("/train", response_model=StatusInfo, summary="Начать тренировку")
135 async def train(train_data: TrainParams):
136     await app_state.train(train_data, restart)
137     return StatusInfo()
138
139
140 @app.get("/training-metrics/{training_name}", response_model=TrainingMetrics, summary="Получить результаты тренировки")
141 async def get_training_metrics(training_name: str = Path(..., description="Имя тренировки")):
142     response = app_state.get_training_metrics(training_name)
143     return response

```

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
модель 'yolov8n.pt', 'yolov8s.pt', 'yolov8m.pt', 'yolov8l.pt', 'yolov8x.pt'")
2 async def set_model(name: ChangeModelRequest):
3     await app_state.set_model(name.name)
4     return StatusInfo()

```

Функция вызова метода обнаружения объектов на изображении:

```

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

```

23 24	<pre> async def predict_to_image(image: UploadFile = File(..., description="Загружаемое изображение для анализа")): response = await app_state.predict_to_image(image) return Response(content=response, media_type="image/png") </pre>
----------	---

Функция вызова метода обучения нейросети:

1 2 3 4	<pre> @app.post("/train", response_model=StatusInfo, summary="Начать тренировку") async def train(train_data: TrainParams): await app_state.train(train_data, restart) return StatusInfo() </pre>
------------------	---

Функция вызова метода получения метрик обучения нейросети:

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

Функция вызова метода получения информации о выбранной модели:

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

Функция вызова метода получения информации о результатах тренировок:

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

Функция вызова метода удаления результатов тренировки:

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

Функция вызова метода получения списка моделей:

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

Функция вызова метода удаления модели:

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

Функция вызова метода сохранения модели по результатам тренировки:

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

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        write_yaml_file(self.train_yaml,
11        generate_train_yaml(train_data.datasetPath, "train", "val", train_data.classes))
12        self.set_model(train_data.modelName)
```

```
13         asyncio.create_task(self.nn.train(train_data, self.train_yaml,
14 self.train_folder, self.hard_lock, on_train_end))
15         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, "results.csv")
13
14         if os.path.exists(results_dir):
15             df = pd.read_csv(results_dir, skipinitialspace=True)
16         else:
17             df = pd.DataFrame()
18
19         metrics = create_training_metrics(df)
20         return metrics
21     except Exception as e:
22         raise TrainingMetricsNotExistError()
```

Метод возвращения информации об активированной модели:

```
1 def get_model_info(self) -> ModelInfo:
2     """
3     Возвращает информацию об активированной модели.
4
5     Returns:
6     ModelInfo: Информация о модели.
7     """
8
9     classes = [
10         ModelClass(classIndex=index, className=name)
11         for index, name in self.nn.model.names.items()
12     ]
13
14
```

```
    return
    ModelInfo(name=self.nn.model.model_name[len(self.models_folder)+1:],
              classes=classes)
```

Метод, возвращающий список доступных моделей

```
1 def get_models(self):
2     """
3     Возвращает список доступных моделей.
4
5     Returns:
6     List[str]: Список имен доступных моделей.
7     """
8
9     return [
10        file_name
11        for file_name in os.listdir(self.models_folder)
12        if file_name.endswith(".pt")
13    ]
```

Метод, возвращающий список результатов обучений:

```
1 def get_training_results(self) -> TrainingResults:
2     """
3     Возвращает список результатов обучений.
4
5     Returns:
6     TrainingResults: Список результатов обучений.
7
8     Raises:
9     TrainingResultsError: Если возникла ошибка при получении
10    результатов обучений.
11    """
12
13    try:
14        folder_names = [
15            folder_name
16            for folder_name in os.listdir(self.train_folder)
17            if os.path.isdir(os.path.join(self.train_folder, folder_name))
18        ]
19
20        training_results = []
21        for folder_name in folder_names:
22            weights_dir = os.path.join(self.train_folder, folder_name, "weights")
23            weight_files = [
```

```

21         Weights(name=file_name)
22         for file_name in os.listdir(weights_dir)
23             if file_name.endswith(".pt")
24         ]
25         training_result = TrainingResult(trainingResultName=folder_name,
26 weights=weight_files)
27         training_results.append(training_result)
28
29         return TrainingResults(trainingResults=training_results)
30     except Exception as e:
31         raise TrainingResultsError()         except:
32         raise ModelSaveError()

```

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,
request.trainingResultName)
6          if not os.path.exists(training_result_path):
7              raise TrainingResultNotExistError()
8
9          weight_file_path = os.path.join(training_result_path, "weights",
request.weightsName)
10         if not os.path.exists(weight_file_path):
11             raise WeightsNotExistError()
12
13         new_model_path = os.path.join(self.models_folder,
request.newModelName)
14         if os.path.exists(new_model_path):
15             raise ModelAlreadyExistError()
16
17         if not request.newModelName.endswith(".pt"):
18             raise ModelNoPTError()
19         try:
20             shutil.copy(weight_file_path, new_model_path)
21         except:
22             raise ModelSaveError()
23
24         return

```

В файле log.py метод инициализация логгера:

```

1  def setup_rich_logger():
2      """
3      Инициализирует логгер
4      """
5      for name in logging.root.manager.loggerDict.keys():
6          logging.getLogger(name).handlers = []
7          logging.getLogger(name).propagate = True
8      logging.basicConfig(

```

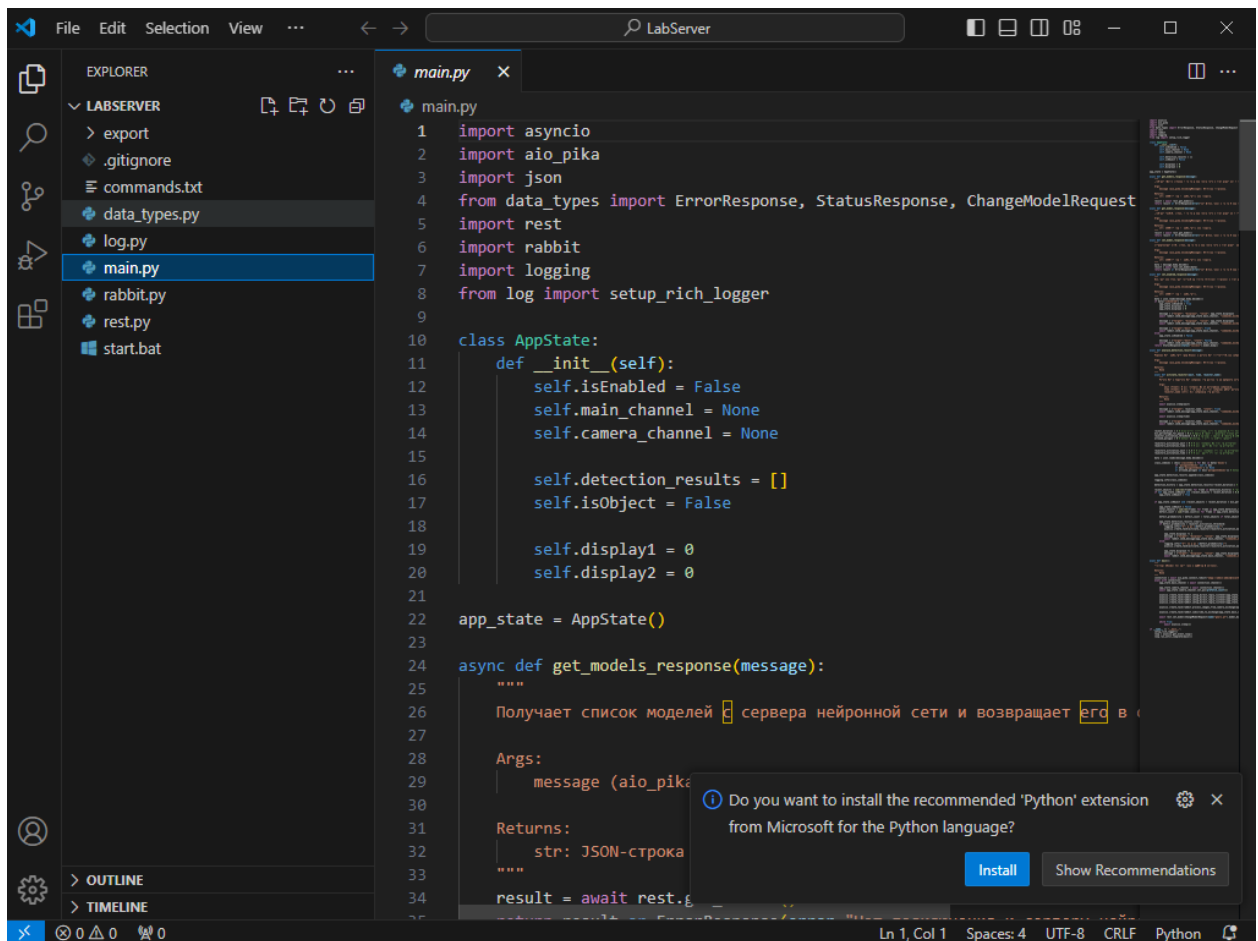
```

9     level="INFO",
10    format="%(message)s",
11    datefmt="[%X]",
12    handlers=[RichHandler(markup=True, rich_tracebacks=True)],
    )

```

Описание сервера лабораторной работы

Сперва рассмотрим код сервера лабораторной работы, а именно файл *main.py*.



Сервер лабораторной работы

Инициализация переменных используемых в лабораторной работе:

```

1 class AppState:
2     def __init__(self):
3         self.isEnabled = False
4         self.main_channel = None
5         self.camera_channel = None
6
7         self.detection_results = []
8         self.isObject = False
9
10        self.display1 = 0

```

```
11 self.display2 = 0
12
13 app_state = AppState()
```

Метод получения списка моделей с сервера нейронной сети:

```
1 async def get_models_response(message):
2     """
3     Получает список моделей с сервера нейронной сети и возвращает его в
4     ответ на сообщение.
5
6     Args:
7         message (aio_pika.IncomingMessage): Входящее сообщение.
8
9     Returns:
10        str: JSON-строка с результатом или ошибкой.
11    """
12    result = await rest.get_models()
13    return result or ErrorResponse(error="Нет подключения к серверу
14    нейронной сети").model_dump()
```

Метод получения текущей модели с сервера нейронной сети:

```
1 async def get_model_response(message):
2     """
3     Получает текущую модель с сервера нейронной сети и возвращает её в
4     ответ на сообщение.
5
6     Args:
7         message (aio_pika.IncomingMessage): Входящее сообщение.
8
9     Returns:
10        str: JSON-строка с результатом или ошибкой.
11    """
12    result = await rest.get_model()
13    return result or ErrorResponse(error="Нет подключения к серверу
14    нейронной сети").model_dump()
```

Метод получения новой модели на сервере нейронной сети:

```
1 async def set_model_response(message):
2     """
3     Устанавливает новую модель на сервере нейронной сети и возвращает
4     результат в ответ на сообщение.
5
6     Args:
```

```

6     message (aio_pika.IncomingMessage): Входящее сообщение.
7
8     Returns:
9         str: JSON-строка с результатом или ошибкой.
10        """
11    data = message.body.decode()
12    result = await rest.set_model(data)
13    return result or ErrorResponse(error="Нет подключения к серверу
нейронной сети").model_dump()

```

Включает или выключает систему на основе входящего сообщения и возвращает статус:

```

1  async def set_enabled_response(message):
2      """
3      Включает или выключает систему на основе входящего сообщения и
4      возвращает статус.
5
6      Args:
7          message (aio_pika.IncomingMessage): Входящее сообщение.
8
9      Returns:
10         str: JSON-строка с результатом.
11         """
12     data = json.loads(message.body.decode())
13     if data["isEnabled"] == True:
14         app_state.isEnabled = True
15         app_state.display1 = 0
16         app_state.display2 = 0
17
18     message = {"target": "display1", "value": app_state.display1}
19     await rabbit.send_message(app_state.main_channel,
20 "commands_exchange", "command.set_value", message, is_debug=False)
21
22     message = {"target": "display2", "value": app_state.display2}
23     await rabbit.send_message(app_state.main_channel,
24 "commands_exchange", "command.set_value", message, is_debug=False)
25
26     message = {"target": "belt", "state": True}
27     await rabbit.send_message(app_state.main_channel,
28 "commands_exchange", "command.set_state", message, is_debug=False)
29     else:
30         app_state.isEnabled = False
31
32     message = {"target": "belt", "state": False}

```

```

32         await rabbit.send_message(app_state.main_channel,
33 "commands_exchange", "command.set_state", message, is_debug=False)
34         return StatusResponse(status="success").model_dump()

```

Анализирует результаты обнаружения и активирует соответствующие механизмы на основе анализа:

```

1  async def analyze_detection_result(message):
2      """
3      Анализирует результаты обнаружения и активирует соответствующие
4      механизмы на основе анализа.
5
6      Args:
7
8      message (aio_pika.IncomingMessage): Входящее сообщение.
9
10     Returns:
11     None
12     """
13
14     async def activate_rejector(wait, time, rejector_name):
15         """
16         Активирует и деактивирует механизм отбраковки через заданные
17         интервалы времени.
18
19         Args:
20
21         wait (float): Время ожидания перед активацией механизма.
22         time (float): Время, в течение которого механизм будет активен.
23         rejector_name (str): Имя механизма отбраковки.
24
25         Returns:
26         None
27         """
28         await asyncio.sleep(wait)
29
30         message = {"target": rejector_name, "state": True}
31         await rabbit.send_message(app_state.main_channel,
32 "commands_exchange", "command.set_state", message, is_debug=False)
33
34         await asyncio.sleep(time)
35
36         message = {"target": rejector_name, "state": False}
37         await rabbit.send_message(app_state.main_channel,
38 "commands_exchange", "command.set_state", message, is_debug=False)

```

```
37 recent_duration = 3 # Количество состояний, которые анализируются для
38 проверки, вышел ли объект за границы камеры
39 min_percentage_for_pause = 0.2 # Какой процент с объектами должен
40 быть, чтобы определить, что объекта больше нет в области видимости
41 камеры
42 rejector_activation_threshold = 0.5 # Начиная с какого процента
43 предполагаемого брака активируется сбрасыватель
44 allowed_polygon = 0 # Индекс полигона, в котором ищутся объекты
45
46 rejector1_activation_wait = 0 # Время ожидания первого сбрасывателя
47 rejector1_activation_time = 4 # Время работы первого сбрасывателя
48
49 rejector2_activation_wait = 3.5 # Время ожидания второго сбрасывателя
50 rejector2_activation_time = 4 # Время работы второго сбрасывателя
51
52 data = json.loads(message.body.decode())
53
54 class_indexes = [box['classIndex'] for box in data['boxes']]
55                 if 'polygonIndexes' not in box
56                 or box['polygonIndexes'] == None
57                 or allowed_polygon in box['polygonIndexes']] # Индексы
58 найденных объектов
59
60 app_state.detection_results.append(class_indexes)
61
62 logging.info(class_indexes)
63
64 detection_history = app_state.detection_results[-recent_duration:] # История
65 за несколько последний состояний
66
67 recent_objects = sum(len(frame) for frame in detection_history) # Общее
68 число срабатываний
69 if not app_state.isObject and (recent_objects > recent_duration * 0.8): #
70 Достаточно ли число срабатываний, чтобы считать это объектом
71     app_state.isObject = True
72
73
74 if app_state.isObject and (recent_objects < recent_duration *
75 min_percentage_for_pause):
76
77     app_state.isObject = False
78     total_objects = sum(len(frame) for frame in app_state.detection_results)
79     defect_count = sum(frame.count(1) for frame in
80 app_state.detection_results)
81
```

```
82     defect_probability = defect_count / total_objects if total_objects > 0 else 0
83 # Шанс того, что объект является браком
84
85     app_state.detection_results.clear()
86     if defect_probability > rejector_activation_threshold:
87         logging.info(f"Это брак ({defect_probability})")
88         asyncio.create_task(activate_rejector(rejector1_activation_wait,
89 rejector1_activation_time, "rejector1"))
90
91         app_state.display1 += 1
92         message = {"target": "display1", "value": app_state.display1}
93         await rabbit.send_message(app_state.main_channel,
94 "commands_exchange", "command.set_value", message, is_debug=False)
95     else:
96         logging.info(f"Это не брак ({defect_probability})")
97         asyncio.create_task(activate_rejector(rejector2_activation_wait,
98 rejector2_activation_time, "rejector2"))
99
100        app_state.display2 += 1
101        message = {"target": "display2", "value": app_state.display2}
102        await rabbit.send_message(app_state.main_channel,
103 "commands_exchange", "command.set value", message, is debug=False)
```



Sk
Resident

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

