ProgramLab
INNOVATIVE DIGITAL SYSTEMS

РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

ВИРТУАЛЬНЫЙ УЧЕБНЫЙ КОМПЛЕКС «РАСПОЗНАВАНИЕ РУКОПИСНЫХ СИМВОЛОВ И РАБОТА С ВНЕШНИМИ ДАТА СЕТАМИ»

PLLLC.RU

ОГЛАВЛЕНИЕ

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.

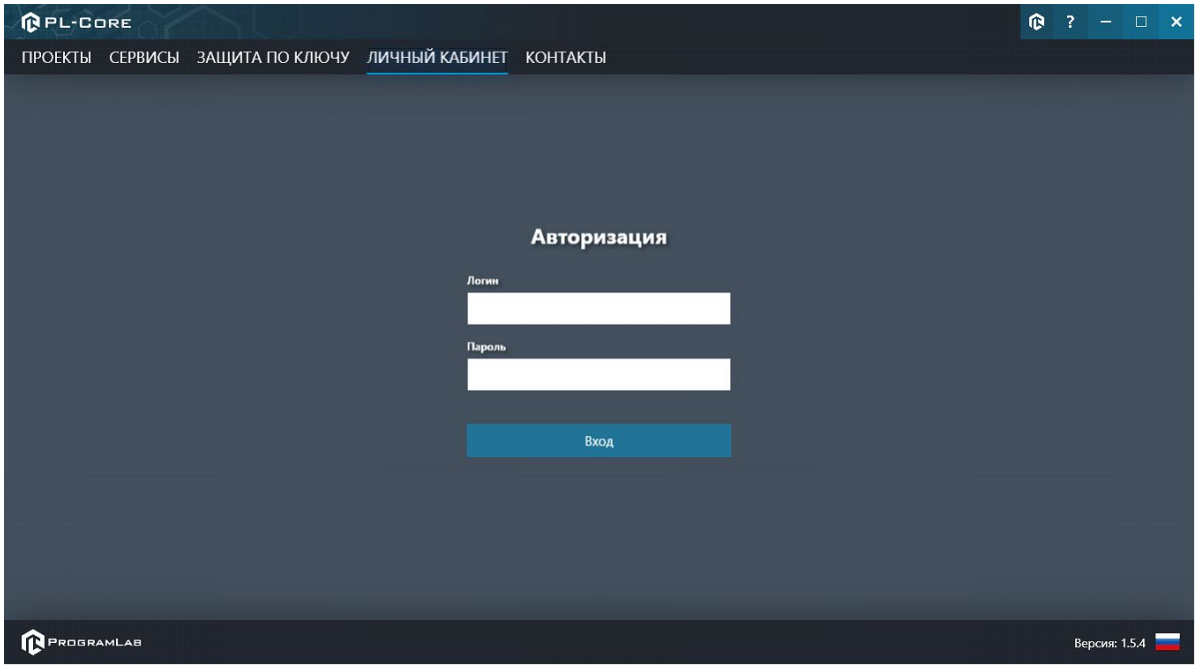
В случае поставки программного комплекса без ПК вам необходимо установить программное обеспечение с USB-носителя.

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»**.

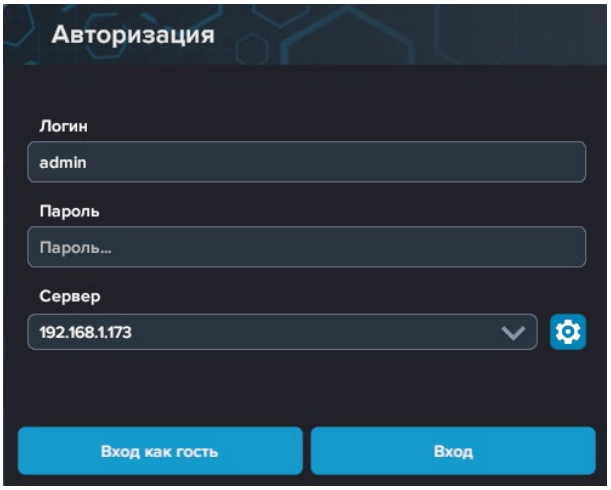
По умолчанию в системе создается пользователь с именем Администратор и ролью Администратор. Этот пользователь не может быть удален, но его параметры могут быть изменены.

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

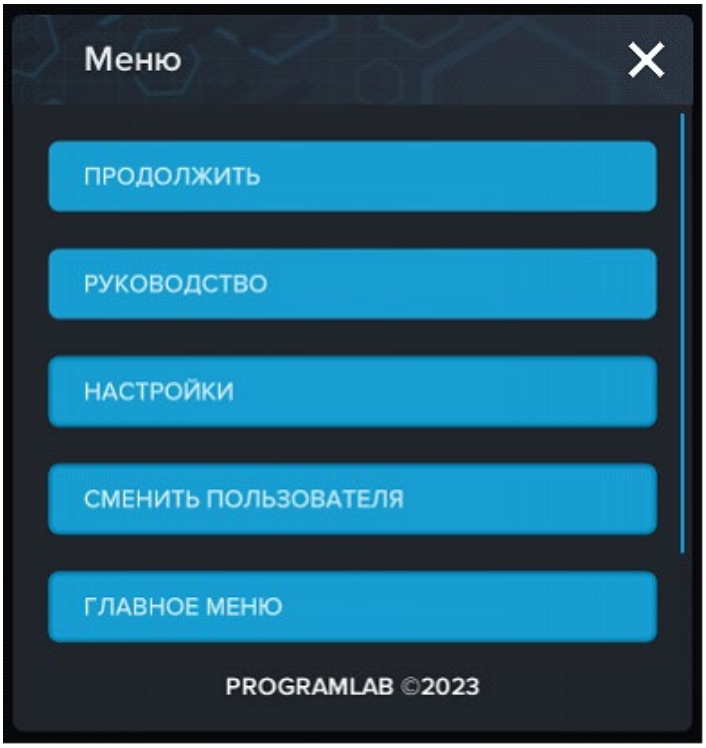
Запуск и управление в программе

 — Левая кнопка мыши – действие, выбор объекта;

 — Правая кнопка мыши – вращение камеры;

 — Вращение колеса мыши – приближение\отдаление камеры;

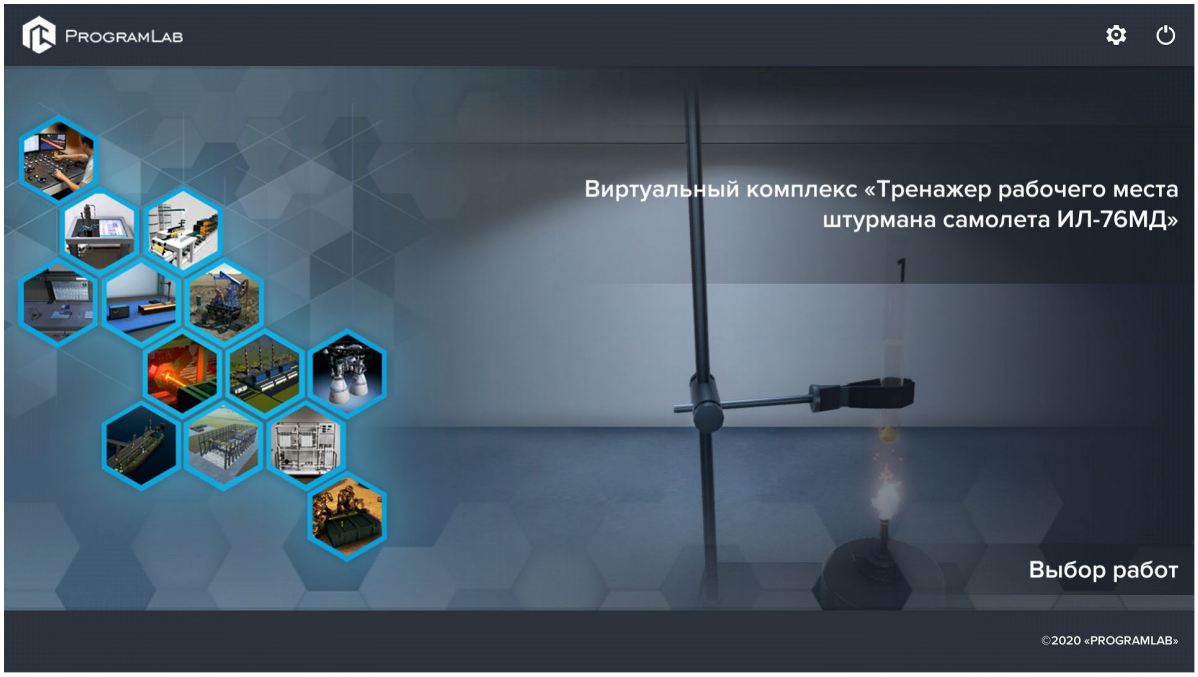
— Вызов меню программы.



Меню программы

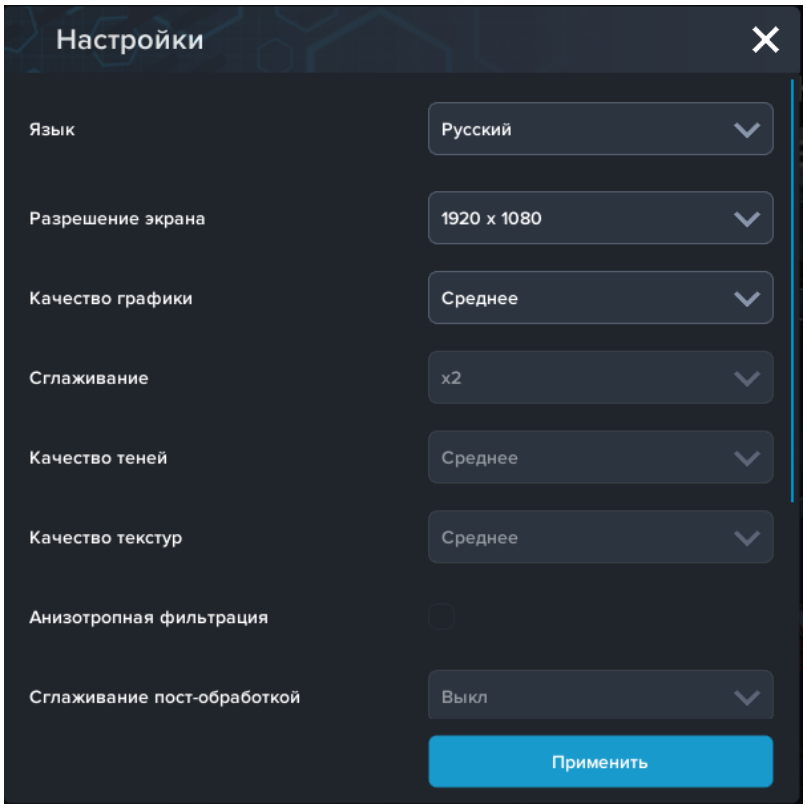
«Продолжить» – вернуться в программу;
«Руководство» – вызвать руководство пользователя;
«Настройки» – настройки параметров графики;
«Сменить пользователя» – пройти авторизацию повторно;
«Главное меню» – выход в главное меню;
«Выход» – выход из программы.

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.

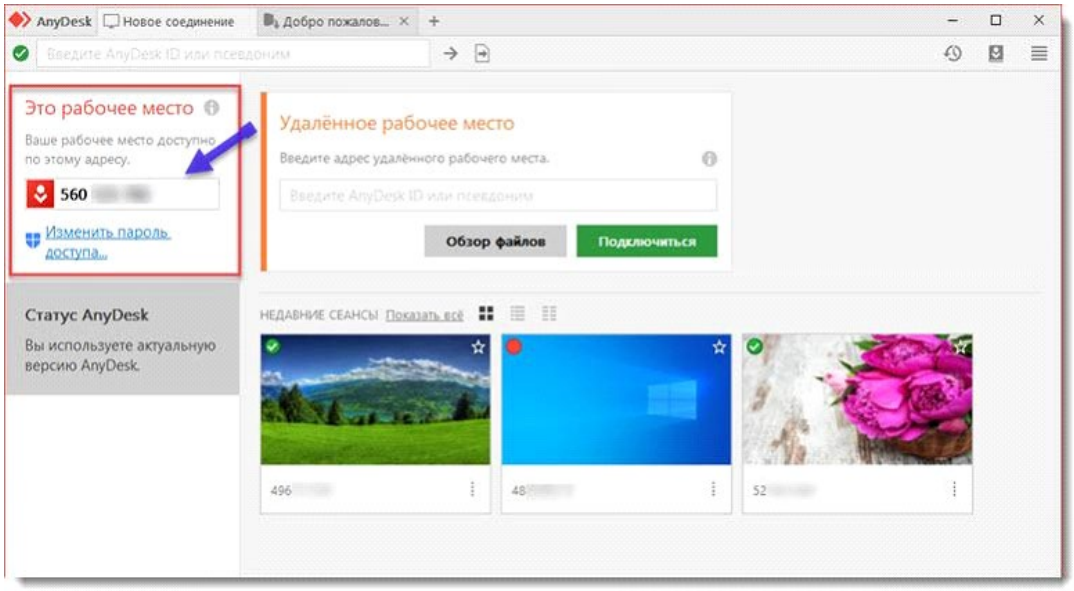
Для того чтобы специалист смог подключиться к вашему ПК и устранить проблемы вам необходимо запустить ПО для дистанционного управления ПК Anydesk и сообщить данные для доступа.

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**».

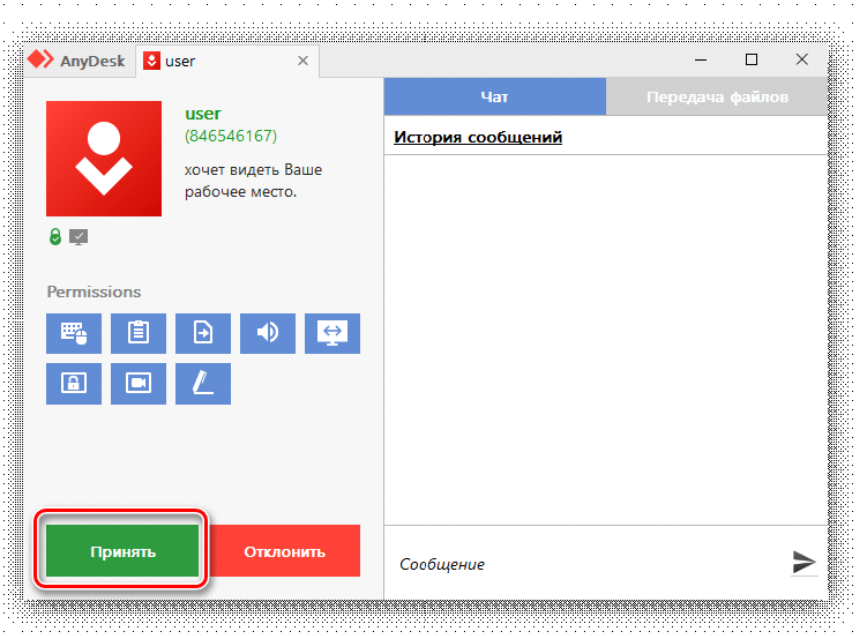
При первом запуске может возникнуть окно с требованием предоставить разрешение. Необходимо нажать на кнопку **Разрешить доступ.**

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.

ервную 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.

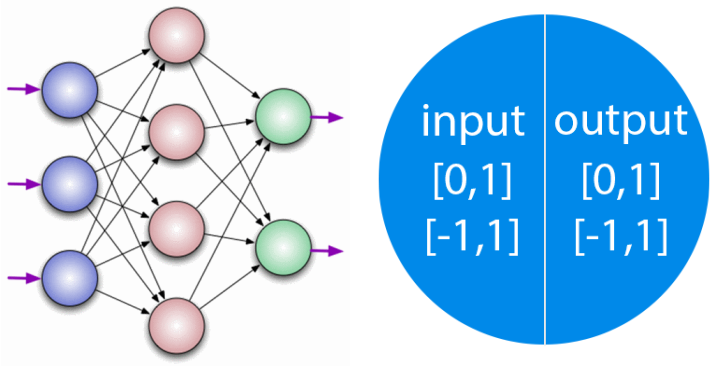
Возрождение интереса к нейронным сетям и революция в глубоком обучении произошли лишь в последние годы благодаря индустрии компьютерных игр. Современные игры требуют сложных вычислений для обработки большого числа операций. В итоге производители начали выпускать **графические процессоры (GPU)**, которые 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

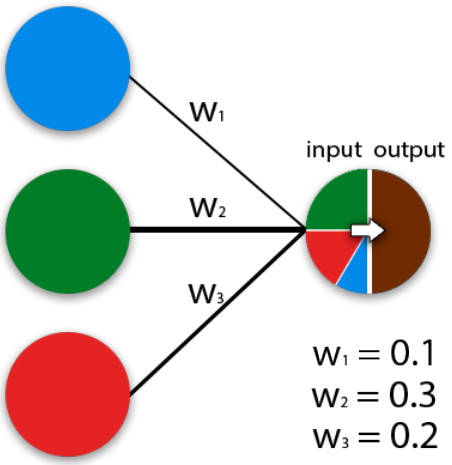
Главное отличие нейросетевых моделей от классических заключается в их структуре. Основные элементы, из которых он состоит – искусственные нейроны и связи между ними.

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.

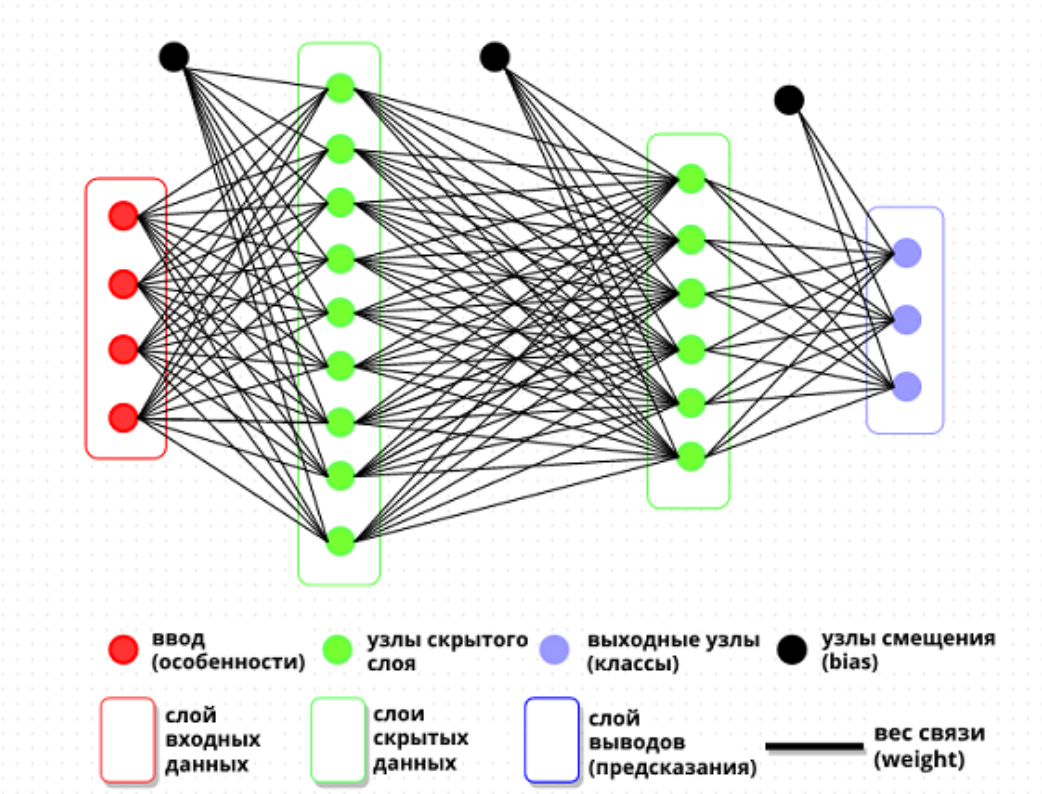
Искусственный интеллект — понятие более широкое. Оно включает в себя не только нейронные сети, но и другие методы обработки информации, в том числе экспертные и логические программы. Нейронные сети — один из видов искусственного интеллекта. Их отличительная особенность — обучение и адаптация в основе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

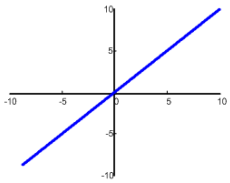
Чем точнее и подробнее запрос, тем быстрее и качественнее получится результат.

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

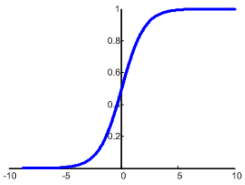
$$f(x) = x$$



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

Сигмоид

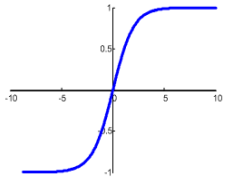
$$f(x) = \frac{1}{1+e^{-x}}$$



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

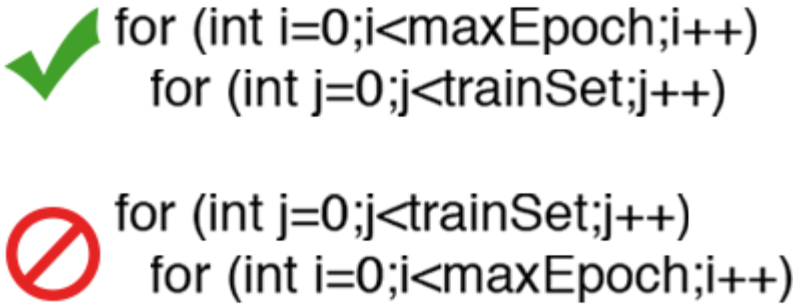
Тренировочный сет — это последовательность данных, которыми оперирует нейронная сеть.

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.



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:

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

Root MSE:

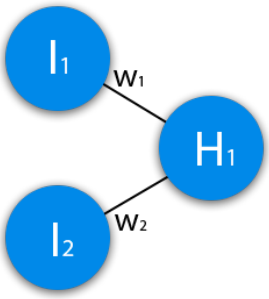
$$\sqrt{\frac{(i_1-a_1)^2+(i_2-a_2)^2+\ldots+(i_n-a_n)^2}{n}}$$

Arctan:

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

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



$$1) H_{1_{input}} = (I_1 * w_1) + (I_2 * w_2)$$

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

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 = sigmoid(0.45)=0.61

H_2input = 1*0.78+0*0.13=0.78

H_2output = sigmoid(0.78)=0.69

O_1input = 0.61*1.5+0.69*-2.3=-0.672

O_1output = sigmoid(-0.672)=0.33

O_1ideal = 1 (0xor1=1)

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.

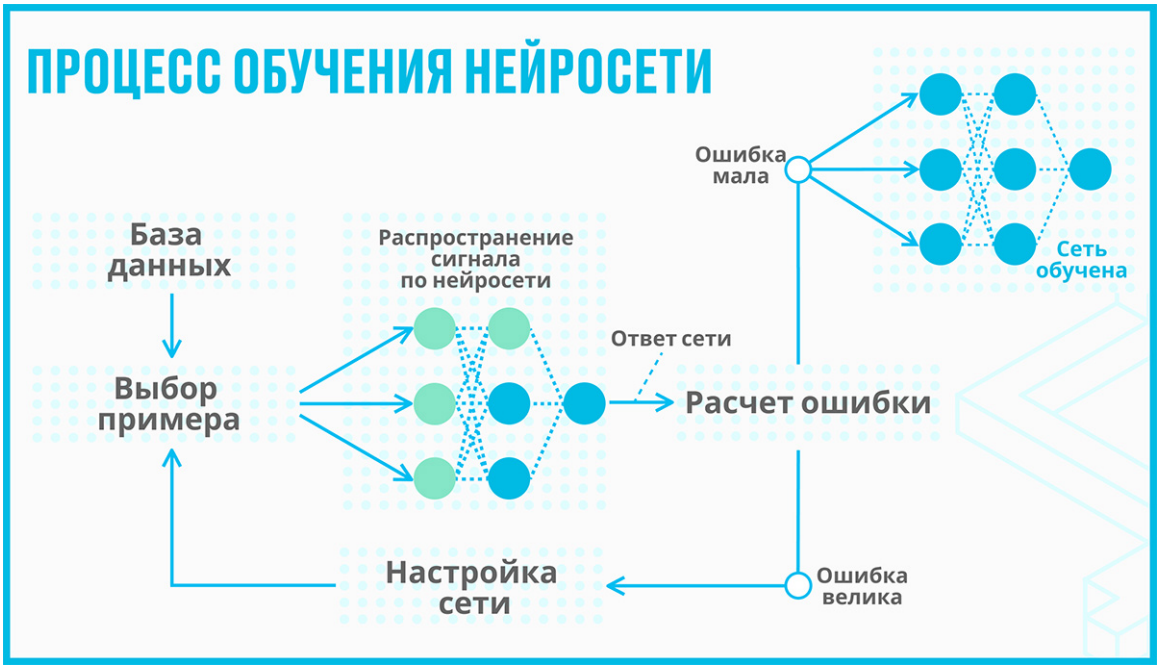
Существуют следующие методы обучения:

- **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.

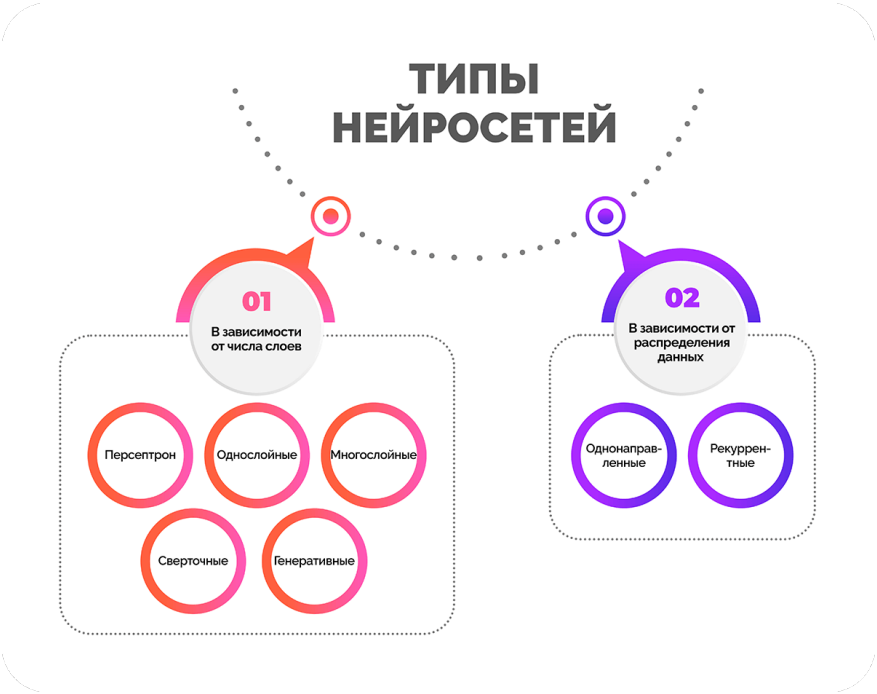
- **Без учителя.** Метод применяют, если нет правильных ответов на входные сигналы. Сеть в этом случае, используя собственную память, делит объекты на классы, то есть начинает кластеризацию. Эталонные ответы при этом не показаны. Данный тип обучения называют глубоким: система все время обучается сама.

- **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:

- **Персептрон** – 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 реккурентные в зависимости от распределения данных по синапсам:

- **Однонаправленные** (прямого распространения). Сигнал движется от входного слоя к выходному, обратного движения нет. Нейросети такого типа используют для распознавания речи, кластеризации, составления прогнозов.
- **Реккурентные** (с обратными связями). Реккурентные нейронные сети предполагают, что любое количество сигналов может перемещаться в разных направлениях, в том числе от выхода к входу.

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

Применения нейросетей

Разные варианты нейросетей создаются для решения нескольких типов различных задач:

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

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

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

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

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

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

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

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

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

Применение:

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

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

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

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

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

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

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

Преимущества и недостатки нейросетей

Преимущества нейросетей:

– незаменимы в сфере автоматизации процессов;

– спасают там, где может навредить человеческий фактор;

– экономят время на выполнении рутинных задач;

– постоянно обучаются.

Недостатки:

– напрямую зависят от вводимых данных, поэтому сильно подвержены влиянию;

– чтобы получить действительно хорошую рабочую сеть, нужно потратить много времени на её обучение;

– занимают много места на сервере и требуют больших вычислительных мощностей.

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

Установка и настройка сервера при использовании комплекса под Astra Linux

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

Данные

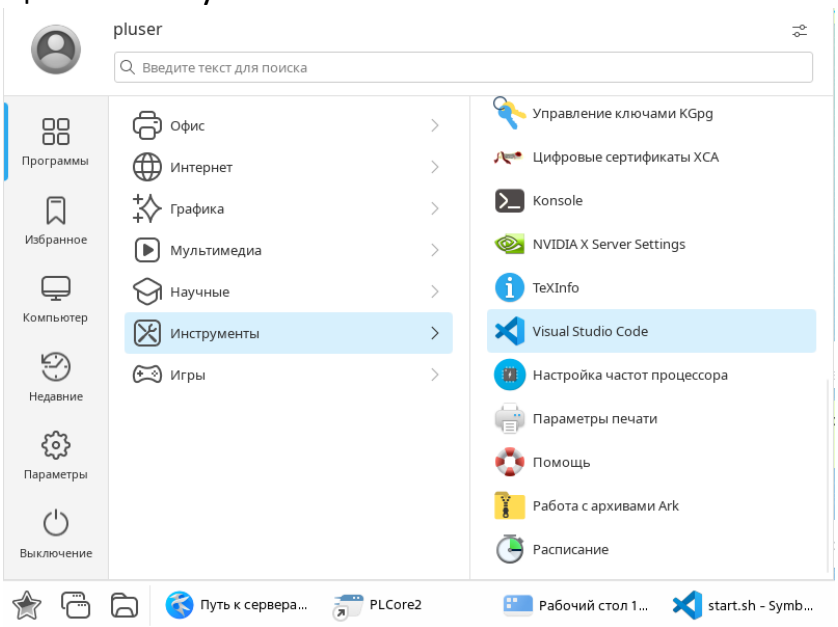
https://www.python.org/downloads/release/python-3124/

https://docs.anaconda.com/miniconda/miniconda-install/

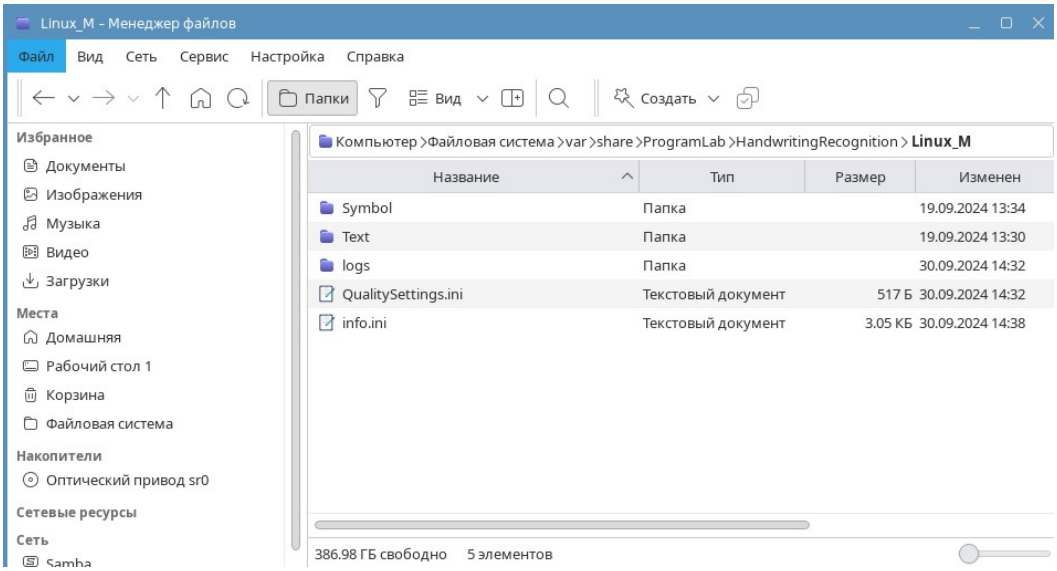
После установки всех программ перезапустите компьютер.

Установка и настройка сервера распознавания текста

Когда все необходимые программы установлены, запустите Visual studio code при помощи меню «Пуск».

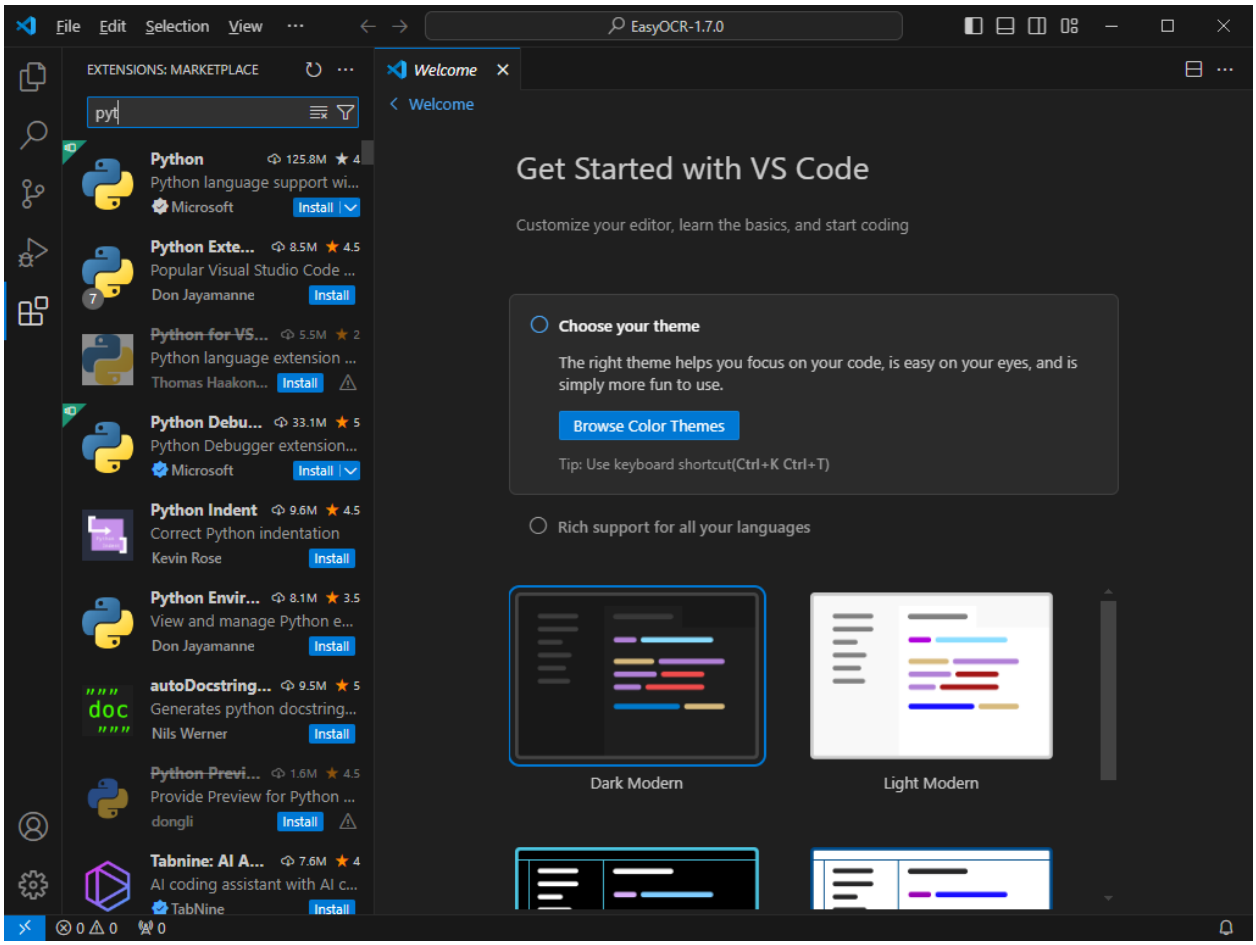


После запуска VSCode нажмите на вкладку File ->Open folder и выберите путь к папке где располагается необходимый вам сервер (по умолчанию сервера находится в папке проекта: var/share/ProgramLab/HandwritingRecognition/Linux_M/Text).



Открытие папки сервера

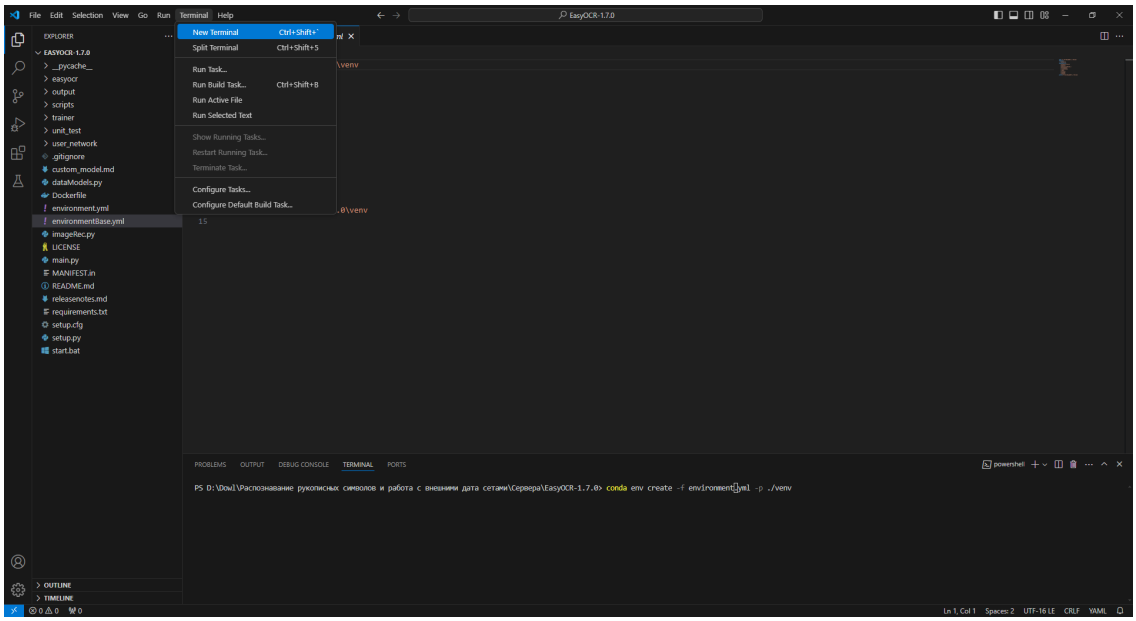
Перейдите во вкладку Extensions и в нем установите Python.



Установка Python

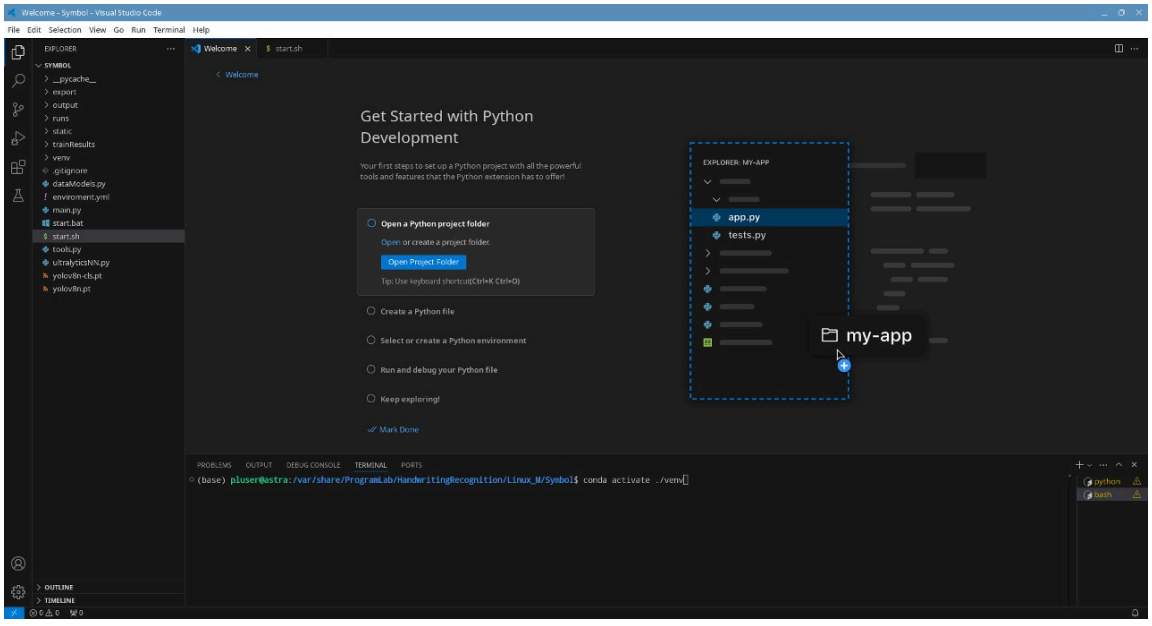
После установки питона нажмите на вкладку Terminal далее нажмите на New terminal.

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

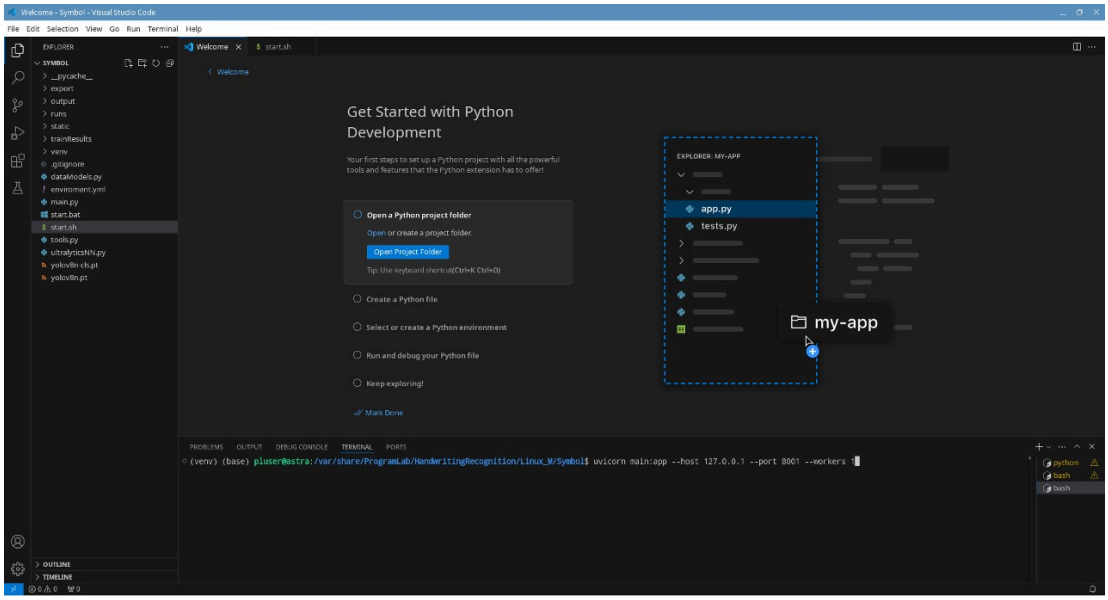


Установка необходимых пакетов

Для запуска сервера необходимо активировать окружение проекта, для этого введите команду:
conda activate ./venv



Вы можете изменить IP адрес сервера, если захотите запустить сервер на отдельном компьютере, для при вводе команды используйте IP адрес компьютера, на котором будет расположен сервер.



Команда запуска

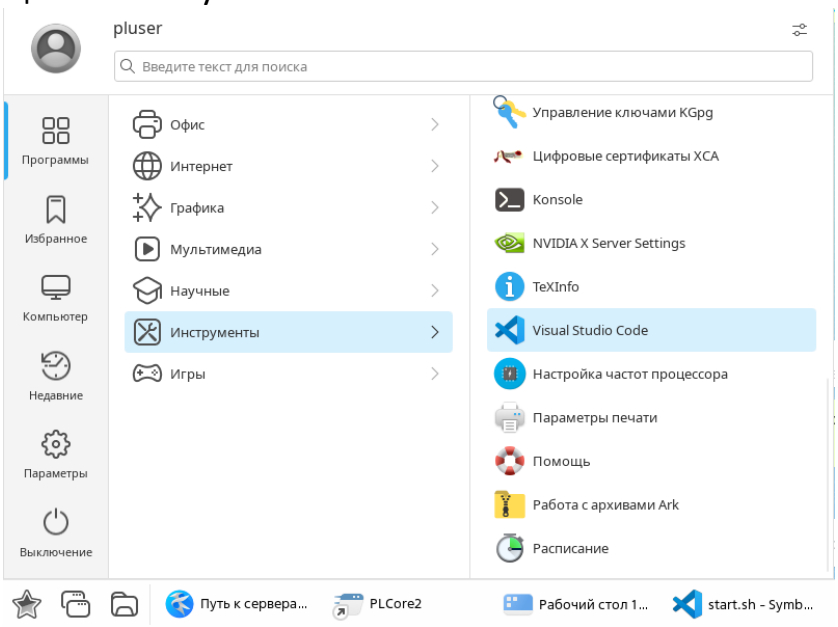
Если после запуска появляется ошибка о недостающем пакете, то его необходимо установить, для этого вернитесь в Visual studio и в терминале введите команду *conda init*, после чего введите команду *conda activate ./venv,* теперь вам необходимо ввести команду для установки необходимого пакета *conda install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

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

Запустите сервер и проверьте, что все работает.

Установка и настройка сервера распознавания символов

Когда все необходимые программы установлены, запустите Visual studio code при помощи меню «Пуск».

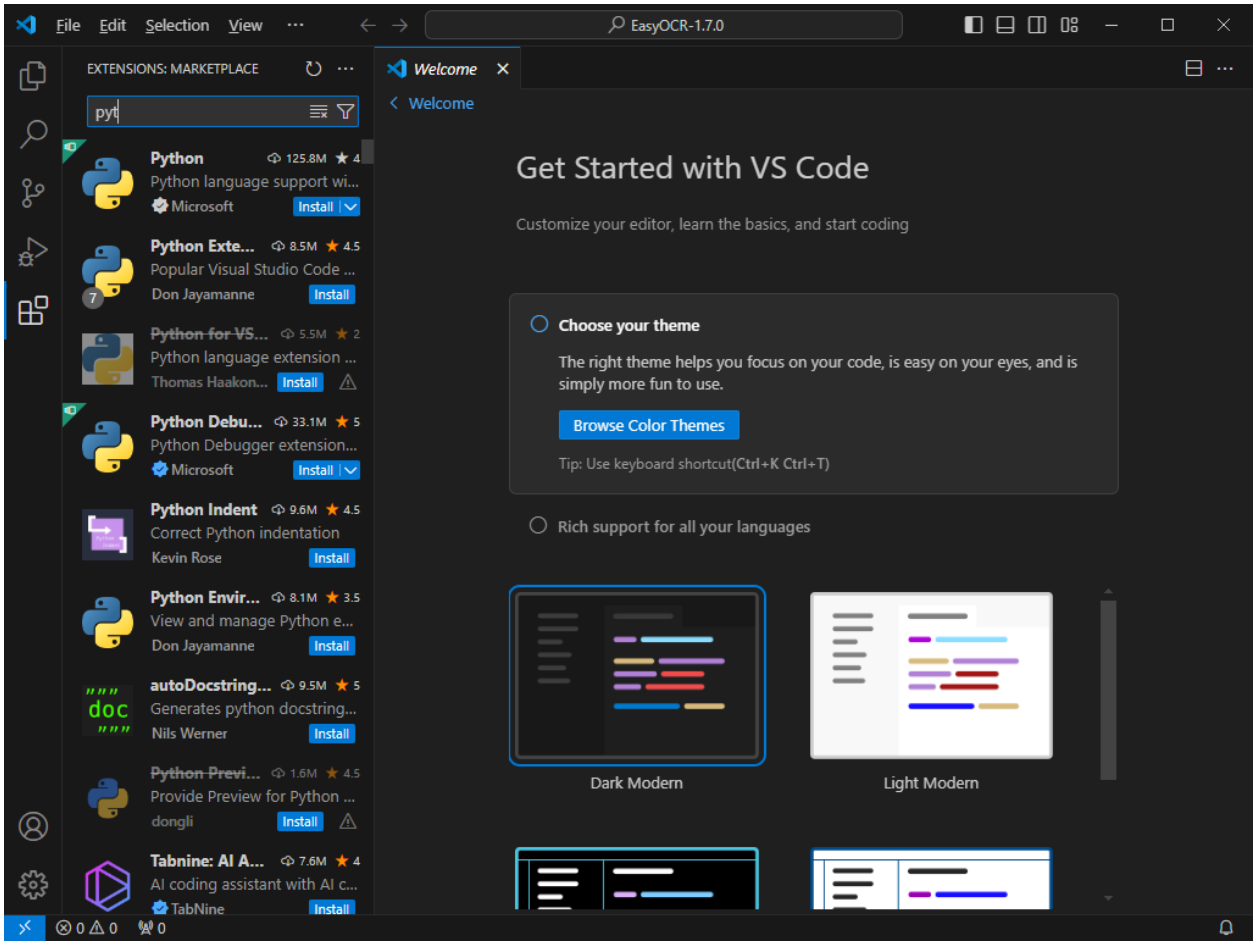


После запуска VSCode нажмите на вкладку File ->Open folder и выберите путь к папке где располагается необходимый вам сервер (по умолчанию сервера

находится в папке проекта:
var/share/ProgramLab/HandwritingRecognition/Linux_M/Symbol).

Открытие папки сервера

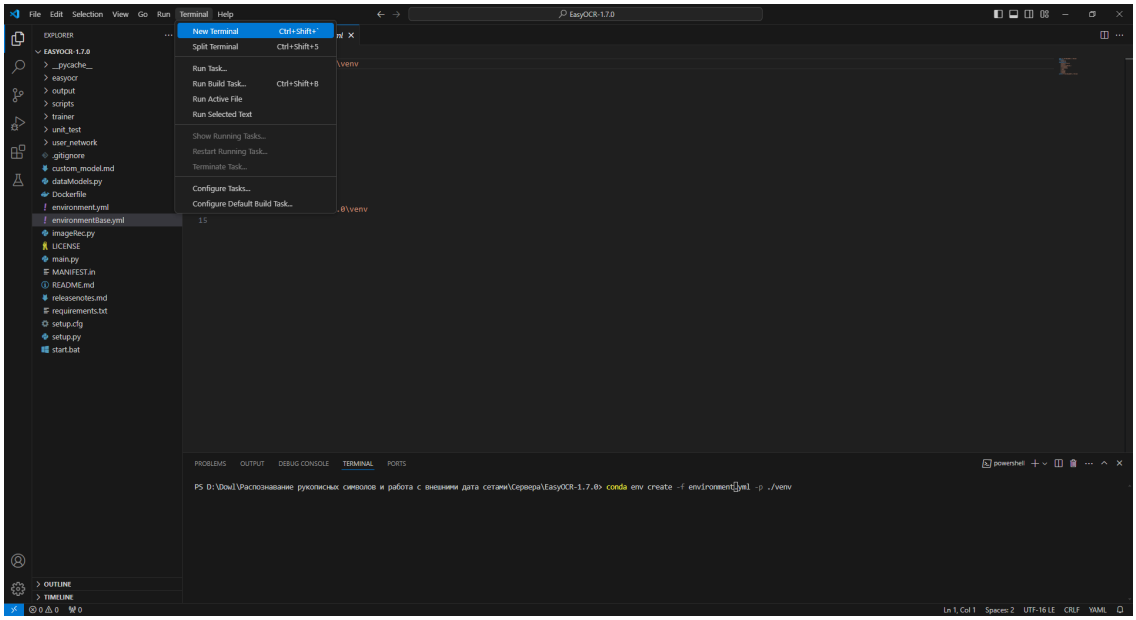
Перейдите во вкладку Extensions и в нем установите Python.



Установка Python

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

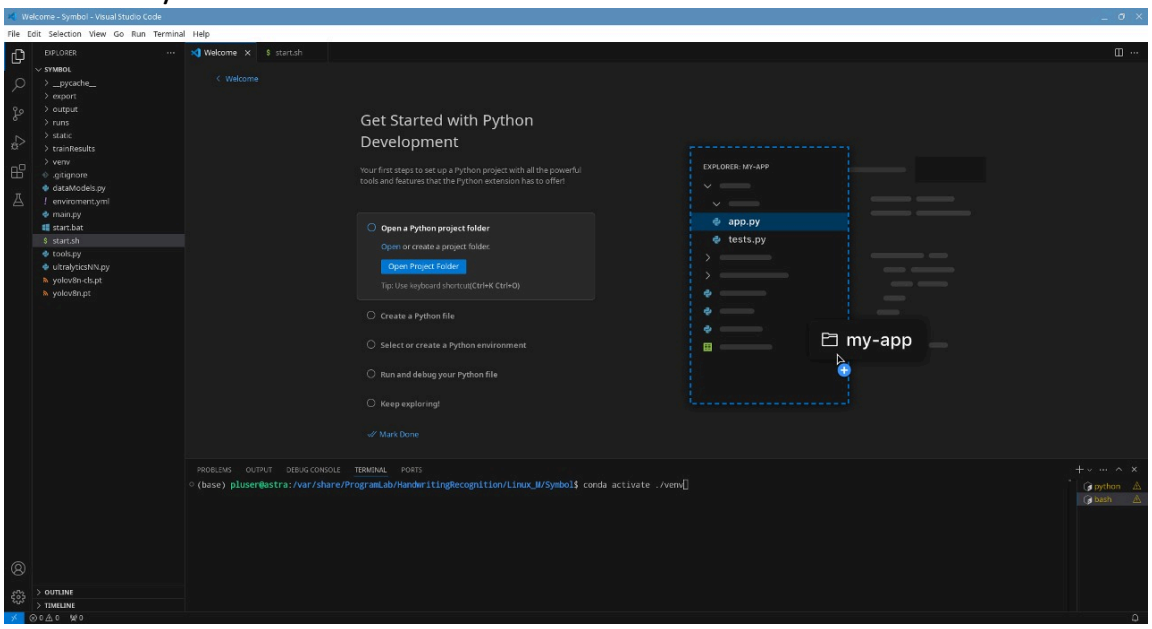
В появившемся поле введите команду « *conda env create -f «Название файла.yml» -p ./venv* » (Можете использовать один из двух файлов а именно «environmentBase.Yml » или « *environment.Yml* »), после чего начнется установка, дождитесь её завершения.



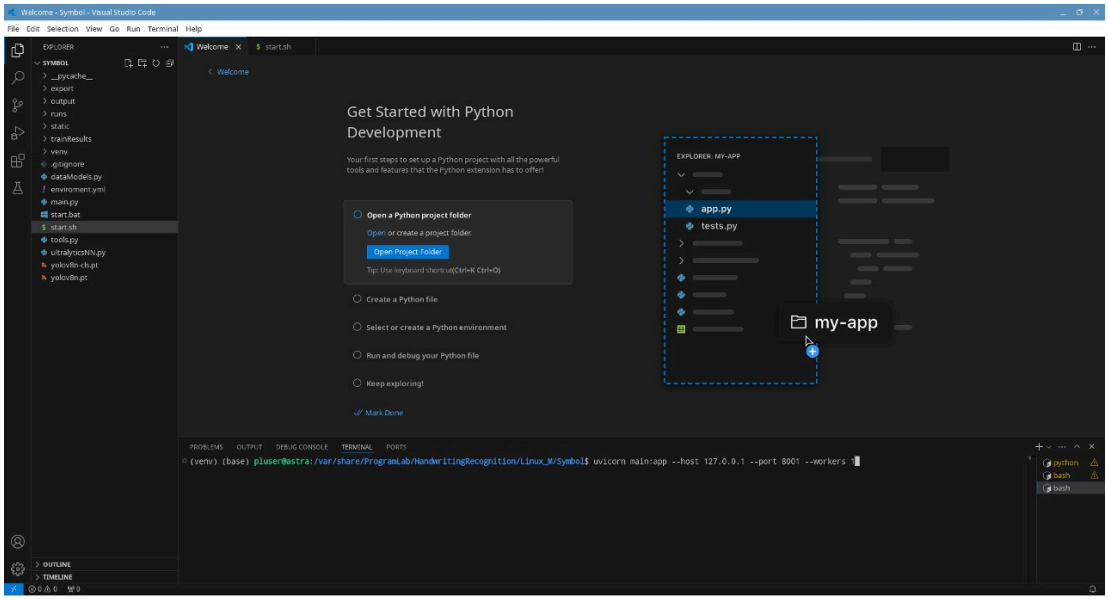
Установка необходимых пакетов

Для запуска сервера необходимо активировать окружение проекта, для этого введите команду:

conda activate ./venv



Вы можете изменить IP адрес сервера, если захотите запустить сервер на отдельном компьютере, для при вводе команды используйте IP адрес компьютера, на котором будет расположен сервер.



Команда запуска

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

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

Запустите сервер и проверьте, что все работает.

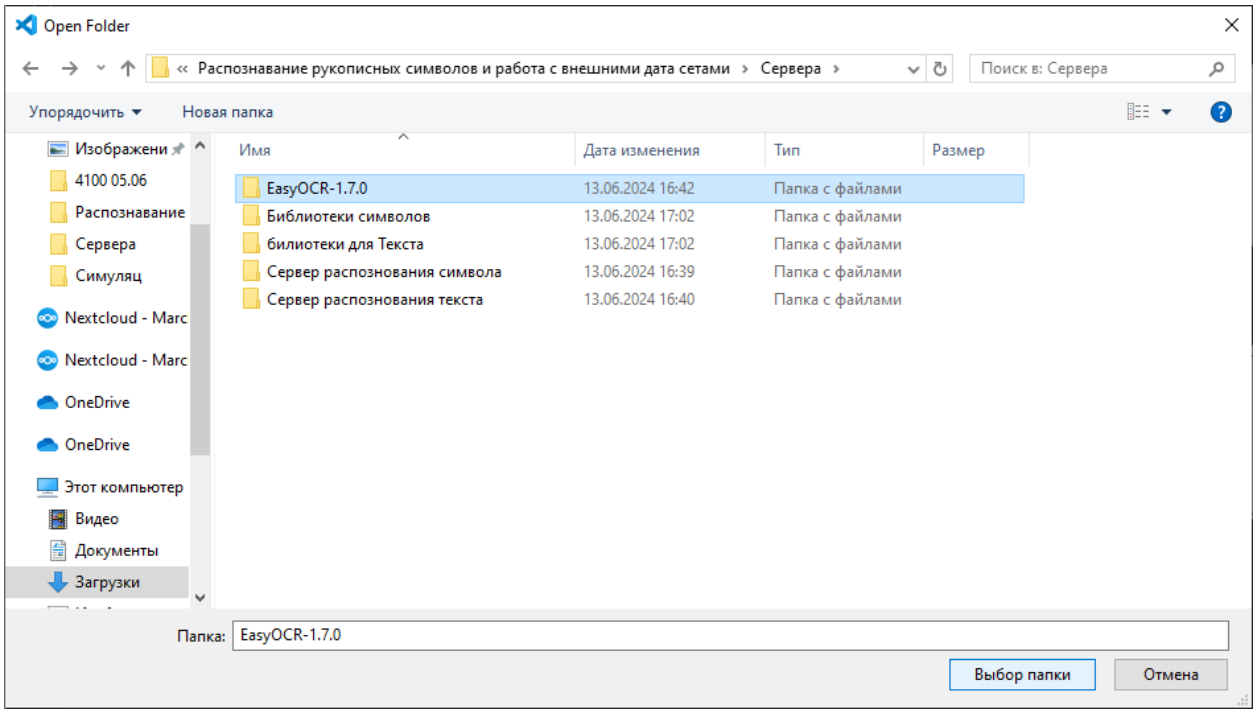
Установка и настройка сервера при использовании комплекса под Windows.

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

Данные программы можно скачать по следующим ссылкам:
https://code.visualstudio.com/docs/?dv=win64user
https://www.python.org/downloads/release/python-3124/
https://docs.anaconda.com/miniconda/
После установки всех программ перезапустите компьютер.

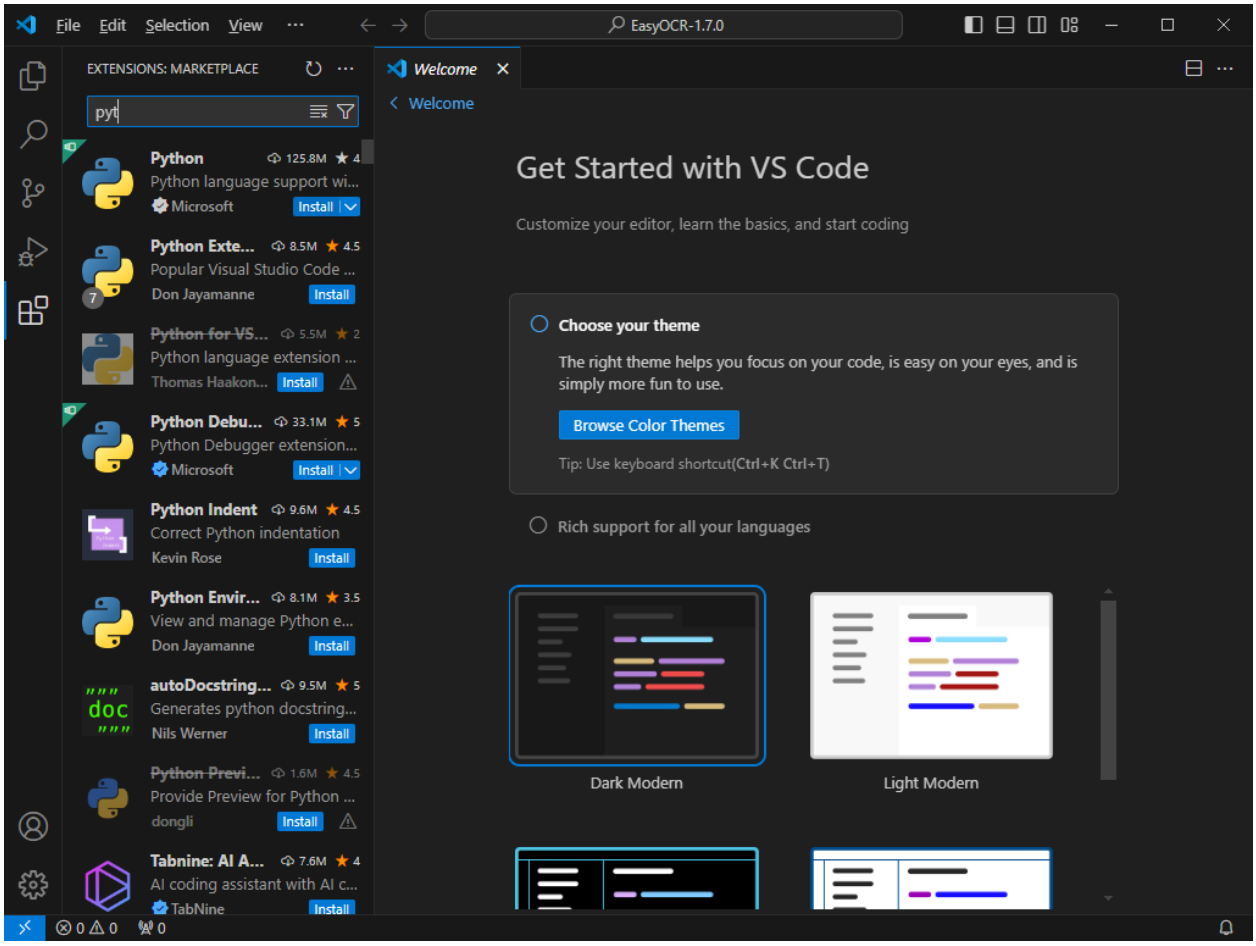
Установка и настройка сервера распознавания текста

Когда все необходимые программы установлены, запустите Visual studio code, нажмите на вкладку File ->Open folder и выберите путь к папке где располагается необходимый вам сервер (по умолчанию сервер находится в папке проекта: C:\Program Files (x86)\ProgramLab\HandwritingRecognition).



Открытие папки сервера

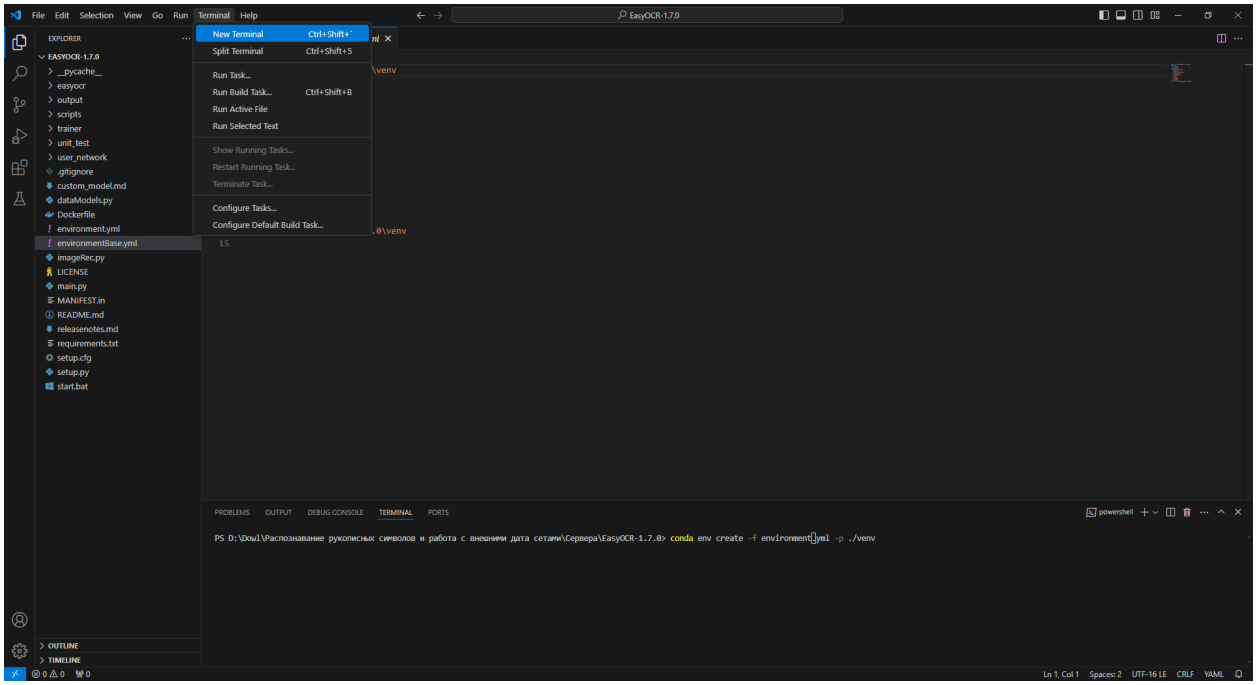
Перейдите во вкладку Extensions и в нем установите Python.



Установка Python

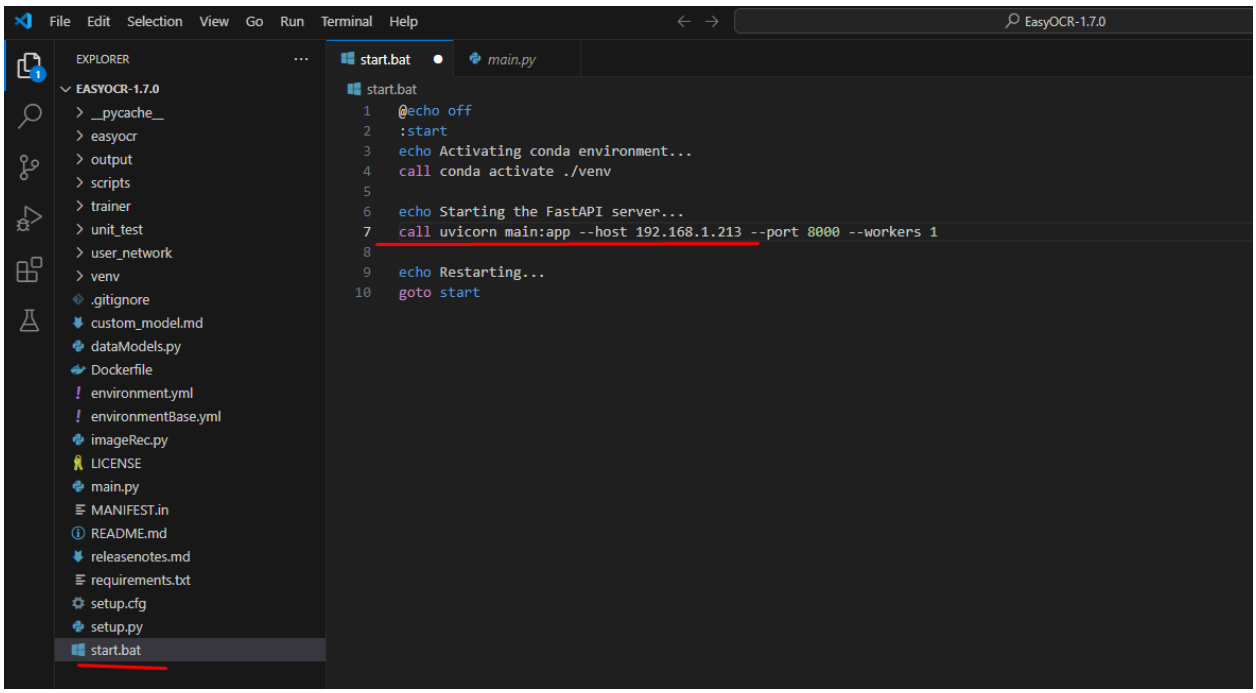
После установки питона нажмите на вкладку Terminal далее нажмите на New terminal.

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



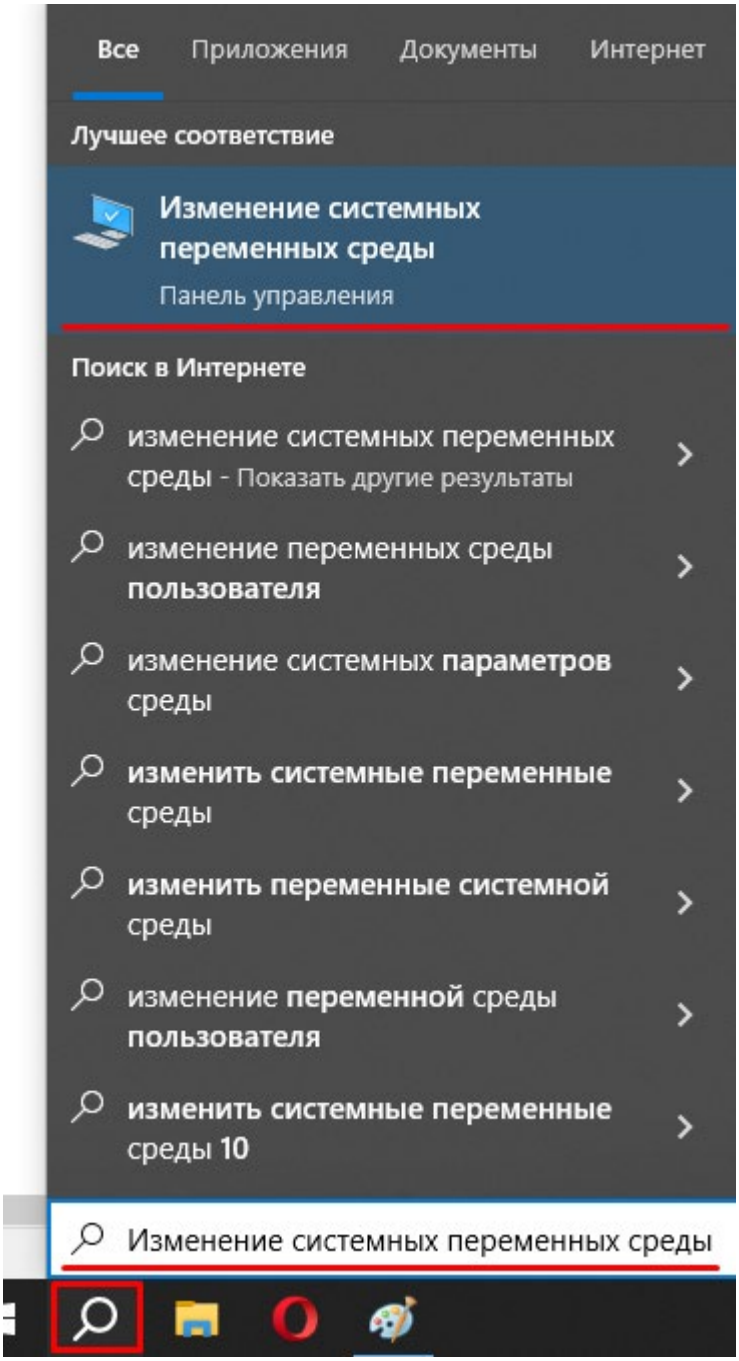
Установка необходимых пакетов

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



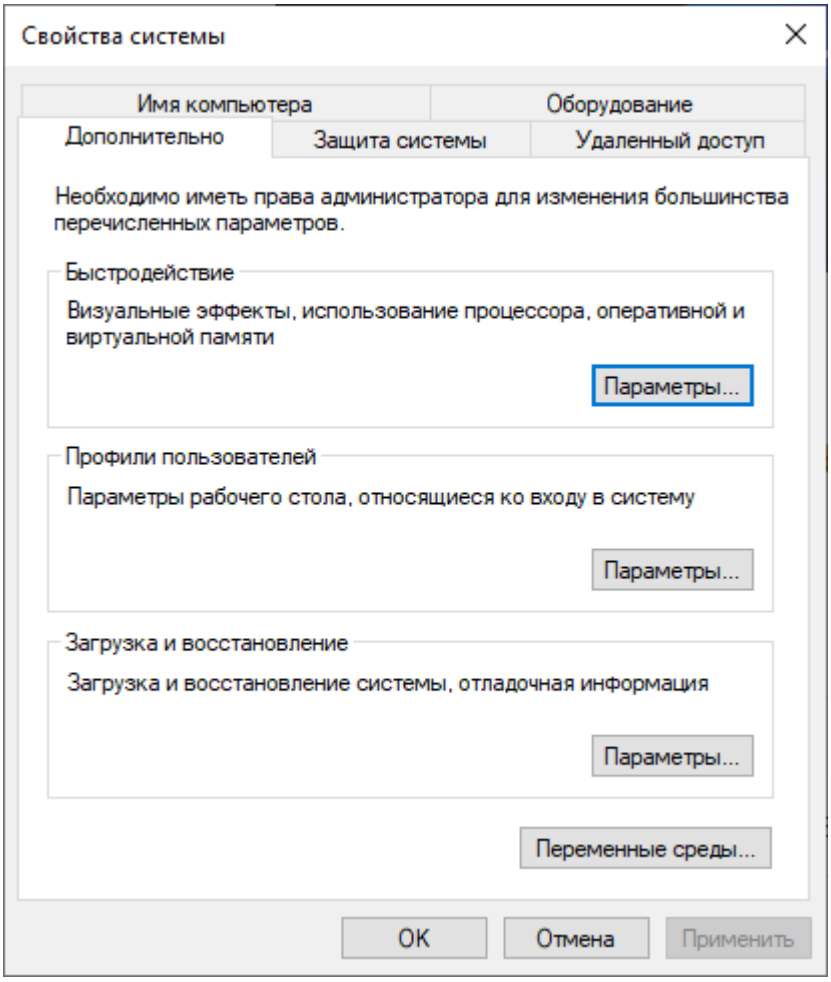
Смена IP адреса

Если у вас возникла ошибка 'Conda' не распознаётся как внутренняя команда, то откройте встроенное в windows приложение изменение системных переменных среды, для этого в поиске приложений windows введите изменение системных переменных среды и откройте его.

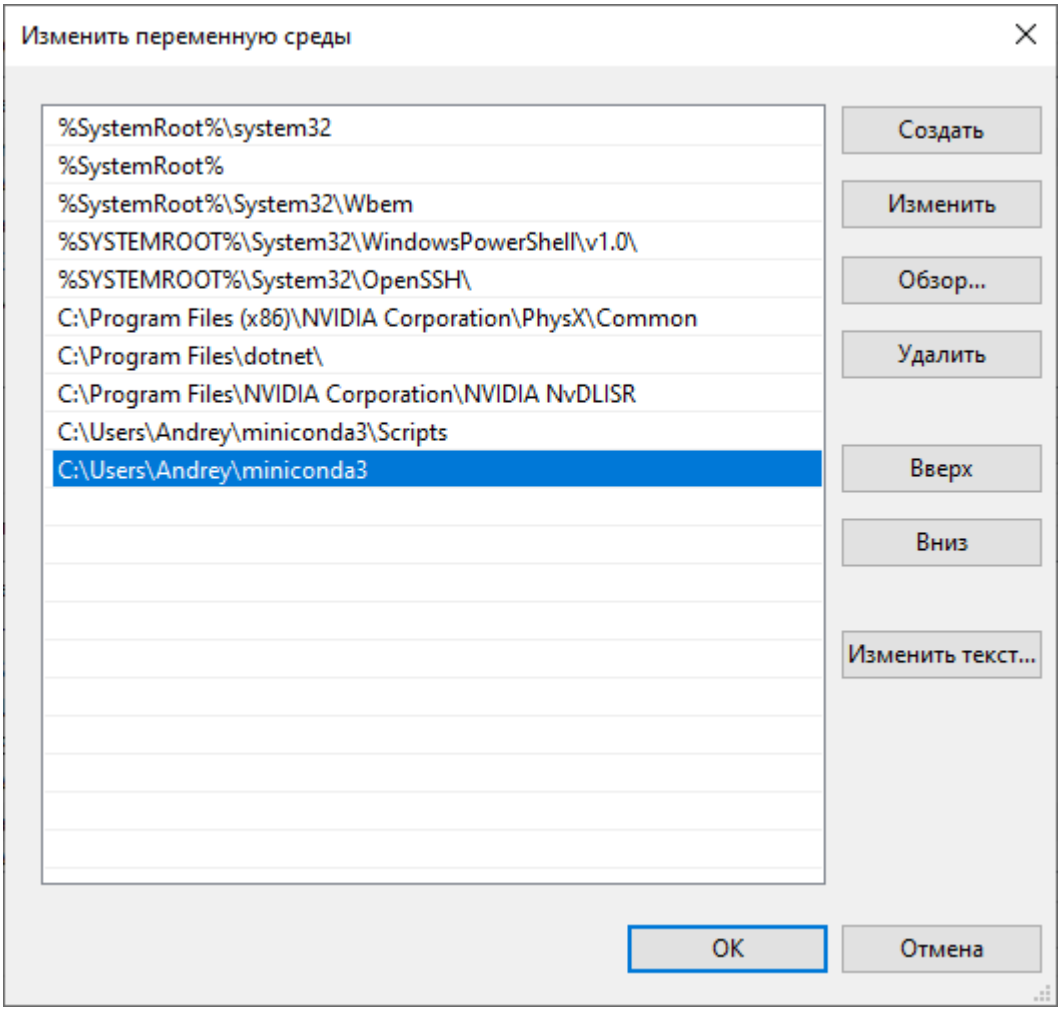


Поиск необходимого приложения

После запуска приложения изменение системных переменных среды нажмите на кнопку *Переменные среды*.

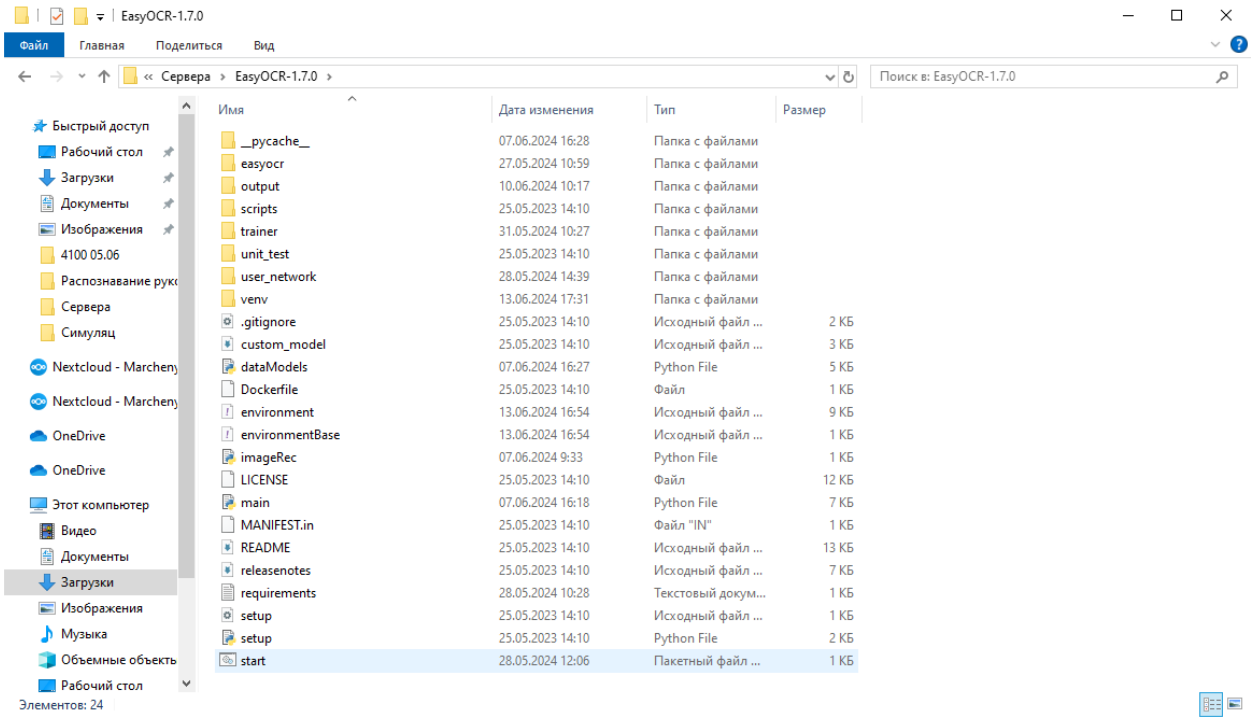


В открывшемся окне в нижнем окне найдите поле Path, нажмите на кнопку изменить и добавьте путь к Anaconda3\Scripts и к Anaconda3.



После чего повторите прошлые пункты с установкой.

После установки зайдите в папку, где расположен сервер и запустите файл *Start.*



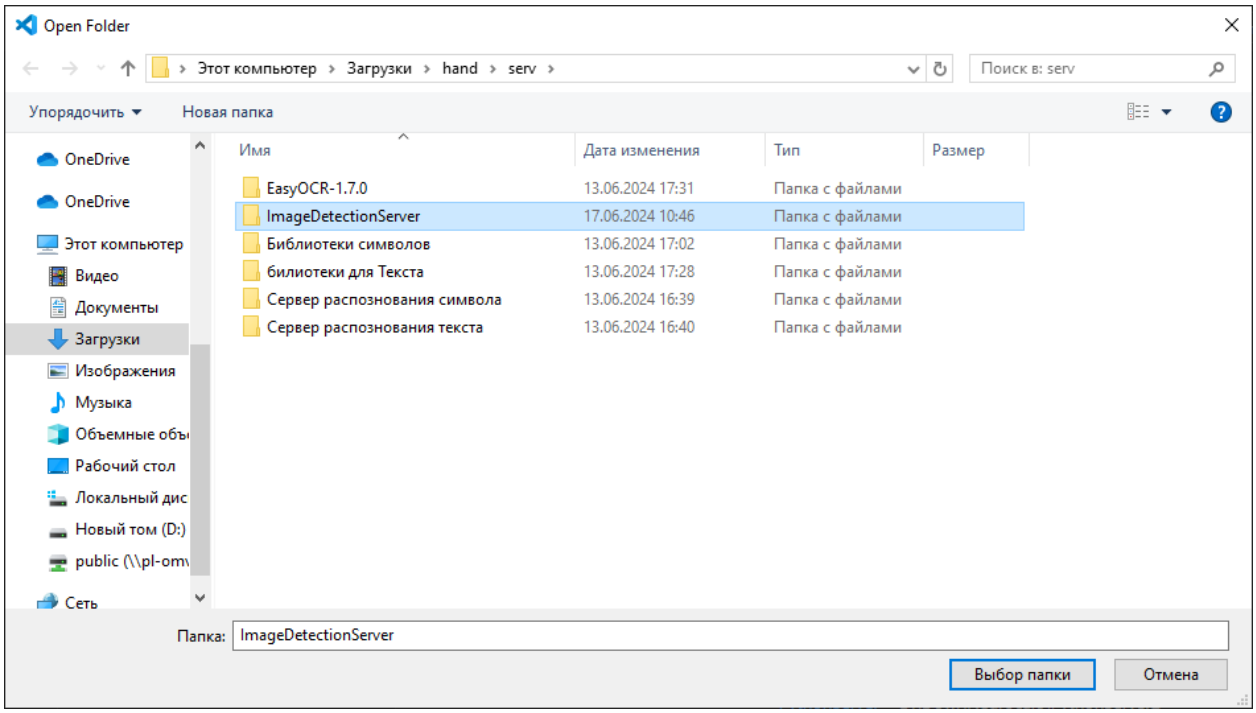
Если после запуска появляется ошибка о недостающем пакете, то его необходимо установить, для этого вернитесь в Visual studio и в терминале введите команду *conda init*, после чего введите команду *conda activate ./venv,* теперь вам необходимо ввести команду для установки необходимого пакета *conda install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

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

Запустите сервер и проверьте, что все работает.

Установка и настройка сервера распознавания символов

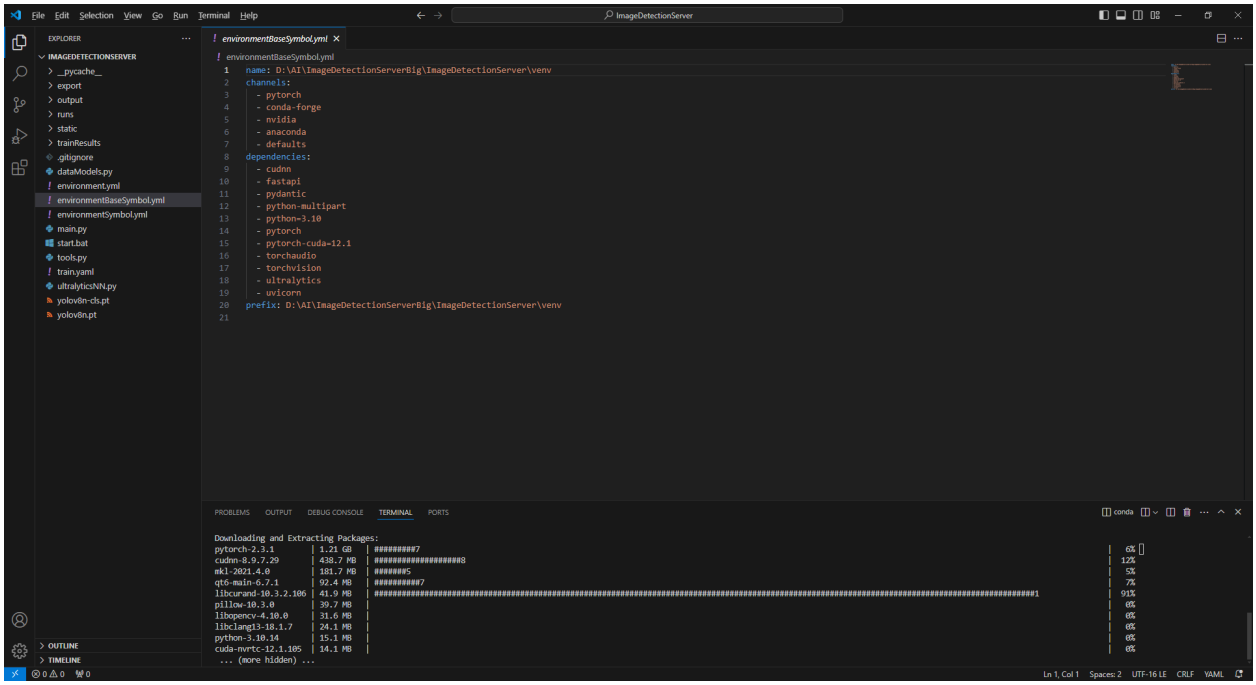
Запустите Visual studio code, нажмите на вкладку File ->Open folder и выберите путь к папке где располагается необходимый вам сервер (по умолчанию сервер находится в папке проекта: C:\Program Files (x86)\ProgramLab\HandwritingRecognition).



Открытие папки с сервером

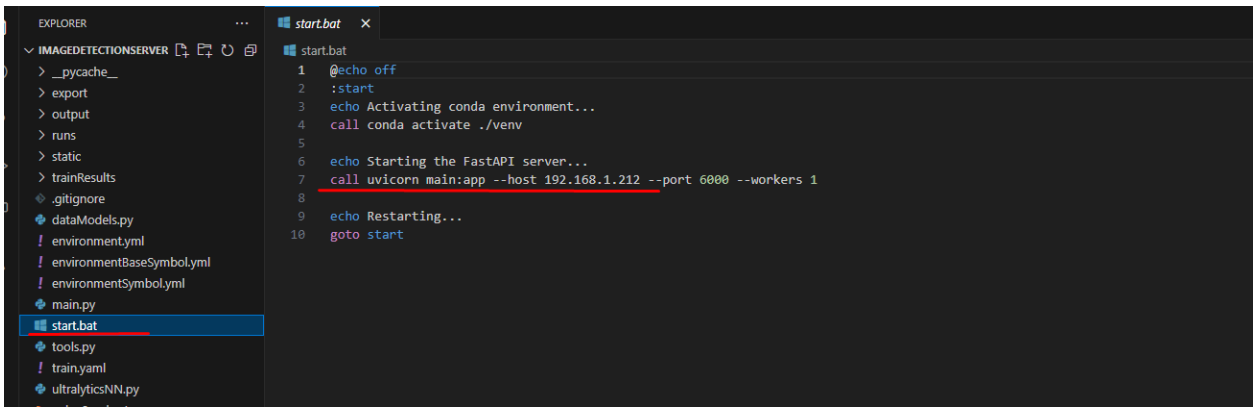
После установки питона нажмите на вкладку Terminal далее нажмите на New terminal.

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



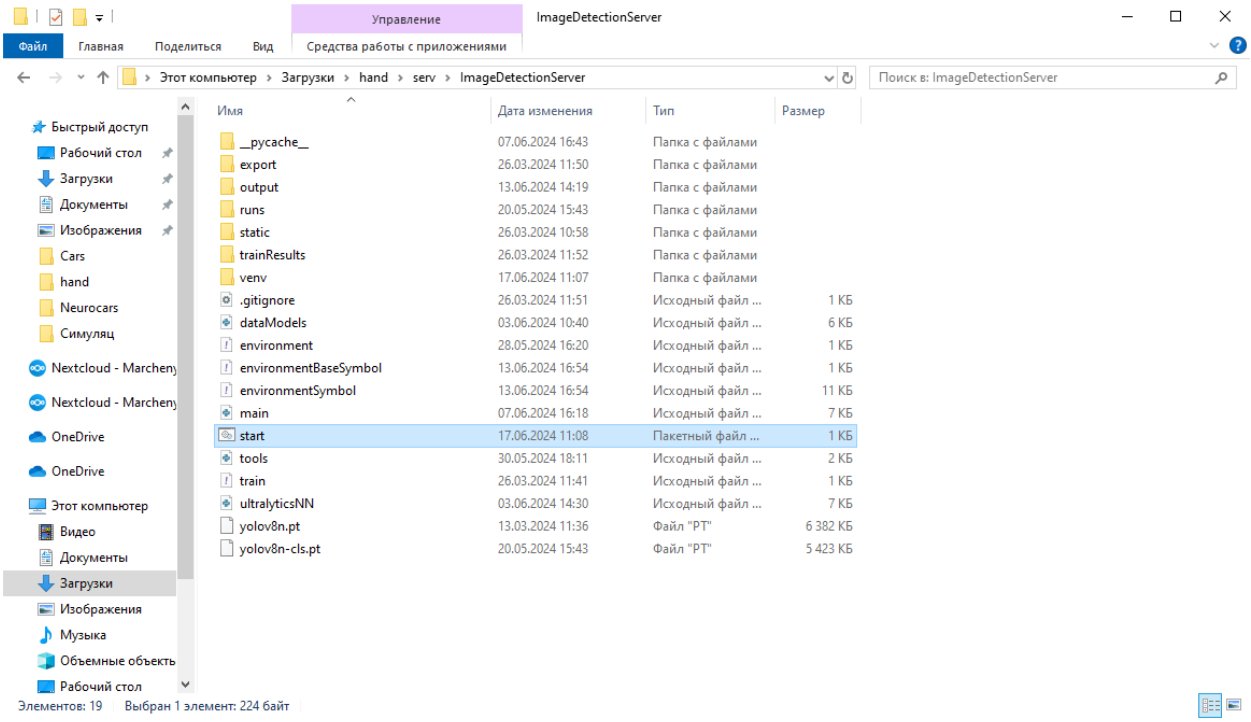
Установка необходимых библиотек

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



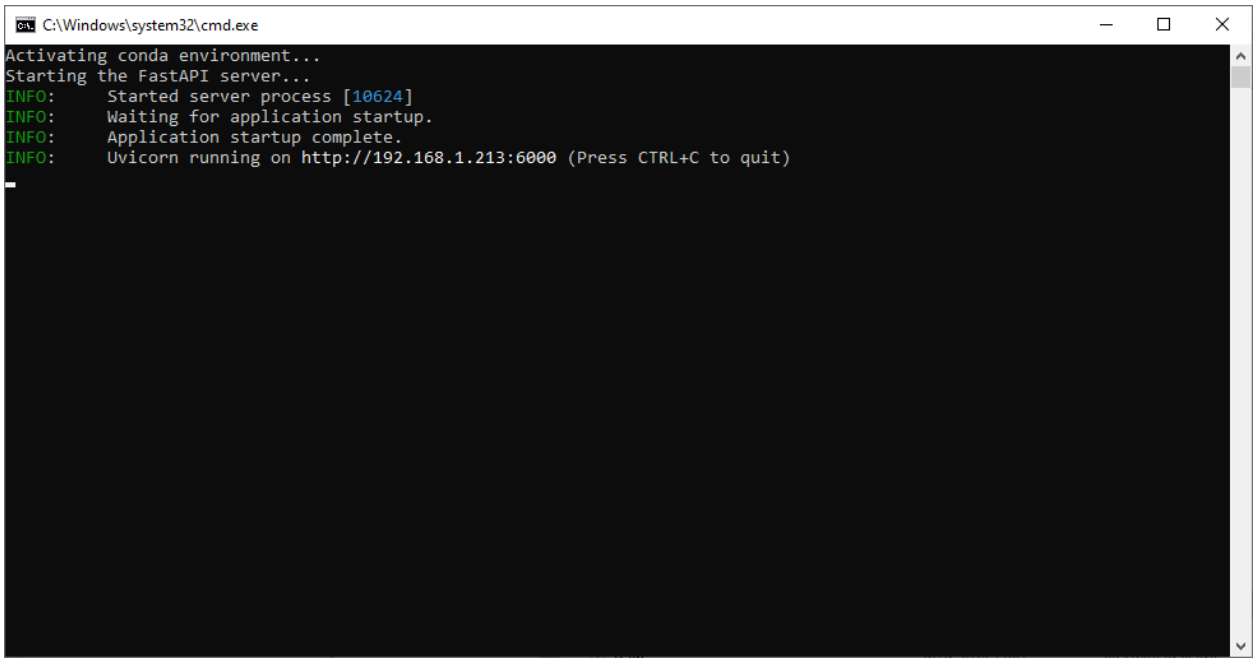
Смена IP адреса

Зайдите в папку сервера и запустите сервер запустив файл *start*.



Запуск сервера

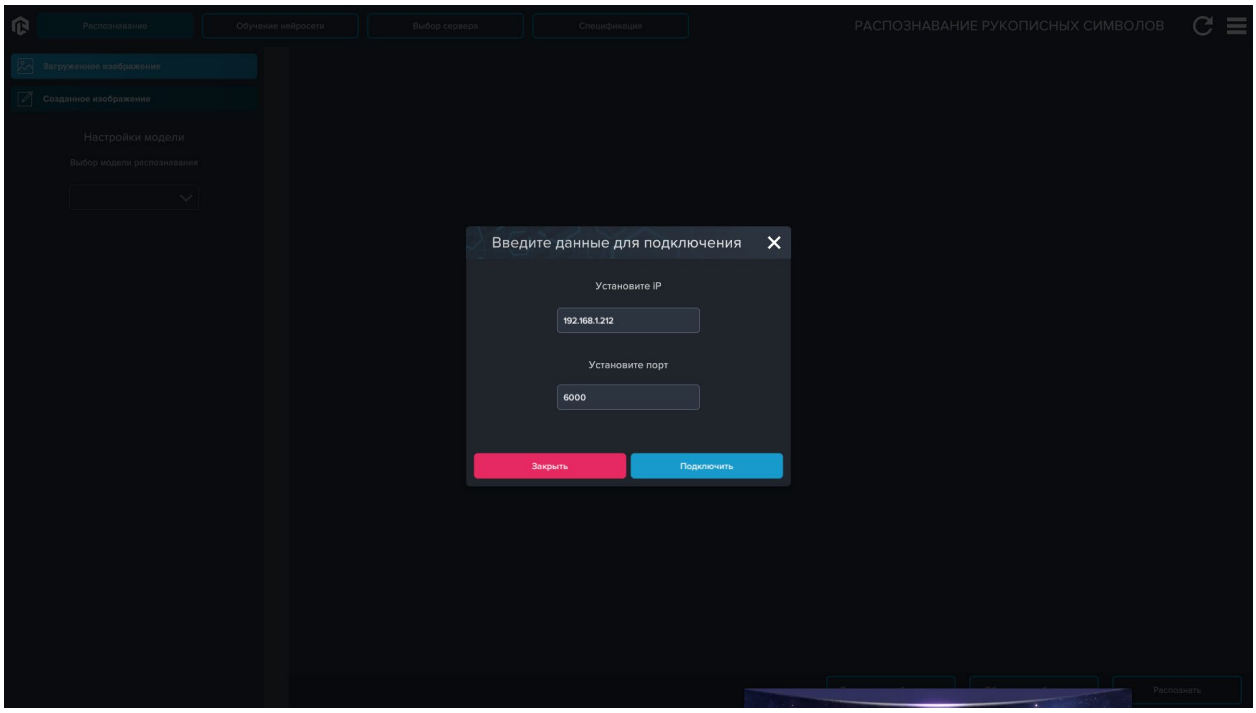
После запуска сервера откроется командная строка, в которой появятся информация о запуске сервера.



Успешный старт сервера

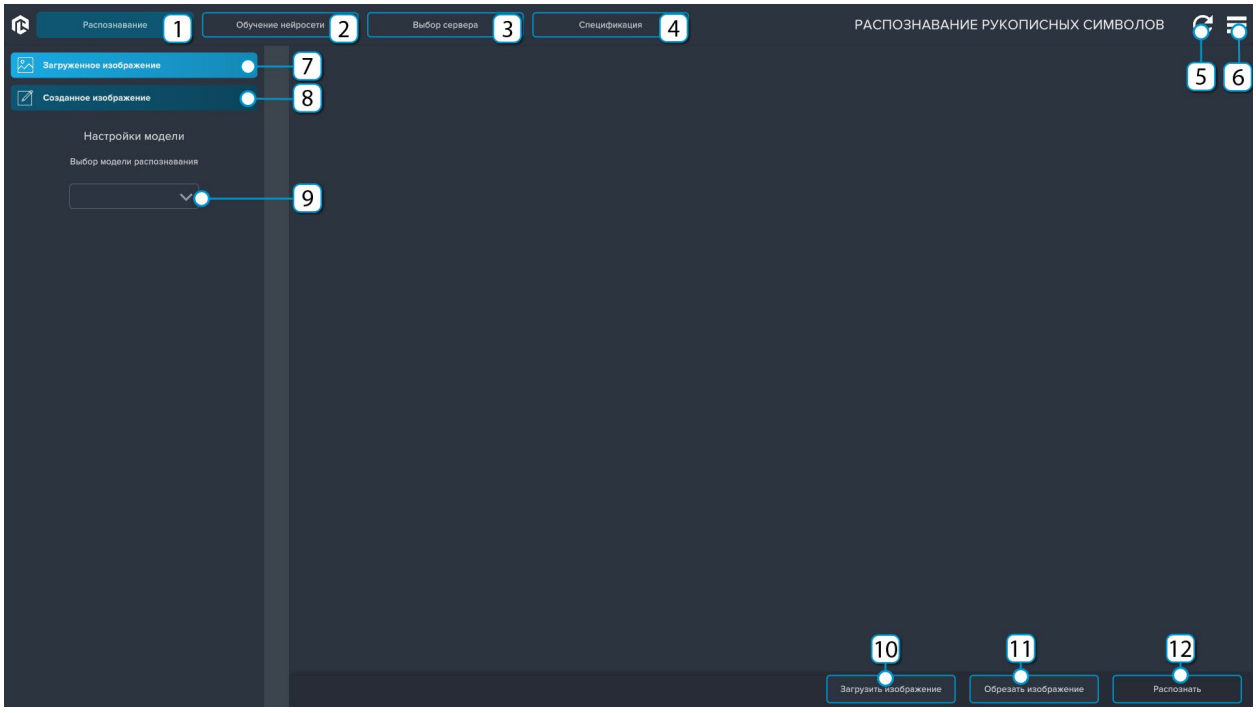
Работа в программе

После запуска программы откроется главный экран. При первом запуске откроется окно с вводом данных для подключения к серверу.



Окно подключения при первом запуске

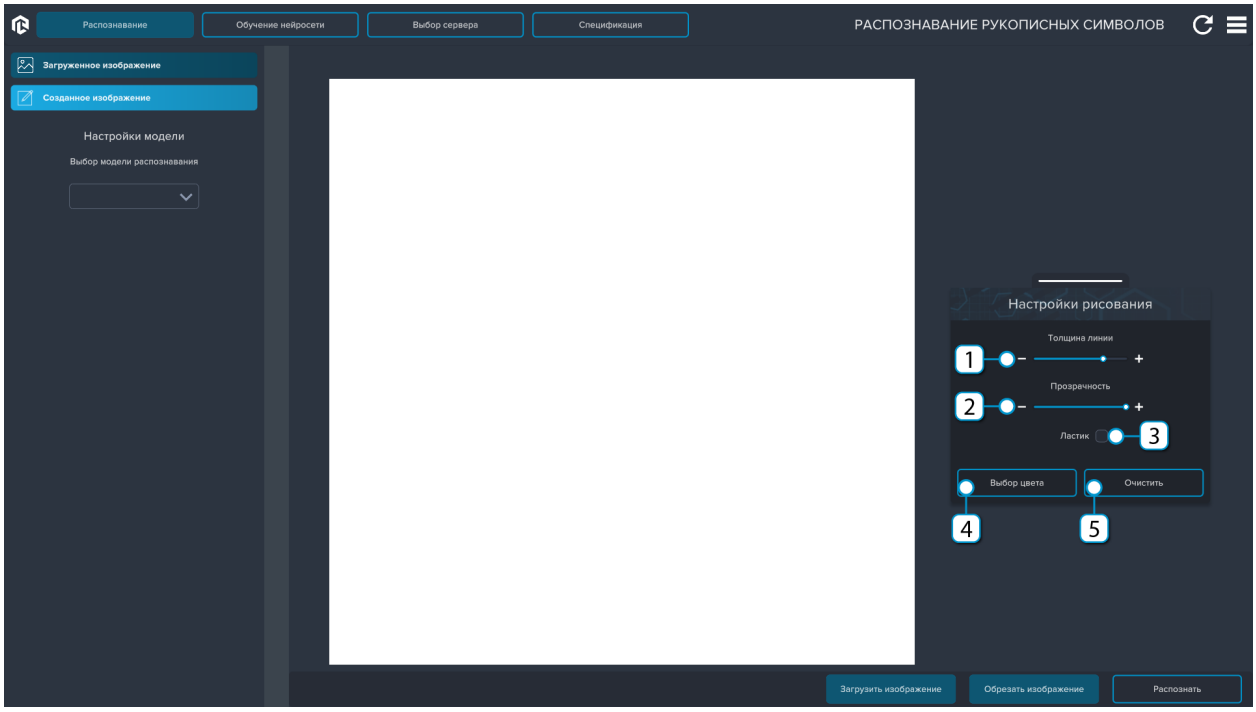
После подключения или закрытия окна подключения вы увидите главный экран приложения. На главном экране представлен интерфейс программы, для управления используйте мышь. Если программа не подключена к серверу с нейросетью, по этой причине многие функции неактивны.



Главный экран

1. Вкладка работы с распознаванием.
2. Вкладка обучения нейросети.
3. Кнопка выбора сервера и порта подключения.
4. Кнопка описании спецификации.
5. Кнопка обновления информации с сервера.
6. Кнопка вызова меню.
7. Работа с загружаемым изображением.
8. Работа с инструментами рисования.
9. Выбор языковой модели нейросети.
10. Кнопка загрузки изображения с компьютера.
11. Кнопка обрезания загруженного изображения.
12. Кнопка запуска распознавания.

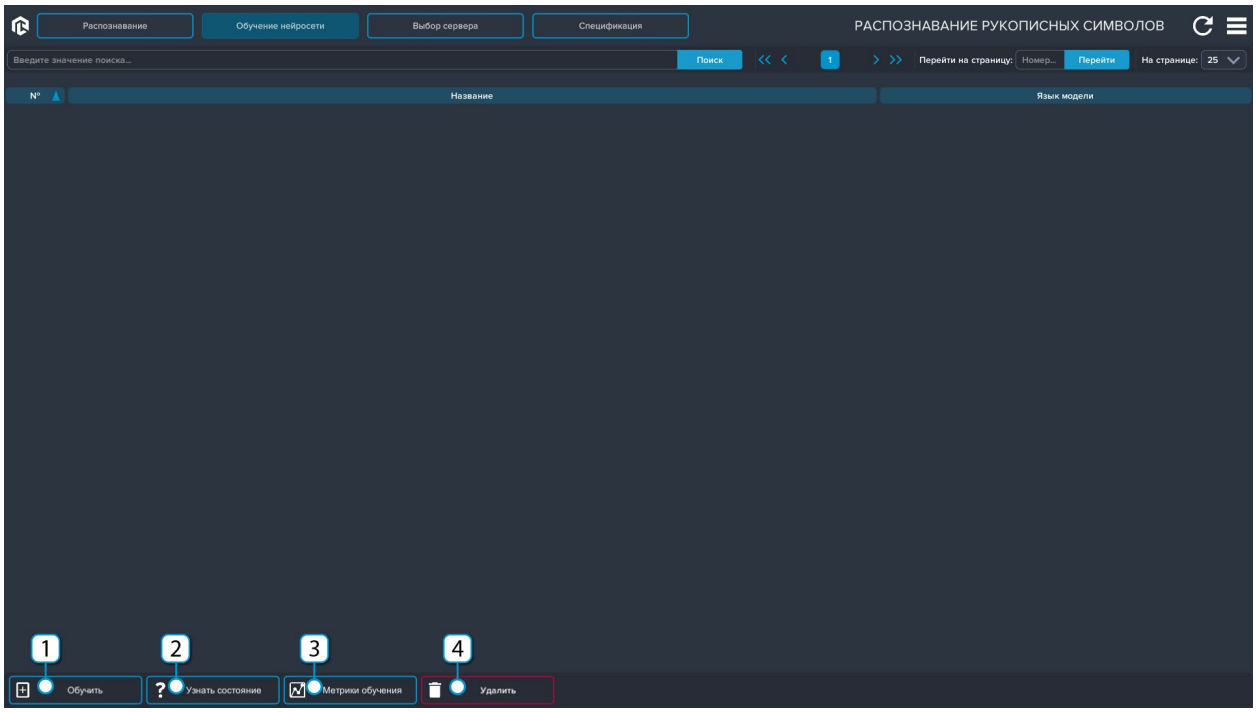
Во вкладке распознавание имеются 2 режима работы, распознавание по картинке и распознавание по рисунку, распознавание по картинке позволяет загружать картинку с компьютера, обрезать и распознавать текст, до подключения к серверу функционал работать не будет, вид и описание кнопок представлены на скриншоте выше. Распознавание по рисунку позволяет написать свой собственный текст или символ для распознавания, вид распознавания представлен на скриншоте ниже.



Рисование символов

1. Ползунок настроек толщины кисти.
2. Ползунок настроек прозрачности кисти.
3. Кнопка включения ластика.
4. Кнопка выбора цвета, позволяет настроить цвет по палитре цветов, либо с помощью введения кода цвета.
5. Кнопка очистки холста, при нажатии полностью очищает холст.

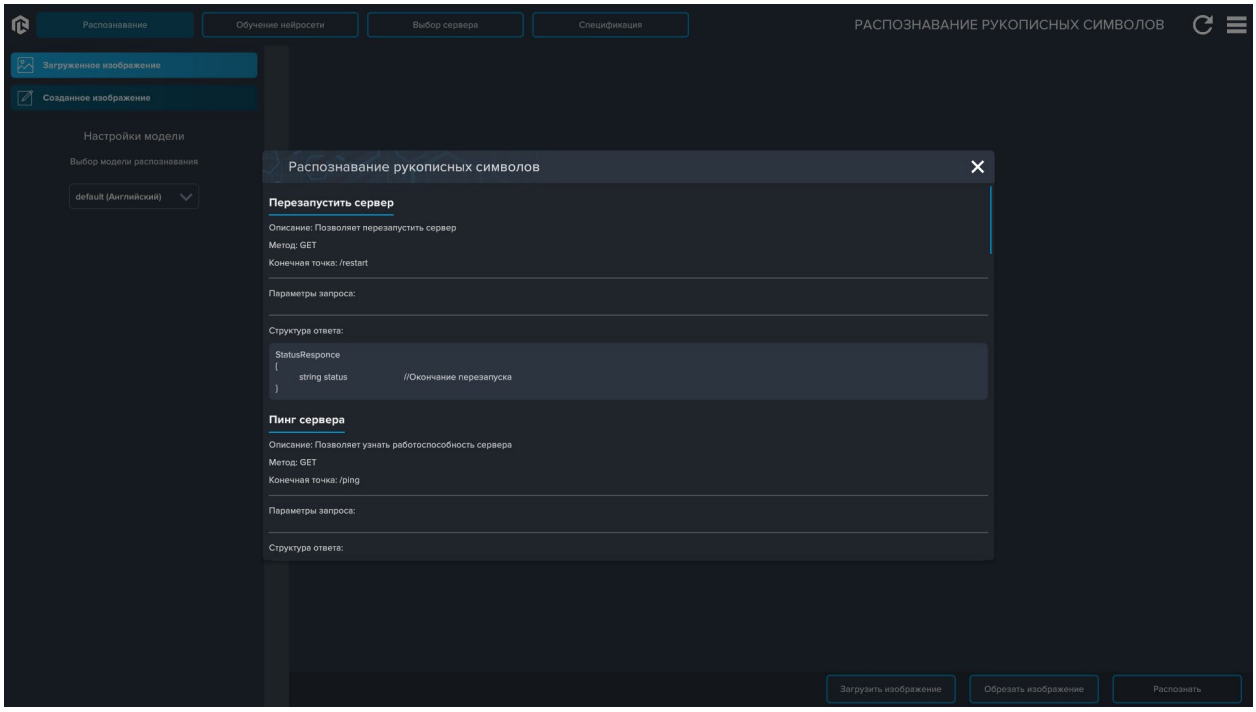
Во вкладке обучения нейросети находится список моделей, так же в данной вкладке находится функционал для обучения нейросети и просмотра этапов и состояния обучения, без подключения к серверу к нейросетью список будет пуст.



Обучение нейросети

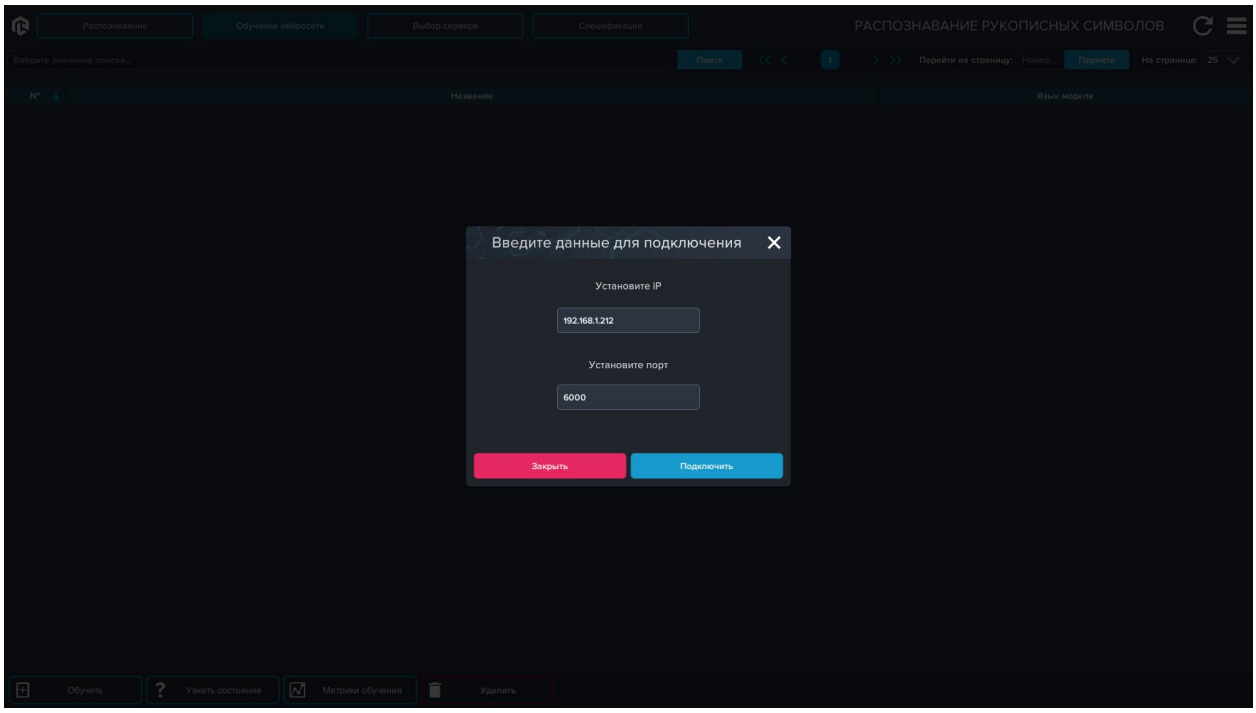
1. Кнопка обучить, при нажатии открывает окно с настройками обучения.
2. Кнопка узнать состояние, при нажатии показывает находится ли нейросеть в состоянии обучения или нет.
3. Кнопка метрики обучения, открывает окно с графиками данных о состоянии обучения.
4. Кнопка удалить, при нажатии удаляет выбранную модель.

Вкладка спецификация содержит информацию о командах запросов отправляемых от приложения к серверу



Спецификация

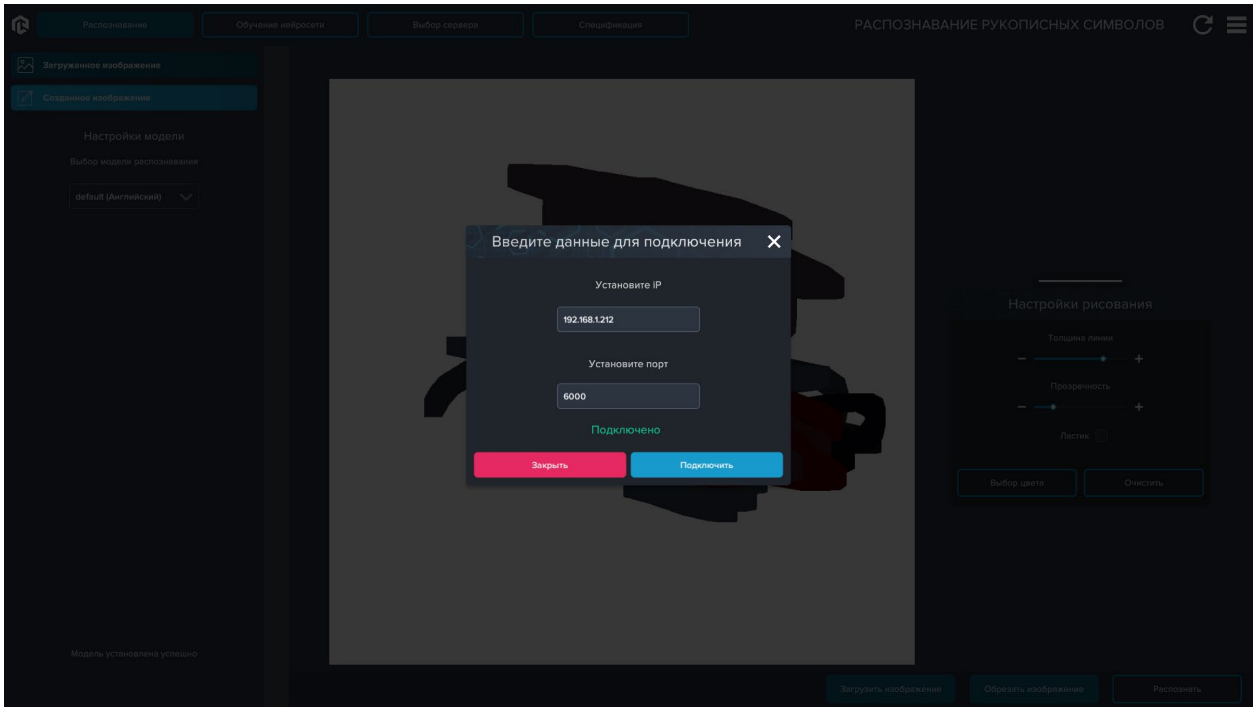
Для работы с нейросетью необходимо подключиться к серверу, на котором располагается нейросеть, для подключения необходимо нажать на кнопку **Выбор сервера**, после нажатия откроется всплывающее окно с настройками подключения к серверу. О установке, запуске и настройках сервера смотрите в соответствующем разделе.



Окно подключения к серверу

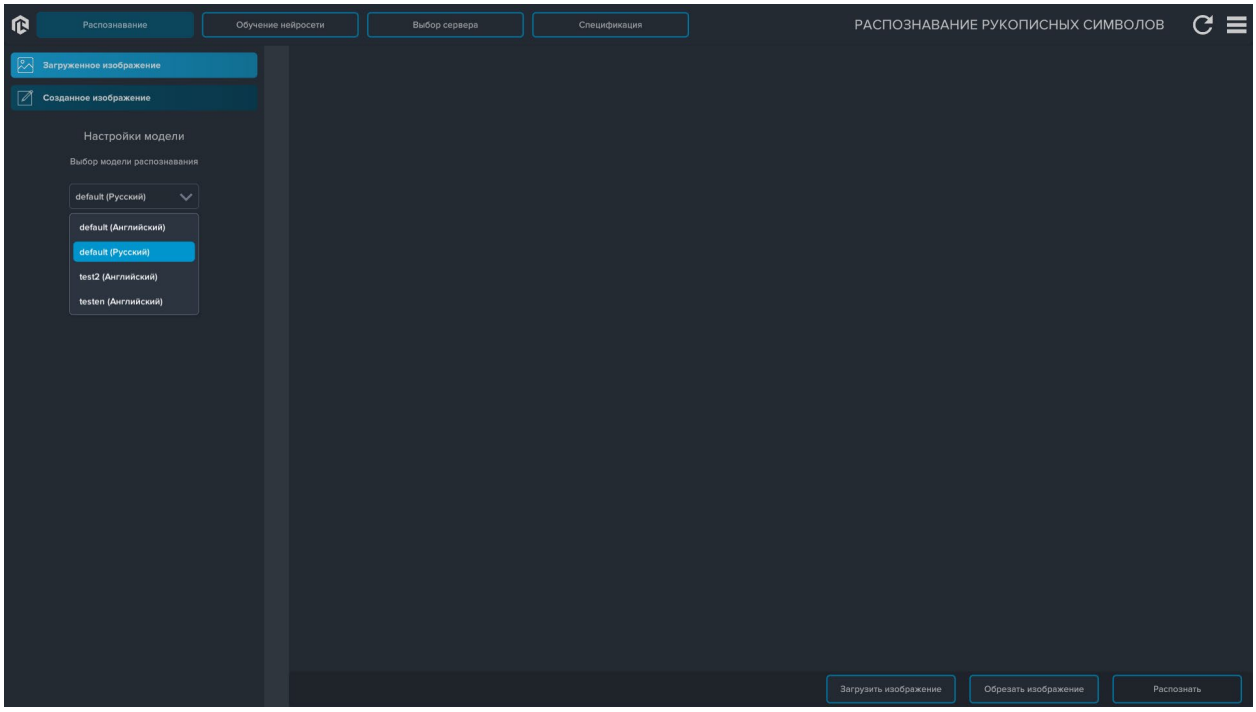
В поле IP введите IP настроенного сервера с нейросетью, перед подключением убедитесь, что сервер включен. В поле порт введите порт подключения к серверу.

При успешном подключении к серверу с нейросетью в окне подключения появится надпись, сигнализирующая о успешном подключении.



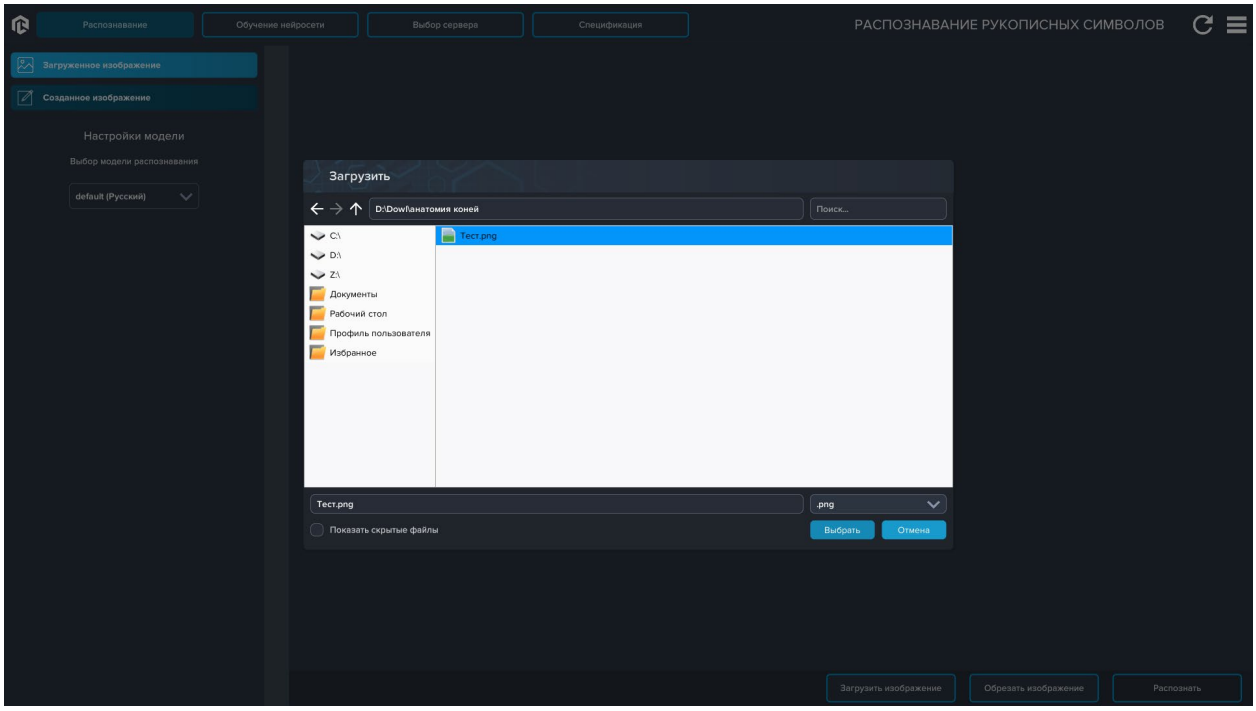
Успешное подключение

При подключении к серверу с нейросетью во вкладке распознавание разблокируется функционал работы с нейросетью, в режиме распознавания символов с изображения в списке выборов моделей для распознавания появляется список обученных языковых моделей, кнопка загрузить изображение, обрезать изображение и кнопка распознать становятся активными.



Список моделей

Для распознавания по картинке, выберите в списке нужную языковую модель, после чего нажмите на кнопку загрузить изображение, откроется окно загрузки, в данном окне необходимо выбрать необходимое вам изображение и нажать на кнопку **Выбрать**.

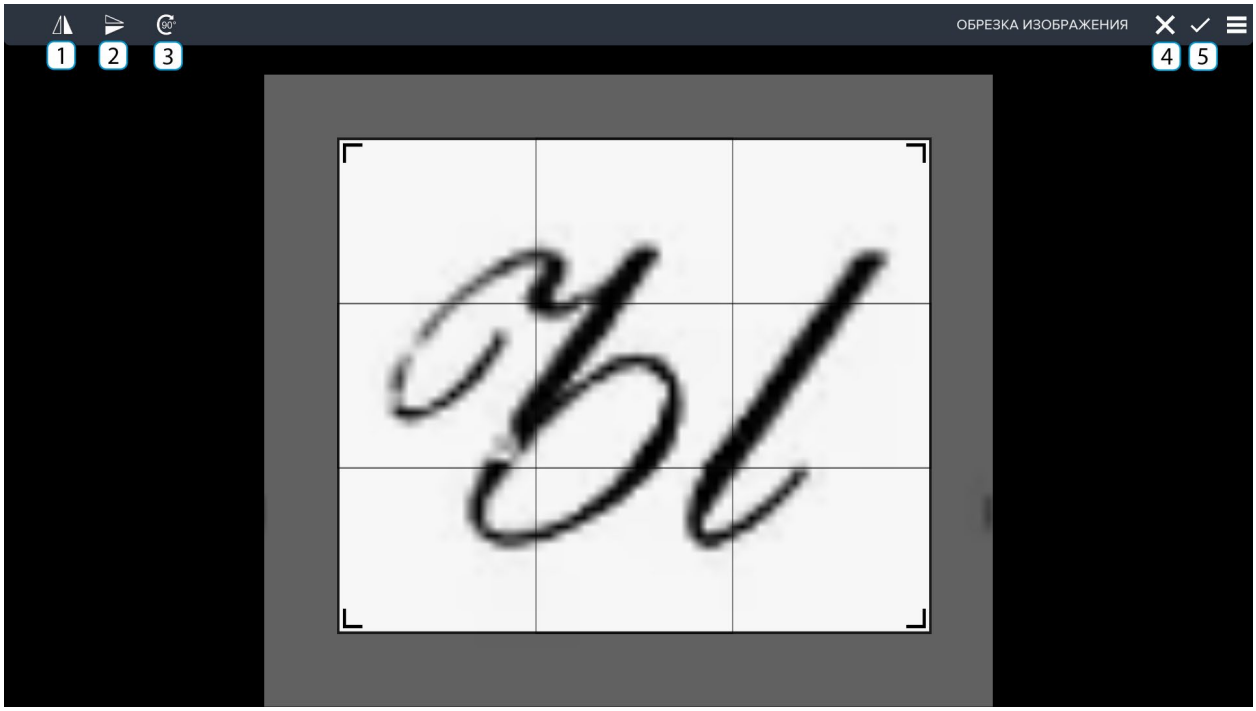


Загрузка изображения

При успешной загрузке изображения оно отобразится в программе.

Заруженное изображение

При необходимости загруженное изображение можно обрезать, для более точного распознавания.



Обрезка изображения

В открывшемся окне обрезки изображения присутствуют следующие кнопки

1. Кнопка поворота изображения по горизонтальной оси.
2. Кнопка поворота изображения по вертикальной оси.
3. Кнопка поворота изображения на 90 градусов.
4. Кнопка отмены, при нажатии закрывает обрезку изображения.

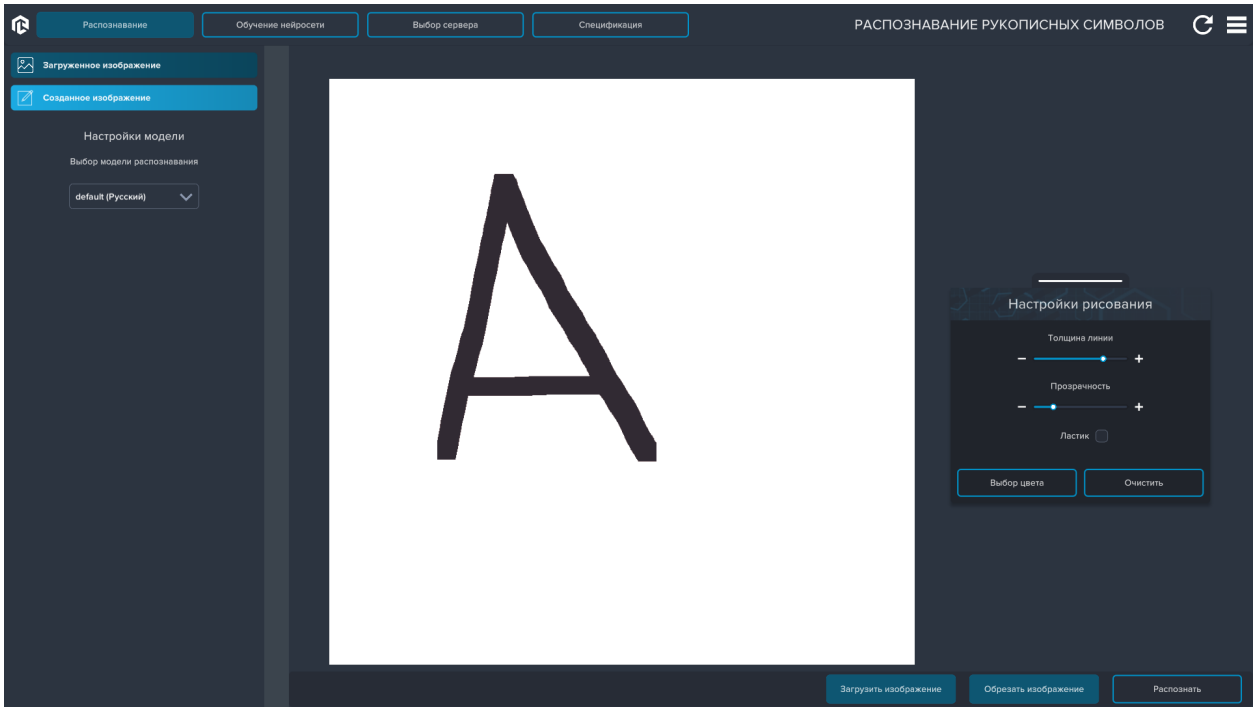
5. Кнопка подтверждения, при нажатии сохраняет обрезку.

После завершения обрезки изображения нажмите на кнопку подтверждения для обрезки и выхода из режима обрезки изображения

После того как изображение загружено нажмите на кнопку **Распознать**, для того чтобы нейросеть распознала текст или символ с загруженной картинки.

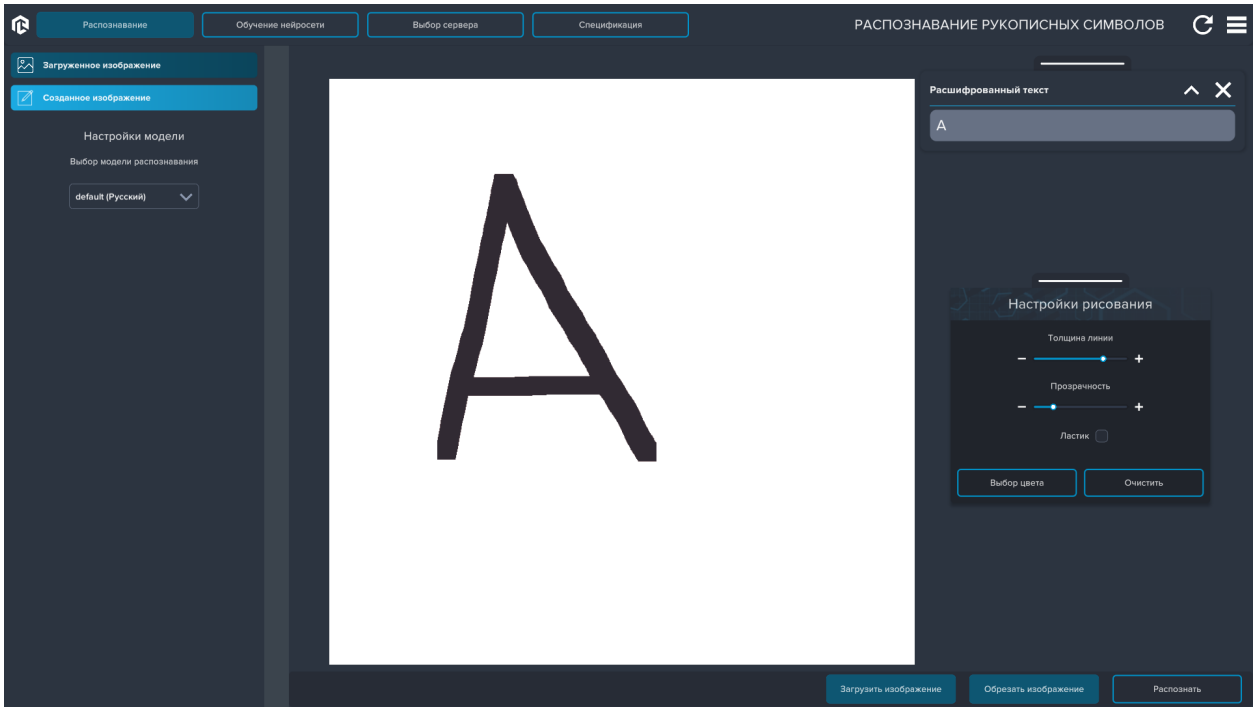
Поле вывода распознанного текста

В режиме распознавания по нарисованному изображению сначала выберите языковую модель, после чего положите руку на компьютерную мышь, наведитесь в зону рисования и зажав *ЛКМ* напишите ваш текст или символ.



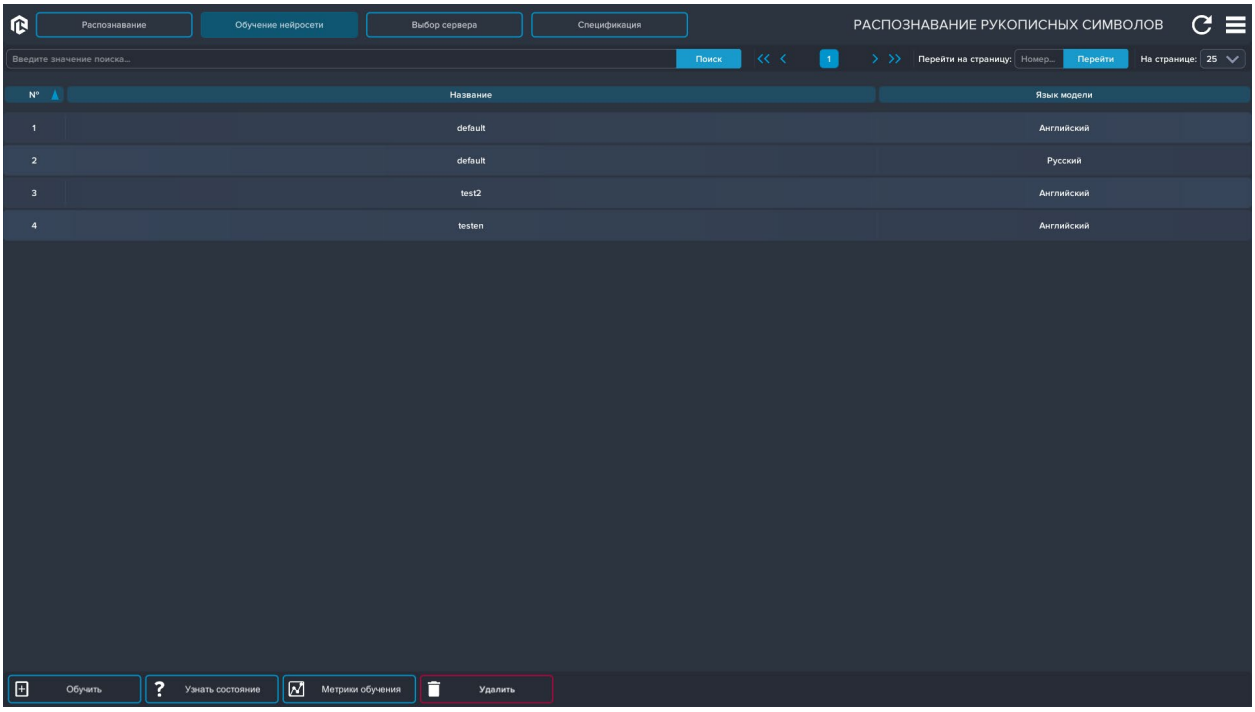
Распознавание по рисунку

Для распознавания текста или символа, нажмите на кнопку **Распознать**, появится окошко с распознанным текстом или символом.



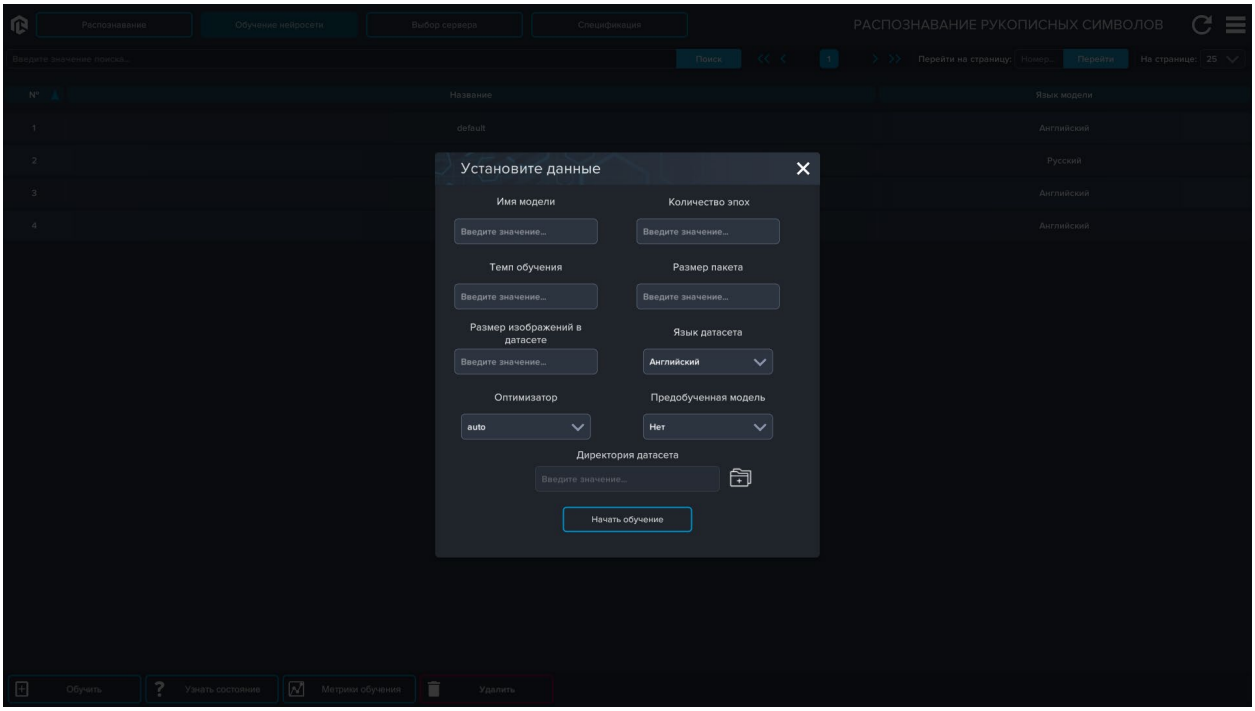
Распознаный текст

После успешного подключения к серверу с нейросетью во вкладке обучение нейросети появятся списки с обученными языковыми моделями, а также станет активным возможность обучения нейросети на своей модели.



Список обученных языковых моделей

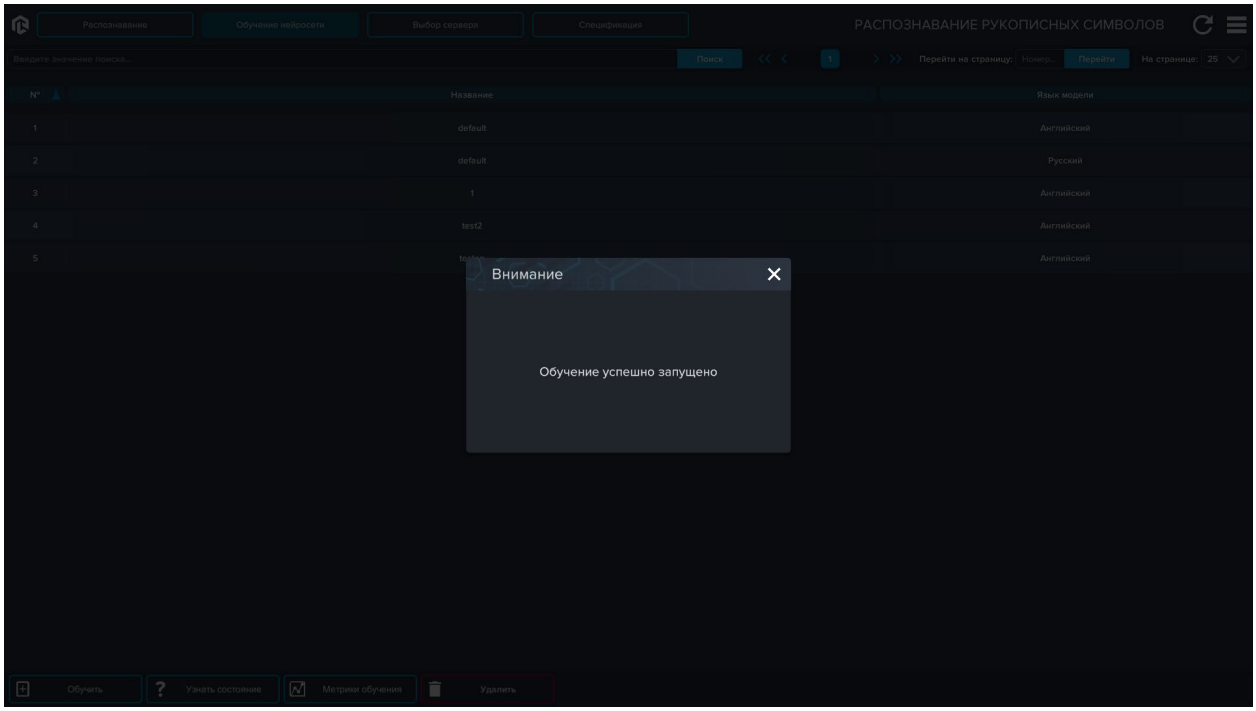
Для обучения на своей языковой модели, нажмите на кнопку **Обучить**, после чего откроется окно настроек обучения.



Окно настроек обучения

В данном окне можно задать имя для своей языковой модели, настроить количество эпох для обучения нейросети, настроить темп обучения нейросети, настроить пакет для определения количества выборок в эпоху обучения, задать размер изображения в датасете, выбрать язык датасета, выбрать оптимизатор (алгоритм оптимизации метода градиентного спуска), выбрать обучение на основе предобученой модели (Предварительно

обученные модели часто используются в качестве отправной точки для разработки моделей машинного обучения, поскольку они предоставляют набор начальных весов и отклонений, которые можно точно настроить для конкретной задачи) ,для начала обучения укажите местонахождения датасета на локальном компьютере, после чего нажмите на кнопку начать обучение.



Запуск обучения

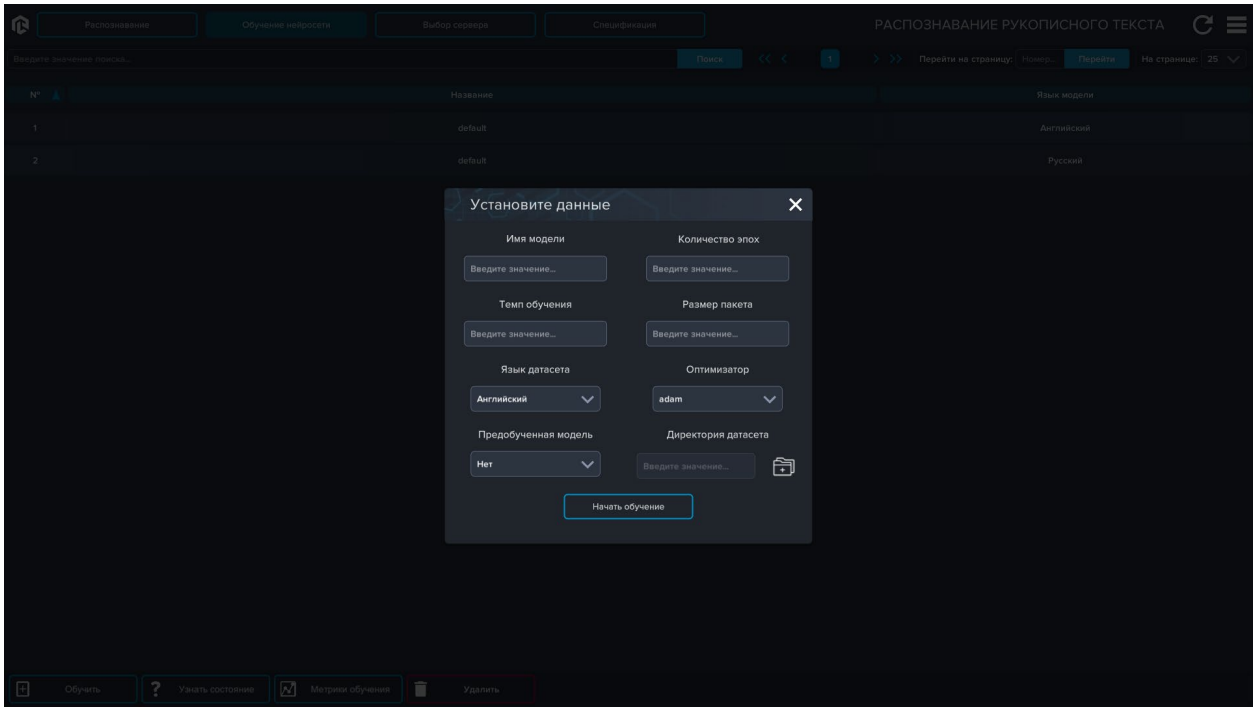
Во время обучения для изучения процесса обучения нейросети, можно нажать на кнопку **Метрики обучения**, после чего откроется окно с метриками, так же можно посмотреть метрики для уже обученных моделей.

Окно с метриками

Для более точного изучения данных метрик, можно указать начальную эпоху, после чего готовый графики метрик перестроятся, и выбранная начальная эпоха станет начальной точкой отсчета

Задание начальной эпохи

Функциональные отличи между распознаванием рукописных символов и рукописного текста заключается в настройках обучения языковой модели, метриках, а также в используемых нейросетях для определения символов и текста. В окне настроек обучения для распознавания рукописного текста находятся настройки имени языковой модели, настройки количества эпох для обучения нейросети, настройка темпа обучения нейросети, настройка пакета для определения количества выборок в эпоху обучения, выбор языка датасета, выбор оптимизатора (алгоритм оптимизации метода градиентного спуска), выбор обучения на основе предобученой модели (Предварительно обученные модели часто используются в качестве отправной точки для разработки моделей машинного обучения, поскольку они предоставляют набор начальных весов и отклонений, которые можно точно настроить для конкретной задачи).



Окно настроек обучения

В окне метрики обучения можно ознакомиться с метриками обучения нейросети на основе выбранной модели, на графиках можно ознакомиться с количеством неудачных и удачных этапов обучения, точности обучения на выбранной модели, информации о нормализованном расстоянии Левенштейна, однако на предустановленных в **ПО** моделях данные метрики отсутствуют.

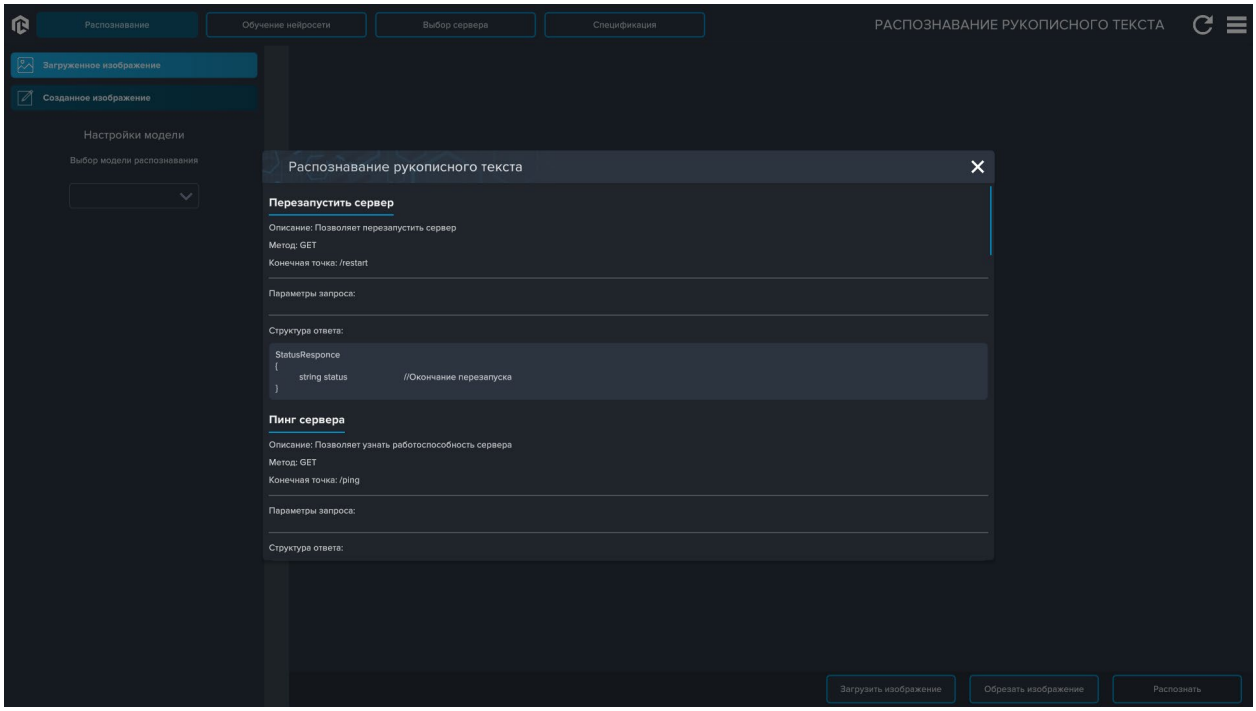
Метрики для предустановленных моделей

Спецификация рукописного текста

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

Рассмотрим спецификацию для распознавания рукописного текста.

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



Спецификация

Перезапустить сервер.

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

Пример JSON ответа:

```
{
string status                    //Окончание перезапуска
}
```

Пинг сервера.

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

Пример JSON ответа:

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

Получить список моделей.

Команда ***Получить список моделей***, приложение посылает запрос к конечной точке ***/get-models***, адресованный к серверу, сервер отсылает информацию о списке обученных моделей, хранящихся на сервере в виде json ответа.

Пример JSON ответа:

```
{
 "models": [
   {
    "modelName": "string",      //Имя модели
    "modelLang": "string"       //Язык модели
   }
 ]
}
```

Получить состояние обучения

Команда ***Получить состояние обучения***, приложение посылает запрос к конечной точке ***/get-train-state***, адресованный к серверу, сервер отсылает информацию о состоянии текущего обучения, в виде Json ответа.

Пример JSON ответа:

```
{
 "state": "string"       //Состояние обучения
}
```

Получить состояние обучения

Команда ***Начать обучение***, приложение посылает запрос к конечной точке ***/train*** с заданными настройками, адресованный к серверу в виде json запроса.

Пример JSON запроса:

```
{
 "datasetName": "string",    //Название тренировки
 "datasetPath": "string",    //Путь к датасету
 "lang": "string",    //Язык обучения
```

```
  "optimizator": "string",
  "epochs": 0,     /Количество эпох обучения
  "learningRate": 0,
  "batchSize": 0,
  "pretrainedModel": {
    "modelName": "string",
    "modelLang": "string"
  }
}
```

Сервер в ответ на запрос отсылает информацию о запуске обучения и о состоянии обучения в виде json ответа.

Пример JSON ответа:

```
{
      "status": "string"    //Состояние запуска обучения
}
```

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

Пример сообщения JSON об ошибке:

```
{
 "detail": [
   {
    "loc": [
     "string",
     0
    ],
    "msg": "string",
    "type": "string"
   }
 ]
}
```

Установить модель

Команда ***Установить модель***, приложение посылает запрос к конечной точке ***/set-model***, адресованный к серверу для выбора модели распознавания в виде json запроса.

Пример JSON запроса:

```
{
 "modelName": "string",    //Имя модели
 "modelLang": "string"   //Язык модели
}
```

В ответ на запрос сервер отсылает информацию о выбранной модели в виде json ответа.

Пример JSON ответа:

```
{
 "status": "string"   //Состояние установки модели
}
```

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

Пример сообщения JSON об ошибке:

```
{
 "detail": [
  {
   "loc": [
    "string",
    0
   ],
   "msg": "string",
   "type": "string"
  }
 ]
}
```

Обнаружение текста на изображении

Команда ***Обнаружение текста на изображении***, приложение посылает запрос к конечной точке ***/predict-to-image***, данный запрос позволяет отослать на сервер изображение в виде массива байтов.

Сервер принимает изображение в виде массива байтов и обрабатывает его, в ответ сервер отсылает распознанный текст в виде json ответа.

Пример JSON ответа:

```
{
 "Line": [
  {
   "lineText": "string",    //Текст распознанной строки
  }
```

```
  ]
}
```

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

Пример сообщения JSON об ошибке:

```
{
 "detail": [
  {
   "loc": [
    "string",
    0
   ],
   "msg": "string",
   "type": "string"
  }
 ]
}
```

Получить результаты обучения

Команда ***Получить результаты обучения***, приложение посылает запрос к конечной точке ***/training-metrics/{training_name},*** адресованный к серверу.

Сервер в ответ на запрос отсылает информацию о обученной модели, при этом сервер отдает словарь метрик обучения, метрики обучения и значения метрики в виде Json ответа.

Пример JSON ответа:

```
{
 "metrics": {     //Словарь метрик обучения
  "additionalProp1": [      //Метрика 1
   0                        //Значение метрики 1
  ],
  "additionalProp2": [
   0
  ],
  "additionalProp3": [
   0
  ]
 }
}
```

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

Пример сообщения JSON об ошибке:

```
{
 "detail": [
  {
   "loc": [
    "string",
    0
   ],
   "msg": "string",
   "type": "string"
  }
 ]
}
```

Удалить модель

Команда ***Удалить модель***, приложение посылает запрос к конечной точке ***/output/{model_name}***, адресованный серверу.

В ответ на запрос сервер удаляет выбранную пользователем модель в виде json ответа.

Пример JSON ответа:

```
{
 "status": "string"    //Состояние удаления
}
```

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

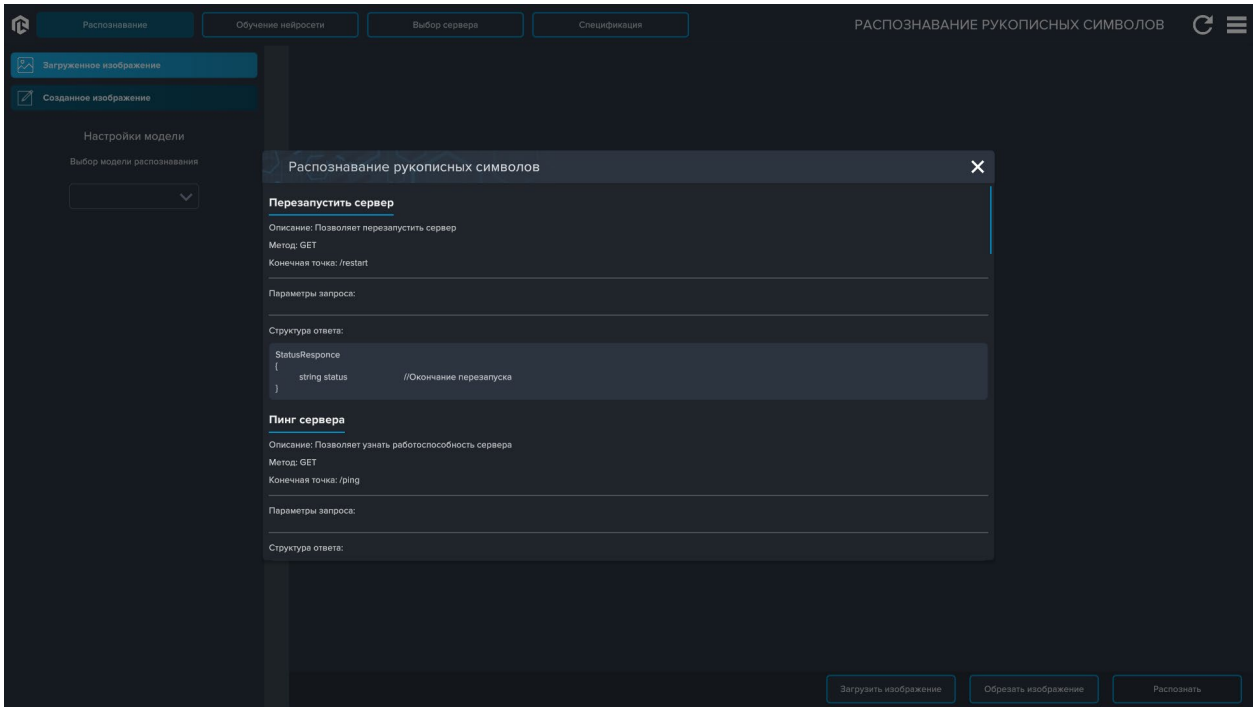
Пример сообщения JSON об ошибке:

```
{
 "detail": [
  {
   "loc": [
    "string",
    0
   ],
   "msg": "string",
   "type": "string"
  }
 ]
}
```

Спецификация рукописных символов

Рассмотрим спецификацию для распознавания рукописных символов.

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



Спецификация

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

Пример JSON ответа:

```
{
string status                    //Окончание перезапуска
}
```

Пинг сервера

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

Пример JSON ответа:

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

Получить список моделей

Команда ***Получить список моделей***, приложение посылает запрос к конечной точке ***/get-models***, адресованный к серверу, сервер отсылает информацию о списке обученных моделей, хранящихся на сервере в виде json ответа.

Пример JSON ответа:

```
{
 "models": [
  {
   "modelName": "string",   //Имя модели
   "modelLang": "string"    //Язык модели
  }
 ]
}
```

Получить состояние обучения,

Команда ***Получить состояние обучения***, приложение посылает запрос к конечной точке ***/get-train-state***, адресованный к серверу, сервер отсылает информацию о состоянии текущего обучения, в виде Json ответа.

Пример JSON ответа:

```
{
 "state": 0  //Состояние обучения
}
```

Начать обучение

Команда ***Начать обучение***, приложение посылает запрос к конечной точке ***/train*** с заданными настройками, адресованный к серверу в виде json запроса.

Пример JSON запроса:

```
{
 "datasetName": "string",      //Название тренировки
 "datasetPath": "string",      //Путь к датасету
 "lang": "string",       //Язык обучения
  "imgsz":  "int",     //Размер изображений в датасете (ширина и длина одинаковы)
 "epochs": "int"       /Количество эпох обучения
}
```

Сервер в ответ на запрос отсылает информацию о запуске обучения и о состоянии обучения в виде json ответа.

Пример JSON ответа:

```
{
 "status": "string"  //Состояние запуска обучения
}
```

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

Пример сообщения JSON об ошибке:

```
{
 "detail": [
  {
   "loc": [
    "string",
    0
   ],
   "msg": "string",
   "type": "string"
  }
 ]
}
```

Установить модель

Команда ***Установить модель***, приложение посылает запрос к конечной точке ***/set-model***, адресованный к серверу для выбора модели распознавания в виде json запроса.

Пример JSON запроса:

```
{
 "modelName": "string",   //Имя модели
 "modelLang": "string"    //Язык модели
}
```

В ответ на запрос сервер отсылает информацию о выбранной модели в виде json ответа.

Пример JSON ответа:

```
{
 "status": "string"   //Состояние установки модели
}
```

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

Пример сообщения JSON об ошибке:

```
{
 "detail": [
  {
   "loc": [
    "string",
    0
   ],
   "msg": "string",
   "type": "string"
  }
 ]
}
```

Обнаружение текста на изображении

Команда ***Обнаружение текста на изображении***, приложение посылает запрос к конечной точке ***/predict-to-image***, данный запрос позволяет отослать на сервер изображение в виде массива байтов.

Сервер принимает изображение в виде массива байтов и обрабатывает его, в ответ сервер отсылает распознанный текст в виде json ответа.

Пример JSON ответа:

```
{
 "Line": [
  {
   "lineText": "string", //Текст распознанной строки
  }
 ]
}
```

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

Пример сообщения JSON об ошибке:

```
{
 "detail": [
  {
   "loc": [
    "string",
    0
   ],
   "msg": "string",
```

```
    "type": "string"
   }
  ]
}
```

Получить результаты обучения

Команда ***Получить результаты обучения***, приложение посылает запрос к конечной точке ***/training-metrics/{training_name}***, адресованный к серверу.

Сервер в ответ на запрос отсылает информацию о обученной модели, при этом сервер отдает словарь метрик обучения, метрики обучения и значения метрики в виде Json ответа.

Пример JSON ответа:

```
{
 "metrics": {    //Словарь метрик обучения
  "additionalProp1": [   //Метрика 1
   0                     //Значение метрики 1
  ],
  "additionalProp2": [
   0
  ],
  "additionalProp3": [
   0
  ]
 }
}
```

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

Пример сообщения JSON об ошибке:

```
{
 "detail": [
  {
   "loc": [
    "string",
    0
   ],
   "msg": "string",
   "type": "string"
  }
 ]
}
```


Удалить модель

Команда ***Удалить модель***, приложение посылает запрос к конечной точке ***/output/{model_name},*** адресованный серверу.

В ответ на запрос сервер удаляет выбранную пользователем модель в виде json ответа:

Пример JSON ответа:

```
{
 "status": "string"    //Состояние удаления
}
```

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

Пример сообщения JSON об ошибке:

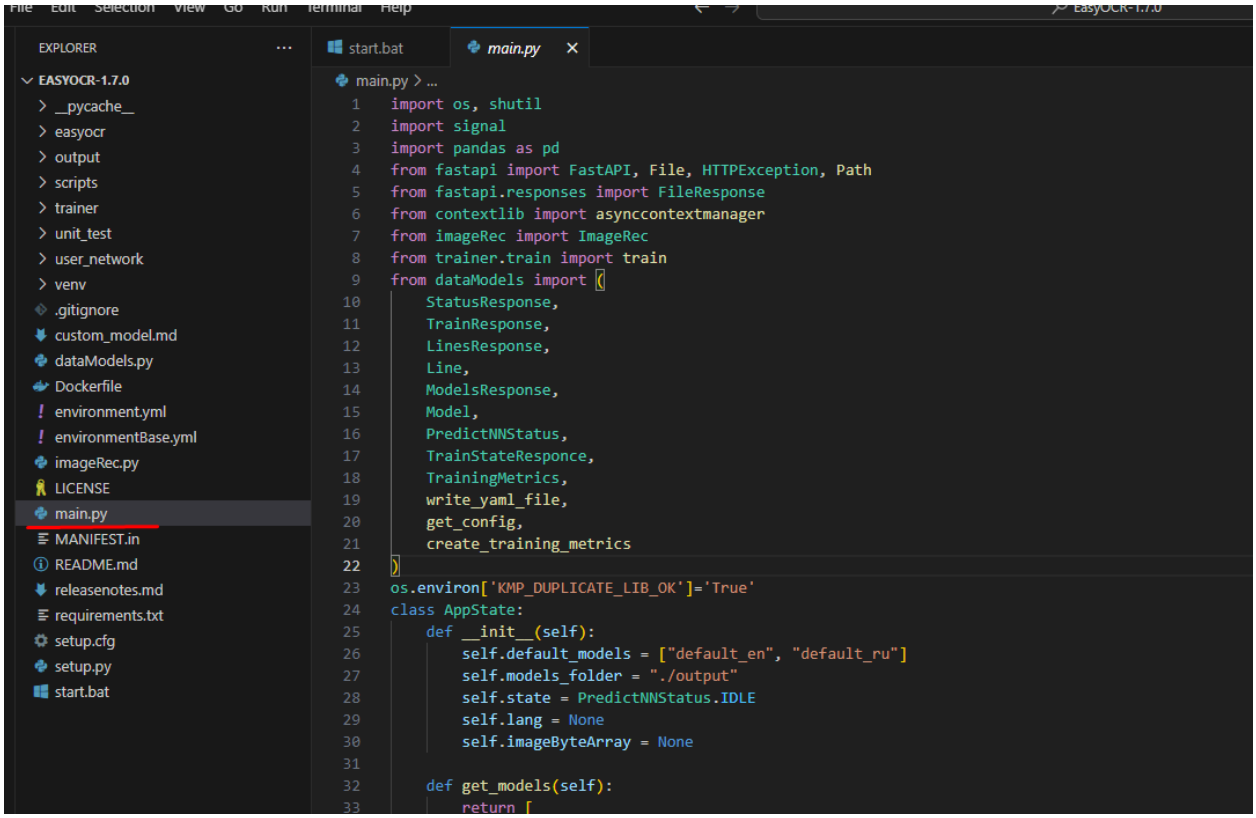
```
{
 "detail": [
  {
   "loc": [
    "string",
    0
   ],
   "msg": "string",
   "type": "string"
  }
 ]
}
```

Описание лабораторной работы

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

Описание лабораторной работы распознавание рукописного текста

Сперва рассмотрим код сервера лабораторной работы по распознаванию рукописного текста а именно файл *main.py*.



Сервер распознавания

Инициализация переменных используемых в лабораторной работе:

```
class AppState:
    def __init__(self):
        self.default_models = ["default_en", "default_ru"]
        self.models_folder = "./output"
        self.state = PredictNNStatus.IDLE
        self.lang = None
        self.imageByteArray = None
```

Инициализация метода получения моделей распознавания:

```
def get_models(self):
        return [
            dir_name
            for dir_name in os.listdir(self.models_folder)
        ]
```

Инициализация метода выбора модели распознавания:

```
def set_model(self, name, lang):
        self.lang = lang
self.nn=ImageRec(f"{self.models_folder}/{name}",self.lang,name.split("_")[0]=="default")
        self.nn.SetModel()
```

Инициализация метода отмены выбора модели:

```
def unset_model(self):
        self.nn = None
```

Метод жизненного цикла сервера, то, что до yield срабатывает при запуске сервера, здесь при запуске сервера устанавливается дефолтная модель, а после выключения сервера дефолтная модель сбрасывается:

```
@asynccontextmanager
async def lifespan(app: FastAPI):
    app_state.set_model(app_state.default_models[0], "en")
    yield
    app_state.unset_model()
```

Инициализация и запуск сервера для распознавания:

```
app = FastAPI(
    title="HandwrittingRecognition",
    description="API для нейронных сетей",
    version="1.0.0",
    lifespan=lifespan
```

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

```
@app.post("/restart", summary="Перезапустить сервер")
def restart():
    os.kill(os.getpid(), signal.SIGINT)
    return StatusResponse(status="restarting")
```

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

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

Метод получения списка моделей, при получении запроса о отправке списка моделей, отправляет список содержащихся на сервере моделей:

```
@app.get("/get-models", response_model=ModelsResponse, summary="Получить список моделей")
def get_models():
    models = []
    for model in app_state.get_models():
        if len(model.split("_")) == 2:
            name, lang = model.split("_")
```

```
            models.append(Model(modelName=name,modelLang=lang))
    return ModelsResponse(models=models)
```

Метод получения состояния обучения модели, при получении запроса отправляет информацию о состоянии обучения:

```
@app.get("/get-train-state", response_model=TrainStateResponce, summary="Получить состояние обучения")
def get_train_state():
    return TrainStateResponce(state = app_state.state)
```

Метод начала обучения нейросети, при получении запроса начинает обучать нейросеть по выбранной модели обучения, в случае успешного старта обучения высылает ответ что обучение началось, в случае ошибки высылает сообщение о ошибке:

```
@app.post("/train", response_model=StatusResponse, summary="Начать тренировку")
def trainModel(train_data: TrainResponse):
        try:
            if train_data.datasetName+"_"+train_data.lang in app_state.get_models():
                raise HTTPException(status_code=400, detail="This name already exist")
            if app_state.state == PredictNNStatus.TRAINING:
                raise HTTPException(status_code=400, detail="Its already in training state, please wait")
            opt = get_config(write_yaml_file(train_data, app_state.models_folder))
            train(opt,appstate=app_state, amp=True)
            return StatusResponse(status="train completed")
        except Exception as e:
            app_state.state= PredictNNStatus.IDLE
            raise HTTPException(status_code=500, detail=f'Failed to train model: {repr(e)}')
```

Метод выбора модели, при получении запроса устанавливает выбранную модель как модель для распознавания:

```
@app.post("/set-model", response_model=StatusResponse, summary="Установить модель")
async def set_model(model_data: Model):
    model_dir = model_data.modelName+"_"+model_data.modelLang
    app_state.set_model(model_dir, model_data.modelLang)
    return StatusResponse(status="model setted")
```

Метод обнаружения текста на полученном изображении, при получении данных от приложения переводит изображение в текст и отправляет обратно в виде строк текста:

```
@app.post("/predict-to-image", summary="Обнаружение текста на изображении")
async def predictToImage(image: bytes = File(..., description="Загружаемое изображение для анализа")):
    if app_state.state == PredictNNStatus.TRAINING:
        raise HTTPException(status_code=400, detail="Its in training state, please wait")
    detected_objects = await app_state.nn.predict_image(image)
```

```
        lines = []
        for line in detected_objects:
            lines.append(Line(lineText=line))
            print(line)
        return LinesResponse(lines=lines)
```

Метод удаления модели, при получении запроса получает список моделей, после получения списка моделей проверяет наличие выбранной модели, в случае если был получен запрос на удаление несуществующей модели выдаст ошибку, в случае если выбрана встроенная модель, то выдаст ошибку о невозможности удалить встроенную модель, если модель существует и не является встроенной то удалит выбранную модель:

```
@app.delete("/output/{model_name}", response_model=StatusResponse, summary="Удалить модель")
async def deleteModel(model_name: str = Path(..., description="Имя модели")):
    if app_state.state == PredictNNStatus.TRAINING:
        raise HTTPException(status_code=400, detail="Its in training state, please wait")
    try:
      if model_name not in app_state.get_models():
        raise HTTPException(status_code=404, detail=f"Model '{model_name}' not found")

      if model_name in app_state.default_models:
        raise HTTPException(status_code=500, detail="Unable to delete the default model")
      model_path = os.path.join(app_state.models_folder, model_name)

      if os.path.exists(model_path):
        shutil.rmtree(model_path)
        return StatusResponse(status="success")
    except:
      raise HTTPException(status_code=500, detail="Failed to delete model")
```

Метод получение результатов тренировки модели, при получении запроса отправляет результаты о обучении модели, независимо от получения запросов отправляет информацию о метриках для построения графиков обучения:

```
app.get("/training-metrics/{training_name}", response_model=TrainingMetrics, summary="Получить результаты тренировки")
async def training_metrics(training_name: str = Path(..., description="Имя тренировки")):
    try:
      if training_name not in app_state.get_models():
        raise HTTPException(status_code=400, detail="This training name does not exist")

      results_dir = os.path.join(app_state.models_folder, training_name, "results.csv")

      if os.path.exists(results_dir):
```

```
        df = pd.read_csv(results_dir, skipinitialspace=True)
    else:
        df = pd.DataFrame()
    metrics = create_training_metrics(df)
    return TrainingMetrics(metrics=metrics)
except HTTPException:
    raise
except Exception as e:
    raise HTTPException(status_code=500, detail="No metrics are available")
```

Рассмотрим блок взаимодействия сервера и нейросети распознавания, для этого откроем файл imageRec.py

Блок взаимодействия

Инициализация переменных данного блока:

```
class ImageRec:
    def __init__(self, rectModelDir, lang, isDefault):
        self.rec_model_dir = rectModelDir
        self.lang = lang
        self.imageByteArray = None
        self.isDefault = isDefault
        self.reader=None
```

Метод отправки модели распознавания с сервера в нейросеть:

```
def SetModel(self):
        if self.isDefault:
            self.reader = easyocr.Reader([self.lang],True, recognizer=True)
        else:
            self.reader                                                    =
easyocr.Reader([self.lang],True,model_storage_directory=self.rec_model_dir,
                    user_network_directory='./user_network',
                    recog_network=self.lang)
```

Инициализация метода получения моделей распознавания:

```
def get_models(self):
        return [
            dir_name
            for dir_name in os.listdir(self.models_folder)
        ]
```

Инициализация метода выбора модели распознавания:

```
 def set_model(self, name):
      self.unset_model()
      self.nn = PredictNN(f"{self.models_folder}/{name}")
```

Инициализация метода отмены выбора модели:

```
def unset_model(self):
        if self.nn is not None:
            self.nn.clear()
            del self.nn
            self.nn = None
        torch.cuda.empty_cache()
```

Метод жизненного цикла сервера, то, что до yield срабатывает при запуске сервера, здесь при запуске сервера устанавливается дефолтная модель распознавания, а после выключения сервера дефолтная модель сбрасывается:

```
@asynccontextmanager
async def lifespan(app: FastAPI):
    app_state.set_model(app_state.default_models[0]+"/weights/best.pt")
    yield
    app_state.unset_model()
```

Инициализация и запуск сервера для распознавания

```
app = FastAPI(
    title="PLNeuro",
    description="API для нейронных сетей",
    version="1.0.0",
    lifespan=lifespan
)
```

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

```
@app.post("/restart", summary="Перезапустить сервер")
def restart():
    os.kill(os.getpid(), signal.SIGINT)
    return StatusResponse(status="restarting")
```

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

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

Метод получения списка моделей, при получении запроса о отправке списка моделей, отправляет список содержащихся на сервере моделей:

```
@app.get("/get-models", response_model=ModelsResponse, summary="Получить список моделей")
def get_models():
    models = []
    for model in app_state.get_models():
            if len(model.split("_")) == 2:
                name, lang = model.split("_")
                models.append(Model(modelName=name,modelLang=lang))
    return ModelsResponse(models=models)
```

Метод получения состояния обучения модели, при получении запроса отправляет информацию о состоянии обучения:

```
@app.get("/get-train-state", response_model=TrainStateResponce, summary="Получить состояние обучения")
def get_train_state():
    return TrainStateResponce(state = app_state.state)
```

Метод начала обучения нейросети, при получении запроса начинает обучать нейросеть по выбранной модели обучения, в случае успешного старта обучения высылает ответ что обучение началось, в случае ошибки высылает сообщение о ошибке:

```
@app.post("/train", response_model=StatusResponse, summary="Начать тренировку")
def train(train_data: TrainResponse):
        try:
            if train_data.datasetName+"_"+train_data.lang in app_state.get_models():
                raise HTTPException(status_code=400, detail="This name already exist")
            if app_state.state == PredictNNStatus.TRAINING:
                raise HTTPException(status_code=400, detail="Its already in training state, please wait")
```

```
            app_state.nn.train_classify(train_data,app_state)

            return StatusResponse(status="train started")
        except Exception as e:
            app_state.state= PredictNNStatus.IDLE
            raise HTTPException(status_code=500, detail=f'Failed to train model: {repr(e)}')
```

Метод выбора модели, при получении запроса устанавливает выбранную модель как модель для распознавания:

```
@app.post("/set-model", response_model=StatusResponse, summary="Установить модель")
async def train(model_data: Model):
    model_name = model_data.modelName+"_"+model_data.modelLang+"/weights/best.pt"
    app_state.set_model(model_name)
    return StatusResponse(status="model setted")
```

Метод обнаружения текста на полученном изображении, при получении данных от приложения переводит изображение в текст и отправляет обратно в виде строк текста:

```
@app.post("/predict-to-image", summary="Обнаружение текста на изображении")
async def predictToImage(image: bytes = File(..., description="Загружаемое изображение для анализа")):
    if app_state.state == PredictNNStatus.TRAINING:
        raise HTTPException(status_code=400, detail="Its in training state, please wait")
    imageFile = Image.open(BytesIO(image))
    detected_objects= await app_state.nn.predict_symbol_image(imageFile)
    lines = []
    for line in detected_objects:
        lines.append(Line(lineText=line))
        print(line)

    print(len(lines))
    return LinesResponse(lines=lines)
```

Метод удаления модели, при получении запроса получает список моделей, после получения списка моделей проверяет наличие выбранной модели, в случае если был получен запрос на удаление несуществующей модели выдаст ошибку, в случае если выбрана встроенная модель, то выдаст ошибку о невозможности удалить встроенную модель, если модель существует и не является встроенной то удалит выбранную модель:

```
@app.delete("/output/{model_name}", response_model=StatusResponse, summary="Удалить модель")
async def deleteModel(model_name: str = Path(..., description="Имя модели")):
    if app_state.state == PredictNNStatus.TRAINING:
        raise HTTPException(status_code=400, detail="Its in training state, please wait")
    try:
```

```
        if model_name not in app_state.get_models():
            raise HTTPException(status_code=404, detail=f"Model '{model_name}' not found")

        if model_name in app_state.default_models:
            raise HTTPException(status_code=500, detail="Unable to delete the default model")
        model_path = os.path.join(app_state.models_folder, model_name)

        if os.path.exists(model_path):
            shutil.rmtree(model_path)
            return StatusResponse(status="success")
    except:
        raise HTTPException(status_code=500, detail="Failed to delete model")
```

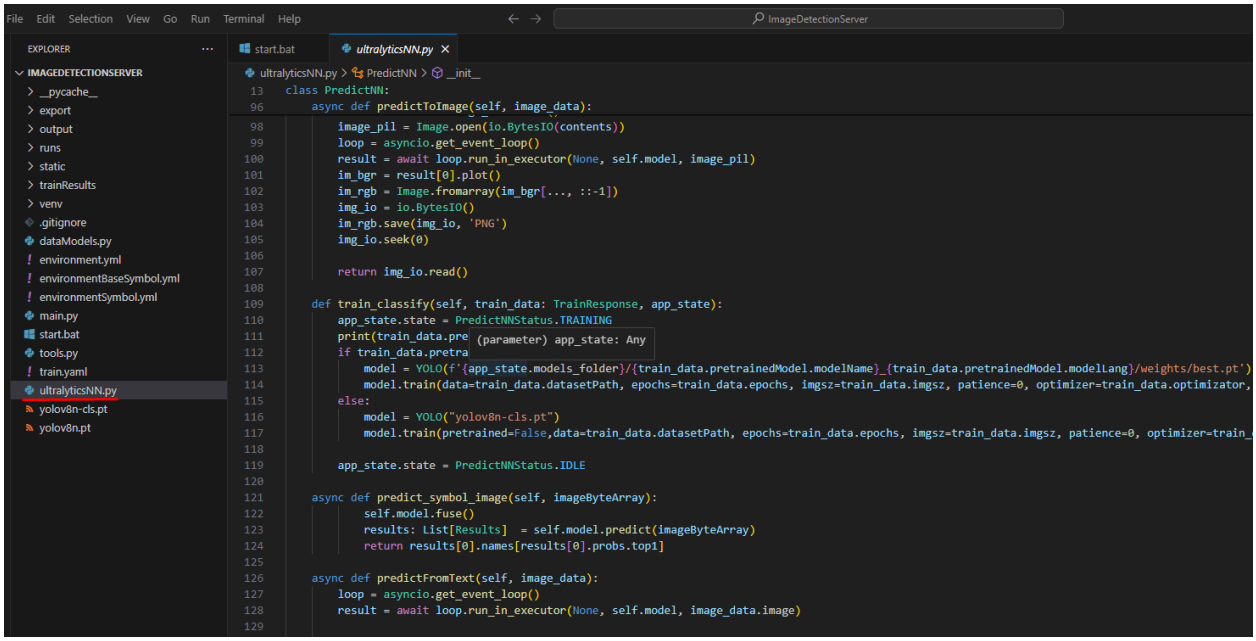
Метод получение результатов тренировки модели, при получении запроса отправляет результаты о обучении модели, независимо от получения запросов отправляет информацию о метриках для построения графиков обучения:

```
@app.get("/training-metrics/{training_name}", response_model=TrainingMetrics, summary="Получить результаты тренировки")
async def training_metrics(training_name: str = Path(..., description="Имя тренировки")):
    try:
        if training_name not in app_state.get_models():
            raise HTTPException(status_code=400, detail="This training name does not exist")

        results_dir = os.path.join(app_state.models_folder, training_name, "results.csv")

        if os.path.exists(results_dir):
            df = pd.read_csv(results_dir, skipinitialspace=True)
        else:
            df = pd.DataFrame()
        metrics = create_training_metrics(df)
        return TrainingMetrics(metrics=metrics)
    except HTTPException:
        raise
    except Exception as e:
        raise HTTPException(status_code=500, detail="No metrics are available")
```

Рассмотрим блок взаимодействия сервера и нейросети распознавания, для этого откроем файл ultralyticsNN.py



Блок взаимодействия

Инициализация переменных данного блока:

```
1   def __init__(self, model_path):
2       self.model = YOLO(model_path)
3       self.detectionTrainer = None
4       self.status = PredictNNStatus.IDLE
5       self.model.add_callback("on_pretrain_routine_start",
6   self.on_pretrain_routine_start)
7       self.model.add_callback("on_pretrain_routine_end", self.on_pretrain_routine_end)
8       self.model.add_callback("on_train_start", self.on_train_start)
9       self.model.add_callback("on_train_epoch_start", self.on_train_epoch_start)
10      self.model.add_callback("on_train_batch_start", self.on_train_batch_start)
11      self.model.add_callback("optimizer_step", self.optimizer_step)
12      self.model.add_callback("on_before_zero_grad", self.on_before_zero_grad)
13      self.model.add_callback("on_train_batch_end", self.on_train_batch_end)
14      self.model.add_callback("on_train_epoch_end", self.on_train_epoch_end)
15      self.model.add_callback("on_fit_epoch_end", self.on_fit_epoch_end)
16      self.model.add_callback("on_model_save", self.on_model_save)
17      self.model.add_callback("on_train_end", self.on_train_end)
18      self.model.add_callback("on_params_update", self.on_params_update)
        self.model.add_callback("teardown", self.teardown)
```

Метод начала обучения нейросети:

```
1   def train_classify(self, train_data: TrainResponse, app_state):
2   app_state.state = PredictNNStatus.TRAINING
3   print(train_data.pretrainedModel.modelName)
4   if train_data.pretrainedModel.modelName != '':
5   model =
    YOLO(f'{app_state.models_folder}/{train_data.pretrainedModel.modelName}_{train_data.pretrainedModel.modelLang}/weights/best.pt')
6   model.train(data=train_data.datasetPath, epochs=train_data.epochs, imgsz=train_data.imgsz,
    patience=0, optimizer=train_data.optimizator, lr0=train_data.learningRate,
```

```
                batch=train_data.batchSize,                    save_period=int(train_data.epochs/10)
7               ,name=train_data.datasetName+"_"+ train_data.lang ,project="./output")
8               else:
9               model = YOLO("yolov8n-cls.pt")
                model.train(pretrained=False,data=train_data.datasetPath,          epochs=train_data.epochs,
                imgsz=train_data.imgsz,          patience=0,          optimizer=train_data.optimizator,
                lr0=train_data.learningRate,  batch=train_data.batchSize,  save_period=int(train_data.epochs/10)
10              ,name=train_data.datasetName+"_"+ train_data.lang ,project="./output")
11
                app_state.state = PredictNNStatus.IDLE
```

Метод получения текста от нейросети с загруженного изображения для распознавания:

```
1   async def predict_symbol_image(self, imageByteArray):
2           self.model.fuse()
3           results: List[Results]  = self.model.predict(imageByteArray)
4           return results[0].names[results[0].probs.top1]
```

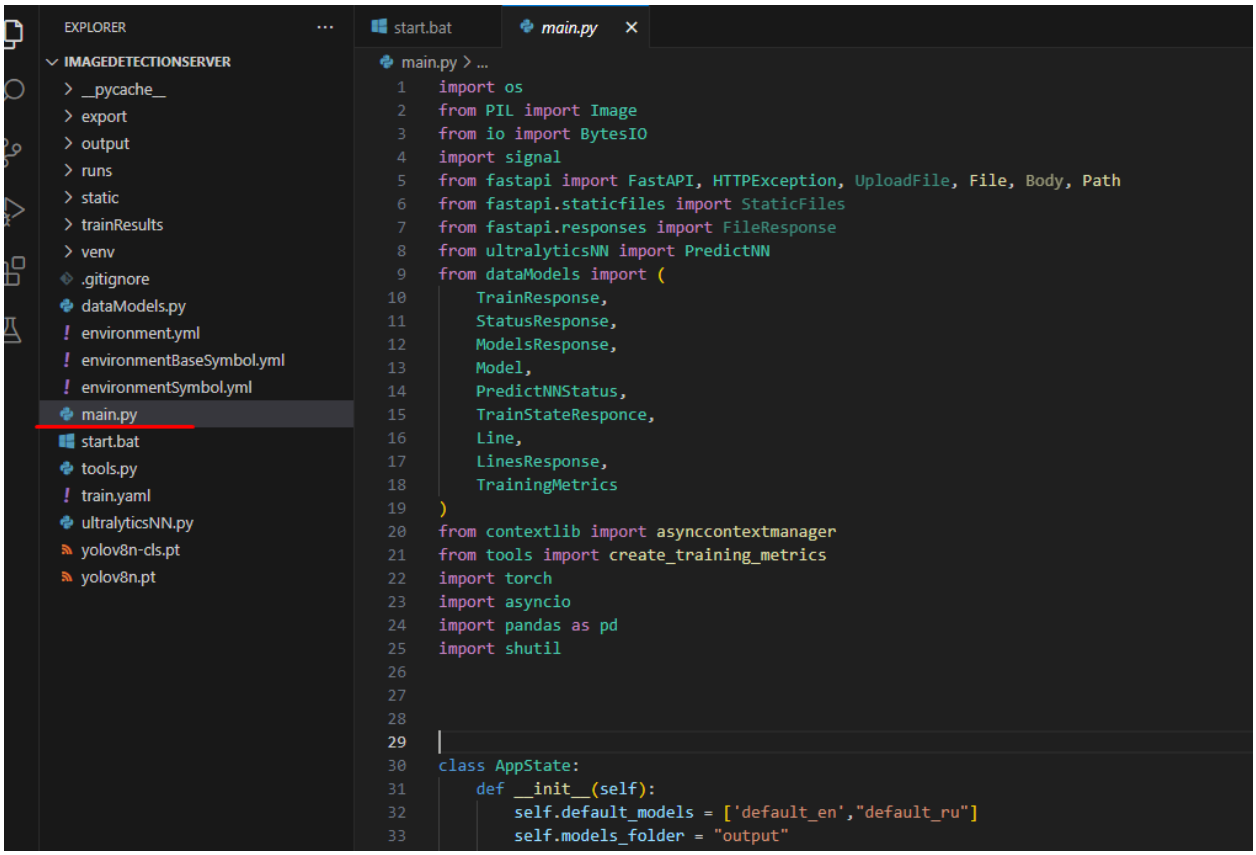
Метод получения текста от нейросети с загруженного изображения для распознавания:

```
async def predict_image(self, imageByteArray):
        print(self.lang)
        print(self.rec_model_dir)

        result = self.reader.readtext(imageByteArray, detail = 0)
        return result
```

Описание лабораторной работы распознавание рукописных символов

Рассмотрим код сервера лабораторной работы по распознаванию рукописных символов, а именно файл *main.py*.



Сервер распознавания

Инициализация переменных используемых в лабораторной работе:

```
class AppState:
    def __init__(self):
        self.default_models = ['default_en',"default_ru"]
        self.models_folder = "output"
        self.state = PredictNNStatus.IDLE
        self.light_lock = asyncio.Lock()
        self.hard_lock = asyncio.Lock()
        self.nn = None
```

Sk Resident

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

ООО “ПРОГРАМЛАБ” INFO@PL-LLC.RU PLLLC.RU 8 (800) 550 89 72 +7 (499) 685 80 00