Министерство науки и высшего образования Российской Федерации
Муромский институт (филиал)
федерального государственного бюджетного образовательного учреждения высшего образования
**«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»**
(МИ ВлГУ)

Факультет Информационных технологий

Кафедра Информационные системы

УТВЕРЖДАЮ

Заведующий кафедрой

Д.Е. Андрианов
(подпись)

«_____» _____ 2021 г.

БАКАЛАВРСКАЯ РАБОТА

Тема Разработка программы имитационного моделирования моделей машинного обучения

МИВУ.09.03.02-05.000 БР

Руководитель

Щаников С.А.
(фамилия, инициалы)

(подпись) (дата)

Студент ИС-117
(группа)

Минеев Р.Р.
(фамилия, инициалы)

(подпись) (дата)

Муром 2021

БЛАНК ЗАДАНИЯ

[Аннотация русскоязычная]

[Аннотация англоязычная]

Министерство науки и высшего образования Российской Федерации
Муромский институт (филиал)
федерального государственного бюджетного образовательного учреждения высшего образования
**«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»**
(МИВлГУ)

Факультет Информационных технологий

Кафедра Информационные системы

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Тема: Разработка программы имитационного моделирования моделей машинного обучения

МИВУ.09.03.02-05.000 ПЗ

Муром 2021

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Значимость работы, как она может улучшить мир

Цель данной бакалаврской работы – разработать программу имитационного моделирования моделей машинного обучения при условии изменения данных внутри нейросети.

Задачи, поставленные на бакалаврскую работу:

- произвести обзор предметной области;
- проанализировать методы ускорения процесса обработки нейросетей;
- …
- проанализировать изменение данных внутри нейросетей при различных отклонениях входных параметров и сделать соответствующие выводы.

1 Анализ технического задания

1.1 Зачем нейросетям краш-тест

Потом блок от преподавателя.

1.2 Обзор методов организации параллельных вычислений

Необходимость разделять вычислительные задачи и выполнять их одновременно (параллельно) возникла задолго до появления первых вычислительных машин.

Существует 3 метода распараллеливания расчетов:

- распараллеливание по задачам (такое распараллеливание актуально для сетевых серверов и других вычислительных систем, выполняющих одновременно несколько функций либо обслуживающих многих пользователей);

- распараллеливание по инструкциям (аппаратно реализовано в современных центральных процессорах общего назначения, поскольку оно эффективно при исполнении программ, интенсивно обменивающихся разнородной информацией с другими программами и с пользователем компьютера);

- распараллеливание по данным.

Потоковая обработка данных особенно эффективна для алгоритмов, обладающих следующими свойствами, характерными для задач физического и математического моделирования:

- большая плотность вычислений — велико число арифметических операций, приходящихся на одну операцию ввода-вывода (например, обращение к памяти). Во многих современных приложениях обработки сигналов она достигает 50:1, причем со сложностью алгоритмов увеличивается;

- локальность данных по времени - каждый элемент загружается и обрабатывается за время, малое по отношению к общему времени обработки, после чего он больше не нужен. В результате в памяти потокового процессора для каждого «вычислителя» можно хранить только данные, необходимые для обработки одного элемента, в отличие от центральных процессоров с моделью произвольно зависимых данных.

Со времени своего появления в начале 1980-х годов, персональные компьютеры развивались в основном как машины для выполнения программ, сложных по внутренней структуре, содержащих большое количество ветвлений, интенсивно взаимодействующих с пользователем, но редко связанных с потоковой обработкой большого количества однотипных данных. Центральные процессоры ПК оптимизировались для решения именно таких задач, поэтому характеризовались следующим:

- большим количеством блоков для управления исполнением программы (кеширование данных, предсказание ветвлений и т.п.) и сравнительно малым количеством блоков для вычислений;

- архитектурой, оптимальной для программ со сложным потоком управления (обработка разнородных команд и данных, организация взаимодействия программ между собой и с пользователем);

- памятью с максимальной скоростью произвольного доступа к данным.

Увеличение производительности CPU в основном было связано с увеличением тактовой частоты и размеров высокоскоростной кеш-памяти (память, расположенная прямо на процессоре). Программирование CPU для ресурсоемких научных вычислений подразумевает тщательное структурирование данных и порядка инструкций для эффективного использования всех уровней кеш-памяти.

Ядра современных центральных процессоров являются суперскалярными, поддерживая векторную обработку (расширения SSE и 3DNow!), сами же CPU обычно содержат несколько ядер. Таким образом, в совокупности центральные процессоры могут реализовывать десятки параллельных вычислительных потоков. Однако графические процессоры включают в себя тысячи параллельных

«вычислителей». Кроме того, при поточно-параллельных расчетах графические процессоры имеют преимущество благодаря следующим особенностям архитектуры:

- память GPU оптимизирована на максимальную пропускную способность (а не на скорость произвольного доступа, как у CPU), что ускоряет загрузку потока данных;

- большая часть транзисторов графического процессора предназначена для вычислений, а не для управления исполнением программы;

- при запросах к памяти, за счет конвейерной обработки данных, не происходит приостановки вычислений.

Однако обработка ветвлений (исполнение операций условного перехода) на GPU менее эффективна, поскольку каждый управляющий блок обслуживает не один, а несколько вычислителей.

Таким образом, производительность одного GPU при хорошо распараллеливаемых вычислениях аналогична кластеру из сотен обычных вычислительных машин, причем графические процессоры сейчас поддерживают практически все операции, используемые в алгоритмах общего назначения:

- распространенные математические операции и функции вещественного аргумента. В рамках SM 4.0 поддерживаются целые числа и логические операции, а в SM 4.1 и CUDA — также и вещественные числа двойной (64-битной) точности;

- организацию циклов. В SM 3 длина циклов ограничена 255 итерациями, в SM 4 длина циклов не ограничена;

- операции условного перехода (которые исполняются сравнительно медленно, поскольку в составе GPU блоков управления меньше, чем вычислительных блоков).

В различных источниках информации можно найти много разных определений процессов и потоков. Такой разброс определений обусловлен, во-первых, эволюцией операционных систем, которая приводила к изменению понятий о процессах и потоках, во-вторых, различием точек зрения, с которых рассматриваются эти понятия.

С точки зрения пользователя, процесс — экземпляр программы во время выполнения, а потоки — ветви кода, выполняющиеся «параллельно», то есть без предписанного порядка во времени.

С точки зрения операционной системы, процесс — это абстракция, реализованная на уровне операционной системы. Процесс был придуман для организации всех данных, необходимых для работы программы. Процесс — это просто контейнер, в котором находятся ресурсы программы:

- адресное пространство;
- потоки;
- открытые файлы;
- дочерние процессы и т.д.

Также, с точки зрения операционной системы, поток — это абстракция, реализованная на уровне операционной системы. Поток был придуман для контроля выполнения кода программы. Это контейнер, в котором находятся:

- счётчик команд;
- регистры;
- стек.

Поток легче, чем процесс, и создание потока стоит дешевле. Потоки используют адресное пространство процесса, которому они принадлежат, поэтому потоки внутри одного процесса могут обмениваться данными и взаимодействовать с другими потоками.

Поддержка множества потоков внутри одного процесса в рамках работы над разрабатываемой программой необходима. В случае, когда одна программа выполняет множество задач, поддержка множества потоков внутри одного процесса позволяет:

- разделить ответственность за разные задачи между разными потоками;
- повысить быстродействие.

Кроме того, часто задачам необходимо обмениваться данными, использовать общие данные или результаты других задач. Такую возможность предоставляют потоки внутри процесса, так как они используют адресное пространство процесса,

которому принадлежат. Конечно, можно было бы создать под разные задачи дополнительные процессы, но:

- у процесса будет отдельное адресное пространство и данные, что затруднит взаимодействие частей программы;

- создание и уничтожение процесса дороже, чем создание потока.

Отличие процесса от потока состоит в том, что процесс рассматривается операционной системой, как заявка на все виды ресурсов (память, файлы и пр.), кроме одного — процессорного времени. Поток — это заявка на процессорное время. Процесс — это всего лишь способ сгруппировать взаимосвязанные данные и ресурсы, а потоки — это единицы выполнения, которые выполняются на процессоре.

1.3 Выбор средства

Для разработки программы были выбраны следующие средства:

1) язык программирования Python;

2) модуль Threading;

3) модуль Multiprosessing.

Язык программирования Python был выбран исходя из того, что данный язык часто используется для написания нейросетей.

В случае с разработкой данной системы использование потоков не подойдет, так как Python имеет GIL.

GIL (Global Interpreter Lock) – это своеобразная блокировка, позволяющая только одному потоку управлять интерпретатором Python. Это означает, что в любой момент времени будет выполняться только один конкретный поток.

Работа GIL может казаться несущественной для разработчиков, создающих однопоточные программы. Но во многопоточных программах отсутствие GIL может негативно сказываться на производительности процессоро-зависимых программ.

Модуль Threading впервые был представлен в Python 1.5.2 как продолжение низкоуровневого модуля потоков. Модуль Threading значительно упрощает работу

с потоками и позволяет программировать запуск нескольких операций одновременно. Потоки в Python лучше всего работают с операциями ввода/вывода, такими как загрузка ресурсов из интернета или чтение файлов на компьютере.

Модуль Multiprocessing был добавлен в Python версии 2.6. Изначально он был определен в PEP 371 Джесси Ноллером и Ричардом Одкерком. Модуль Multiprocessing позволяет создавать процессы таким же образом, как при создании потоков при помощи модуля Threading. Использование данного модуля позволяет обойти GIL и воспользоваться возможностью использования нескольких процессоров на компьютере.

1.4 Требования к разрабатываемой программе

На вход разрабатываемой программе должна поступать модель нейронной сети, а также тестовые значения алгоритма рандомизации данных.

Разрабатываемая программа должна иметь следующие функции:

- функцию загрузки модели из класса программы (а также, описание входных данных тестирования нейронной сети: на чем она обучалась, каким образом загружается тестовая выборка);

- функция определения и вывода временных данных (например, весов нейросетей на всех эпохах обучения);

- функция загрузки полученных данных в исходную модель (таким образом создается новая модель, с новыми данными);

- функция, возвращающая показатели нейросети (точность или расхождение) после проверки модели.

Задачей разрабатываемой программы является повышение скорости расчётов большого количество данных (большого количества тестов, которые будут проводиться для модели нейронной сети).

На выходе пользователь программы должен получать график и возможность сохранения готового ответа, выданного программой.

Добавить картинок

2 Проектирование

На рисунке … представлено различие между последовательной схемой решения (верхняя часть схемы) и способом распределенных вычислений (нижняя часть схемы). Последовательная схема решения в данном случае не подходит, так как предполагается довольно большой объем обрабатываемых данных.

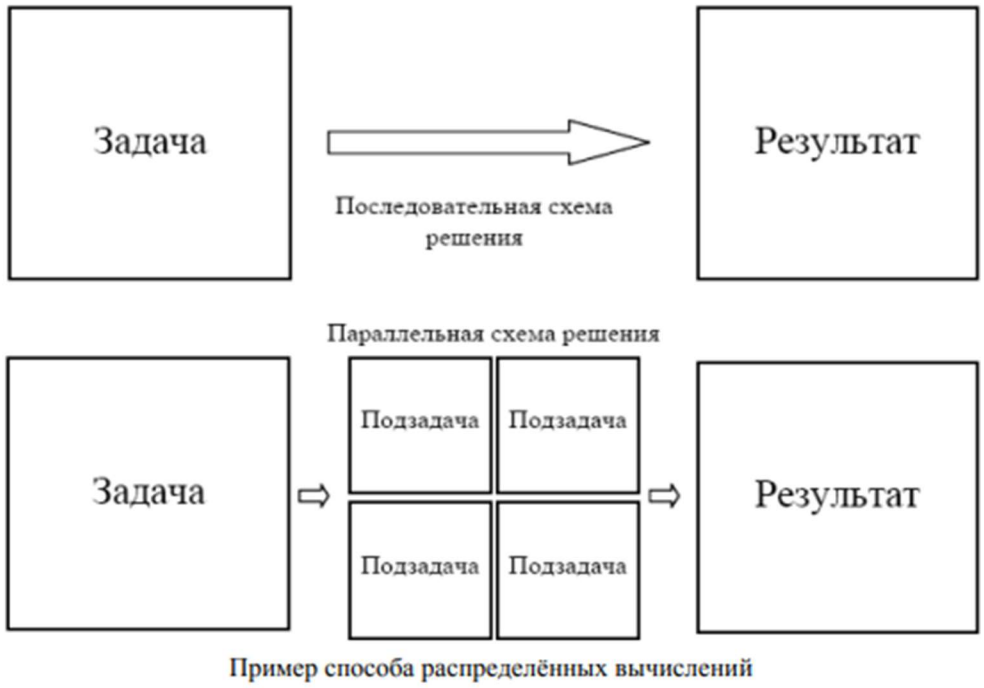


Рисунок … - Схема последовательных и распределенных вычислений

Процессор компьютера, на котором производится разработка программы, содержит 12 ядер. В процессе тестирования работы программы будут использоваться все ядра системы. Таким образом, теоретически, производительность программы должна вырасти.

Для генерации случайных значений весов будут использоваться функции нормального распределения, равномерного распределения и распределения Пуассона. Функцию, с которой будет работать модель, выбирает пользователь.

Нормальным называется распределение вероятностей, которое для одномерного случая задаётся функцией Гаусса.

Нормальное распределение играет важнейшую роль во многих областях знаний. Случайная величина подчиняется нормальному закону распределения,

когда она подвержена влиянию большого числа случайных факторов, что является типичной ситуацией в анализе данных. Поэтому нормальное распределение служит хорошей моделью для многих реальных процессов.

Нормальное распределение зависит от 4-х параметров:

- математическое ожидание - «центр тяжести» распределения;

- дисперсия - степень разброса случайной величины относительно математического ожидания;

- коэффициент асимметрии - параметр формы распределения, определяющий его симметрию относительно математического ожидания;

- коэффициент эксцесса - параметр распределения, задающий «остроту» пика распределения.

Типичные формы нормального распределения для различных средних и дисперсии представлены на рисунке …:

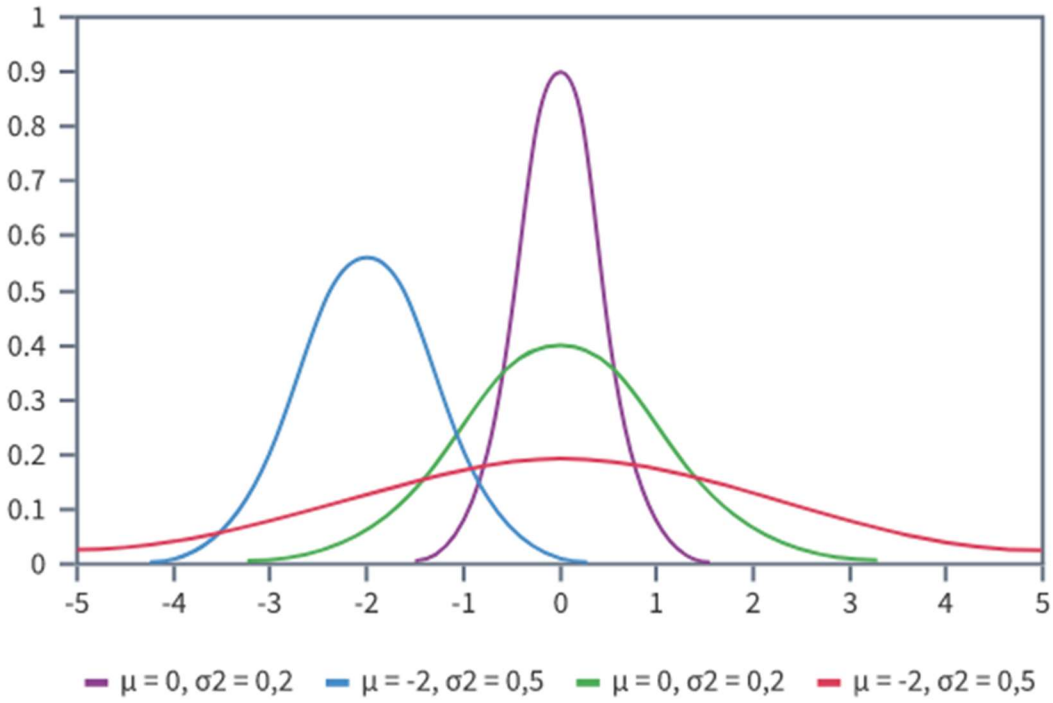


Рисунок … - Примеры форм нормального распределения

Формула данной функции – формула 1:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right), \quad (1)$$

где μ — математическое ожидание, σ^2 — дисперсия.

Коэффициент асимметрии определяется следующим образом (формула 2):

$$\gamma 1 = E\left[\left(\frac{x-\mu}{\sigma}\right)^3\right], \quad (2)$$

где E - знак усреднения. Если коэффициент асимметрии положителен, то правый «хвост» распределения длиннее левого, и отрицателен в противном случае. Если распределение симметрично относительно математического ожидания, то его коэффициент асимметрии равен нулю.

Коэффициент эксцесса вычисляется по формуле 3:

$$\gamma 2 = E\left[\left(\frac{x-\mu}{\sigma}\right)^4\right], \quad (3)$$

Равномерным распределением непрерывной случайной величины называется распределение, в котором значения случайной величины с двух сторон ограничены и в границах интервала имеют одинаковую вероятность. Это означает, что в данном интервале плотность вероятности постоянна.

Таким образом, при равномерном распределении плотность вероятности имеет вид (формула 4):

$$f(x) = \begin{cases} \frac{1}{b-a}, & \text{если } x \in [a,b], \\ 0, & \text{если } x \notin [a,b]. \end{cases} \quad (4)$$

Значения $f(x)$ в крайних точках a и b участка (a, b) не указываются, так как вероятность попадания в любую из этих точек для непрерывной случайной величины равна нулю.

Кривая равномерного распределения имеет вид прямоугольника, опирающегося на участок (a,b) (рисунок …), в связи с чем равномерное распределение иногда называют «прямоугольным».

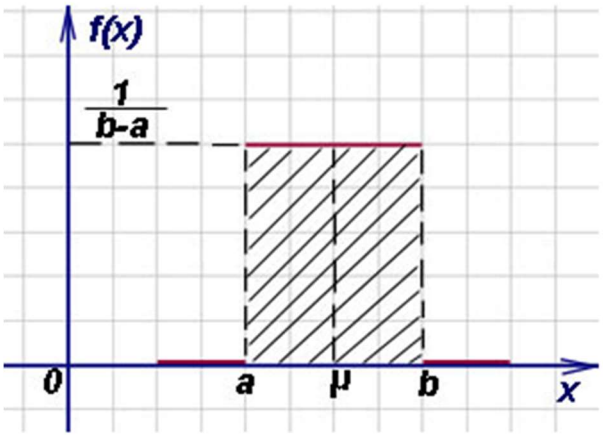


Рисунок … - График кривой равномерного распределения

Функция распределения $F(x)$ непрерывной случайной величины при равномерном распределении имеет вид (формула 5):

$$F(x) = \begin{cases} 0, & \text{если } x < a, \\ \frac{x-a}{b-a}, & \text{если } x \in [a, b], \\ 1, & \text{если } x > b. \end{cases} \quad (5)$$

Характеристики равномерного распределения:

- среднее значение (математическое ожидание) (формула 6):

$$\mu = \frac{a+b}{2}; \quad (6)$$

- дисперсия (формула 7):

$$\sigma^2 = \frac{(b-a)^2}{12}; \quad (7)$$

- стандартное отклонение (формула 8):

$$\sigma = \frac{(b-a)}{\sqrt{12}}; \quad (8)$$

- равномерное распределение не имеет моды.

(тут добавить про пуассоновское распределение)

На рисунке … представлена общая схема работы программы:

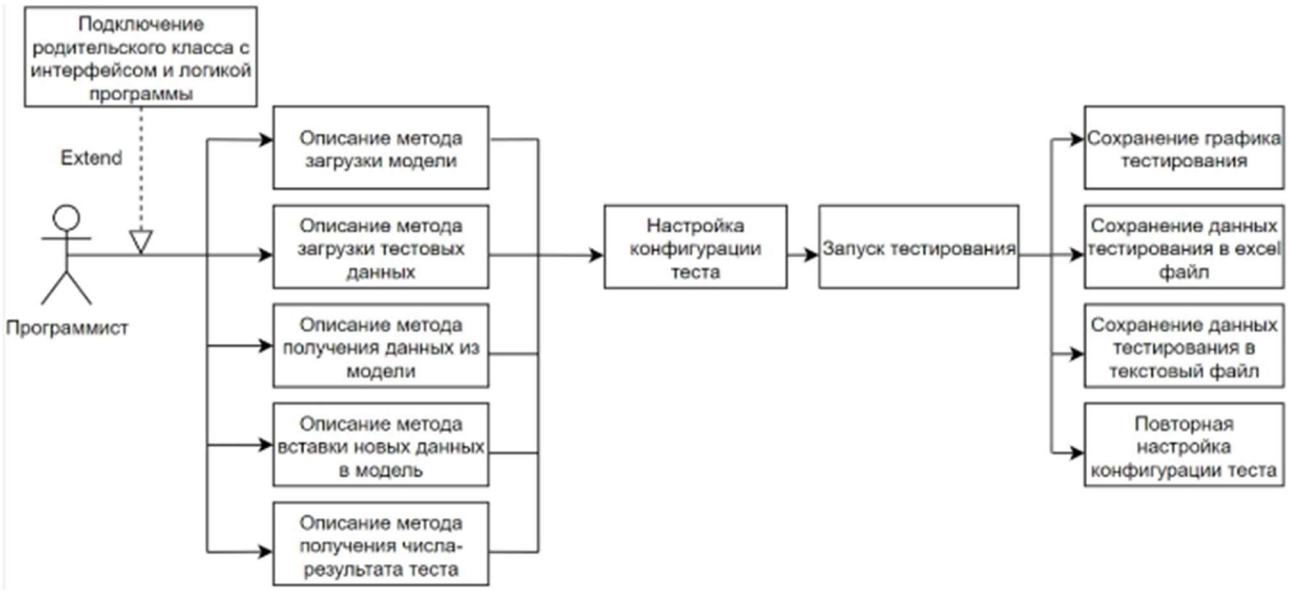


Рисунок … - Схема работы программы

Отсюда видно, что пользователь программы – это программист, перед которым стоит задача написать свой класс, унаследованный от родительского класса, в разрабатываемой программе.

Работа с программой начинается с подключения родительского класса с интерфейсом и логикой программы. Далее пользователь выполняет следующие действия:

- описывает метод загрузки модели (загрузка обученной модели для последующего тестирования);

- описывает метод загрузки тестовых данных (либо загружает готовые данные (изображения, сигналы и т.д.), либо описывает код-генератор тестовых данных);

- описывает метод получения данных из модели (данные (веса и пр.), которые будут изменяться в зависимости от выбранного алгоритма рандомизации данных);

- описывает метод загрузки новых данных в модель;

- описывает метод получения числа – результата теста (вывод точности распознавания, расчет ошибки при распознавании или др.).

Далее происходит настройка конфигурации теста, в процессе которой пользователь задает следующие параметры:

- количество ядер, которые будут задействованы в процессе работы программы (данная характеристика ограничена числом ядер, имеющихся в системе, по умолчанию выбрано максимальное количество);

- алгоритм обработки данных (нормальное распределение, равномерное распределение или распределение Пуассона);

- список значений для алгоритма генерации случайных чисел.

Генерация последнего параметра происходит при помощи 4-х данных:

- начального значения (по умолчанию - 0);

- конечного значения;

- шага;

- количества тестов на каждом шаге (в процессе тестирования полученные данные усредняются).

Далее происходит сам процесс тестирования, после чего на экране программы отображается график, отражающий результат тестирования.

После этого пользователь может сделать следующее:

- сохранить полученный в результате тестирования график в виде изображения;

- сохранить полученные данные тестирования в файл .xlsx;

- сохранить полученные данные тестирования в текстовый файл;

- повторно настроить конфигурацию теста.

На рисунке … приведена диаграмма последовательности, которая описывает последовательность работы пользователя в программе.

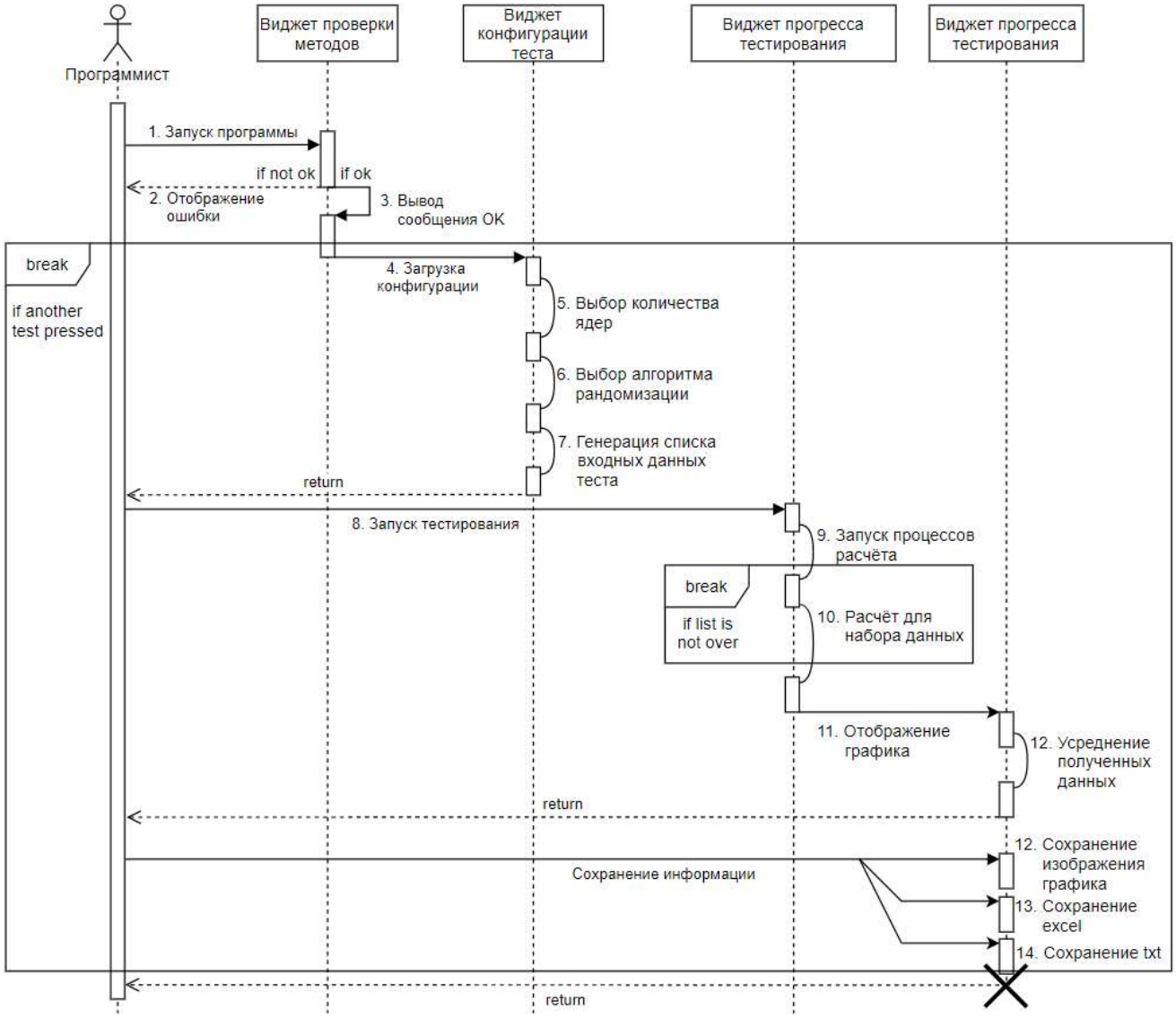


Рисунок … - Диаграмма последовательности

На данном этапе процесс проектирования можно считать завершенным. Следующий этап – разработка программы.

3 Разработка

В данном разделе пояснительной записки описан процесс разработки программы имитационного моделирования моделей машинного обучения.

Проверка функций, описанных пользователем, производится следующим образом:

```
            try:
                if self.name == 'load_model':
                    self.neuralconfig.load_model()
                    try:
                        pickle.dumps(self.neuralconfig.model)
                    except TypeError:
                        raise PickleProblem

                elif self.name == 'load_testdata':
                    self.neuralconfig.load_testdata()

                elif self.name == 'get_tested_values':
                    self.neuralconfig._basedata =
self.neuralconfig.get_tested_values()
                    if type(self.neuralconfig._basedata) is not
list:
                        raise NumpyProblem
                    for array in self.neuralconfig._basedata:
                        if type(array) is not np.ndarray:
                            raise NumpyProblem

                elif self.name == 'set_tested_values':

self.neuralconfig.set_tested_values(self.neuralconfig._basedata)

                elif self.name == 'test_model':
```

```
                    self.neuralconfig._test_value =
self.neuralconfig.test_model(self.neuralconfig.model,
self.neuralconfig.testdata)
                self.function_checked_signal.emit(self.name, 'OK')
            except PickleProblem:
                self.function_checked_signal.emit(self.name,
'ERROR')
                self.error_signal.emit('Your model is not
pickleable.\nTry to pickle your model before launch the program',
self.name)
            except NumpyProblem:
                self.function_checked_signal.emit(self.name,
'ERROR')
                self.error_signal.emit('This method must returns a
list of numpy arrays (numpy.ndarray)', self.name)
            except Exception as e:
                self.function_checked_signal.emit(self.name,
'ERROR')
                self.error_signal.emit(str(e), self.name)

            self.return_new_state.emit(self.neuralconfig)
            self.done.emit()
```

Для каждого метода, описанного пользователем, код данной функции запускается последовательно. Первоначально проверяется метод load_model, который загружает модель в класс.

Данная функция выведет сообщение об ошибке, если:

- метод load_model не описан (его не существует);

- в коде этого метода обнаружена синтаксическая ошибка;

- модель нельзя представить в байтовом виде.

Модуль Multiprocessing (стандартная библиотека Python) выполняет функцию распределения полученных данных на процессы, поэтому все данные, которые нужны для выполнения функции, должны иметь возможность быть представлены в байтовом виде. Таким образом, если модель, загруженная

пользователем, не может быть переведена в байтовый вид, пользователь увидит сообщение об ошибке.

Далее проверяется функция load_testdata. Если метода нет, либо он содержит синтаксическую ошибку, выводится сообщение об ошибке. В данном методе должна быть описана переменная self.testdata. Если она будет описана неправильно, сообщение об ошибке выведет проверка метода test_model.

Следующая на очереди – проверка метода get_tested_values, которая также вернет ошибку при наличии синтаксической ошибки либо отсутствии данного метода.

После этого происходит проверка метода set_tested_values. Программа уведомит пользователя об ошибке, если метод не сможет подставить в модель данные, которые вернул метод get_tested_values.

Последняя проверка - работа метода test_model. Его задача - подставить модель из метода set_tested_values и данных из метода load_testdata, и вернуть ошибку, если на каком-либо из этих этапов произойдет сбой.

Если ошибок не обнаружено, происходит загрузка виджета настройки конфигурации тестов. Пользователь выбирает количество используемых ядер и алгоритм распределения данных, а также задает настройки списка значений для генерации ряда случайных чисел: начальное и конечное значения, шаг и число, отражающее количество тестов на каждом шаге.

Код генерации ряда случайных чисел следующий:

```
def update_config(self):
        numbers = [edit.text() for edit in self.line_edits]
        try:
            start = float(numbers[0])
            stop = float(numbers[1])
            step = float(numbers[2])
            ntes = int(numbers[3])
        except ValueError:
            self.setError('Wrong numbers')
            return
```

```
            try:
                if step <= 0:
                    raise ZeroDivisionError
            except ZeroDivisionError:
                self.setError('Step must be over 0')
                return
            if start >= stop:
                self.setError('Start must be less than Stop')
                return
            self.line_edits[3].setText(str(ntes))
            if ntes < 1:
                self.setError('NTES must be over 0')
                return
            if (stop - start) / step * ntes > 100000:
                self.setError('Too large amount of data')
                return
            self.config = np.arange(start, stop, step)
            self.config = np.around(self.config, 6)
            if ntes > 1:
                self.config = np.repeat(self.config, ntes)
            if len(self.config) > 8:
                self.preview.setText('[' + ', '.join(list(map(str,
self.config[:3]))) + ', ..., ' + ', '.join(list(map(str,
self.config[-3:]))) + '] | ' + str(len(self.config)) + ' values')
            else:
                self.preview.setText('[' + ', '.join(list(map(str,
self.config))) + ']')
            self.config_changed.emit()
```

На данном этапе происходит проверка введенных пользователем данных. Возможно возникновение следующих ошибок:

- введены не числа;
- конечное число ряда меньше, чем начальное;
- шаг меньше или равен 0;
- количество тестов меньше 1.

Хранением и выполнением алгоритмов генерации случайных чисел занимается класс GeneratingRandomAlgorithms. Его код выглядит следующим образом:

```
class GeneratingRandomAlgorithms():
    names = [
        'normal',
        'uniform',
        'poisson',
    ]

    def __init__(self, algorithm='normal'):
        if not self.set_algorithm(algorithm):
            self.algorithm = 'normal'
            self.value_name = 'sigma'

    def set_algorithm(self, algorithm):
        if algorithm in self.names:
            self.algorithm = algorithm
            if algorithm == 'normal':
                self.value_name = 'sigma'
            elif algorithm == 'uniform':
                self.value_name = 'deviation'
            elif algorithm == 'poisson':
                self.value_name = 'lambda'
            return True
        else:
            return False

    def generate(self, weight, value):
        if self.algorithm == 'normal':
            return self._random_numpy_normal(weight, value)
        elif self.algorithm == 'uniform':
            return self._random_numpy_uniform(weight, value)
        elif self.algorithm == 'poisson':
```

```
                return self._random_numpy_poisson(weight, value)
        def _random_numpy_normal(self, weight, sigma):
            return [np.random.normal(array, sigma) for array in
weight]
        def _random_numpy_uniform(self, weight, deviation):
            def generate_number(number):
                return number + np.random.uniform(-deviation,
deviation, size=1)
            return [generate_number(array) for array in weight]

        def _random_numpy_poisson(self, weight, _lambda):
            def generate_number(number):
                return number + np.random.poisson(_lambda)
            return [generate_number(array) for array in weight]
```

В зависимости от выбранного алгоритма (Normal, Uniform, Poisson), будет сохранена информация о названии параметра данного алгоритма (sigma, deviation, lambda соответственно) и выполнен соответствующий метод (_random_numpy_normal, _random_numpy_uniform, _random_numpy_poisson соответственно).

Далее пользователь запускает процесс тестирования, что вызывает загрузку виджета прогресса тестирования и старт самого тестирования. Происходит процесс передачи конфигурации данных в класс MultiprocessExecution. Код данного класса представлен ниже:

```
    class MultiprocessExecution():
        def __init__(self, data, neuralconfig, shm_name):
            self.data = data
            self.neuralconfig = neuralconfig
            self.shm_name = shm_name

        def _create_new_model(self, value):
            new_weights =
self.data['algorithm'].generate(self.neuralconfig._basedata, value)
```

```
            return self.neuralconfig.set_tested_values(new_weights)

        def _compute_models(self, values):
            data = list()
            shm = shared_memory.ShareableList(name=self.shm_name)
            for value in values:
                model = self._create_new_model(value)
                data.append((value,
self.neuralconfig.test_model(model, self.neuralconfig.testdata)))
                shm[0] += 1
            return data

        def multiprocessingCalc(self):
            values = self.data['config']
            cpu_count = self.data['processors']
            process_step = ceil(len(values) / cpu_count)
            list_in_data = list()
            for i in range(ceil(len(values) / process_step)):
                end = i * process_step + process_step if i +
process_step <= len(values) else len(values)
                start = i * process_step
                list_in_data.append(values[start:end])

            with
multiprocessing.get_context("spawn").Pool(cpu_count) as p:
                data = p.map(self._compute_models, list_in_data)

            return data
```

Поступившие данные конфигурации делятся на части, равные количеству ядер, которые были заданы пользователем, с последующим запуском функции _compute_models для каждой из этих частей. Каждый раз, когда функция _compute_models выполняет расчет для новой модели, она прибавляет 1 к общему прогрессу рассчитанных моделей, что в последующем можно использовать для отображения прогресса пользователю. Для этого используется класс ShareableList

библиотеки Multiprocessing. Он позволяет иметь доступ к одной ячейке памяти для всех процессов, которые в данный момент выполняются программой.

Каждый процесс для каждого числа берет следующее число из списка данных, выполняет функцию генерации новых значений модели, тем самым создавая новую модель. Происходит расчет модели на тестовых данных, заданных пользователем, и сохранение пары «значение-результат».

После окончания работы всех процессов все данные собираются в общий список и передаются в виджет отображения результатов тестирования.

При создании данного виджета происходит сбор и расчет данных для построения графика – результата тестирования. Код расчета данных для построения графика следующий:

```
    def getXY_values(self):
            XY_data = {}
            for lst in self.data:
                for sigma_result in lst:
                    if sigma_result[0] in XY_data:

XY_data[sigma_result[0]].append(sigma_result[1])
                    else:
                        XY_data[sigma_result[0]] =
[sigma_result[1]]
                del(sigma_result)
            del(lst)
            for key in XY_data.keys():
                # counting Y
                XY_data[key] = np.average(XY_data[key])

            self.data = XY_data
```

В случаях, когда пользователем было указано количество тестов больше 1, при расчете координат точек графика, каждый полученный результат в рамках

одного диапазона данных усредняется при помощи операции нахождения среднего арифметического.

Отображение полученного графика в окне программы реализовано средствами библиотеки Matplotlib:

```
    def getGraph(self):
            matplotlib.use('Qt5Agg')
           sc = MplCanvas(self, width=5, height=5, dpi=100)
            sc.axes.plot(list(self.data.keys()),
list(self.data.values()))
    sc.axes.set_xlabel(self.algorithm.value_name.capitalize() + '
(numpy.' + self.algorithm.algorithm + ')')
            sc.axes.set_ylabel('Test results')
            sc.axes.set_title(self.name + ' | ' + self.test_time)
            toolbar = NavigationToolbar2QT(sc, self)
            sc.fig.tight_layout()
            return toolbar, sc
```

В данном коде происходит создание графика. Параллельно оси X будет отображаться название алгоритма, при помощи которого была осуществлена генерация случайных чисел с соответствующим названием параметра этого алгоритма. Название графика отображается вверху графика и представляет собой название модели, заданное пользователем, а также включает в себя дату и время окончания тестирования.

Далее пользователь может либо сохранить график в виде изображения, либо вывести данные графика в текстовом виде или в формате .xlsx, либо продолжить работу с программой с другой конфигурацией теста. Это реализовано при помощи следующего кода:

```
    def save(self):
            if self._save_type == 'excel':
                self._save_to_excel()
            elif self._save_type == 'txt':
```

```
                self._save_to_txt()

        def _save_to_excel(self):
            wb = openpyxl.Workbook()
            ws = wb.worksheets[0]
            ws.cell(row=1, column=1).value = 'Name of model: ' +
self._data['name']
            ws.cell(row=2, column=1).value = 'Tested: ' +
self._data['time']
            ws.cell(row=3, column=1).value = 'Your model tested
value: ' + str(self._data['test_value'])
            ws.cell(row=4, column=1).value = 'Generating random
algorithm: numpy.' + self._data['algorithm'].algorithm
            ws.cell(row=6, column=1).value =
self._data['algorithm'].value_name.capitalize()
            ws.cell(row=6, column=2).value = 'Tested value'
            for row, key in
enumerate(list(self._data['data'].keys())):
                ws.cell(row=row + 7, column=1).value = key
                ws.cell(row=row + 7, column=2).value =
self._data['data'][key]

            filepath = QFileDialog.getSaveFileName(caption='Save
your test result to excel file',
                                                   directory=f'./',
                                                   filter='Excel
file (*.xlsx)')
            wb.save(filepath[0])

        def _save_to_txt(self):
            filepath = QFileDialog.getSaveFileName(caption='Save
your test result to txt file',
                                                   directory=f'./',
                                                   filter='Text
file (*.txt)')
            with open(filepath[0], 'w') as f:
```

```
                f.write('Name of model: ' + self._data['name'] +
'\n')
                f.write('Tested: ' + self._data['time'] + '\n')
                f.write('Your model tested value: ' +
str(self._data['test_value']) + '\n')
                f.write('Generating random algorithm: numpy.' +
self._data['algorithm'].algorithm + '\n\n')
    f.write(f'{self._data["algorithm"].value_name.capitalize()}\tTe
sted value\n')
                for key, value in self._data['data'].items():
                    f.write(f'{key}\t{value}\n')
```

Сохранение в формат .xlsx реализовано при помощи библиотеки Openpyxl. При сохранении в этот формат в файл сохраняются следующие данные:

- название модели, заданное пользователем (если название не задано, по умолчанию модели будет присвоено название «NoName»);

- дата и время окончания тестирования;

- значение, которое вернул метод test_model, на неизмененной модели, загруженной методом load_model, и тестовых данных, загруженных методом load_testdata;

- выбранный пользователем алгоритм распределения данных;

- список данных, полученных в результате выполнения тестирования, в виде пары «значение-результат».

Аналогичным образом производится сохранение результатов тестирования в текстовый файл.

Далее пользователю дается возможность выполнить повторное тестирование с другой конфигурацией над той же моделью.

4 Тестирование

4.1 Тест общей работы модуля

Для тестирования работы программы используются методы библиотеки Tensorflow. Была создана модель – классификатор стандартного датасета библиотеки Tensorflow – Fashion-MNIST.

Работа начинается с подключения разработанного модуля и описания дочернего от этого модуля класса. Тест-кейсом будет проверка загрузки модели Fashion-MNIST.

```
class NeuralCrashTest(NeuralCrash):
    def __init__(self):
        super().__init__()

    def load_model(self):
        self.model = load_model('../2 II
Creation/fashion_mnist.h5')

    def load_testdata(self):
        (_, _), (testX, testY) = fashion_mnist.load_data()
        testX = testX.reshape(testX.shape[0], 784) / 255
        self.testdata = testX, testY

    def get_tested_values(self):
        return self.model.get_weights()

    def set_tested_values(self, values):
        new_model = self.model
        new_model.set_weights(values)
        return new_model

    def test_model(self, model, data):
        results = model.evaluate(data[0], data[1],
batch_size=64)
```

```
            return results[-1]
```

В данном коде происходит создание дочернего класса NeuralCrashTest от родительского класса NeuralCrash. В данном классе описаны пять методов, требующиеся для корректной работы программы, и один метод инициализирующий код создания объекта класса NeuralCrash.

После вызова метода run у родительского класса происходит создание интерфейса с проверкой написанных ранее методов.

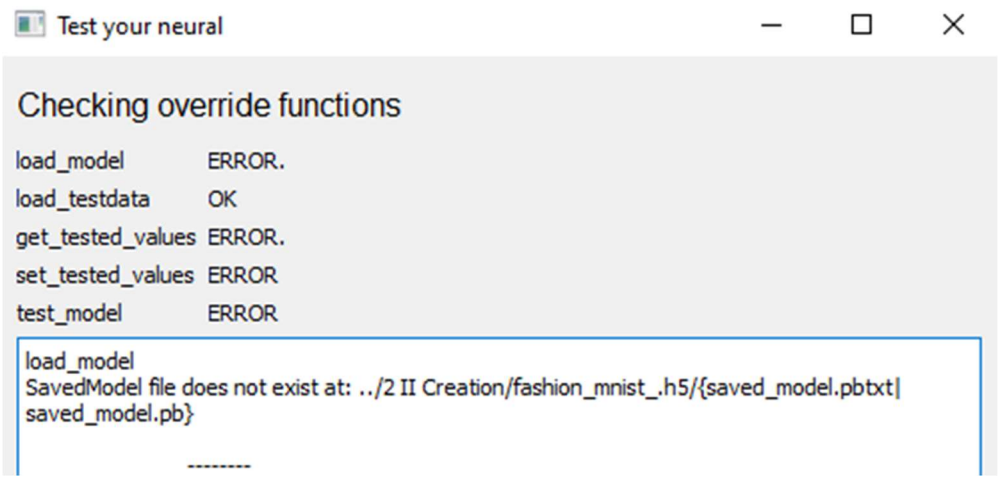
Попробуем специально исказить написанные методы, чтобы произошло событие отображения ошибок.

```
        def load_model(self):
            self.model = load_model('../2 II
Creation/fashion_mnist.h5')
```

В вышеприведенном коде попробуем заменить путь к модели на следующий:

```
        def load_model(self):
            self.model = load_model('../2 II
Creation/fashion_mnist_notModel.h5')
```

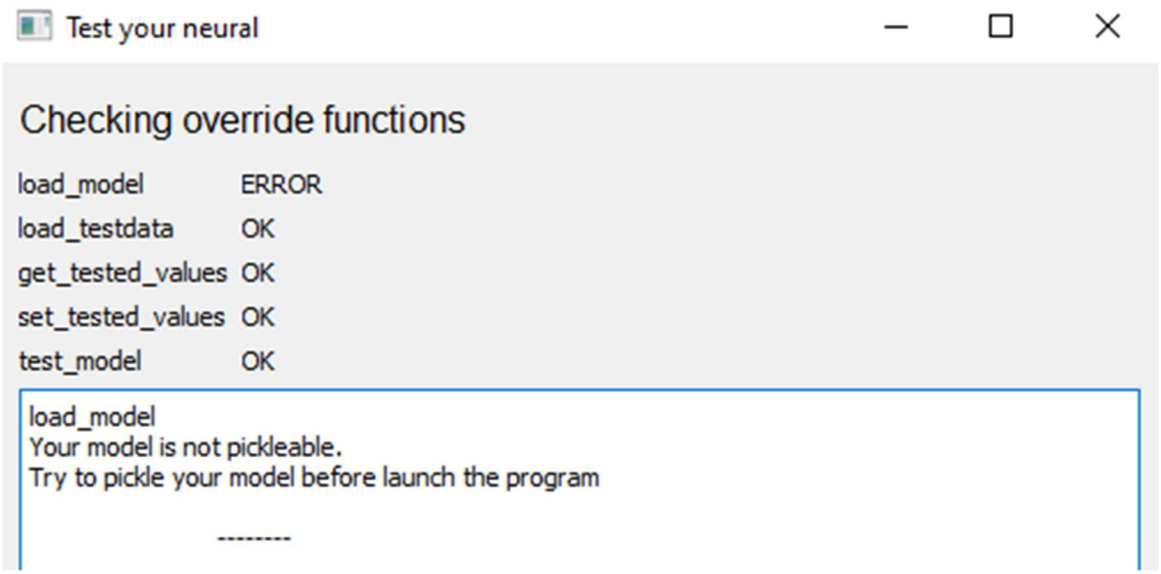
Класс проверки корректно обработал ошибку, что отображено на рисунке…



Рисунок… - Виджет обработки ошибок

Данное окно (рис…) показывает, что пользователь совершил ошибку в описании метода load_model, что повело за собой появление ошибок и в других методах, которые прямо зависят от load_model (get_tested_values, set_tested_values, test_model).

Далее необходимо проверить еще один важный момент. Модель должна иметь возможность быть приведена в двоичный вид библиотекой pickle. Если данной возможности нет, то данный виджет обработки ошибок также предупредит об этом пользователя. Это отображено на рисунке…



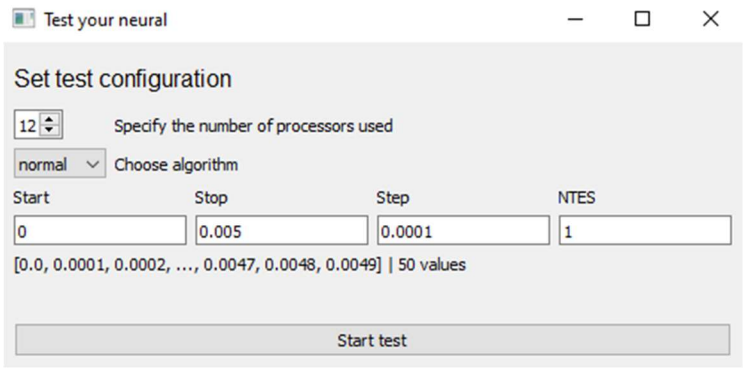
Рисунок… - Обработка ошибки метода pickle

В данном случае, чтобы исправить данную ошибку, необходимо для библиотеки tensorflow, модель которого мы пытаемся загрузить, переописать методы deserialize и serialize. Это приведено в коде ниже:

```
def unpack(model, training_config, weights):
  restored_model = deserialize(model)
  if training_config is not None:
    restored_model.compile(
      **saving_utils.compile_args_from_training_config(
        training_config))
```

```
    restored_model.set_weights(weights)
    return restored_model
  def make_keras_picklable():
    def __reduce__(self):
      model_metadata = saving_utils.model_metadata(self)
      training_config=model_metadata.get("training_config", None)
      model, weights = serialize(self), self.get_weights()
      return (unpack, (model, training_config, weights))
    cls, cls.__reduce__ = Model, reduce__
  make_keras_picklable()
```

После вызова данного кода, модель стало возможно использовать в программе. После успешной проверки всех остальных методов происходит отображения виджета настройки конфигурации теста. Его внешний вид показан на рисунке…



Рисунок… - Виджет настройки конфигурации теста

При создании данного виджета (рис…) во все элементы управления расставляются значения по умолчанию:

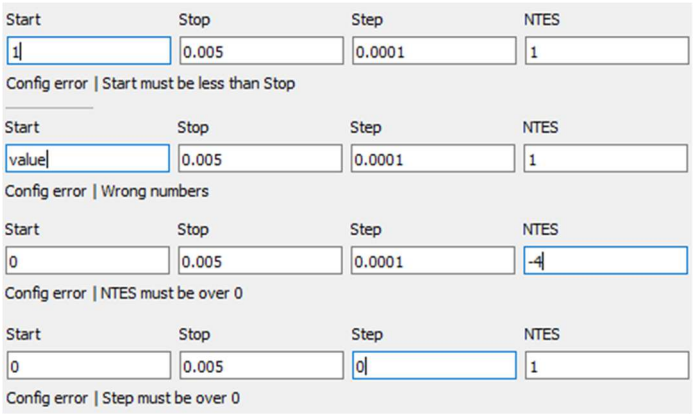
- количество ядер процессора равно максимальному значению, определенного системой;

- алгоритм рандомизации данных - normal;

- старт, стоп, шаг и количество тестов - 0, 0.005, 0.0001, 1 соответственно.

Каждое изменение значений списка данных для теста повлечет за собой обработку события update_config, которое проверит возможность создания списка с данными параметрами, а также отобразит ошибку в противном случае.

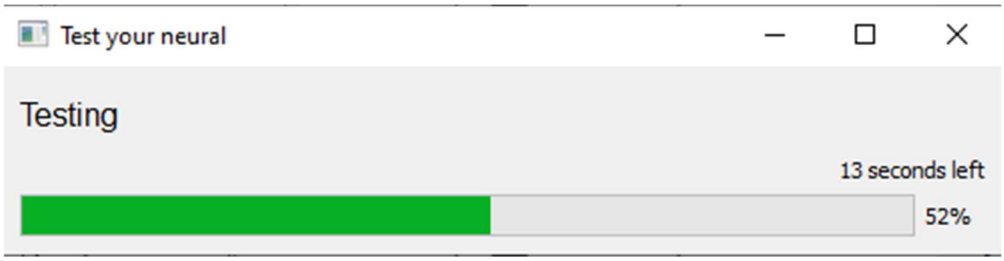
Примеры ошибок, которые может совершить пользователь при конфигурации списка, представлены на рисунке …



Рисунок… - Обработка ошибок при конфигурации списка входных данных теста

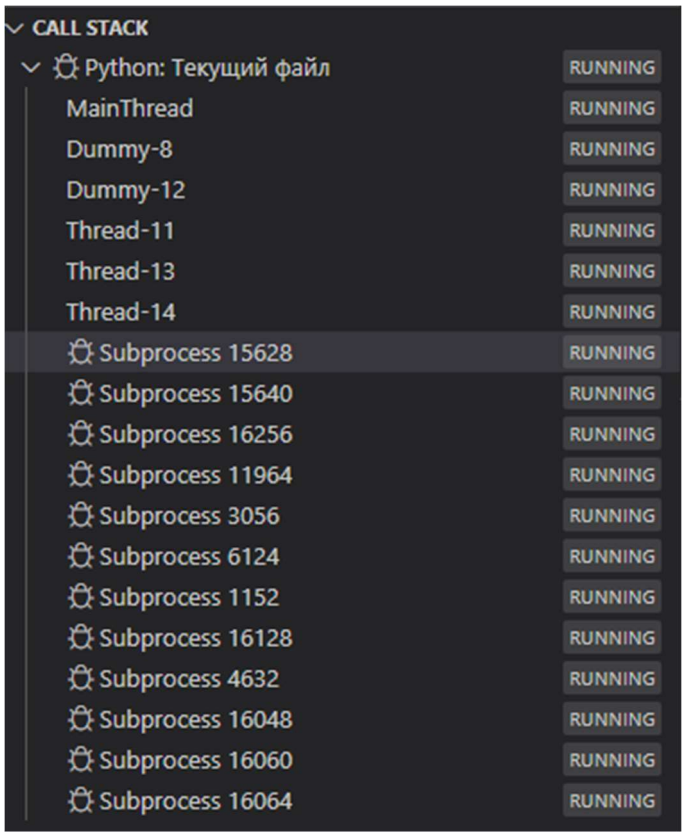
Далее происходит процесс запуска теста, где задача разбивается на подзадачи и отправляется на выполнение каждая в свой процесс. Количество таких подзадач пользователь задал в элементе управления для выбора количества ядер процессора.

Пользователю отображается виджет отображения прогресса тестирования (рис…) с процентным индикатором выполнения общей задачи, а также примерное время до конца тестирования.

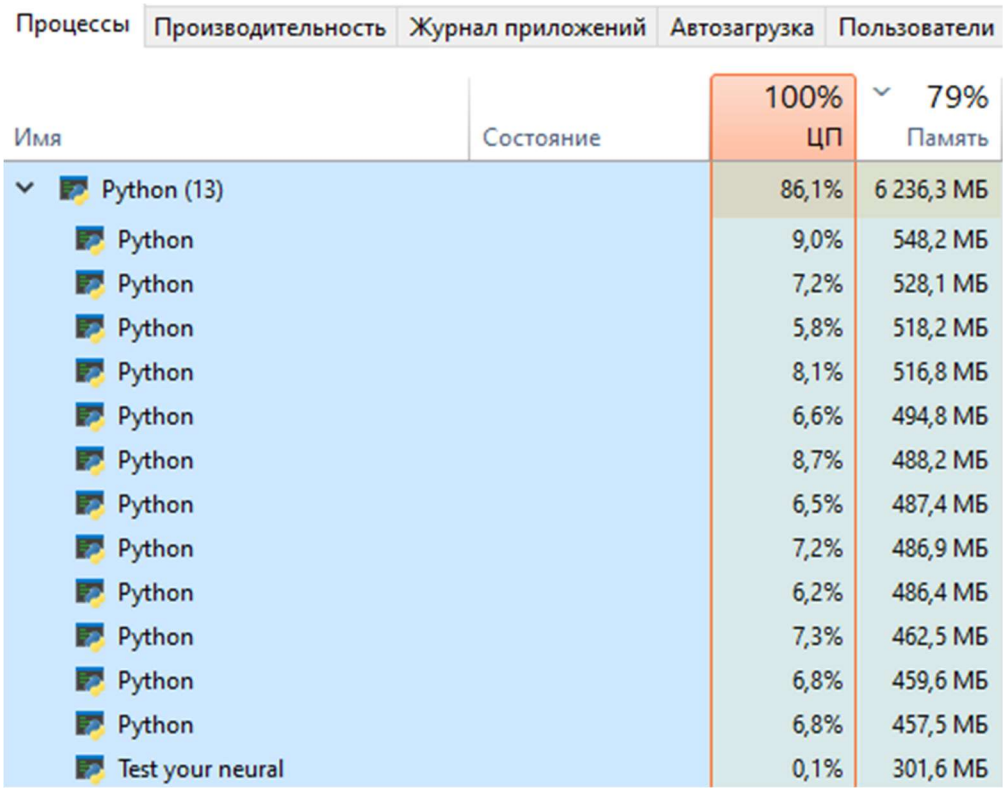


Рисунок… - виджет отображения прогресса тестирования

Работу программы на данном этапе можно отследить отладчиком кода, либо с помощью диспетчера задач Windows.



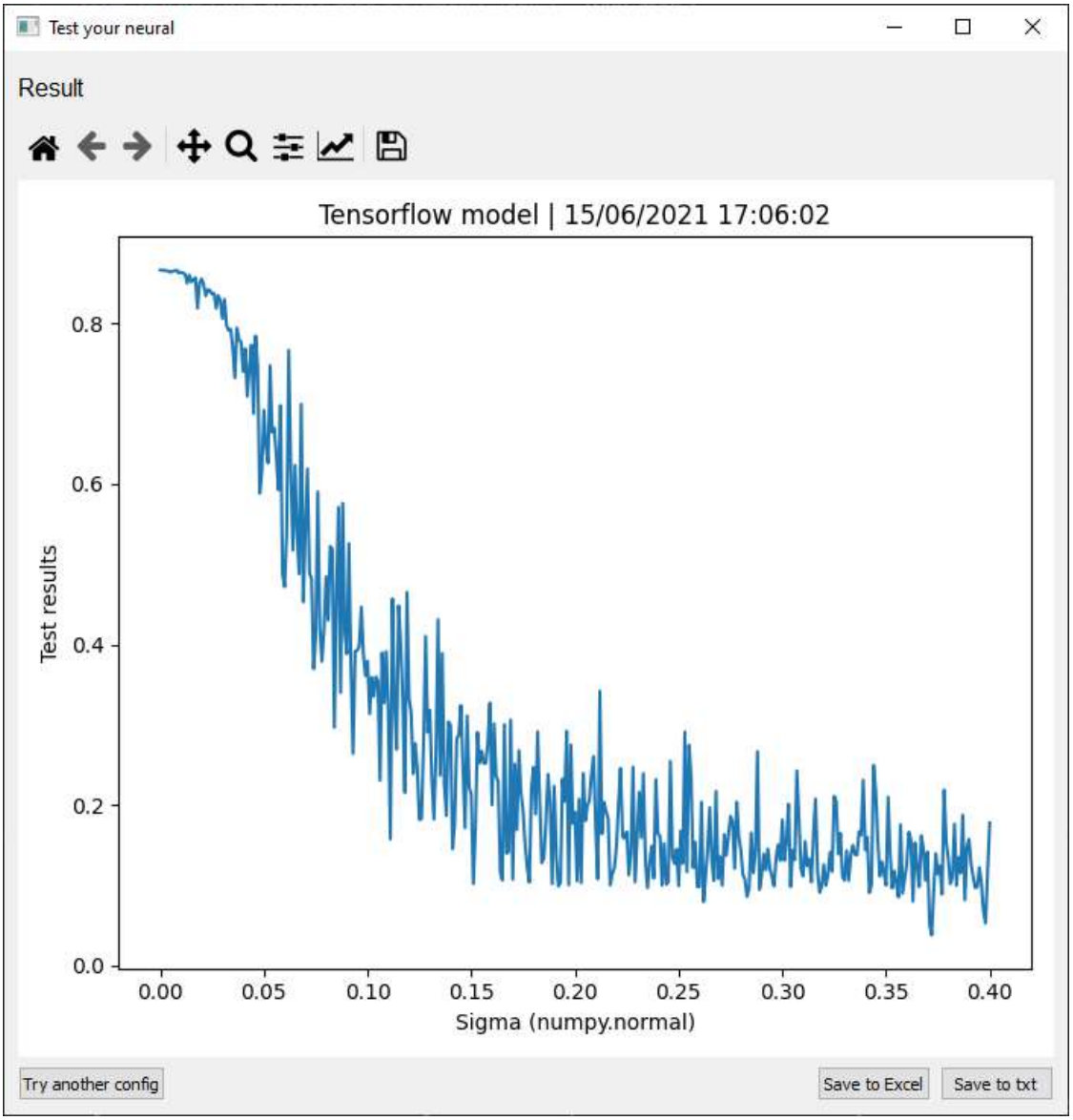
Рисунок… - Отображение созданных процессов в отладчике кода VS Code



Рисунок… - Отображение созданных процессов в диспетчере задач Windows

Из рисунков … и … можно заметить, что тестирование запустилось на 12-ти ядрах процессора и заняло работу процессора системы на 100%. Из этого можно сделать вывод о том, что программа разделила задачу на подзадачи, количество которых было сконфигурировано пользователем.

После тестирования на экране отображается виджет результатов тестирования (рис…).

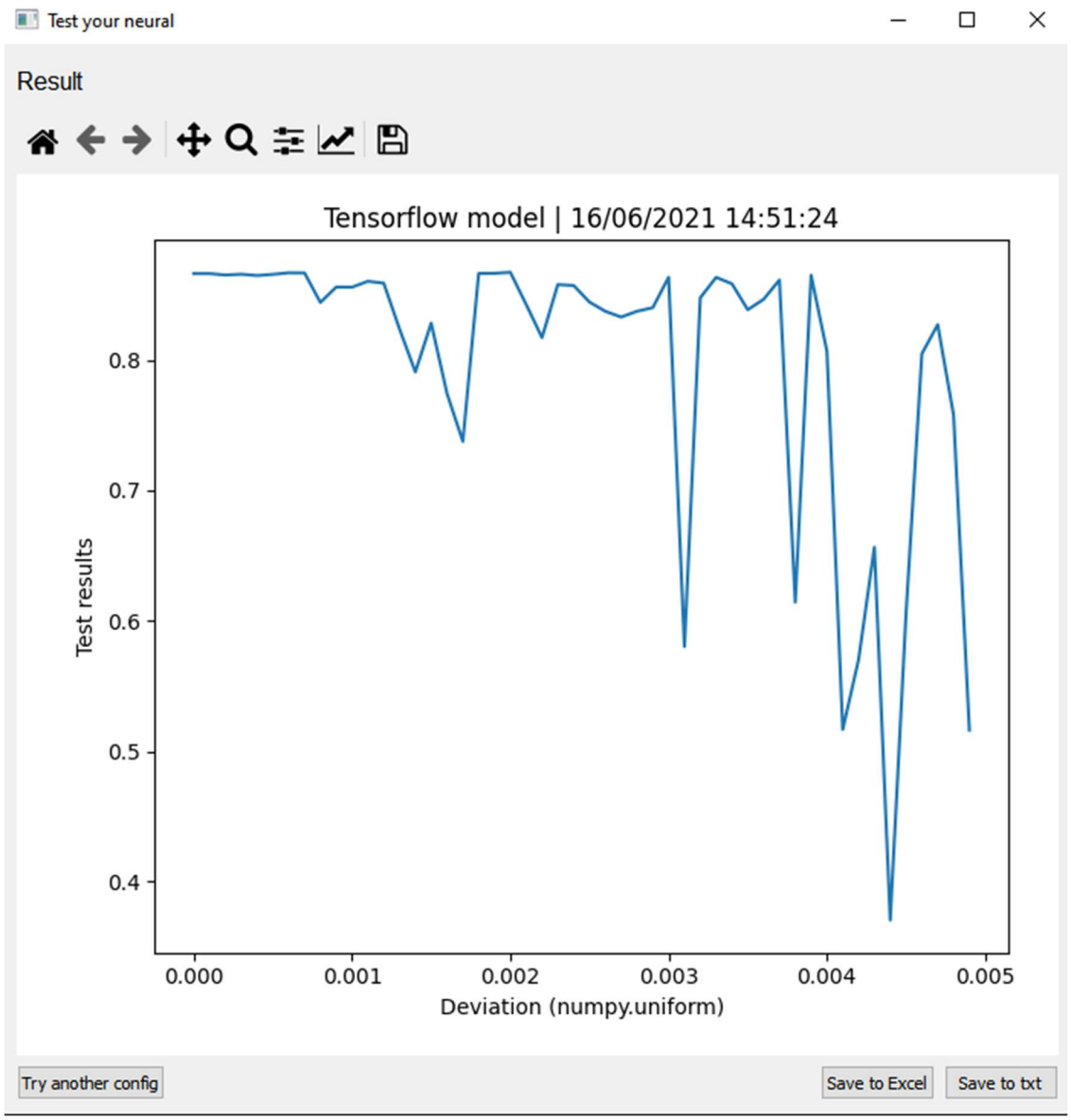


Рисунок… - Виджет отображения результатов тестирования

Как видно из рисунка… , модель с названием «Tensorflow model» была успешно протестирована (рядом верно установлена дата окончания тестирования) с обработкой значений алгоритмом numpy.normal. На графике по оси X отражается

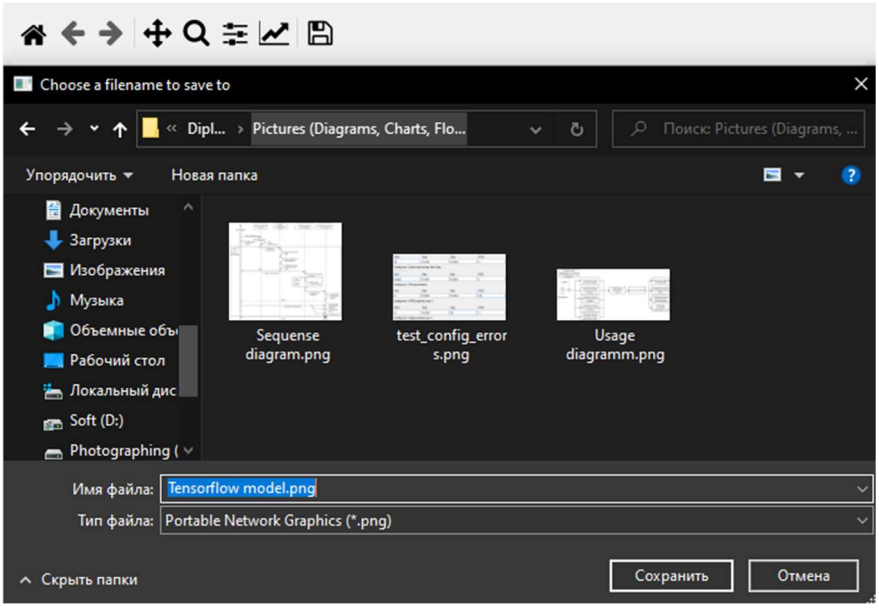
значение sigma, а по оси Y – рассчитанные точности созданных во время тестирования моделей.

Необходимо проверить работу и других алгоритмов. Было проведено тестирования работы алгоритма uniform с параметром deviation. Необходимо задать новую конфигурацию теста. Это можно сделать после нажатия на кнопку Try another config. После нажатия на кнопку снова произойдёт создания виджета настройки конфигурации без необходимости перезапускать программу или переописывать методы. Далее необходимо ввести новые параметры тестирования и запустить тест. Результат появится на следующим виджете при завершении тестирования (рис…).

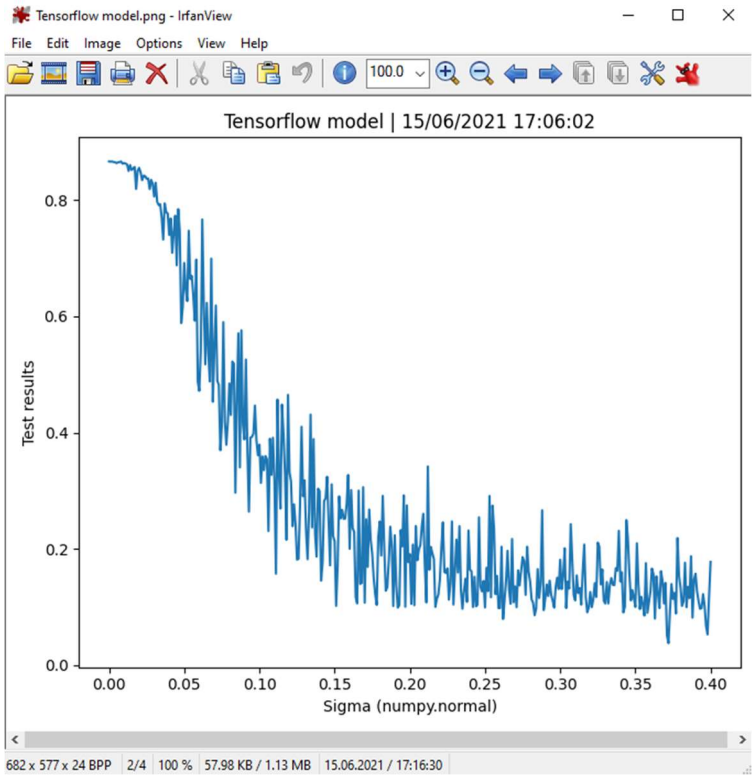


Рисунок… - Результат тестирования алгоритма uniform

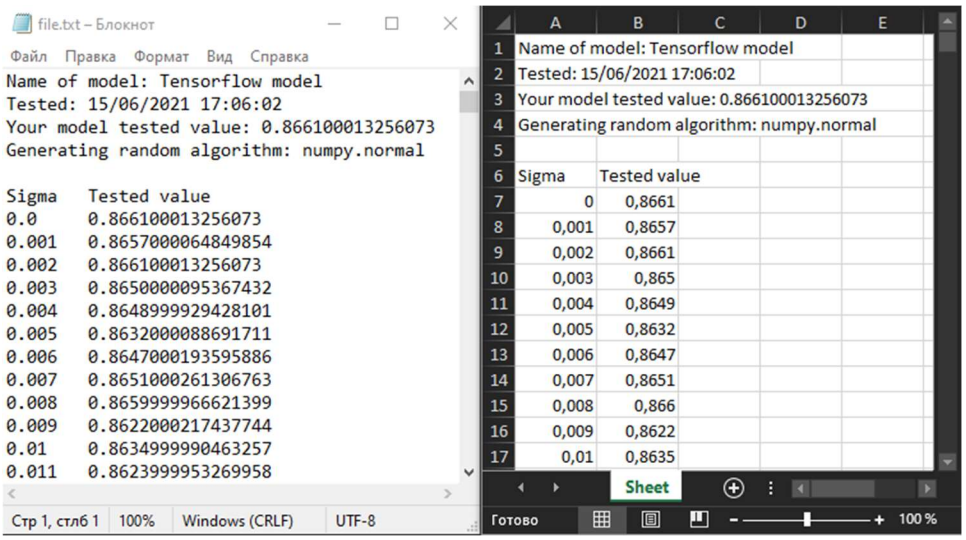
Далее модель можно сохранить в виде изображения (рис… и…) или в файлы excel и txt (рис…).



Рисунок… - Выбор места для сохранения графика в виде изображения



Рисунок… - Сохранённый график



а) б)

Рисунок… - Сохранение данных графиков: а – сохранение в txt;
б – сохранение в excel

4.2 Анализ производительности в зависимости от входных данных и аппаратных данных устройства

После проверки корректности работы данного модуля необходимо провести тестирование на различных аппаратных конфигурациях системы, а также на нескольких моделях. Для тестирования было выбрано два компьютера, работающих под управлением операционной системы Windows 10.

Компьютер с объёмом оперативной памяти 16 Гб и частотой 3200 МГц и процессором AMD Ryzen 5 3600, имеющим тактовую частоту 3,6 ГГц и 6 физических (12 виртуальных) ядер (в дальнейшем Компьютер).

Ноутбук с объёмом оперативной памяти 8 Гб и частотой 2400 МГц и процессором Intel® Core(TM) i5-7200U, имеющим тактовую частоту 2,5 ГГц и 4 физических ядра (в дальнейшем Ноутбук).

Будет протестировано две модели нейронной сети:

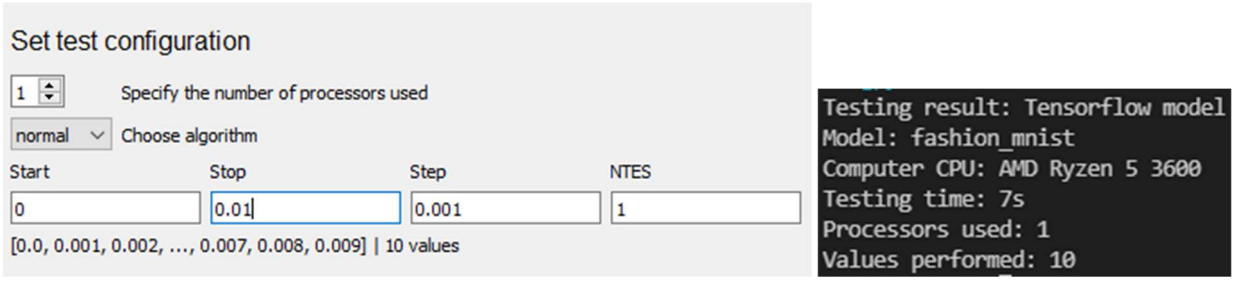
- модель Fashion_MNIST с двумя слоями: 800 нейронов на входном слое и 10 нейронов - на выходном (в дальнейшем Fashion_mnist);

- модель Fashion_MNIST, имеющая 4 слоя: 800 нейронов на входном слое, 2 скрытых слоя, по 1000 нейронов в каждом, и выходной слой, состоящий из 10 нейронов (в дальнейшем Fashion_mnist_big).

Алгоритм будет всегда установлен normal (так как это не повлияет на процесс тестирования). Время выполнения каждого алгоритма различается несущественно и не сильно скажется на результатах тестирования.

4.2.1 Тестирование работы модуля на Компьютере

Первый тест будет задействовать всего одно ядро системы и иметь 10 входных данных для тестирования Fashion_mnist (рис…).



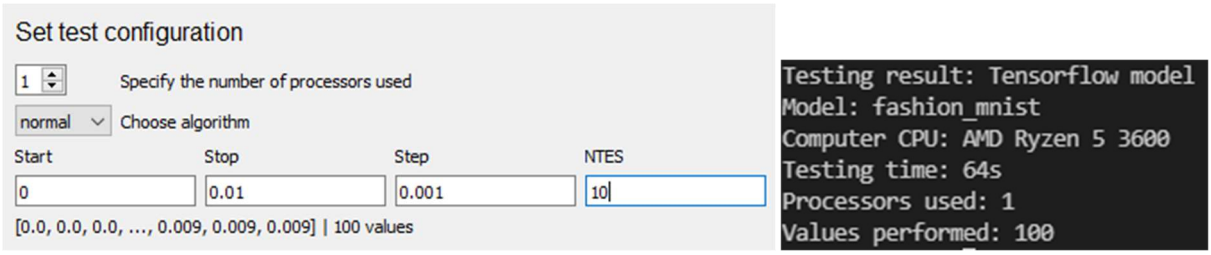
а) б)

Рисунок… - Конфигурация теста: а – конфигурация теста;

б – результат тестирования

Из результатов тестирования видно, что Компьютер на одном ядре обработал всю поставленную задачу за 7 секунд.

Второй тест будет задействовать также одно ядро, но количество входных данных вырастет до 100 для модели Fashion_mnist (рис…).



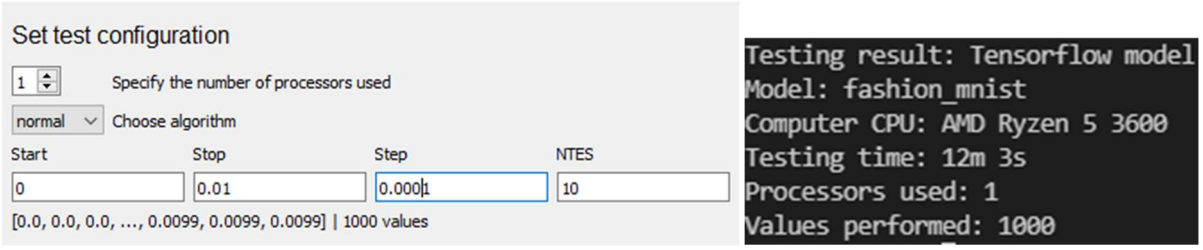
а) б)

Рисунок… - Конфигурация теста: а – конфигурация теста;

б – результат тестирования

В результате этого теста время тестирования выросло до 64 секунд.

Третьим тестом будет 1000 значений тестирования на одном ядре и на Fashion_mnist.



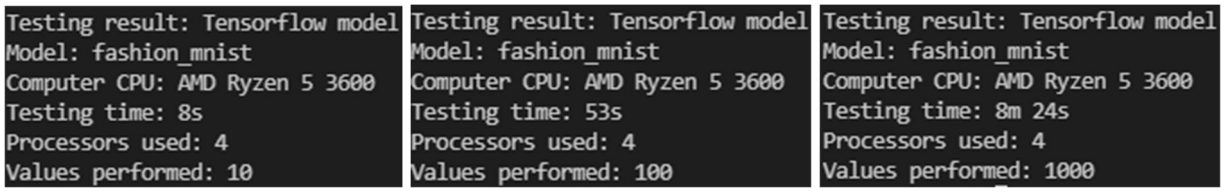
а) б)

Рисунок… - Конфигурация теста: а – конфигурация теста;

б – результат тестирования

В результате этого теста время выросло до 12-ти минут.

На рисунках ….а, ….б и ….в представлены аналогичные тесты для 4 используемых ядер.

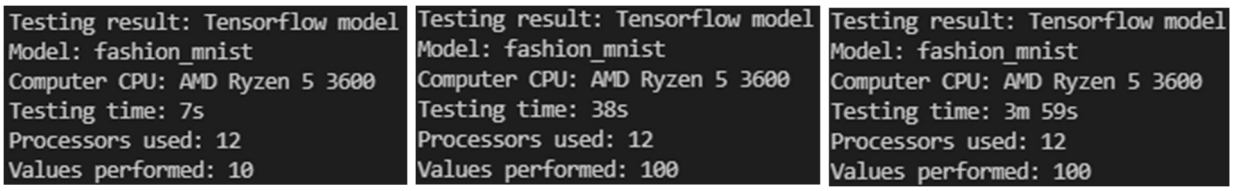


а) б) в)

Рисунок… - Тестирование Fashion_mnist: а – 10 тестовых данных;

б – 100 тестовых данных; в – 1000 тестовых данных

Далее необходимо проверить работу программы на системе, нагрузив процессор на 100%, чтобы увеличить скорость обработки тестов. На рисунках ….а, ….б и ….в представлены аналогичные тесты для максимального количества используемых ядер в системе Компьютер.



а) б) в)

Рисунок… - Тестирование Fashion_mnist: а – 10 тестовых данных;

б – 100 тестовых данных; в – 1000 тестовых данных

В результате проведённых тестов можно построить трёхмерный график зависимости времени тестирования от количества используемых при тестировании ядер и количества значений, которые нужно будет обработать (таб…).

Таблица … - Зависимость времени тестирования от ядер и количества операций

Ядра / Значения	1	4	12
10	7	8	7
100	64	53	38
1000	723	504	239

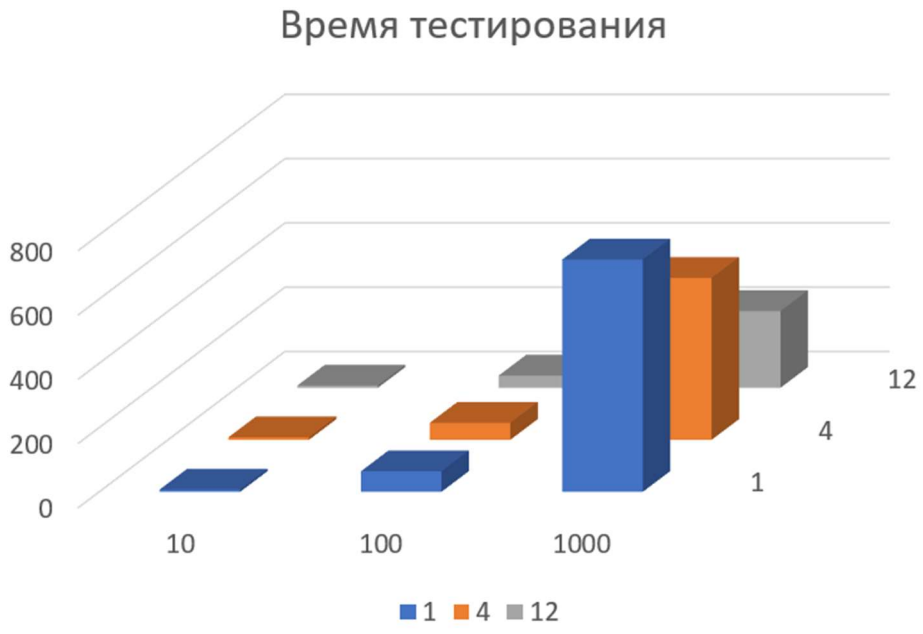


Рисунок … - График времени тестирования

Далее будут приведены аналогичные тесты для модели Fashion_mnist_big. Данные тестирования отображены в таблице ….

Таблица … - Зависимость времени тестирования от ядер и количества операций

Ядра / Значения	1	4	12
10	16	13	12
100	102	87	56
1000	2152	1628	890

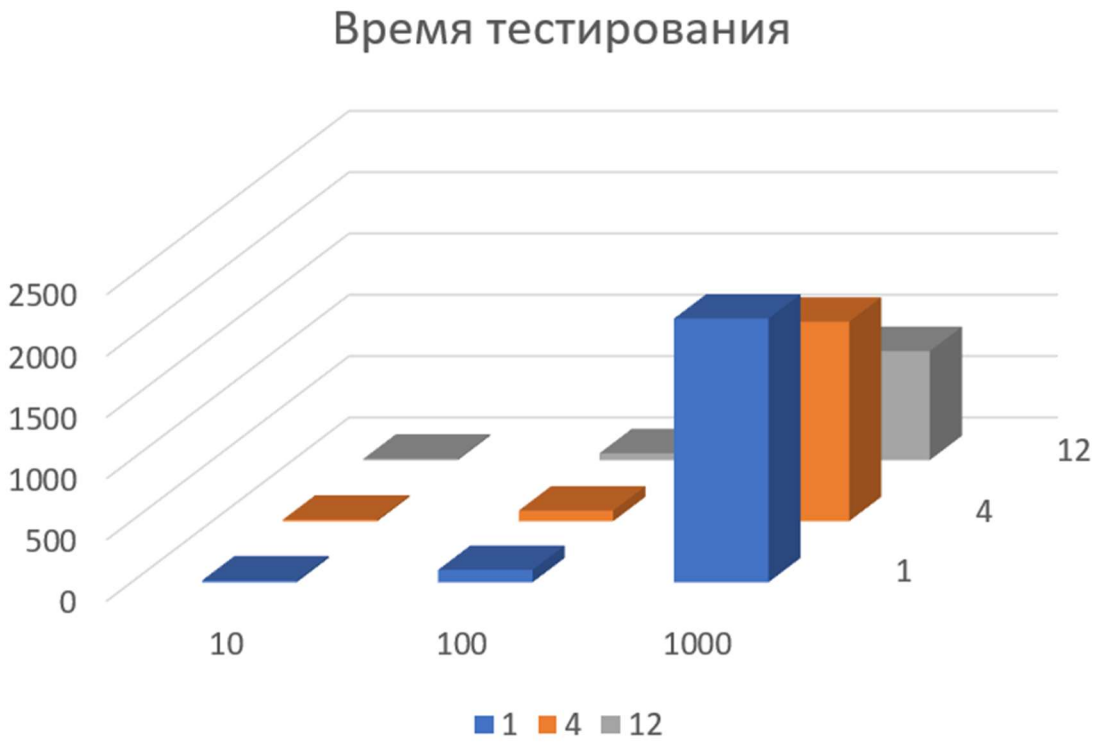


Рисунок … - График времени тестирования

Тестирование показало, что при использовании одноядерного решения поставленной задачи требуется в среднем на 150% больше времени, чем при разделении задачи на подзадачи и использовании системы на 100%.

Чтобы понять, насколько важно использовать многоядерную и мощную систему для тестирования нейронных сетей данным модулем, необходимо произвести аналогичные тесты на Ноутбуке – менее производительной системе.

4.2.2 Тестирование работы модуля на Ноутбуке

Тесты будут отличаться от прошлых только системой, на которой эти тесты будут выполняться и количеством используемых ядер (на Ноутбуке их 4). Вначале будет протестирована Fashion_mnist. Результаты представлены в таблице … и на рисунке ….

Таблица … - Зависимость времени тестирования от ядер и количества операций

Ядра / Значения	1	2	4
10	18	13	17
100	176	148	113
1000	2042	1593	1142

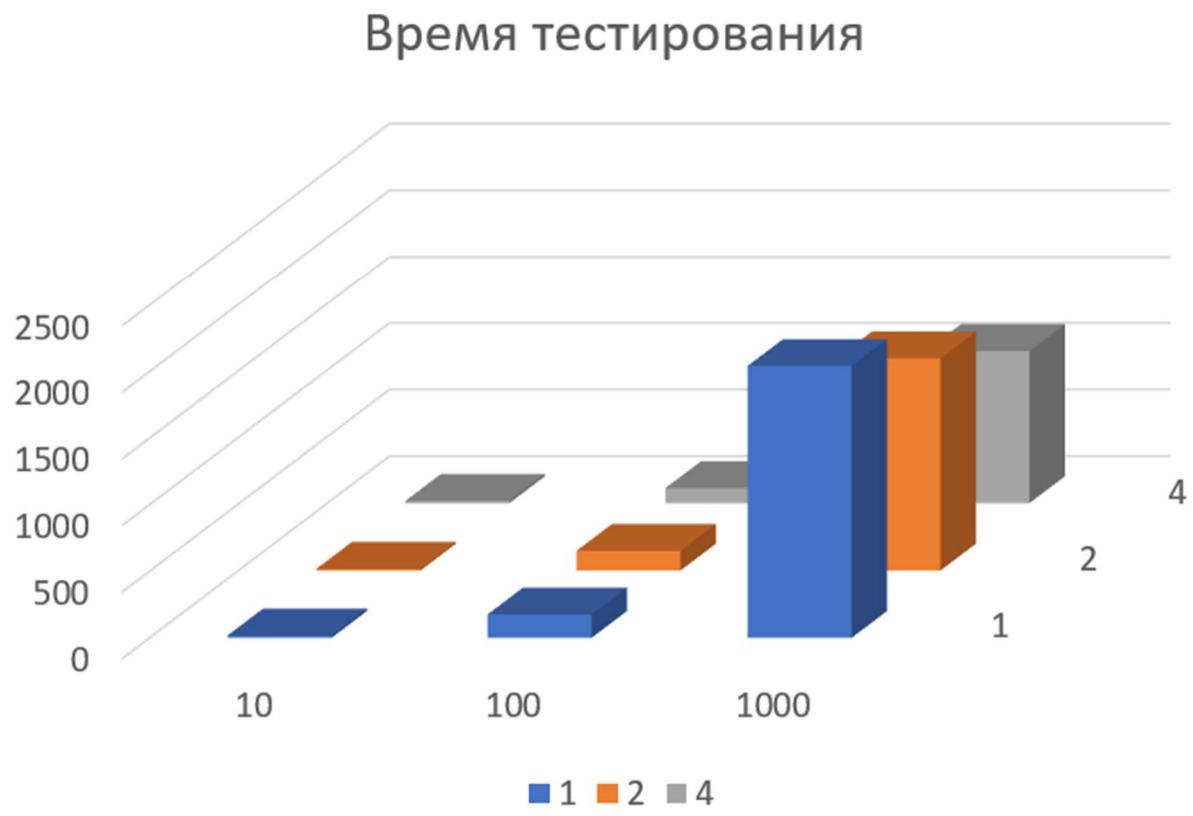


Рисунок … - График времени тестирования

Далее будет протестирована модель Fashion_mnist_big. Результаты представлены в таблице … и на рисунке ….

Таблица … - Зависимость времени тестирования от ядер и количества операций

Ядра / Значения	1	2	4
10	31	25	24
100	299	231	170
1000	3102	2615	2097

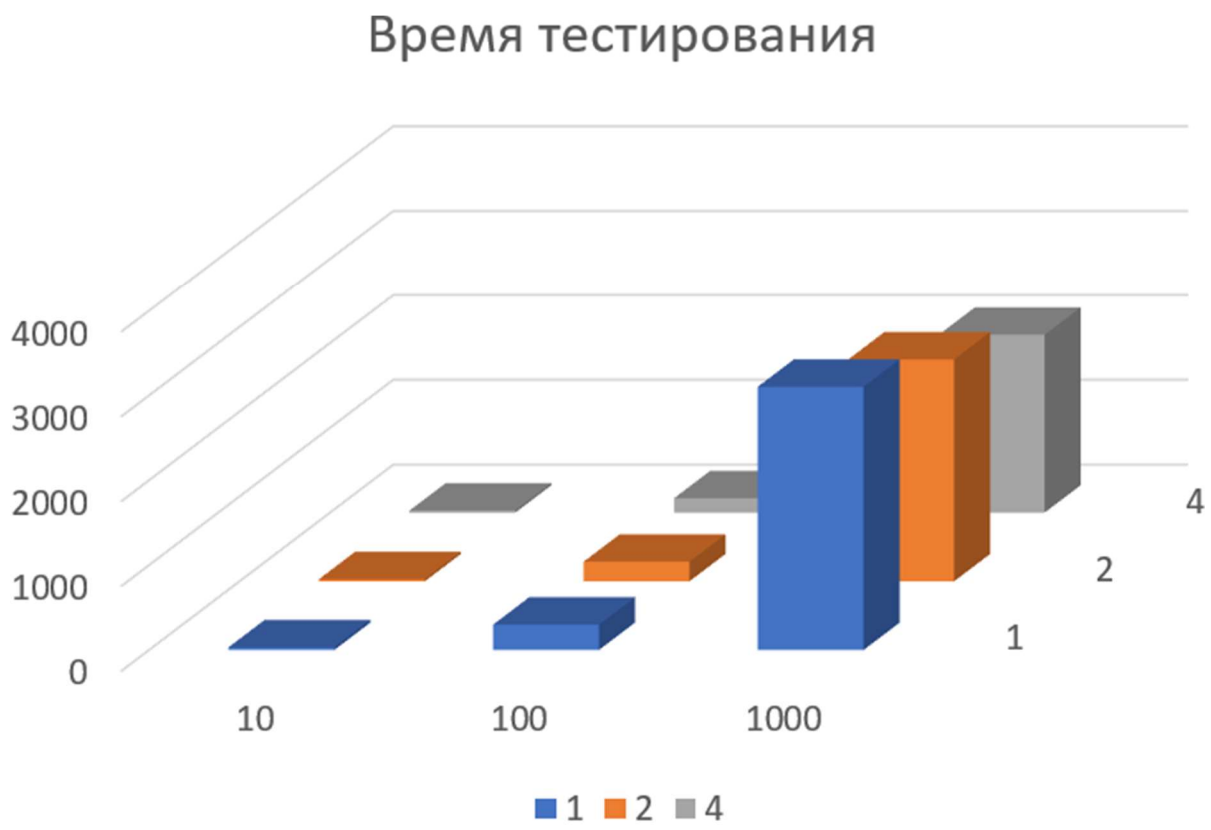


Рисунок … - График времени тестирования

4.2.3 Сравнение получившихся результатов

Все тесты проводились на системе, на которой не было запущено никаких других программных средств или утилит, кроме данной программы. Поэтому проведённые тесты можно назвать «чистыми» тестами.

Согласно проведённому тестированию, можно сделать вывод о том, что производительность системы сильно влияет на результаты. Чем больше ядер в системе, тем быстрее она может выполнить поставленную задачу, разбив её на множество подзадач.

От количества тестов зависит будущее время тестирования. Если таких тестов мало (в данном случае проверялось время выполнения 10-ти тестов), то время тестирования практически не изменяется, так как больше времени тратится на процесс разбития задачи на подзадачи. Но если увеличивать количество тестов, то прирост производительности увеличивается если увеличивать количество используемых ядер.

Также производительность данной программы напрямую зависит от системы, на которой производится тестирование. С увеличением аппаратных характеристик, таких как оперативная память (её объём и частота), процессор (его частота и количество физических ядер), также ускоряется процесс проведения тестов.

Таким образом, было проведено тестирование разработанной программы на двух системах: Компьютере и Ноутбуке. Программа корректно обрабатывает ошибки пользователя при описании дочернего класса программы, разбивает задачу на подзадачи, из-за чего сильно увеличивается производительность и уменьшается время тестирования.

ЗАКЛЮЧЕНИЕ

В данной работе приведено описание процесса разработки программы имитационного моделирования моделей машинного обучения.

Были выполнены следующие задачи, поставленные перед началом работы:

- произведен обзор предметной области;
- проанализированы методы ускорения процесса обработки нейросетей;
- …
- проанализировано изменение данных внутри нейросетей при различных отклонениях входных параметров, в результате были сделаны следующие выводы:

(тут выводы, которые можно сделать, исходя из тестирования)

Таким образом, задачи, поставленные на выпускную квалификационную работу, можно считать выполненными в полном объеме.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Акопов, А.С. Имитационное моделирование: учебник и практикум для академического бакалавриата / А. С. Акопов — М. : Издательство Юрайт, 2017. — 389 с.

2 Кобелев, Н.Б. Имитационное моделирование объектов с хаотическими факторами: Учебное пособие / Кобелев Н.Б. - М.: КУРС, НИЦ ИНФРА-М, 2016. - 192 с.

3 Боев, В.Д. Имитационное моделирование систем : учеб. пособие для прикладного бакалавриата // М. : Издательство Юрайт, 2017. 253 с.

4 Вьюненко, Л.Ф. Имитационное моделирование: учебник и практикум для академического бакалавриата / Л. Ф. Вьюненко, М. В. Михайлов,; под ред. Л. Ф. Вьюненко. — М. : Издательство Юрайт, 2017. — 283 с.

5 Кораблев, Ю.А. Имитационное моделирование: учебник / Ю.А. Кораблев. — Москва: КНОРУС, 2017. — 146 с.

6 Таненбаум, Э.С. Современные операционные системы: [пер. с англ.]. Питер, 2011. – 1115 с.

7 Флорес, А., Организация вычислительных машин, пер. с англ., М., 1972. – 634 с.

8 Каган, Б. М., Каневский, М. М., Цифровые вычислительные машины и системы, 2-е изд., М., 1973. – 347 с.

9 Справочник по цифровой вычислительной технике, под ред. Б. Н. Малиновского, К., 1974. – 638 с.

10 Калачев А. В. Многоядерные процессоры; Интернет-университет информационных технологий, Бином. Лаборатория знаний - М., 2015. - 248 с.

11 Янюшкин, В.В. Программные компоненты и архитектурные решения распределенных информационных систем на основе применения технологии cloud computing и WCF. / мат.межвуз.научн.техн.конф. – Новочеркасск: НВВКУС, 2009.- с.239-241.

12 Калачев, А.В. Многоядерные процессоры. Учебное пособие; Интернет-Университет Информационных Технологий (ИНТУИТ) - М., 2017. - 132 с.

13 Перепелкин, Е.Е. Вычисления на графических процессорах (GPU) в задачах математической и теоретической физики; Огни - Москва, 2017. - 750 с.

14 Фет, Я.И. Массовая обработка информации в специализированных однородных процессорах; Новосибирск: СО АН СССР - М., 2016. - 199 с.

15 Галушкин, А. И. Нейрокомпьютеры. Учебное пособие / А.И. Галушкин. - М.: Альянс, 2014. - 528 с.

16 Редько, В.Г. Эволюция, нейронные сети, интеллект: Модели и концепции эволюционной кибернетики / В.Г. Редько. - Москва: СИНТЕГ, 2017. - 224 с.

17 Рутковская, Д. Нейронные сети, генетические алгоритмы и нечеткие системы / Д. Рутковская, М. Пилиньский, Л. Рутковский. - М.: Горячая линия - Телеком, 2013. - 384 с.

18 Тархов, Д.А. Нейросетевые модели и алгоритмы. Справочник / Д.А. Тархов. - М.: Радиотехника, 2014. - 397 с.

19 Jesse, Russell Искусственная нейронная сеть / Jesse Russell. - М.: VSD, 2012. - 630 с.

20 Гелиг, А. Х. Введение в математическую теорию обучаемых распознающих систем и нейронных сетей. Учебное пособие / А.Х. Гелиг, А.С. Матвеев. - М.: Издательство СПбГУ, 2014. - 224 с.