МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования
«Гомельский государственный технический университет имени П.О. Сухого»

Факультет автоматизированных и информационных систем

Кафедра «Информационные системы»

направление специальности 1-40 05 01-01 «Информационные системы и технологии (в проектировании и производстве)»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к дипломной работе
на тему
«Программный комплекс для анализа и классификации сортов твердой пшеницы по электрофоретическому спектру глиадина на основе искусственных нейронных сетей с глубинным обучением»

Разработал студент гр. ИТ-41	_____ (подпись)	Гончаров П.В. (Ф.И.О.)
Руководитель работы	_____ (подпись)	доцент, к.ф.-м.н, Цитринов А.В. (учёное звание, учёная степень, Ф.И.О.)
Консультант по экономической части	_____ (подпись)	Астраханцев С.Е. (учёное звание, учёная степень, Ф.И.О.)
Консультант по охране труда, энергосбережению и технике безопасности	_____ (подпись)	Прищепов С.С. (учёное звание, учёная степень, Ф.И.О.)
Нормоконтроль	_____ (подпись)	доцент, к.ф.-м.н, Цитринов А.В. (учёное звание, учёная степень, Ф.И.О.)
Рецензент	_____ (подпись)	_____ (учёное звание, учёная степень, должность, организация Ф.И.О.)

Дипломная работа (_____ стр.) допущена к защите в Государственной экзаменационной комиссии.

Зав. кафедрой
«Информационные технологии» _____ доцент, к.т.н. Курочка К.С.
(подпись) (учёное звание, учёная степень, Ф.И.О.)

Гомель 2017

РЕФЕРАТ

Дипломная работа: 75 страниц, 37 рисунков, 9 таблиц, 28 источников, 7 приложений.

Ключевые слова: электрофорез, электрофоретический спектр белка, искусственная нейронная сеть, глубокое обучение, автоэнкодер, сверточная сеть, *TensorFlow*, твердая пшеница.

Объектом исследования являются спектры белков зерна твердой пшеницы, полученные в результате электрофореза.

Цель работы: разработать эффективный метод классификации сортов твердой пшеницы по электрофоретическому спектру глиадина на основе искусственных нейронных сетей с глубинным обучением и написать программный комплекс с графическим интерфейсом пользователя, реализующий разработанный метод.

В ходе выполнения дипломной работы был проведен анализ литературных источников на предмет поиска эффективного метода распознавания сортов растений по электрофоретическому спектру глиадина. В ходе аналитического обзора выяснилось, что ни один из существующих методов не удовлетворяет требованиям генетиков и было принято решение разработать собственный. Новый метод основан на использовании двух искусственных нейросетей: автоэнкодер для предварительного сжатия данных и сеть-классификатор для предсказания сорта. Выполнено сравнение сверточных и глубоких нейросетей, которое показало, что использование сверток более эффективно. Разработанный метод превосходит все существующие в литературе аналоги по эффективности классификации и скорости работы. Было написано веб-приложение с применением микросервисного подхода для удобства использования разработанного метода классификации.

Студент-дипломник подтверждает, что приведенный в дипломной работе материал объективно отражает состояние разрабатываемого объекта, все заимствованные из литературных и других источников теоретические и методологические положения и концепции сопровождаются ссылками на их авторов.

Резюме

Тема работы – «Программный комплекс для анализа и классификации сортов твердой пшеницы по электрофоретическому спектру глиадина на основе искусственных нейронных сетей с глубинным обучением».

Объектом исследования являются спектры белков зерна твердой пшеницы, полученные в результате электрофореза.

Цель работы заключается в создании программного комплекса на основе искусственных нейронных сетей, позволяющего выполнять классификацию сортов твердой пшеницы.

Основным результатом работы является веб-приложение, использующее алгоритмы глубинного обучения для определения принадлежности растения к определенному сорту, имея данные электрофореза белков зерна.

Рэзюмэ

Тэма працы – «Праграмны комплекс для аналізу і класіфікацыі гатункаў цвёрдай пшаніцы па электрафарэтычнаму спектру гліадзіна на аснове штучных нейронных сетак з глыбінным навучаннем».

Аб'ектам даследавання з'яўляюцца спектры бялкоў збожжа цвёрдай пшаніцы, атрыманыя ў выніку электрафарэзу.

Мэта работы заключаецца ў стварэнні праграмнага комплексу на аснове штучных нейронных сетак, які дазваляе выконваць класіфікацыю гатункаў цвёрдай пшаніцы.

Асноўным вынікам працы з'яўляецца вэб-праграма, якая выкарыстоўвае алгарытмы глыбокага навучання для вызначэння прыналежнасці расліны да пэўнага гатунку, маючы дадзеныя электрафарэзу бялкоў збожжа.

Abstract

The theme of the work is a «Software for durum wheat varieties analysis and classification according to the electrophoretic spectrum of gliadin using artificial neural networks with deep learning».

The object of the study are the spectra of durum wheat seed’s proteins obtained by electrophoresis.

The purpose of the work is to create a software using artificial neural networks that allows durum wheat classification.

The main result is a web application that uses deep learning to determine the belonging of a plant to a certain class, having the seed’s protein electrophoresis.

СОДЕРЖАНИЕ

Перечень условных обозначений и сокращений 7
ВВЕДЕНИЕ 8
1 ОБЗОР СУЩЕСТВУЮЩИХ МЕТОДОВ КЛАССИФИКАЦИИ ГЕНЕТИЧЕСКОЙ ИНФОРМАЦИИ 10
1.1 Классификация белков. Метод гель-электрофореза 10
1.2 Методы классификации и разделения спектральных данных 12
1.3 Обзор и сравнение программных решений для классификации белковых последовательностей 18
2 РАЗРАБОТКА АЛГОРИТМА КЛАССИФИКАЦИИ С ПРЕДВАРИТЕЛЬНОЙ КОМПРЕССИЕЙ ДАННЫХ 21
2.1 Нелинейный метод главных компонент 21
2.2 Глубокая нейронная сеть 23
2.3 Сверточная сеть 23
2.4 Связные веса и стохастический градиентный спуск для решения проблемы сходимости 24
2.5 Подготовка данных для обучения и тестов 26
2.6 Разработка функциональной схемы приложения 29
3 РАЗРАБОТКА ВЕБ-ПРИЛОЖЕНИЯ ДЛЯ КЛАССИФИКАЦИИ СОРТОВ ТВЕРДОЙ ПШЕНИЦЫ 33
3.1 Микросервисный подход для разработки веб-приложения 33
3.2 Библиотеки машинного обучения TensorFlow и Keras 35
3.3 Облачные вычисления для ускорения процесса обучения нейросетей 37
3.4 Разработка интерфейса пользователя 39
4 ПОДБОР ПАРАМЕТРОВ И ТЕСТИРОВАНИЕ МОДЕЛИ КЛАССИФИКАЦИИ. ВЕРИФИКАЦИЯ РАБОТЫ ПРИЛОЖЕНИЯ 44
4.1 Подбор параметров модели 44
4.2 Верификация программного обеспечения. Оценка эффективности классификации 52
5 ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ДИПЛОМНОЙ РАБОТЫ 55
5.1 Технико-экономическое обоснование целесообразности разработки программного комплекса для анализа и классификации сортов твердой пшеницы 55

5.2 Оценка трудоемкости работ по созданию программного обеспечения 56
5.3 Расчет затрат на разработку программного продукта..... 57
5.4 Расчет отпускной цены разрабатываемого программного продукта. Стоимость платной подписки 64
6 ЭЛЕКТРОМАГНИТНОЕ ИЗЛУЧЕНИЕ КОМПЬЮТЕРНОЙ ТЕХНИКИ. МЕТОДЫ СНИЖЕНИЯ УРОВНЯ ИЗЛУЧЕНИЙ..... 66
6.1 Электромагнитные излучения компьютера 66
6.2 Электромагнитные излучения портативных компьютеров 68
6.3 Безопастные уровни излучений. Средства защиты от излучений 69
7 РЕСУРСО- И ЭНЕРГОСБЕРЕЖЕНИЕ ПРИ ИСПОЛЬЗОВАНИИ ИНФОРМАЦИОННЫХ СИСТЕМ..... 71
ЗАКЛЮЧЕНИЕ 74
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 75
ПРИЛОЖЕНИЕ А Блок-схемы алгоритмов..... 78
ПРИЛОЖЕНИЕ Б Программный код 81
ПРИЛОЖЕНИЕ В Руководство пользователя..... 99
ПРИЛОЖЕНИЕ Г Руководство программиста..... 103
ПРИЛОЖЕНИЕ Д Руководство системного программиста..... 104
ПРИЛОЖЕНИЕ Ж Примеры работы программы..... 106
ПРИЛОЖЕНИЕ К Результаты расчета экономического обоснования 110

Перечень условных обозначений и сокращений

В настоящей пояснительной записке применяются следующие термины, обозначения и сокращения.

Автоэнкодер – автоассоциативная нейронная сеть с обучением без учителя, реализующая нелинейный метод главных компонент, позволяющий извлекать из данных вектор наиболее значимых признаков размерностью меньше, чем исходные данные.

Батч – часть набора данных, подаваемая на вход нейросети.

Денситограмма – кривая изменения оптической плотности.

ИНС – Искусственная Нейронная Сеть.

ОЗУ – Оперативное Запоминающее Устройство.

ЦП – Центральный Процессор.

Adam (*Adaptive Moment Estimation*) – метод стохастического градиентного спуска, основанный на адаптивной оценке момента.

DDoS (*Distributed Denial of Service*) – тип сетевой атаки, основанной на небезграничности ресурсов атакуемой службы, к которой организуется масса запросов, с целью доведения ее до отказа.

HMM (*Hidden Markov Model*) – скрытая модель Маркова.

REST (*Representational State Transfer*) – передача состояния представления.

SOM (*Self-Organizing Map*) – карты самоорганизации.

UPGMA (*Unweighted Pair-Group Method Using Arithmetic Averages*) – метод невзвешенного попарного среднего.

ВВЕДЕНИЕ

В последнее десятилетие с развитием технологии Интернет количество данных, производимых человечеством, растет в геометрической прогрессии, как и потребность в их обработке. Рекламные компании изучают статистику запросов для организации новых предложений; венчурные фонды платят за предоставление информации по мониторингу трендов, чтобы делать более выгодные вложения; банки изучают историю клиентов для расчета риска выдачи кредитов; музыкальные сервисы и социальные сети подбирают контент, основанный на предпочтениях отдельных пользователей, исходя из истории их веб-запросов. Объем информации, непрерывный поток которой необходимо ежедневно анализировать, просто колоссальный, в связи с этим, ученые именуют это эрой «Больших данных».

Как показывает практика, обычные статистические методы уступают технологиям, основанным на применении искусственных нейронных сетей (ИНС), по эффективности и скорости обработки данных. ИНС представляют собой системы, которым требуется предварительное обучение на большой выборке данных – это нужно для того, чтобы алгоритм нашел такие корреляции в данных, с помощью которых можно выполнить прогнозирование, построить аппроксимацию или сделать кластеризацию, в зависимости от проблемы. Спектр задач, выполняемых ИНС, растет с каждым днем. И такие алгоритмы обучения, черные ящики, показывают лучшие результаты в классификации, кластеризации, распознавании речи, обработке текстовой информации и даже игре против человека.

Пропорционально увеличению объемов обрабатываемых данных и повышению сложности поставленных задач, растет и «глубина» ИНС, т.е. число слоев скрытых нейронов, что позволяет достичь особо высокого уровня нелинейности и повышает эффективность работы алгоритма, чего нельзя достичь использованием обычных сетей прямого распространения (персептрон) с одним-двумя скрытыми слоями. Вместе с тем, возникает проблема обучения глубоких нейронных сетей.

Существует несколько подходов для решения проблемы размерности. Первые попытки были описаны в [1,2] и заключаются в применении особых способов по оптимальной инициализации параметров глубоких сетей. Другой подход представляет собой предобучение глубокой ИНС за счет использования слоев из рекуррентных сетей, ограниченных машин Больцмана, обучаемых последовательно, слой за слоем, по методу Монте-Карло для марковских цепей с оптимизацией в виде алгоритма *Contrastive Divergence* [3]. Однако даже при использовании всех этих уловок, если постоянно наращивать количество слоев, то сеть начнет обучаться все хуже и хуже. Последние исследования открыли

новый подход, именуемый *Residual learning* [4]. Благодаря этому методу остаточного обучения можно предсказывать разницу между выходным сигналом на слое и тем, что должно быть. Это очень схоже с идеей применения машин Больцмана за одной лишь разницей, что не требуется восстанавливать распределение, можно просто вычесть или прибавить разницу на выходе.

Развитие глубокого обучения и наращивание вычислительных мощностей за счет использования графических процессоров позволяет выполнять сложные задачи по обработке больших объемов данных. В дипломной работе представлены способы решения задачи классификации твердых сортов пшеницы с помощью глубокого обучения, на основе генетической информации, представленной оцифрованными белковыми электрофорезами.

Актуальность темы определяется потребностями биологии и радиобиологии, где одной из важнейших задач является задача анализа генетической изменчивости белковых структур. Решение такой задачи включает в себя получение электрофоретических спектров исследуемых белков с последующим распознаванием по изображениям этих спектров генетической принадлежности образца. В силу многокритериальности этой задачи и необходимости автоматизации процессов распознавания и классификации, было предложено использовать для ее решения нейросетевые подходы. Выполненные ранее работы по этой теме показали неэффективность применения прямоточных нейросетей с одним скрытым слоем [5]. Кроме того, большой объем электрофоретической информации серьезно ограничивает возможности ее непосредственного ввода в нейросеть и требует предварительного сжатия вводимых данных с сохранением их информативных свойств.

Таким образом, целью дипломной работы явилось создание программного комплекса для сжатия и классификации генетической информации сортов твердой пшеницы с дальнейшим распознаванием этих сортов. Программный комплекс должен включать в себя реализацию нейросетей, обеспечивающих решение этой задачи и иметь удобный графический интерфейс.

1 ОБЗОР СУЩЕСТВУЮЩИХ МЕТОДОВ КЛАССИФИКАЦИИ ГЕНЕТИЧЕСКОЙ ИНФОРМАЦИИ

1.1 Классификация белков. Метод гель-электрофореза

Одной из основных задач генетики является разделение и классификация белков. Первичные структуры белка пока известны лишь для некоторых живых организмов. В качестве методов классификации используют несовершенные и даже часто перекрещивающиеся методы.

Белки принято подразделять на две основных категории: глобулярные и фибриллярные, в зависимости от растворимости и строении молекулы.

Глобулярные белки хорошо растворяются в воде и разбавленных соленых растворах, имеют шарообразную форму молекулы. Хорошую растворимость белков можно объяснить локализацией на поверхности глобулы заряженных аминокислотных остатков, которые, окружая себя оболочкой из гидрата, обеспечивают хороший контакт с растворителем. К глобулярным белкам относятся все ферменты и, за исключением структурных, большинство других биологически активных белков.

В свою очередь глобулярные белки делятся на альбумины, глобулины, гистоны, протамины, глутелины и проламины, которые чаще всего встречаются в белках зерна [6, с. 345].

Выделение нерастворимых фибриллярных белков не является особо трудной задачей, в то время как выделение и очистка отдельных глобулярных белков из тканей живых организмов сильно затруднена из-за присутствия в растворе многих других белков, липидов и других биомолекул. К этому добавляется чувствительность белка к влиянию температуры, что вызывает его распад – денатурацию.

Различают несколько основных методов разделения белков:

– осаждение;
– распределительная хроматография;
– диализ и ультрафильтрация;
– центрифугирование;
– гель-хроматография;
– электрофорез;
– афинная хроматография и др.

В дипломной работе информация о сортах твердой пшеницы представлена оцифрованными спектрами электрофореза спирторастворенного белка – глиадина. Метод электрофореза представляет собой движение заряженных частиц белков с различной, зависящей от отношения заряда к массе скоростью к аноду или катоду. Таким образом фракции белка могут быть отделены друг от

друга, создавая уникальный маркер, однозначно характеризующий генетическую природу данного белка.

Наиболее распространенный метод электрофореза – с применением геля. Гель-электрофорез проводится в агарозном или полиакриламидном геле. Это исключительно гибкий метод разделения; варьируя структуру геля и состав буферного раствора, можно проводить разделение на основе различия в молекулярных массах изоэлектрических точках и биоспецифическом сродстве. Особенно высокого разрешения достигают при гель-электрофорезе в полиакриламиде, так как здесь дополняют друг друга электрофорез и молекулярно-ситовой эффект [6]. На рисунке 1.1 [5] представлен пример электрофореграммы сорта твердой пшеницы.

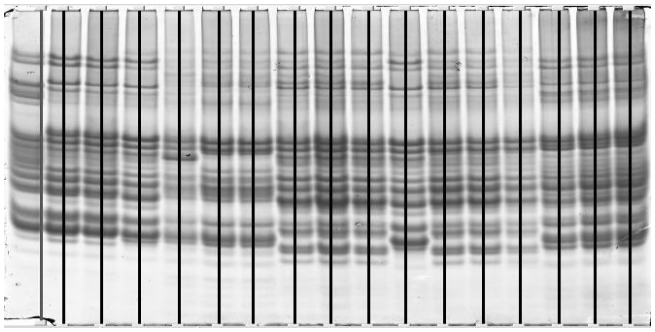


Рисунок 1.1 – Дорожки гель-электрофореза, иллюстрирующие разделение белков для твердой пшеницы

Преимущества белка как генетического маркера связано с тем, что белок является первичным и непосредственным продуктом генетической системы. Белки в наименьшей мере подвержены фенотипической изменчивости и обладают хорошо выраженной биологической специфичностью. В качестве белковых маркеров для пшеницы используют белки семян, в частности мономерные проламины, такие как глиадин [7].

После того, как произведен электрофорез и собраны данные для каждого сорта, их следует оцифровать для ввода в компьютер.

В качестве способа оцифровки применяется денситограмма – кривая изменения оптической плотности. Она представляет собой кривую, описывающую изменение величины плотности пикселей на картинке, в данном случае, более темные полоски на дорожках электрофореза (рисунок 1.1) будут являться пиками спектра денситограммы.

На рисунке 1.2 изображен пример денситограммы для одной из дорожек электрофореза, где по оси ординат обозначена интенсивность пикселя исходной фотографии дорожки гель-электрофореза.

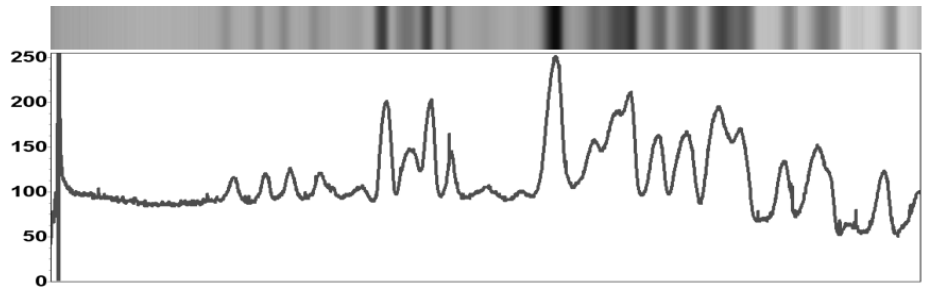


Рисунок 1.2 – Денситограмма дорожки гель-электрофореза

1.2 Методы классификации и разделения спектральных данных

Самоорганизующиеся карты Кохонена. *Self-organizing map* (*SOM*), карты самоорганизации – вид ИНС, предложенный Кохоненом [8], основной задачей которого является преобразование входного сигнала любой размерности в одно- или двумерную дискретную карту. На рисунке 1.3 [9] представлена схема *SOM*.

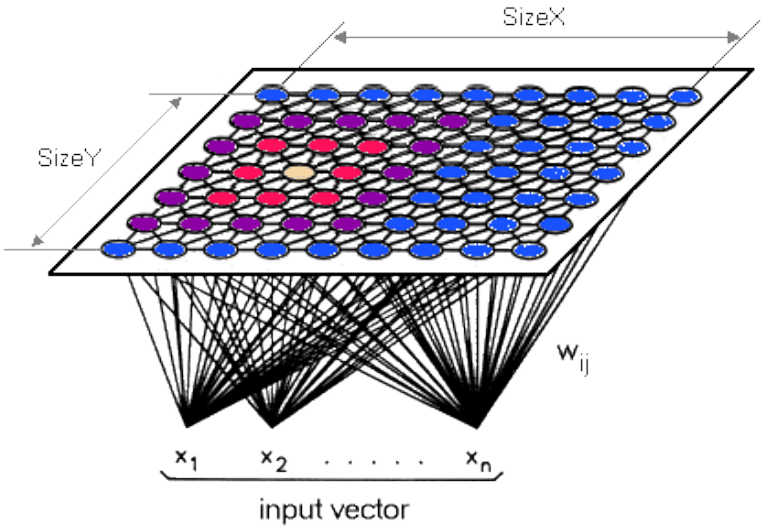


Рисунок 1.3 – Схема *SOM*, где *X* – это входной вектор, а *W* – матрица весов для перевода в дискретную карту

Алгоритм *SOM* включает в себя три основных шага: подвыборка (*sampling*), поиск максимального соответствия (*similarity matching*) и корректировка весов (*weights updating*).

Цикл алгоритма можно разделить на 5 шагов, которые следует итеративно повторять до сходимости:

– *Инициализация*. Для исходных весов $w_j(0)$ синаптических связей

выбираются случайные значения. Вектора для отдельных $j = 1,2, \ldots, l,$ где l – это общее число нейронов в решетке, должны быть различными. При этом следует сохранять малую амплитуду значений инициализации.

– *Подвыборка*. С определенной вероятностью избирается вектор x из входного пространства. Этот вектор представляет собой возбуждение, которое применяется к решетке нейронов. Размерность вектора x равна m.

– *Поиск максимального подобия.* Находится наиболее подходящий (победивший) нейрон $i(x)$ на шаге n, используя критерий минимума Евклидова расстояния:

$$i(x) = arg\min_j \|x - w_j\|, j = 1,2,\ldots,l. \quad (1.1)$$

– *Коррекция*. Корректируем векторы синаптических весов всех нейронов, используя следующую формулу:

$$w_j(n+1) = w_j(n) + \eta(n)h_{j,i(x)}(n)(x - w_j(n)), \quad (1.2)$$

где $\eta(n)$ – параметр скорости обучения;

$h_{j,i(x)}$ – функция окрестности с центром в победившем нейроне $i(x)$.

Оба эти параметра динамически изменяются во время обучения с целью получения лучшего результата.

– *Продолжение.* Возвращаемся к шагу 2 и продолжаем вычисления до тех пор, пока в карте признаков не перестанут происходить заметные изменения [10, с. 587].

На рисунке 1.4 [10, с.599] представлены изменения состояния выходной решетки в процессе обучения.

Алгоритм позволяет топологически упорядочить распределение входного сигнала, что позволяет решать задачи кластеризации и классификации. Несмотря на то, что *SOM* – это сильный алгоритм кластеризации, который также позволяет бороться с зашумлением, все же имеет целый ряд недостатков:

– сильно полагается на выбранную меру расстояния;

– долгий процесс обучения;

– плохая эффективность для категориальных данных;

– нет общепринятых правил для назначения параметров сети.

Метод невзвешенного попарного среднего. *Unweighted Pair-Group Method Using Arithmetic Averages* [11] (*UPGMA*) – один из методов иерархичного кластерного анализа.

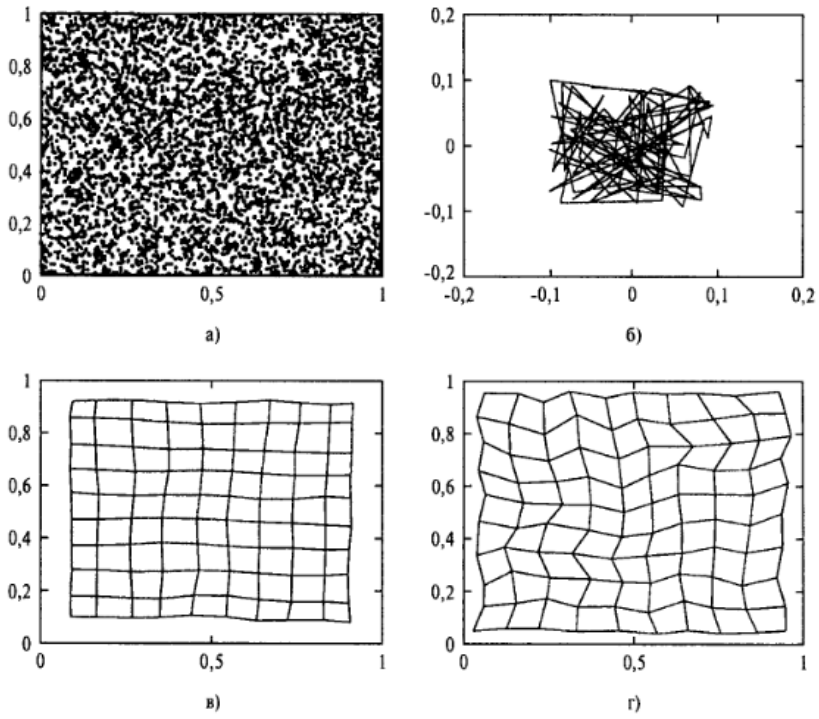


Рисунок 1.4 – а) распределение входных данных; б) начальное состояние двумерной решетки; в) состояние решетки в конце этапа упорядочивания; г) состояние решетки в конце этапа сходимости

На первом шаге каждый объект входных данных представляет собой отдельный кластер. Далее высчитывается матрица попарных расстояний между центрами всех кластеров. Расстояние (*similarity*) рассчитывается по формуле:

$$similarity(cluster1, cluster2) = \frac{\sum cosine(d1, d2)}{size(cluster1) * size(cluster2)}, \quad (1.3)$$

где $\sum cosine(d1, d2)$ – сумма косинусных расстояний между объектами каждого кластера, вычисляемая по формуле 1.4:

$$cosine(d1, d2) = \frac{d1 \cdot d2}{\|d1\|_2 \|d2\|_2} = \frac{\sum_{i=1}^{n} d1_i d2_i}{\sqrt{\sum_{i=1}^{n} d1_i^2} \sqrt{\sum_{i=1}^{n} d2_i^2}} \quad (1.4)$$

После того, как найдена матрица попарных расстояний, находят два кластера с наименьшим расстоянием и объединяют их в отдельный кластер. Далее снова вычисляют матрицу попарных расстояний, используя формулу 1.3 и повторяют предыдущий шаг. Так повторяется до тех пор, пока не останется два кластера, которые объединяются в один.

На выходе формируется дендрограмма (рисунок 1.5). Дендрограмма – это дерево, построенное по матрице мер расстояний между объектами, она позволяет отобразить иерархичную структуру связей между объектами в заданном множестве.

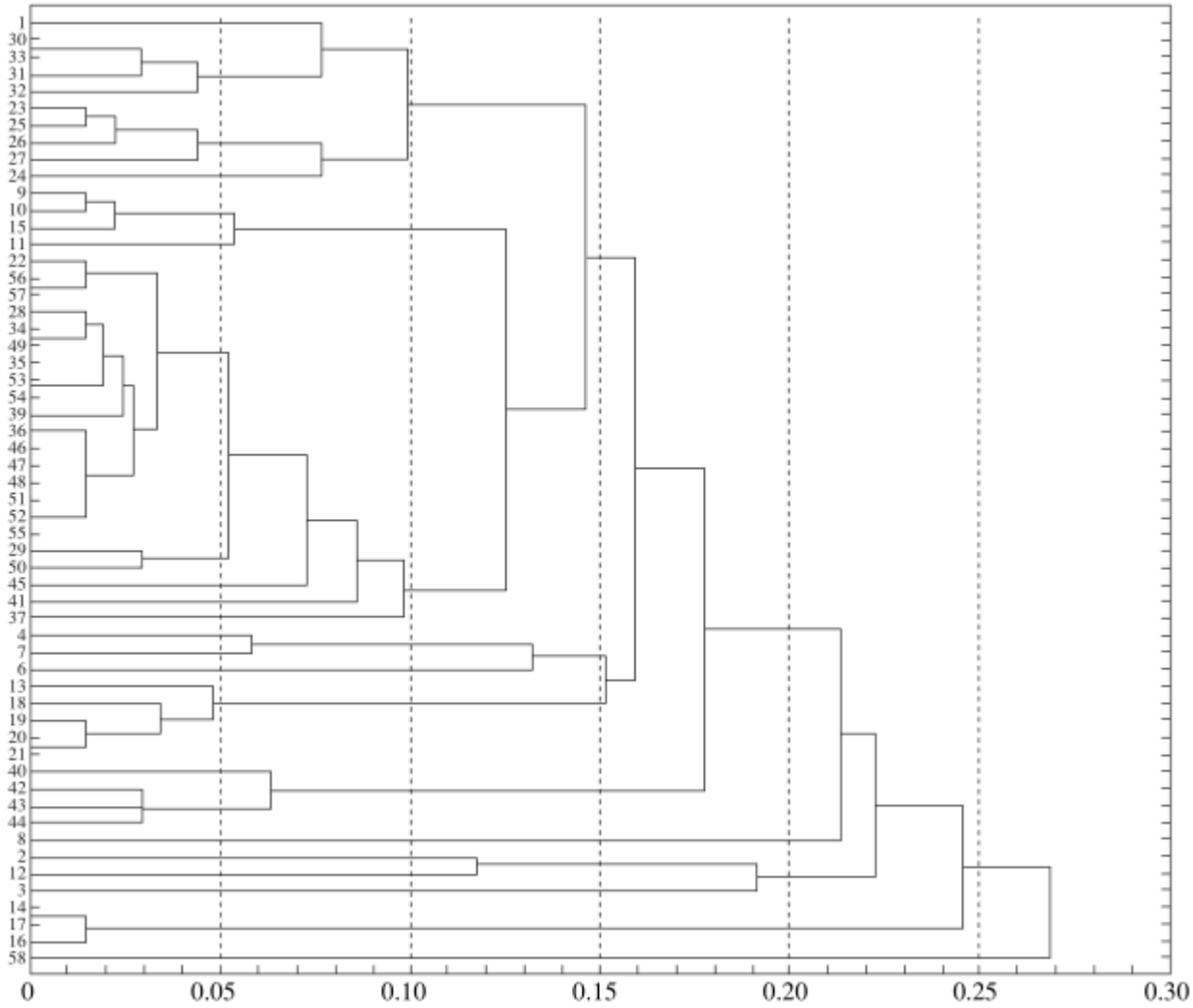


Рисунок 1.5 – Дендрограмма *UPGMA* образцов стручкового перца [12]

Недостатками данного метода являются:

– метод не надежен, если данные не являются ультраметрическими;

– могут быть получены отличные друг от друга результаты, если подавать на вход данные в разном порядке.

Последний недостаток довольно существенный, так как сорта растений должны быть классифицированы однозначно верно, как и другие методы кластеризации, качество зависит от меры расстояния. В связи с этим, следует обратить внимание на методы классификации, которые используются в задачах распознавания сортов: скрытые марковские модели [13] и персептрон [14].

Скрытая марковская модель. *Hidden Markov Model* (*HMM*) – это статистическая модель, в которой основной задачей является определить неизвестные параметры на следующей отсечке времени, на основе наблюдаемого «настоящего». Модель пытается смоделировать марковский процесс – случайный процесс, в котором каждый следующий шаг $t+1$ не зависит от эволюции, предшествовавшей *t*, при условии, что значение процесса в каждый отдельно взятый момент времени фиксировано.

На рисунке 1.6 представлена схема *HMM*.

Рассмотрим пример простейшего марковского случайного процесса. Пускай человек делает шаг вперед или назад, в зависимости от результата броска монеты: назад, если решка – вперед, если орел. После каждого шага снова бросается монета и делается следующий случайный шаг. Такое «блуждание» назад-вперед представляет собой марковский процесс, так как следующее положение человека зависит только от текущего положения и не зависит от прошлых.

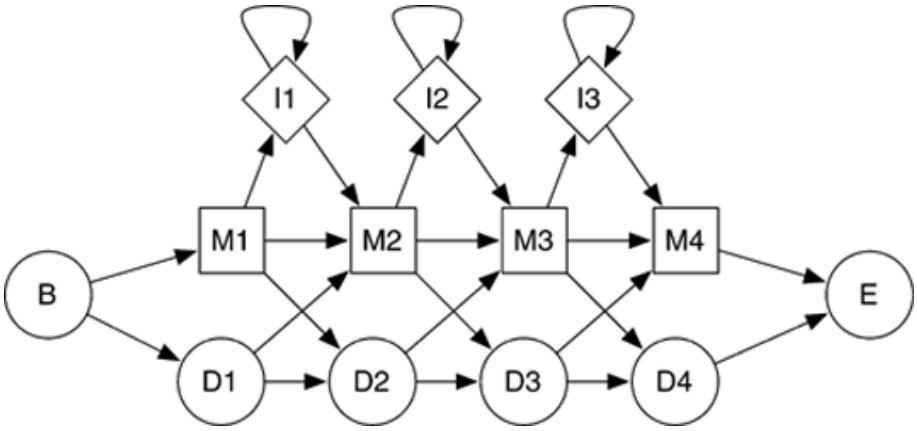


Рисунок 1.6 – схема цепи для скрытой марковской модели

Так как стоит задача определить наиболее вероятный сорт твердой пшеницы по последовательности, представленной денситограммой, то важной величиной является вероятность определения сорта для последовательности:

$$P(c \mid s, \theta, \phi) = P(s, c \mid \theta, \phi) / P(s \mid \theta), \qquad (1.5)$$

где *c* обозначает вектор классов;

ϕ – вероятности классов;

θ – параметры *HMM;*

s – входная последовательность [15].

Класс с самой высокой вероятностью и будет являться выходным значением *HMM*.

Процесс обучения стоит в минимизации функции ошибки, представленной формулой 1.8 [15]. Функция ошибки называется функцией логарифмического правдоподобия или отрицательного логарифмического правдоподобия.

$$L_f = -log\, P(s \mid \theta), \tag{1.6}$$

$$L_c = -log\, P(c,s \mid \theta,\phi), \tag{1.7}$$

$$L = -log \frac{P(c,s \mid \theta,\varphi)}{P(s \mid \theta)} = L_c - L_f . \tag{1.8}$$

Обновление параметров модели происходит по формуле:

$$\theta_i^{new} = Normalize(\theta_i^{old} + \alpha(m_k^{\mu} - n_k^{\mu})), \tag{1.9}$$

где «*Normalize* » – это функция нормализации;

α – это параметр скорости обучения;

m_k^{μ} – это ожидаемое число раз, когда *k*-ый параметр используется в последовательности наблюдений *s* и обеспечивает верную классификацию *c*;

n_k^{μ} – это остальные случаи использования параметра *k*.

HMM обеспечивает достаточно высокую эффективность в задачах категориальной классификации, однако имеет очень большой недостаток в виду длительного времени обучения и скорости классификации. Однако является весьма распространенной моделью для классификации генетических последовательностей.

Персептрон. Персептроном, называют простейшую ИНС прямого распространения. На рисунке 1.7 представлена модель многослойного персептрона с одним скрытым слоем.

Процедура обучения персептрона заключается в минимизации функционала ошибки (неправильного ответа сети). Для этого используется градиентный метод обратного распространения ошибки [16]. Эволюция сети происходит по формуле:

$$w(n) = w(n) - \alpha \frac{\partial L}{\partial w}, \tag{1.10}$$

где α – это параметр скорости обучения;

L – это функция ошибки сети (*loss*).

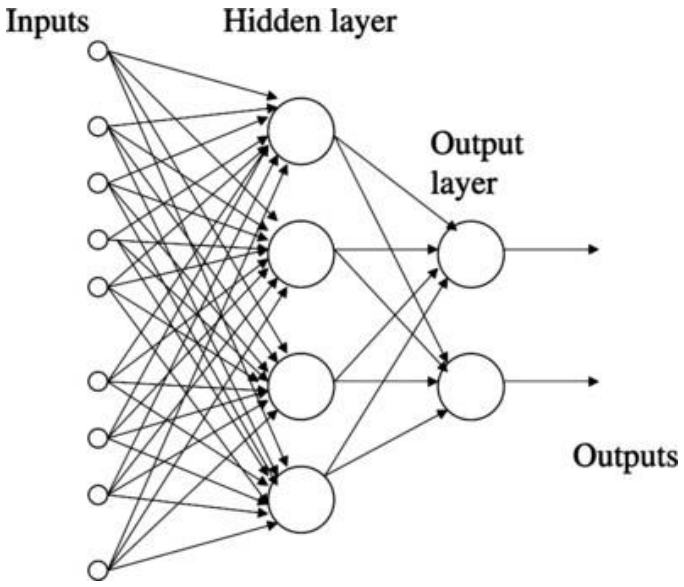


Рисунок 1.7 – Многослойный персептрон с одним скрытым слоем

Персептрон позволяет достигать высокой эффективности классификации на данных, не обладающих сложной внутренней структурой. Имеет высокую скорость обработки данных.

1.3 Обзор и сравнение программных решений для классификации белковых последовательностей

В литературе существует множество примеров использования методов, описанных выше, для решения задач обработки биологической информации.

В работе [12] авторы попытались сравнить применение *SOM* и *UPGMA* для задачи классификации разновидностей стручкового перца. Результаты исследования показали эффективность распознавания порядка 90%, причем оба метода имеют сравнительно равные показатели точности классификации. Однако следует заметить, что число сортов, исследуемых в статье, всего 8.

Исследования [17,18] показывают возможность применения однослойного персептрона в задаче распознавания сортов растений по электрофоретическим спектрам.

В работе [17] стоит задача, аналогичная данной дипломной работе. У авторов получилось добиться эффективности классификации на тестовой выборке в пределах 70 – 90%, что, как утверждают авторы, соответствует результату, который показывают эксперты.

Исследование [18] является примером, как можно использовать комбинированный подход в классификации, путем предварительной кластеризации данных с помощью *SOM*. Эффективность классификации, которая была достигнута авторами этой статьи 87.5 для 12 сортов картофеля.

Существуют и другие готовые программные решения, которые, однако, не выполняют задачу классификации, но также ориентируются на анализ генетических и белковых последовательностей. Это продукты *InterPro* [19] и *PANTHER* [20]. В основе данных программных решений лежит использование *HMM*, что позволяет достичь требуемой эффективности, однако очень сильно проигрывает по скорости работы другим статистическим методам.

На основе этих данных составлена таблица 1.1, в которой указаны качественные сравнительные характеристики алгоритмов анализа спектральной информации в задачах генетики. Самыми важными являются два показателя: эффективность классификации и скорость работы алгоритма.

Таблица 1.1 – Качественные сравнительные характеристики алгоритмов

Характеристика	*SOM*	*UPGMA*	*HMM*	Персептрон
Высокая скорость обработки	+	+	–	+
Высокий показатель эффективности	–	–	+	–
Выполняет задачу классификации	–	–	–	+

Как можно заметить из таблицы выше, алгоритмы, работающие быстро, не могут обеспечить высокой точности классификации для большого числа сортов. В то время, как скрытая марковская модель отвечает всем требованиям показателей эффективности, она не позволяет проводить быструю обработку данных.

В работе [5] для решения задачи классификации сортов твердой пшеницы используются также и другие методы, такие как:

– быстрое преобразование Фурье;

– дискретное вейвлет преобразование;

– метод главных компонент;

– поиск по эталону.

Как и в случае с персептроном все вышеперечисленные методы не позволяют перейти порог в 10 сортов. Если количество распознаваемых классов больше 10, то эффективность резко снижается.

Можно заметить, что ни один из предложенных методов не отвечает всем поставленным требованием. Это означает потребность в разработке программного комплекса, который будет использовать модель классификации, обеспечивающую быструю скорость обработки данных и высокие показатели эффективности для большого числа сортов.

В работе [21] автором дипломной работы был предложен новый подход для классификации данных с помощью ИНС. Суть подхода заключается в использовании двух нейросетей: одна для представления данных в сжатой форме путем извлечения признаков – вторая для классификации. Данный подход показал лучшие результаты распознавания по сравнению с персептроном и также обеспечивает высокую скорость обработки данных, так как оперирует сжатыми представлениями объектов на входе сети.

Разработанный в дипломной работе программный комплекс реализует подход, предложенный в [21] с модернизацией в виде перехода на ИНС, использующие свертки.

2 РАЗРАБОТКА АЛГОРИТМА КЛАССИФИКАЦИИ С ПРЕДВАРИТЕЛЬНОЙ КОМПРЕССИЕЙ ДАННЫХ

2.1 Нелинейный метод главных компонент

Собственная размерность данных – это наименьшее измерение, описывающее этот набор данных без существенной потери информации. Биометрические базы данных, такие как изображения лиц, являются типичными примерами многомерных данных с низкой собственной размерностью. Использование всех пикселей изображения в качестве входного вектора для нейронной сети избыточно из-за корреляций и нелинейных зависимостей между пикселями, формирующими это изображение и, следовательно, намного больше, чем собственная размерность изображения. Снижение размерности изображения до его собственной размерности значительно упрощает обработку. Это особенно важно для глубоких нейронных сетей, где количество межнейронных связей резко увеличивается пропорционально размеру вектора на входе сети. Увеличения числа нейронов делает процесс обучения очень долгим и приводит к плохой сходимости функции стоимости ИНС, а также способствует переобучению. Однако использование линейных методов сокращения размерности, таких как метод главных компонент является менее эффективным, чем поиск нелинейных корреляций в данных.

Для сжатия исходных данных можно использовать разные методы. В работе [5] описаны несколько подходов для извлечения признаков из спектров:

– метод генетических эталонов;
– метод главных компонент;
– быстрое преобразование Фурье;
– дискретное вейвлет преобразование;
– метод извлечения пиков.

Метод извлечения пиков показывает самые высокие результаты классификации. Идея его заключается в том, чтобы охарактеризовать каждый пик тремя признаками: позицией в денситограмме, интегралом (площадью под пиком) и высотой, соответственно. Потом авторы делают отсеивание маловажных пиков и для каждой денситограммы остается максимум 37 пиков. Точность классификации для 5 сортов – 100% и начинает падать только в случае классификации более чем 8 сортов [5].

В качестве алгоритма компрессии данных, в дипломной работе применяется автоэнкодер с тремя скрытыми слоями. Автоэнкодер – это автоассоциативная ИНС с обучением без учителя, реализующая нелинейный метод главных компонент (НМГК), позволяющий извлекать из данных вектор наиболее значимых признаков размерностью меньше, чем исходные данные [22].

В отличие от идеи использования иерархии связанных автоэнкодеров для обучения сети-классификатора [23], мной предлагается новый подход использования автоэнкодера, как отдельной ИНС, что позволяет:

– провести более тонкую настройку параметров сети;

– получить одинаковое сжатое представление данных для любого классификатора.

Была использована модель автоэнкодера, представленная Крамером [22], в которой нелинейность достигается за счет добавления трех скрытых слоев, в том числе среднего слоя, извлекающего вектор признаков. Схема такой сети представлена на рисунке 2.1.

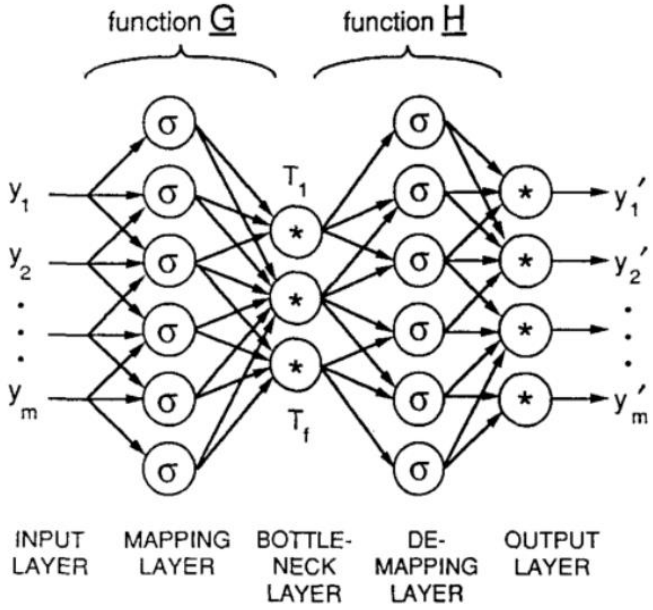


Рисунок 2.1 – Автоассоциативная нейронная сеть-автоэнкодер

Слой «бутылочное горлышко» *T* извлекает *f* главных компонент. Обучение проводится с сигмоидными посредством минимизации суммы квадратичных ошибок (*SSE, sum of squared errors*):

$$E = \sum_{p=1}^{n} \sum_{i=1}^{m} (Y_i - Y_i^{'})_p^2. \quad (2.1)$$

Подход Крамера был модифицирован за счет замены сигмоидных активаций на *ReLU* [24] и применения специальной нормализации весов, а в качестве алгоритма обучения использовался специальный метод стохастического градиентного спуска, известный как *Adam* [25].

2.2 Глубокая нейронная сеть

Глубокая нейронная сеть представляет собой многослойный персептрон с большим количеством скрытых слоев. Как было сказано во введении, глубокие нейронные сети трудно обучить, если не использовать определенные уловки. Для того, чтобы построить классификатор с множеством скрытых слоев, была использована специальная нормализованная функция активации [1]:

$$W = U\left[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\right], \tag{2.2}$$

где $U[-a,a]$– это равномерное распределение на интервале $(-a,a)$;

n_j – это число нейронов на j - м скрытом слое.

2.3 Сверточная сеть

Сверточные сети [26] – это отдельный вид глубоких ИНС, состоящих из комбинирования слоев свертки и пулинга (разбиения на меньшие подвыборки) в связке с обычными слоями нейронов на выходе. На рисунке 2.2 представлена схема архитектуры сверточной сети *LeNet*-5 [26], используемой для задачи распознавания символов и цифр.

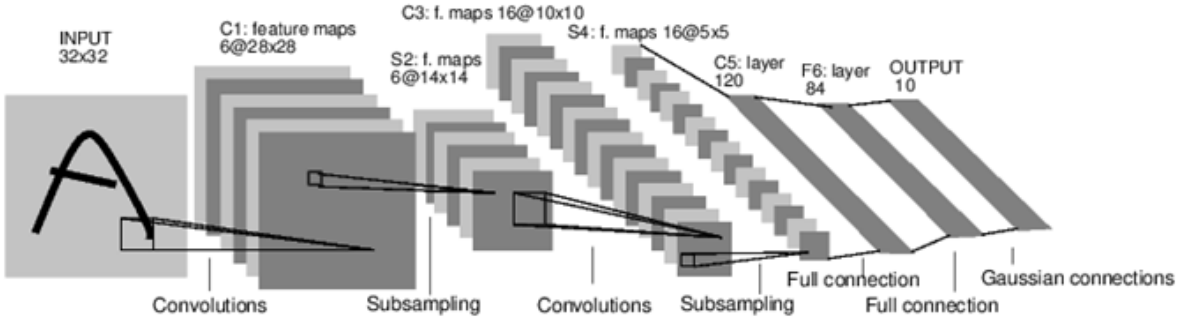


Рисунок 2.2 – Архитектура сверточной сети *LeNet-5*

Сверточный слой представляет собой определенное количество фильтров, которые в процессе обучения становятся детекторами неких признаков, нелинейных корреляций в данных. Операция свертки является простым матричным перемножением ограниченной матрицы весов небольшого размера, которую сдвигают по всему обрабатываемому слою (в самом начале – по входному изображению), вычисляя после каждого сдвига сигнал активации для

нейрона следующего слоя с тем же индексом. Перед обучением сети задается размер окна свертки и шаг движения окна по площади входных данных. После выполнения операции свертки, информация будет преобразована к многомерному массиву с глубиной, равной количеству фильтров.

Слой пулинга (иначе подвыборки, субдискретизации) используется для уменьшения размерности, как это показано на рисунке 2.3.

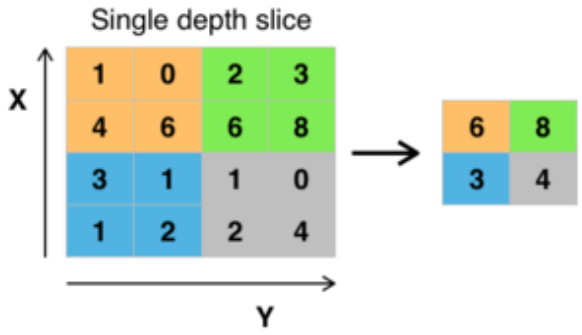


Рисунок 2.3 – Пулинг 2x2 с шагом 2 по максимуму

Перед обучением устанавливается размер окна для пулинга и шаг. Алгоритм заключается в том, что по всей площади входных данных (для слоя пулинга) скользит окно заданной размерности, представляющее из себя функцию поиска максимума (или среднего, в зависимости от вида пулинга) значения в окне. Таким образом, если входное изображение имеет размерность 8x8, а окно пулинга 2x2 с шагом 2, то размер выходных данных будет 2x2. Это эффективный метод снижения размерности, потому что изображения обладают свойством локальной скоррелированности пикселей – соседние пиксели, как правило, не сильно отличаются друг от друга.

2.4 Связные веса и стохастический градиентный спуск для решения проблемы сходимости

Связные веса. Сверточная сеть позволяет резко сократить число параметров модели, но при использовании обычного автоэнкодера с тремя скрытыми слоями может возникнуть проблема. Большое число параметров усложняет модель, уменьшая вероятность стабилизации в глобальном минимуме, а также требует большего числа памяти и вычислительных ресурсов.

Метод связывания весов очень просто объяснить на примере персептрона с одним скрытым слоем, выход которого равен:

$$f(x) = \sigma_2(b_2 + W_2\sigma_1(b_1 + W_1 x)), \quad (2.3)$$

где σ – сигмоидная функция активации;

b – смещения;

W – синаптические веса.

Так как W_1 и W_2 независимы друг от друга, то после обучения они имеют разные значения. Чтобы избежать этого, нужно использовать связные веса, как в формуле ниже:

$$f(x) = \sigma_2(b_2 + W_1^T \sigma_1(b_1 + W_1 x)), \tag{2.4}$$

Полагается, что $W_2 = W_1^T$ – это позволяет резко сократить число степеней свободы. Связные веса работают быстрее, дают более точные результаты в поиске глобального минимума и сокращают число параметров сети-автоэнкодера вдвое.

Метод *Adam*. Обычный метод градиентного спуска производит вычисление производной для каждого примера в обучающей выборке, что очень медленно и избыточно, так как повторно вычисляются градиенты для аналогичных примеров в выборке перед каждым обновлением параметров. Стохастический градиентный спуск (СГС), наоборот, выполняет по одному обновлению за каждый случайно выбранный тренировочный пример. Это многократно повышает скорость работы и, кроме того, колебание СГС, скачки по функции потерь позволяют перейти к новым и потенциально лучшим локальным минимумам. Но с другой стороны, повышается риск постоянного «выпрыгивания» из минимума в силу случайности выбора тренировочного примера. Одним из способов оптимизации SGD и демпфирования его колебаний является добавление к текущему вектору обновления «импульса инерции» как доли вектора обновления на последнем временном шаге.

В работе используется метод *Adaptive Moment Estimation* (*Adam*) для эффективной стохастической оптимизации, для которого требуются только градиенты первого порядка [25]. *Adam* вычисляет адаптивные скорости обучения для каждого параметра, а также сохраняет экспоненциально затухающее среднее из прошлых градиентов, подобное импульсу. Важным свойством правила обновления метода *Adam* является тщательный выбор величины шага.

На рисунке 2.4 представлен результат применения связных весов и метода *Adam*. В качестве данных для обучения и тестов был использован популярный бенчмарк *MNIST* – база данных рукописных цифр. Префикс «*Tied*/*Untied*» означает вид весов ИНС: связные и независимые веса, соответственно. «*GD*» – это аббревиатура градиентного спуска.

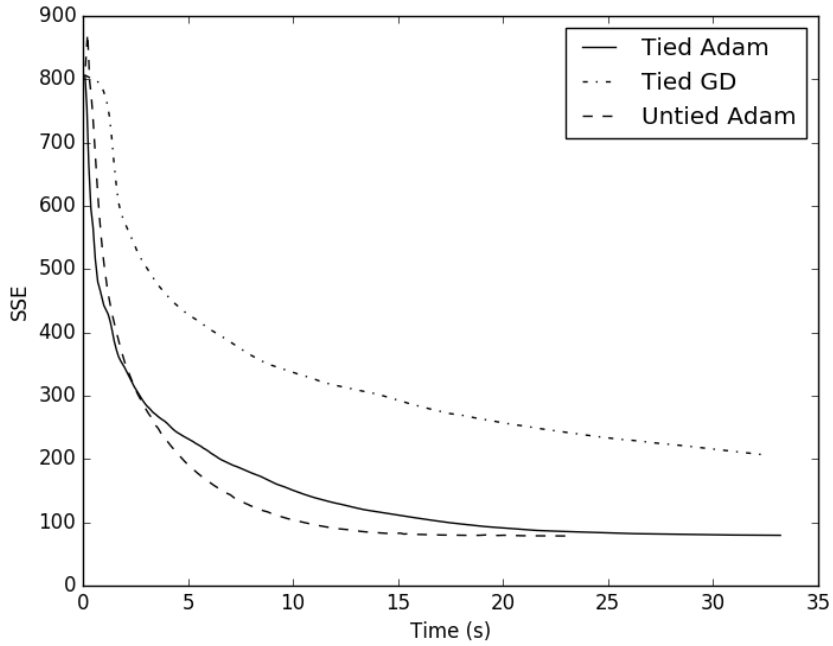


Рисунок 2.4 – Уменьшение суммы квадратичных ошибок во времени в зависимости от вида весов и алгоритма минимизации

Как можно видеть на рисунке выше, *Adam* существенно сокращает скорость сходимости по сравнению с обычным обратным распространением ошибки и даже позволяет достичь минимума функции потерь для сети с несвязными весами, в то время как, обратное распространение ошибки не позволяет этого сделать. Ошибка сети с несвязными весами после нескольких итераций обычного градиентного спуска резко подскакивает до запредельных величин, поэтому не представлена на графике.

2.5 Подготовка данных для обучения и тестов

Исходная база данных, для обучения содержит 67 текстовых файлов, где имя каждого файла обозначает индекс сорта. В каждом файле данные денситограмм записаны по одному значению в отдельной строке по 4096 отсчетов каждый спектр. Между спектрами – пустая строка.

На рисунке 2.5 проиллюстрировано количество примеров денситограмм для каждого сорта в базе. И как можно заметить, выборка сильно несбалансированная – разное число тренировочных образцов для каждого сорта.

В связи с тем, что число примеров для обучения сильно отличается для каждого сорта, возникают две проблемы:

– при разбиении на выборки для тренировки и теста нужно сохранить баланс;

– в процессе обучения сеть-классификатор может отдать предпочтение классу с большим числом примеров.

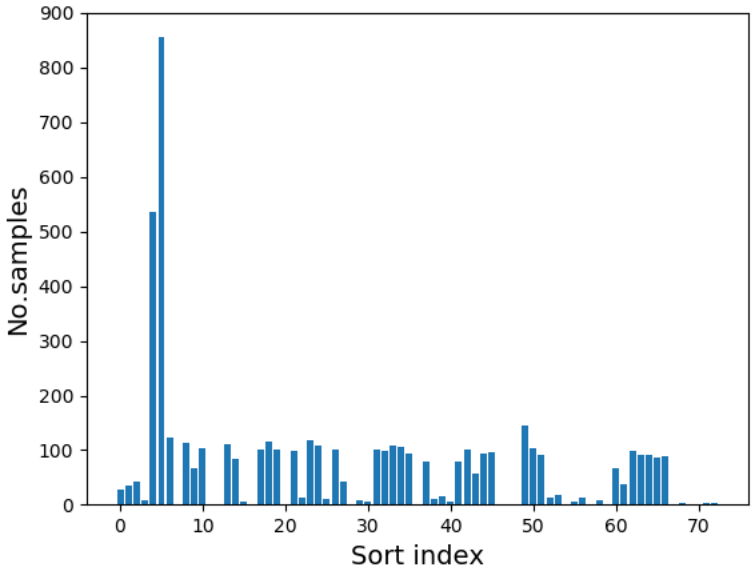


Рисунок 2.5 – Распределение числа денситограмм для каждого сорта в базе данных

Для того чтобы решить первую проблему, нужно создать обучающую и тестовую выборки для каждого сорта отдельно. Сначала вся исходная выборка примеров сорта перемешивается – это делается для того, чтобы в базу для обучения и теста входили разные примеры. После того, как данные перемешаны, выборка делится на две части в соотношении 80 на 20, где 80% – это размер массива для обучения, а 20% – тестовый массив. Так повторяется для каждого сорта.

После того, как все сорта разделены на подвыборки, все обучающие массивы объединяются в один, то же самое для тестовых примеров. Получается две базы, которые тоже перемешиваются, чтобы сорта не шли при обучении последовательно.

Для решения второй проблемы существует несколько путей:

– собрать большее число примеров;

– использовать другой показатель эффективности (*F1 Score, ROC*);

– сбалансировать базу путем добавления копий для малочисленных классов или удаления примеров многочисленных;

– создать модель для генерации новых искусственных данных;

– использовать систему штрафов для функции ошибок.

Собрать большее число примеров не такая простая задача, требующая большого количества времени. К тому же, иногда в природе отдельный сорт может быть очень редким, что обуславливает малое число примеров для обучения, так что такой подход не отвечает нужным требованием. Использование другого показателя эффективности, конечно, покажет объективную картину классификации, но не заставит ИНС повысить качество

распознавания. Создание копий денситограмм для малочисленных классов может помочь в решении проблемы, но требует специальных усилий для верификации полученных распределений этих копий. Разработка модели для генерации новых примеров обучения довольно распространенный подход, от которого пришлось отказаться в виду временных затрат на построение такой системы. Выходом из ситуации стало ограничение функции потерь за счет присвоения каждому классу его веса, вычисляемого по формуле:

$$w_i = \frac{\sum n_i}{n_i}, \tag{2.5}$$

где n_i – число обучающих примеров i-го класса;

w_i – это целочисленный вес каждого класса.

Таким образом, сеть будет штрафоваться каждый раз, когда будет отдавать предпочтение сорту с большим числом примеров и получать поощрение, если верно распознает сорт с меньшим количеством данных.

Некоторые данные совершенно в исходной базе некорректны или и вовсе отсутствуют. На рисунке 2.6 можно увидеть визуализацию десяти спектров из обучающей выборки для сорта с индексом 1.

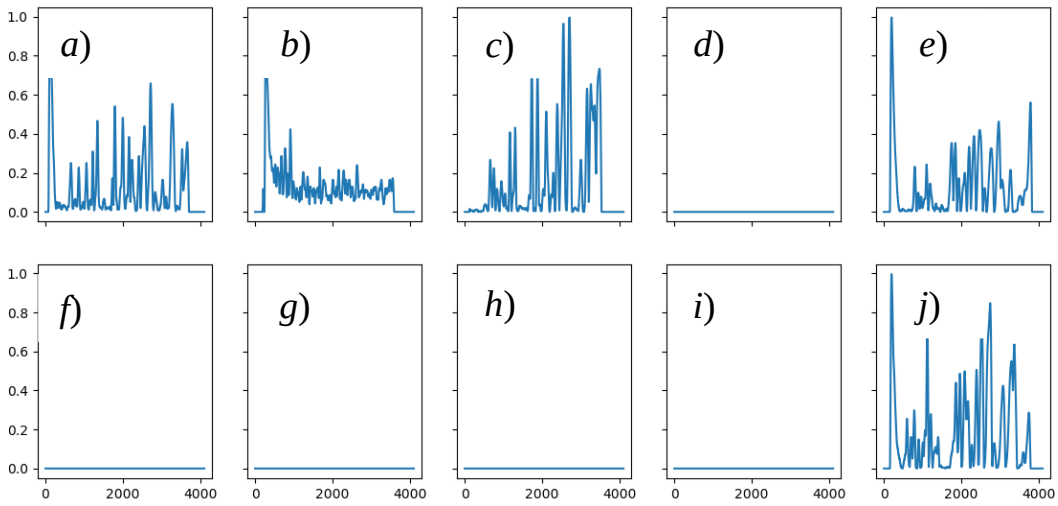


Рисунок 2.6 – Визуализация 10-ти денситограмм сорта с индексом 1

Из рисунка выше можно заметить, что денситограммы *d*, *f*, *g*, *h*, *i* имеют нулевое значение на всех 4096 отсчетах. Возможно, это обусловлено какими-то ошибками при произведении электрофореза или же при записи в файл исходной базы данных. Спектры *a*, *e*, *j* имеют одинаковую структуру и ярко выраженные отдельные пики, в то время как, объекты *b* и *c* сильно отличаются и явно непохожи на представителей одного сорта.

Нулевые спектры были исключены из обучающей и тестовой выборок, а те спектры, что отличаются от эталона, оставлены в виду того, что допускается погрешность при проведении электрофореза.

2.6 Разработка функциональной схемы приложения

На основании выбранных алгоритмов обработки данных формируется функциональная схема приложения. В дипломной работе используется подход, описанный в [21]: для классификации используется ансамбль из двух нейросетей для произведения предварительной компрессии данных и дальнейшей классификации.

Программный комплекс должен состоять из двух основных блоков-функций:

– обработка спектра на входе и предсказание сорта;

– обучение модели на новых данных.

Перед тем, как подать данные на классификатор, нужно произвести их предварительную компрессию с помощью одного из двух видов автоэнкодера, в зависимости от того, выбран глубокий метод или сверточный:

– автоассоциативная сеть с тремя скрытыми слоями;

– сверточный автоэнкодер.

Так как автоэнкодер обучен извлекать признаки из спектральных данных, то он будет работать для любых видов электрофореграмм и его не нужно каждый раз переобучать. Поэтому при запуске программы происходит начальная инициализация, включающая в себя загрузку в оперативную память компрессионных моделей и, в зависимости от метода, сверточного или глубокой сети классификатора – все загружаемые модели уже обучены. Начальная инициализация нужна для того, чтобы не тратить каждый раз время на выгрузку графов вычислений в оперативную память.

На рисунке 2.7 представлена блок-схема функции начальной инициализации, функция «*init*».

Операция «очистка графов вычислений» в блок-схеме функции «*init*» означает, что производится удаление из оперативной памяти всех существующих графов и вычислительных сессий. Делается это по двум причинам:

– чтобы избежать ошибок при использовании двумя разными графами одних и тех же переменных;

– освободить больше места в оперативной памяти сервера.

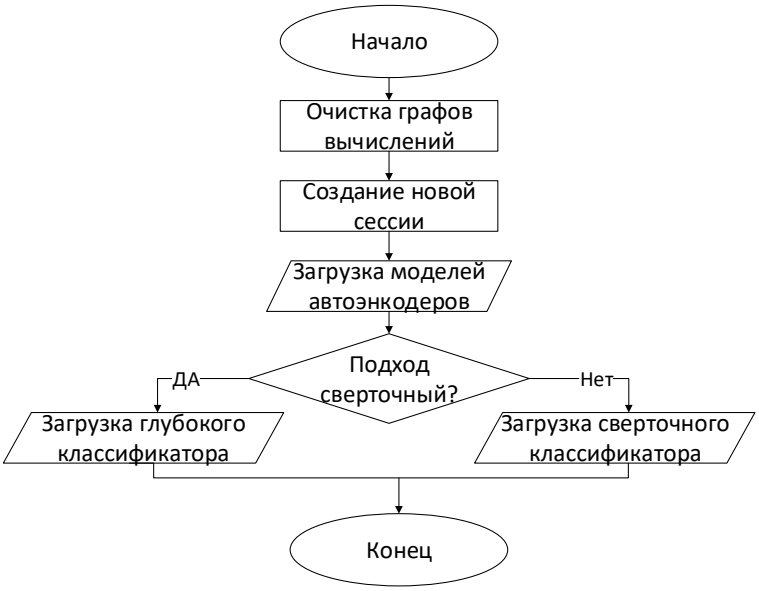


Рисунок 2.7 – Блок-схема функции начальной инициализации

Так как сжимающие сети загружены в оперативную память, то можно в любой нужный момент вызвать функцию компрессии данных для обработки входных спектров. Блок-схема функции сжатия данных, «*compress_data*» представлена на рисунке 2.8.

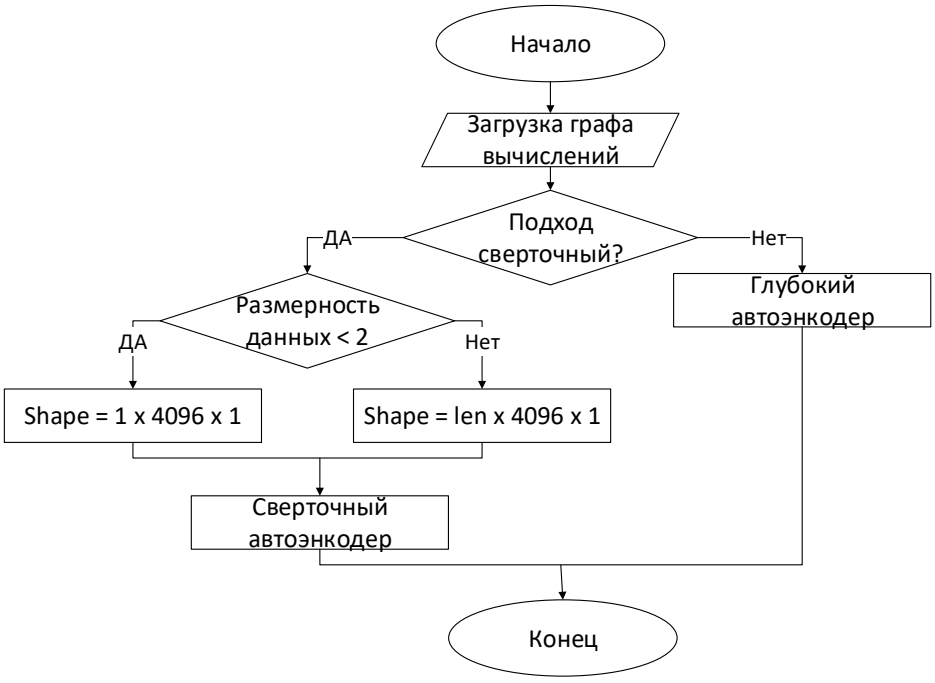


Рисунок 2.8 – Блок-схема функции компрессии

Функция компрессии возвращает исходные данные в сжатом формате, представленном главными компонентами. Сжатые данные могут поступать на

вход сети классификатора для предсказания наиболее вероятного номера сорта. Блок-схема функции предсказания сорта, метод «*predict*» содержится в приложении А.

Также в программе должна быть предусмотрена возможность обучить модель на других данных или обучить уже имеющуюся модель на новых данных для увеличения числа распознаваемых сортов, например. Для этого пользователь должен будет выбрать папку с исходными файлами для обучения, которые будут обработаны с помощью функции «*read_dataset*», возвращающей тренировочные данные в нужном для модели формате, которые сохраняются в папке на сервере. Блок-схема функции чтения базы представлена в приложении А.

На рисунке 2.9 проиллюстрирована блок-схема функции обучения сети, метод «*train*», где градиентный шаг включает в себя корректировку синаптических весов сети по методу *Adam*.

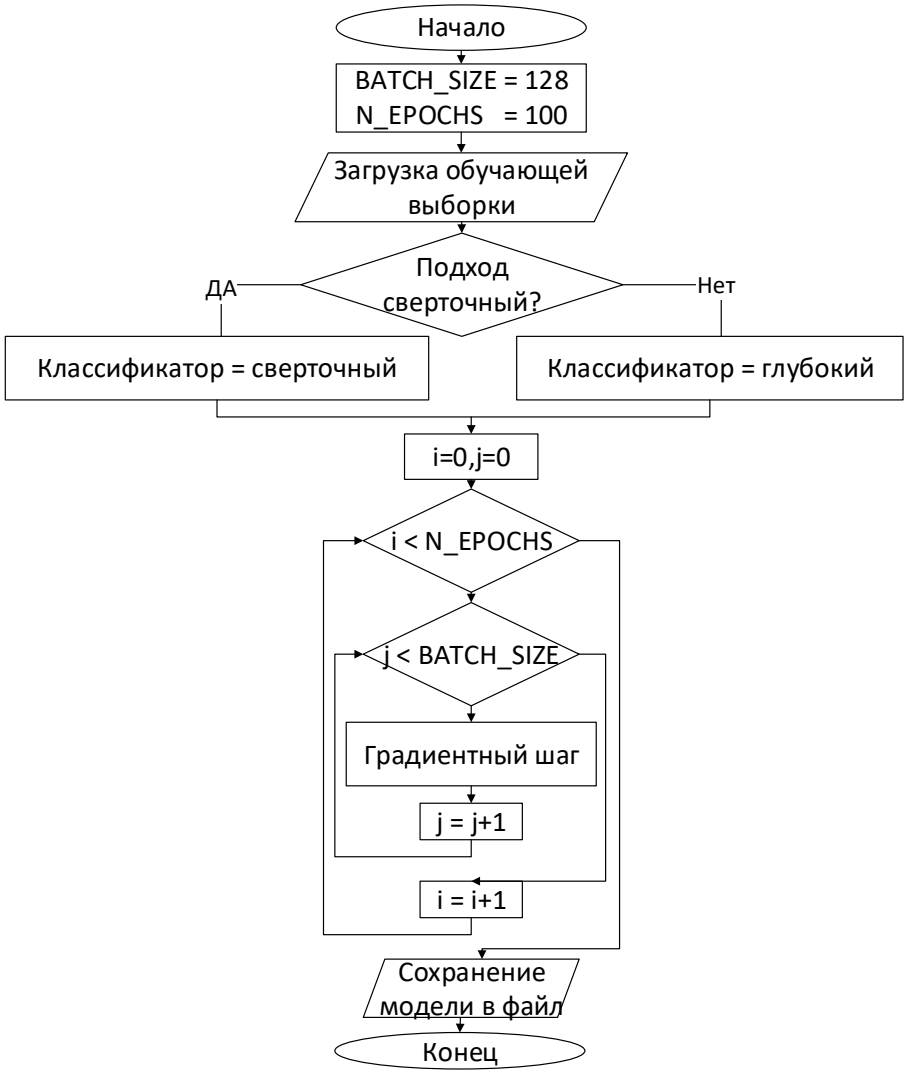


Рисунок 2.9 – Блок схема функции обучения сети

Таким образом основная функциональная схема приложения включает в себя модули «*init*», «*train*», «*predict*», «*read_dataset*», «*compress_data*». В общем виде блок-схема работы приложения представляет собой вечный цикл, в котором происходит обработка условий – выбор пользователем того или иного действия:

– узнать сорт по файлу с исходным спектром;

– обучить модель на новых данных.

На рисунке 2.10 изображена функциональная схема работы приложения.

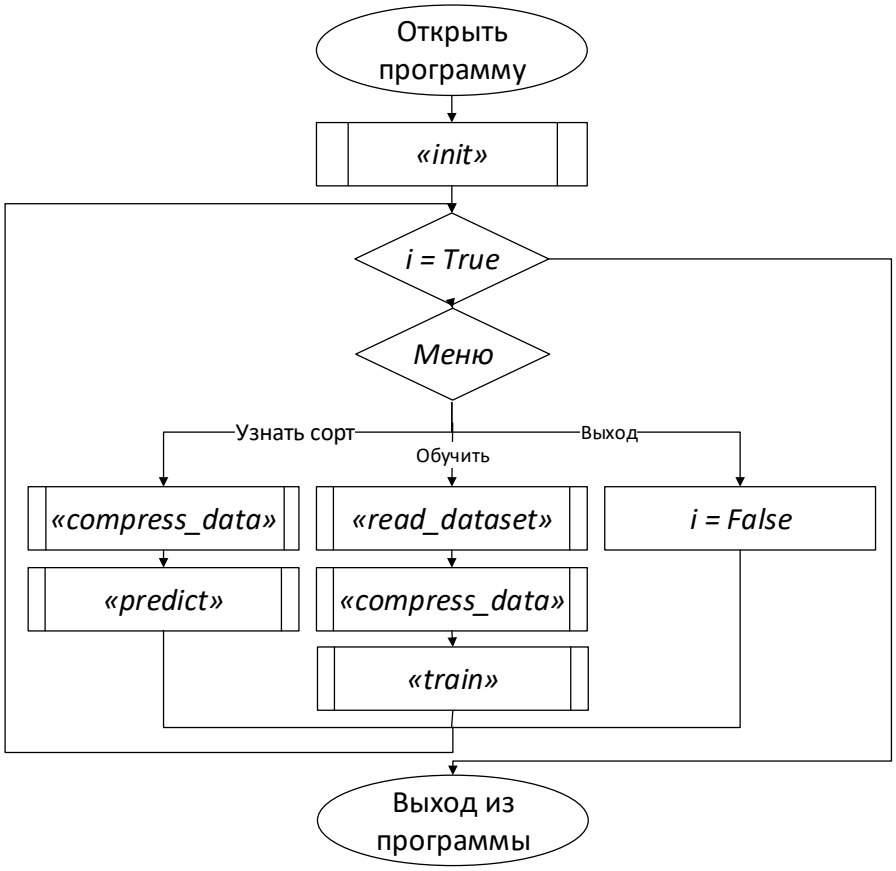


Рисунок 2.10 – Функциональная схема приложения

По разработанной функциональной схеме, нужно написать приложение с графическим интерфейсом пользователя. Приложение должно быть кроссплатформенным, включать в себя реализацию работы алгоритмов выбранных моделей. Пользователь должен иметь возможность выбора одного из подходов для классификации. Процесс обучения должен иметь визуальную составляющую, иллюстрирующую падение ошибки и рост эффективности с увеличением числа эпох обучения. Организовать проверку правильно вводимых данных, чтобы избежать *DDoS*-атак или отказа приложения.

3 РАЗРАБОТКА ВЕБ-ПРИЛОЖЕНИЯ ДЛЯ КЛАССИФИКАЦИИ СОРТОВ ТВЕРДОЙ ПШЕНИЦЫ

3.1 Микросервисный подход для разработки веб-приложения

При классическом подходе разработки приложения выбирается фреймворк, внутри которого реализуются все компоненты программного комплекса. Идея микросервисной архитектуры заключается в модульности: для каждого компонента создается отдельное приложение и подбираются нужные инструменты. Микросервисы – это когда каждая компонента программного комплекса работает в своем отдельном потоке и компоненты общаются через *REST API*.

REST (*Representational State Transfer*) – передача состояния представления, архитектурный стиль взаимодействия компонент в распределенных системах. Как правило, в роли *REST API* выступают обычные *HTTP*-запросы.

На рисунке 3.1 представлено сравнение классической монолитной архитектуры приложения и микросервисного подхода.

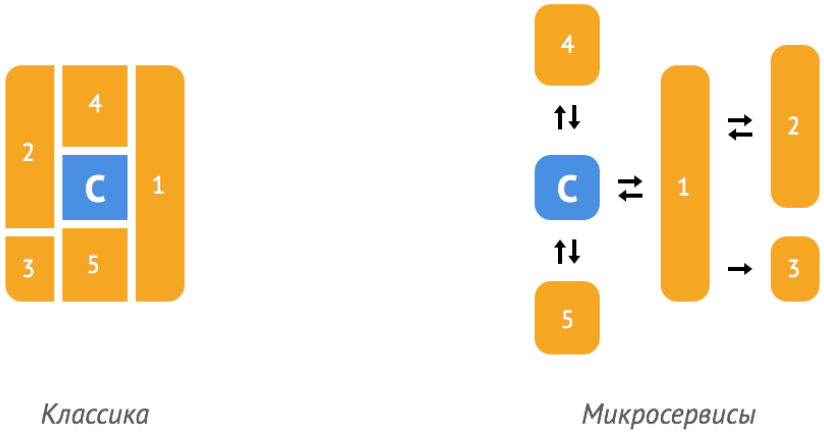


Рисунок 3.1 – Сравнение подходов к разработке приложения

Программный комплекс, разработанный в ходе выполнения дипломной работы можно разделить на несколько структурных блоков:

– ядро приложения, включающее серверную и клиентскую часть с веб-интерфейсом;

– модуль для обучения модели на новых данных;

– сервис, отвечающий за все вычисления, производимые нейронными сетями;

– программа, отвечающая за отрисовку процесса обучения сети;

– модуль, выполняющий обработку и распознавание входного спектра.

Можно отобразить взаимосвязь всех блоков приложения в виде структурной схемы, представленной на рисунке 3.2.

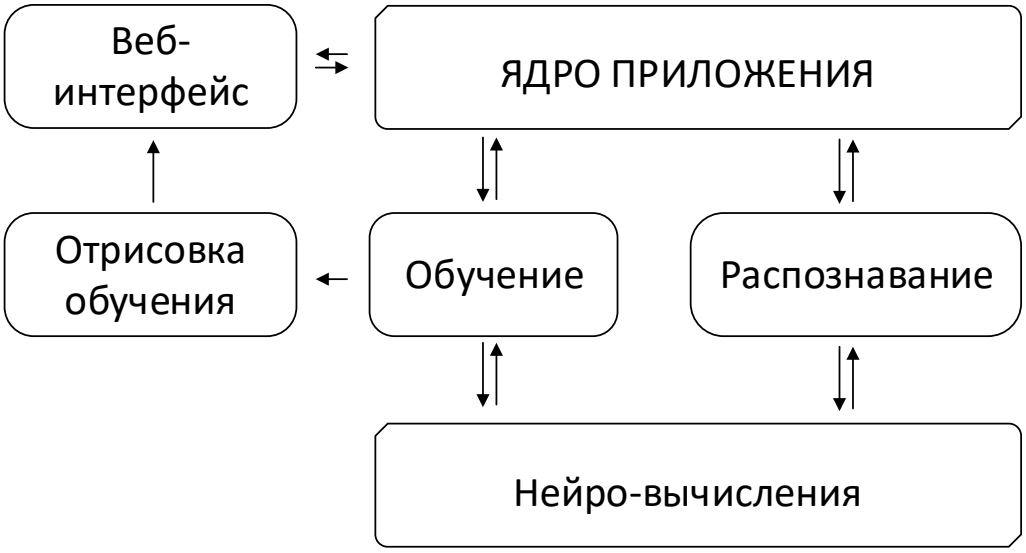


Рисунок 3.2 – Структурная схема приложения. Микросервисы

Каждый модуль отвечает за свои функции. Это позволяет избежать прерывания главного потока во время выполнения, например, операции обучения. Таким образом, пользователь может продолжать работу с программой, пока в фоне производятся необходимые вычисления. Также это позволяет избежать падения основного приложения в случае ошибки на одном из микросервисов – просто потеряется связь с основным узлом программного комплекса.

Преимущества микросервисного подхода:

– четкое разделение задач каждого модуля: для реализации простых функций не нужно использовать сложный фреймворк;

– автономность работы отдельных сервисов: один модуль проще переписать или заменить, чем менять все приложение, при отключении одного микросервиса приложение продолжает работать;

– «каждому свое»: простым примером является случай, когда приложение должно иметь несколько видов интерфейсов для разных групп пользователей;

– удобство командной разработки: тестирование становится проще, когда приходится тестировать отдельно взятый сервис и возможно использование разных программных языков и подходов, когда все они независимы друг от друга.

Из минусов можно выделить только проблемы с распределенностью: ядро должно обеспечивать хорошую согласованность модулей.

3.2 Библиотеки машинного обучения *TensorFlow* и *Keras*

Для написания частей программы, отвечающих за процесс обучения, распознавания и сервис, выполняющий все нейровычисления, используются библиотеки машинного обучения *TensorFlow* [27] и *Keras* [28] – это библиотеки с открытым исходным кодом, хорошей документацией и большим сообществом разработчиков.

Библиотека машинного обучения *TensorFlow* представляет собой фреймворк для выполнения математических операций, использующий графы потоков данных. В качестве потоков данных выступают тензоры – многомерные массивы данных. Отсюда и название – «*TensorFlow*». Узлы в вычислительном графе представляют собой математические операции, а ребра графа служат для обозначения потоков тензоров между узлами. Функции *TensorFlow* поддерживают многопоточные вычисления: с использованием одного или нескольких *CPU* или *GPU*; распределенные вычисления. *TensorFlow* позволяет разрабатывать приложения для настольных устройств, серверов и даже мобильных устройств. *TensorFlow* имеет поддержку для таких языков программирования, как:

– *Python;*
– *C++;*
– *Java;*
– *Go*.

Сообщество ведет разработку *API* и для других языков программирования: *Rust, C#, Haskell, Julia, Ruby* и *Rust*.

Из преимуществ *TensorFlow*:

– скорость;
– мультипоточность;
– поддержка большого кол-ва языков;
– хорошая документация;
– высокоуровневые *API*, такие как *tf.layers*;
– возможность организации распределенных вычислений;
– поддержка большого числа известных моделей ИНС.

Все эти преимущества послужили в пользу выбора *TensorFlow* в качестве инструмента программирования алгоритмов ИНС.

Keras – это высокоуровневый *API* на основе таких фреймворков машинного обучения, как *TensorFlow* и *Theano*, написанный на языке *Python*.

Для того, чтобы начать работу с *TensorFlow* и *Theano*, нужно создать сессию вычислений. На рисунке 3.3 приведен пример инициализации сессии для *TensorFlow* и использование созданной сессии для *Keras*.

```
1. sess = tf.InteractiveSession()  # create session variable
2. K.tensorflow_backend.set_session(sess) # set as Keras session
3. init = tf.global_variables_initializer() # initialize all vars
4. sess.run(init) # run session
```

Рисунок 3.3 – Пример инициализации сессии вычислений

Именно переменная, хранящая сессию вычислений и передается в качестве аргумента в структуре микросервисов – это делается намеренно, чтобы не хранить множество сессий в оперативной памяти, а использовать только одну. На рисунке 3.4 отображен пример, как использовать сессию, переданную в качестве аргумента функции.

```
1. with sess.graph.as_default():
2.     x_compressed = sess.run(ae['z'], feed_dict={ae['x']:[x]})
```

Рисунок 3.4 – Пример использования сессии, переданной в качестве аргумента функции

В примере выше запрограммирован запуск сессии с входными данными [x] для вычисления значения $ae[‘z’]$, где ae – это словарь, содержащий параметры автоэнкодера, а параметр z – это вектор извлеченных из данных главных компонент.

Все параметры внутри вычислительной сессии представляют собой графы или операции над графами, поэтому, чтобы узнать значение какой-то переменной, нужно произвести запуск сессии, указав, значение какой именно переменной требуется узнать.

Почти все модели ИНС были написаны с помощью функций библиотеки *Keras*, которая использует операции *TensorFlow* в качестве базы. Однако *Keras* не поддерживает возможность связывания весов, хоть и позволяет передавать матрицу коэффициентов в качестве параметра для построения слоя. В связи с этим, автоассоциативная сеть автоэнкодер с тремя скрытыми слоями была запрограммирована на *TensorFlow*. Стоит отметить, что это не влияет на производительность, так как *Keras* – это всего лишь надстройка. Использование *Keras* в качестве инструмента построения нейро-архитектуры обусловлено простым и понятным синтаксисом библиотеки, а также возможностью конфигурирования архитектуры сети путем добавления модулей – новых слоев.

В подразделе приведены только примеры программного кода, весь листинг программы содержится в приложении Б.

3.3 Облачные вычисления для ускорения процесса обучения нейросетей

Как правило, процесс тренировки ИНС является очень длительным. Не каждый разработчик имеет под рукой вычислительную мощь суперкомпьютера, а стоимость процессорного времени на таких машинах является очень дорогой. Все это может существенно повлиять на повышение цены на разработку программного продукта.

Комбинирование двух различных нейросетей: автоэнкодера и сети-классификатора позволяет выполнять предварительное сжатие данных. Благодаря этому, обучение и скорость обработки данных в процессе классификации существенно возрастает. Однако прежде чем выполнять сжатие, нужно обучить сеть автоэнкодер.

Разработка программного устройства велась на базе ноутбука *Dell Vostro* 5480 со следующими техническими характеристиками:

– ЦП*: Intel Core i*3-4500*U* @ 1.70 ГГц *(4 CPU)*;

– ОЗУ*:* 4096 Мбайт;

– Видеокарта*: Intel HD Graphics* 4400.

Длительность одной эпохи обучения глубокого автоэнкодера составляет порядка 70 – 80 секунд, что в общей сложности составляет 6 часов. Это не сильно критичная цифра, если обучить модель один раз. Однако при настройке параметров, при их переборе, приходится многократно обучать модель, что еще больше увеличивает время, затрачиваемое на разработку.

Чтобы оптимизировать процесс разработки, было принято решение обратиться за помощью к облачным сервисам. Все крупные компании, такие как *Amazon*, *Google*, *Microsoft* имеют свои облачные сервисы, однако для пользования ими нужно указывать данные своей кредитной карты даже в случае получения студенческой подписки.

Для ускорения обучения было принято решение воспользоваться бесплатной облачной платформой – *Floydhub* [29]. *Floydhub* позволяет обучать модели в облаке с использованием графических ускорителей *Nvidia Tesla K80,* которые имеют 4992 ядра *CUDA* для параллельных вычислений.

Floydhub поддерживает запуск и обучение моделей написанных с применением различных популярных библиотек машинного обучения, таких как:

– *TensorFlow;*

– *torch;*

– *theano;*

– *Keras;*

– *Caffe.*

Скорость выполнения одной эпохи обучения глубокого автоэнкодера на *Floydhub* в параллельном *GPU*-режиме составила две десятых секунды, что в 300 раз быстрее, чем обучение с использованием вычислительной мощности четырехъядерного *Intel Core-i*3.

Floydhub предлагает пользователям возможность следить за ходом выполнения всех задач, запущенных в облаке, а также просматривать их статус. Сделать это можно, если войти в систему под своим аккаунтом и перейти на страницу *https://www.floydhub.com/experiments*. После того, как пользователь перейдет по этому адресу, он может увидеть панель управления запущенными задачами – скриншот продемонстрирован на рисунке 3.5.

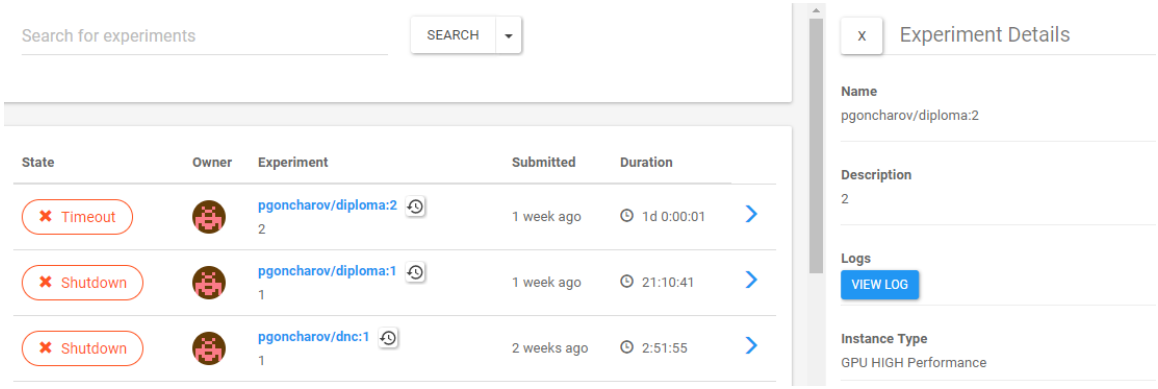


Рисунок 3.5 – Панель управления задачами, запущенными в облаке

Каждая отдельно взятая задача запускается в своем *docker*-контейнере. *Docker* – это открытая платформа для развертывания приложений. Каждый контейнер изолирован и создается из образа – это шаблон, содержащий нужную операционную систему и программное обеспечение для запуска приложения. Образы являются компонентами сборки в *docker*.

Все образы можно скачать из реестра, например, открытый реестр *Docker Hub*. Можно создавать свои образы, а также использовать те, что созданы другими.

Из образа создается контейнер. Контейнер, в простом понимании, – папка, директория, в которой хранится образ и все файлы приложения. Как было сказано выше, все контейнеры изолированы. Образ операционной системы – это *read-only* шаблон, что означает, что его нельзя изменять, только читать.

При каждом изменении приложения создается новый уровень над образом. Такие изменения наслаиваются друг на друга и взаимодействуют посредством *union file system* – объединенной файловой системы. Тогда при каждом изменении меняется только отдельно взятый уровень, а не вся система.

3.4 Разработка интерфейса пользователя

Как было сказано ранее, программный комплекс имеет микросервисную архитектуру. Ядром программы является веб-приложение, вся логика работы которого находится на стороне сервера, а клиентская часть представляет собой веб-страницу. Общение с сервером происходит путем передачи *http* запросов. Веб-приложение написано с использованием микрофреймворка *Flask*.

Flask – это фреймворк для создания веб-приложений на языке *Python*, использующий шаблонизатор *Jinja2*.

Шаблонизатор представляет собой программное обеспечение, основная цель которого отделить программный код от представления. *Jinja2* позволяет встраивать в шаблон веб-страницы программный код. Вставки кода заключаются в двойные фигурные скобки и во время компиляции, движок обрабатывает места с программным кодом. Использование шаблонизатора позволяет улучшить удобочитаемость кода и ускоряет процесс разработки.

На *Flask* также написан модуль, отвечающий за отрисовку процесса обучения и представляет собой отдельный микросервис, получающий данные в процессе обучения. Схема взаимодействия была изображена на рисунке 3.2.

После запуска веб-приложения, открывается окно в браузере с главным меню программы. На рисунке 3.6 представлен скриншот главного окна программного комплекса.

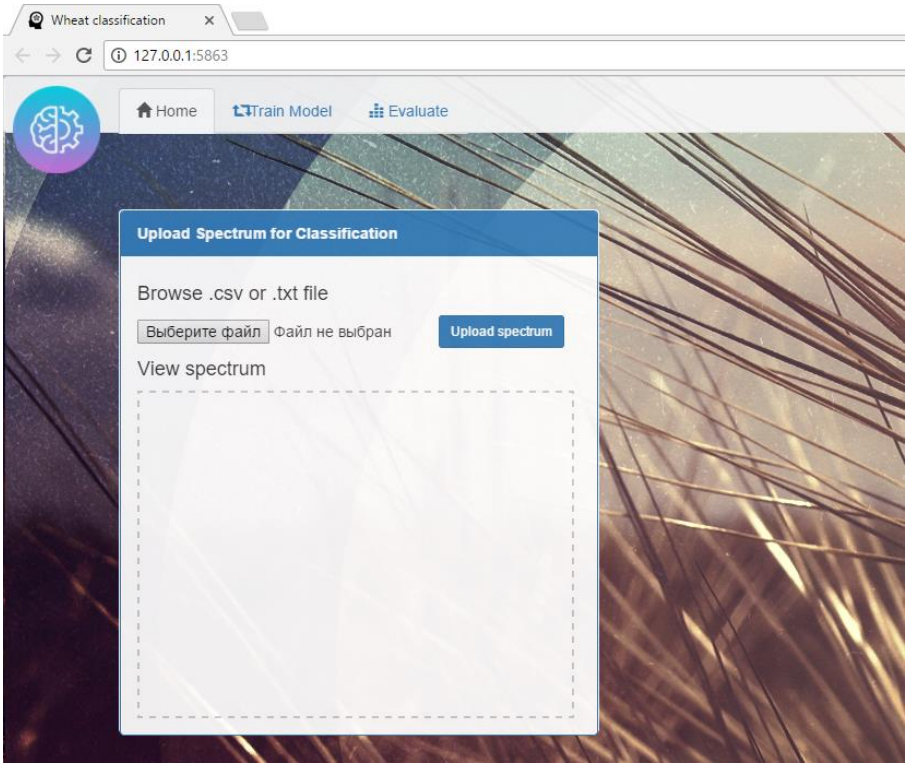


Рисунок 3.6 – Главное окно разработанного приложения

Как можно сразу заметить, страница имеет англоязычный вариант. Это обусловлено тем, что рассчитано на то, что приложением будет пользоваться не только русскоговорящий сегмент – английский язык имеет статус международного.

На главной странице содержится форма загрузки исходного файла со спектром для распознавания. Сверху расположено меню приложения. Оно имеет три страницы:

– главная страница (*Home*);

– страница для обучения нейронной сети на новых данных (*Train*);

– страница, отображающая сравнительный анализ качественных показателей двух использованных подходов распознавания (*Evaluate*).

Приложение В содержит подробное руководство для пользователя программы, а приложения Г и Д – руководства программиста и системного программиста, соответственно. Все основные функции и варианты использования программного обеспечения подробно описаны в этих приложениях, поэтому не требуют детального описания в данной главе.

Если пользователь произведет действие по загрузке исходного файла для распознавания сорта, следуя инструкции в приложении В, то он увидит результат, представленный на рисунке 3.7.

Рисунок 3.7 – Скриншот экрана приложения после выполнения процедуры распознавания

Если загруженный файл прошел проверку на корректность данных, то на графике в окне «*View spectrum*» отобразится загруженная денситограмма, а справа появится надпись с номером распознанного сорта в базе.

Вторая страница, пункт меню «*Train*», предназначена для загрузки пользователем папки, в которой содержится база для обучения нейронной сети, выбора метода обучения и непосредственного обучения классификатора на новых данных. На рисунке 3.8 изображена страница «*Train*».

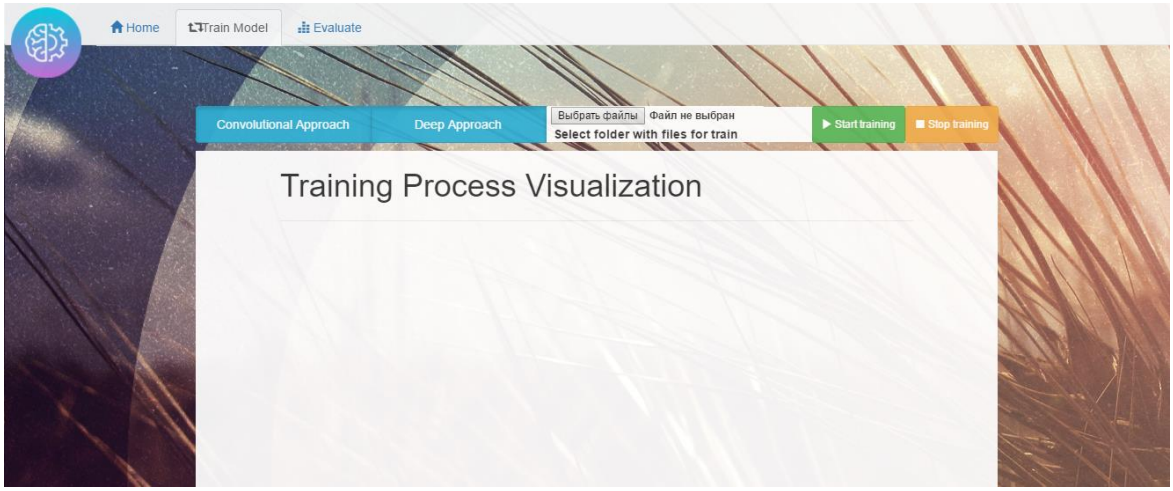


Рисунок 3.8 – Страница веб-приложения для обучения классификатора на новых данных

Так как процесс обучения еще не запущен, окно визуализации процесса тренировки сети пока что пустое. На рисунке 3.9 детально отображены пункты меню обучения, которые располагаются над окном визуализации.

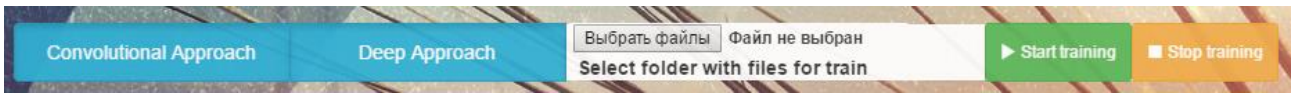


Рисунок 3.9 – Меню для запуска процесса обучения

Меню обучения представляет собой несколько кнопок:

– «*Convolutional Approach*» для выбора сверточной сети классификатора;

– «*Deep Approach*» для выбора глубокой сети классификатора;

– кнопка выбора папки с файлами для обучения: «выбрать файлы»;

– «*Start training*» для запуска процедуры тренировки сети;

– «*Stop training*» для остановки процесса обучения.

Если пользователь загрузит папку с исходными файлами для обучения: файлы с расширением *txt*, где в каждом файле через пробельную строку содержатся 4096 отсчетов электрофореграммы, – выберет метод обучения и нажмет на кнопку начала процедуры тренировки сети, то после загрузки обучающей выборки на сервер, в окне визуализации начнет отрисовываться график, представленный на рисунке 3.10.

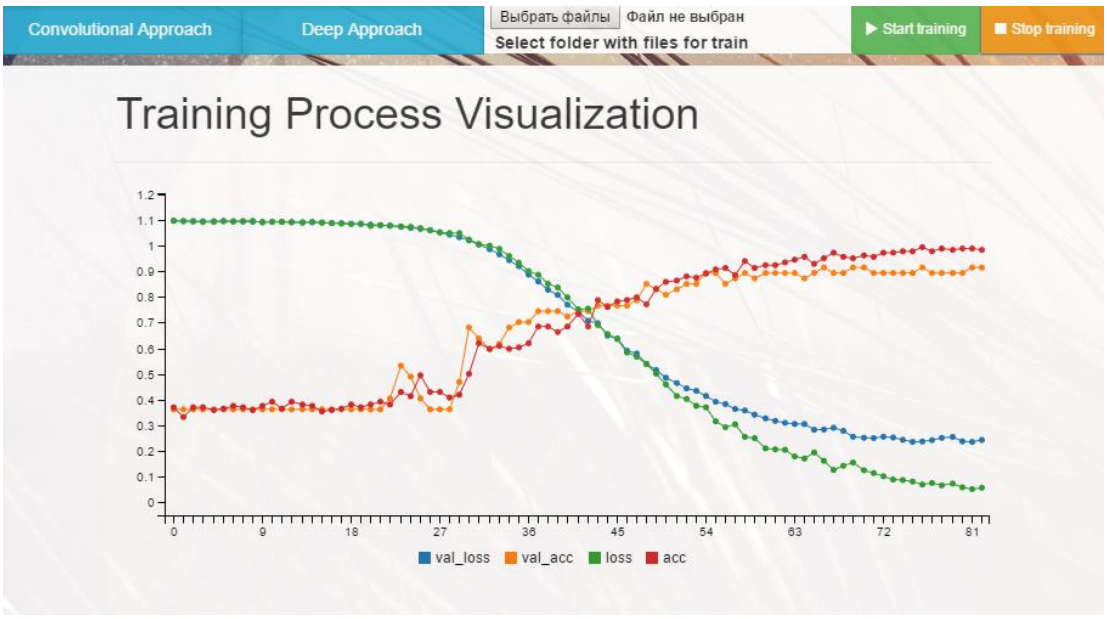


Рисунок 3.10 – Визуализация процедуры обучения

График отображает, как падает ошибка на тестовой и тренировочной выборке с каждой эпохой и как растет эффективность классификации с увеличением числа эпох. На рисунке 3.10 представлены следующие обозначения:

– «*val_loss*» означает ошибку на тестовой выборке;

– «*val_acc*» представляет собой эффективность на тестовой выборке;

– «*loss*» означает ошибку на обучающей выборке;

– «*acc*» является эффективностью классификации для выборки обучения.

Чтобы посмотреть значения кривых на отдельно взятой эпохе, нужно курсор мыши в нужное место на графике и в небольшой табличке рядом с курсором будут отображены значения на выделенной итерации процедуры обучения. Пример этого представлен на рисунке 3.11.

Рисунок 3.11 – Отображение значений кривых обучения

Последняя страница приложения «*Evaluate*» иллюстрирует сравнение качественных характеристик моделей из двух разных подходов. При этом график в конце страницы показывает эффективность распознавания, которая одинаково верна для двух подходов при указанных параметрах в таблицах на странице. На рисунке 3.12 изображена третья страница приложения.

Рисунок 3.12 – Страница «*Evaluate*» разработанного веб-приложения

Как можно видеть из таблицы на скриншоте, эффективность классификации даже для 50 сортов составляет около 90%, в то время, как авторы подходов, описанных в первой главе [5, 17, 12] добивались такой точности только для 10 – 15 сортов.

4 ПОДБОР ПАРАМЕТРОВ И ТЕСТИРОВАНИЕ МОДЕЛИ КЛАССИФИКАЦИИ. ВЕРИФИКАЦИЯ РАБОТЫ ПРИЛОЖЕНИЯ

4.1 Подбор параметров модели

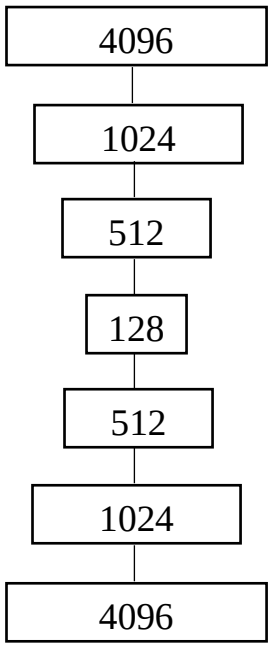
Нейросети имеют много настраиваемых параметров, которые влияют на работу сети. Для того, чтобы определить, какое число слоев и нейронов на скрытом слое необходимо сети, был проведен ряд экспериментов.

Сжатие исходных данных. В качестве метода компрессии использовалась автоассоциативная нейронная сеть с тремя скрытыми слоями, описанная в разделе 2.1.

Эксперимент 1. Так как размерность входного вектора равна 4096, то было принято решение о плавном и постепенном снижении размерности среднего слоя сети. Для достижения плавного снижения размерности, использовалась ИНС с 5 скрытыми слоями. В таблице 4.1 представлены параметры сети.

Батч – это партия, часть набора данных, подаваемая на вход ИНС. Обучение по батчам используется для минимизации затрат на хранение всей выборки в оперативной памяти и повышает скорость работы алгоритма в следствие того, что не приходится рассчитывать ошибку для всего набора исходных данных – используются только локальные градиенты.

Таблица 4.1 – Параметры ИНС в первом эксперименте

Блок-схема сети	Параметр	Значение
	Число эпох	300
	Размер батча	128
	Функция активации	*ReLU*
	Алгоритм оптимизации	*Adam*
	Функция ошибок	*SSE*

Сходимость при обучении на всех имеющихся спектрах очень медленная. После пятидесятой эпохи ошибка стабилизируется и перестает уменьшаться с прежней скоростью, но остается все такой же высокой. Сумма квадратичных ошибок после обучения равняется 3452. На рисунке 4.1 представлены результаты восстановления спектров с помощью ИНС, описанной выше. На каждой из картинок *a*) – исходный спектр, а *b*) – его восстановленный сетью вариант.

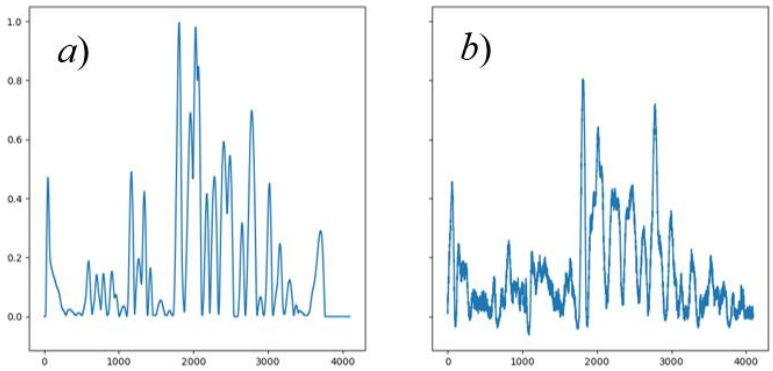


Рисунок 4.1 – Пример восстановленного спектра с помощью ИНС с пятью скрытыми слоями

Эксперимент 2. В ходе второго эксперимента было принято решение вернуться к модели с тремя скрытыми слоями. Были убраны слои с 512-ю нейронами и количество извлекаемых компонент увеличено до 256. Сокращение размерности происходило в пропорциях 4 к 1 – каждый скрытый слой вплоть до среднего имел в 4 раза меньше нейронов, чем предыдущий. Остальные параметры сети остались такими же, как в таблице 4.1.

На рисунке 4.2 представлены примеры восстановления спектров с помощью сети с тремя скрытыми слоями (*a, c* – исходные *b, d* – восстановленные сетью спектры).

При использовании только 3 скрытых слоев сумма квадратичных ошибок падает на два порядка, в сравнении с вариантом в первом эксперименте. В ходе данного эксперименты данные для обучения и тестов были сжаты и преобразованы в вектора из 256 главных компонент. Степень сжатия 6,25% от исходного размера данных. Число параметров сети: 1 075 053 568.

Для того, чтобы добиться лучшего сжатия при меньшем числе параметров следует использовать сверточную сеть в качестве автоэнкодера.

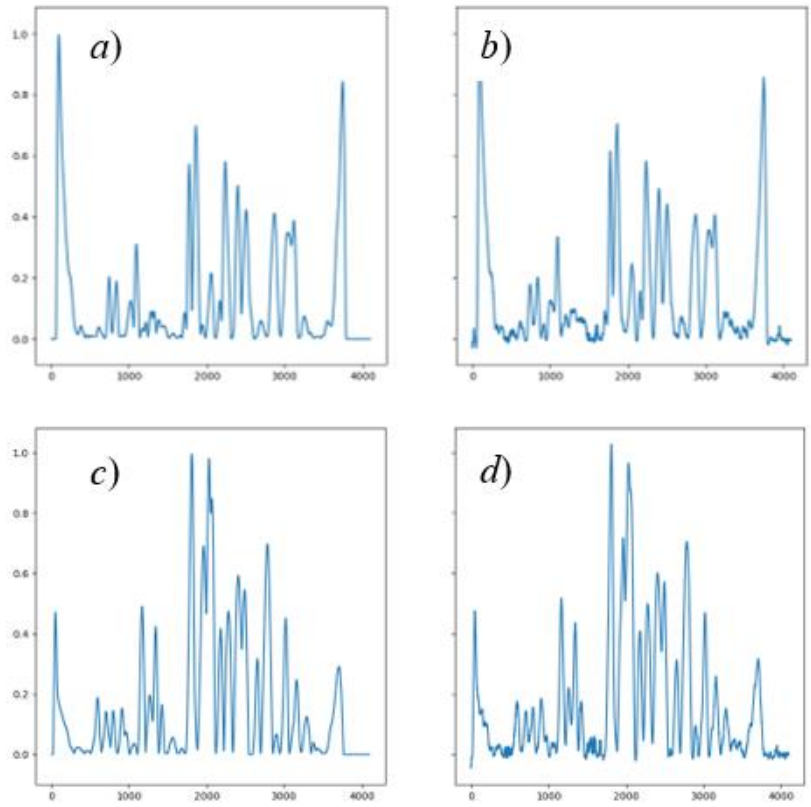


Рисунок 4.2 – Примеры восстановленных спектров с помощью ИНС с тремя скрытыми слоями

Эксперимент 3. Для третьего эксперимента использовалось два варианта сверточного автоэнкодера с разными функциями ошибок: среднеквадратичное отклонение (4.1) и бинарная кросс-энтропия (4.2).

$$y = \frac{1}{n}\sum_{p=1}^{n}\sum_{i=1}^{m}(Y_i - Y_i^{'})_p^2, \tag{4.1}$$

$$y = \frac{1}{n}\sum_{i=1}^{n}\left(y_i\, log(\,h_\theta(\,x_i\,)) + (1-y_i\,)log(1-h_\theta(\,x_i\,))\right), \tag{4.2}$$

где h_θ – это функция активации.

Каждый слой одномерных сверток не уменьшал размерность – это было достигнуто за счет добавления нулей по краям денситограмм – пэддинг – и использования 32 фильтров с размером окна 3 и шагом 1. Слои макспулинга выполняли снижение размерности до $\frac{1}{4}$ от входного вектора. Для увеличения размерности после среднего слоя, использовались слои апсемплинга (обратный пулинг) с аналогичными пулингу параметрами. В таблице 4.2 представлена топология сетей.

Таблица 4.2 – Топология сверточного автоэнкодера

Тип слоя	Размерность на выходе	Число параметров
Input	4096 x 1	0
Conv1D	4096 x 32	128
MaxPooling1D	1024 x 32	0
Conv1D	1024 x 32	3104
MaxPooling1D	256 x 32	0
Conv1D	256 x 32	3104
UpSampling1D	1024 x 32	0
Conv1D	1024 x 32	3104
UpSampling1D	4096 x 32	0
Conv1D	4096 x 1	97
Всего параметров:	9 537	

Общее число параметров сверточной ИНС меньше в 100 тысяч раз, по сравнению с сетью во втором эксперименте, что дает преимущество в скорости обучения и уменьшает расходы на хранение сети в оперативной памяти.

На рисунках 4.3 и 4.4 представлены варианты восстановления денситограмм с помощью сверточных автоассоциативных ИНС. На рисунке 4.3 отображен спектр, восстановленный после обучения сети с среднеквадратичной функцией ошибки, а 4.4 – с бинарной кросс-энтропией в качестве ошибки.

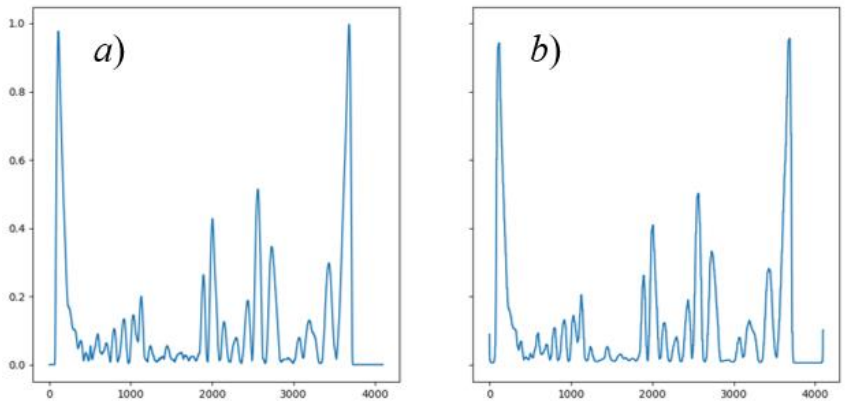


Рисунок 4.3 – Спектр, восстановленный сверточной ИНС с среднеквадратичной функцией ошибок

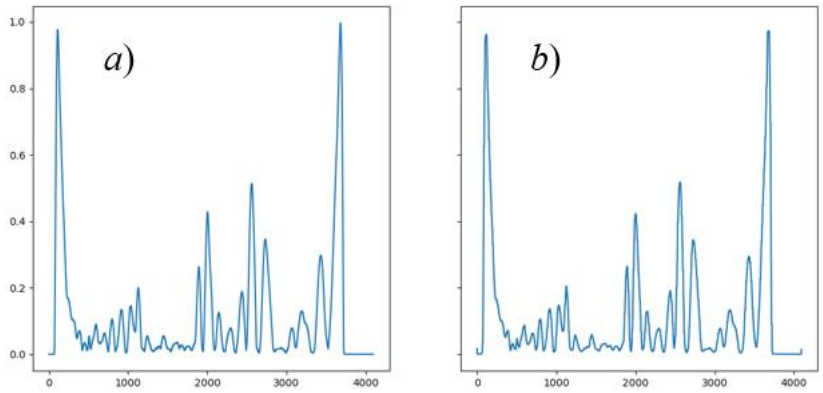


Рисунок 4.4 – Спектр, восстановленный сверточной ИНС с бинарной кросс-энтропией

Было замечено, что в некоторых местах спектр, восстановленый сетью с среднеквадратичным отклонением в качестве ошибки, представлен с более гладкими кривыми, а мелкие спектры и вовсе усекаются. Для того, чтобы не потерять информацию при сжатии, был выбран вариант автоэнкодера с бинарной кросс-энтропией в качестве функции потерь.

На среднем слое такой сети, мы получаем информацию, кодированную в виде активации 2 фильтров. Данные были сжаты в восемь раз по сравнению с исходным размером. Число параметров модели всего 315, что в три с половиной миллиона раз меньше, чем у автоассоциативной сети с тремя скрытыми слоями.

Для подбора оптимальных параметров сети, был проведен ряд экспериментов, в которых изменялось число слоев, размеры сверток и пулинга и был выбран наилучший вариант топологии, представленный в таблице 4.3.

Таблица 4.3 – Топология сверточного автоэнкодера

Тип слоя	Размерность на выходе	Число параметров
1	2	3
Input	4096 x 1	0
Conv1D	4096 x 8	32
MaxPooling1D	1024 x 8	0
Conv1D	1024 x 4	100
MaxPooling1D	256 x 4	0
Conv1D	256 x 2	26
Conv1D	256 x 4	28
UpSampling1D	1024 x 4	0

Продолжение таблицы 4.3

1	2	3
Conv1D	1024 x 8	104
UpSampling1D	4096 x 8	0
Conv1D	4096 x 1	25
Всего параметров:	315	

Автоэнкодер с топологией, приведенной в таблице 4.3, бинарной кросс-энтропией в качестве функции ошибок и размером батча – 128, сходится уже после 50 эпох обучения, что гораздо быстрее, чем 300 эпох для автоассоциативной сети с 3 скрытыми слоями.

Классификация. В ходе проведения экспериментов со сжимающими сетями было создано две нейронных сети автоэнкодера: одна – глубокая сеть с тремя скрытыми слоями, а другая – сверточная. Для того, чтобы произвести классификацию, требуется создать еще две сети: глубокую и сверточную, – сверточный классификатор нужен, так как глубокая сеть не может принять на вход двумерный массив данных, полученный при компрессии сверточным автоэнкодером.

Основными исследуемыми параметрами для построения сверточной сети-классификатора является число слоев, количество и размер фильтров сверток, размер окна макспулинга. Также важно наличие обычных слоев нейронов для того, чтобы перейти от многомерного представления данных в сети к вектору на выходе.

Чтобы не терять никакой информации при выполнении операции свертки, решено было, как и в случае с автоэнкодером, использовать пэддинг, окно свертки размером в 3 единицы и шаг равный 1. Основная мощь сверточных сетей заключается в том, чтобы плавно уменьшать размерность данных с помощью пулинга и одновременно увеличивать глубину за счет добавления новых фильтров – благодаря этому не происходит потеря информации при сокращении параметров.

Отдельное внимание следует уделить операции *dropout* [30]. *Dropout* – это «выбрасывание» определенного количества случайно выбранных нейронов на слое. Например, если *dropout*=0.2, то исключаются 20% всех нейронов на слое.

Эксперимент 1. Первый эксперимент заключался в сравнении двух архитектур: *VGG*-подобная сеть [31] и с общепринятой топологией. *VGG*-подобная сеть отличается от обычной тем, что объединеняются сразу несколько сверточных слоев.

Качество классификации проверялось для 50 сортов в выборке, без предварительной обработки по поиску эталонных спектров каждого сорта и

кластеризации на подвыборки по 5-8 сортов как в работе [5]. Размер батча во всех экспериментах общий и равен 128, число эпох обучения – 100, метод оптимизации – *Adam*.

В таблице 4.4 представлена топология *VGG*-подобной сети.

Таблица 4.4 – Топология *VGG*-подобной сети

Тип слоя	Размерность на выходе	Число параметров
Input	256 x 8	0
Conv1D	256 x 8	200
Conv1D	256 x 8	200
MaxPooling1D	128 x 8	0
Dropout 0.25	128 x 8	0
Conv1D	128 x 16	400
Conv1D	128 x 16	784
MaxPooling1D	64 x 16	0
Dropout 0.25	64 x 16	0
Flatten	1024	0
Dense	256	262 400
Dropout 0.5	256	0
Dense	50	12 850
Всего параметров:	276 834	

Flatten – слой, производящий векторизацию многомерного массива.
Dense – слой обычных нейронов.

После окончания обучения эффективность классификации равна 79,8%. Одна тренировочная эпоха длилась порядка 2 секунд.

Второй вариант сети с общепринятой топологией. Отличие от сети, описанной выше, только в том, что нет парных сверточных слоев, идущих один за другим. Соответственно, число параметров уменьшилось до 270 710, скорость обучения выросла в два раза, при этом качество классификации на тестовой выборке – 81,4%. Сеть с обычной топологией показала эффективность выше, чем *VGG*-подобная сеть.

Эксперимент 2. Второй эксперимент – исследование параметров *Dropout*. Было выбрано несколько вариантов исключающих слоев. Результаты эксперимента представлены в таблице 4.5.

Таблица 4.5 – Подбор величины *Dropout*

Dropout слои			Эффективность классификации (%)
1 слой	2 слой	3 слой	
0	0	0	79
0,25	0,25	0,2	81,7
0,25	0,25	0	78,4
0	0	0,3	82
0	**0**	**0,5**	**82,8**
0	0	0,7	82

Жирным шрифтом отмечены значения в строке с самым высоким показателем эффективности. Стоит отметить, что без использования *Dropout* классификатор сильно переобучается, так как качество классификации на обучающей выборке достигает 100% для 50 сортов, но на тестовой выборке падает до 79%. Это вызвано особым свойством исключающих слоев, которое помогает предотвратить переобучение.

Как можно заметить из таблицы 5, применение *Dropout* не играет особой роли в случае расположения исключающего слоя после слоя пулинга. Это объясняется тем, что сама по себе операция пулинга ведет к сокращению вероятности переобучения, а вот использование *Dropout* целесообразно на том уровне, на котором происходит векторизация данных, в виду очень большого числа межнейронных связей – 1024 x 256.

Эксперимент 3. Третий эксперимент заключался в том, чтобы проверить, как поведет себя сеть, если добавить еще несколько скрытых слоев с той же величиной окна для свертки и пулинга.

Самый высокий показатель эффективности – 83% был достигнут при добавлении еще одного сверточного слоя и пулинга. Окончательный вариант топологии сверточного классификатора представлен в таблице 4.6.

После того, как была подобрана оптимальная топология и параметры сверточного классификатора, остается проверить эффективность классификации для разного числа сортов.

Таблица 4.6 – Топология сверточного классификатора

Тип слоя	Размерность на выходе	Число параметров
1	2	3
Input	256 x 8	0
Conv1D	256 x 8	200

Продолжение таблицы 4.6

1	2	3
MaxPooling1D	128 x 8	0
Conv1D	128 x 16	400
MaxPooling1D	64 x 16	0
Conv1D	64 x 32	1568
MaxPooling1D	32 x 32	0
Flatten	1024	0
Dense	256	262 400
Dropout 0.5	256	0
Dense	50	12 850
Всего параметров:	277 418	

Следующим этапом стало построение обычной глубокой сети с нормализованной функцией инициализации, на вход которой подаются данные сжатых с помощью пятислойной автоассоциативной сети денситограмм.

Размерность входных данных составляет вектор длиной 256. В ходе ряда исследований была определена максимально эффективная размерность и параметры сети. Классификатор содержит 4 скрытых слоя по 512, 1024, 512, и 256 скрытых нейронов, соответственно. После каждого из первых трех слоев установлен *Dropout* = 0,3. Все остальные параметры такие же, как в сверточном классификаторе.

Тестирование глубокой сети показало не менее высокие показатели эффективности, различие значений составляет от минус единицы до трех десятых процента в сравнении со сверточной сетью. Время, затраченное на одну эпоху обучения, равно 1 секунде. Но при всем этом, модель имеет 1 318 164 параметров, что почти в 5 раз больше, чем у сверточного аналога.

Следовательно, можно сделать вывод, что несмотря на то, что сокращение размерности с помощью глубокой сети автоэнкодера является достаточно эффективным способом, но все же уступает сверточным сетям по качеству сжатия, скорости обучения и числу параметров моделей. Ко всему прочему, сверточная сеть классификатор может быть использована без автоэнкодера.

4.2 Верификация программного обеспечения. Оценка эффективности классификации

Программный комплекс для распознавания сортов твердой пшеницы по электрофоретическим спектрам должен включать в себя функцию загрузки и

распознавания спектра. Пункт меню, отвечающий за обучение модели на новых данных. Модель, предложенная для решения задачи классификации, должна иметь высокие показатели эффективности распознавания: выше, чем у аналогов, описанных в главе 1, – и высокую скорость обработки данных.

Чтобы проверить, насколько хорошо работает обученный классификатор, загрузим спектр, принадлежащий сорту под номером 11. На рисунке 4.5 представлен результат классификации спектра сорта номер 11.

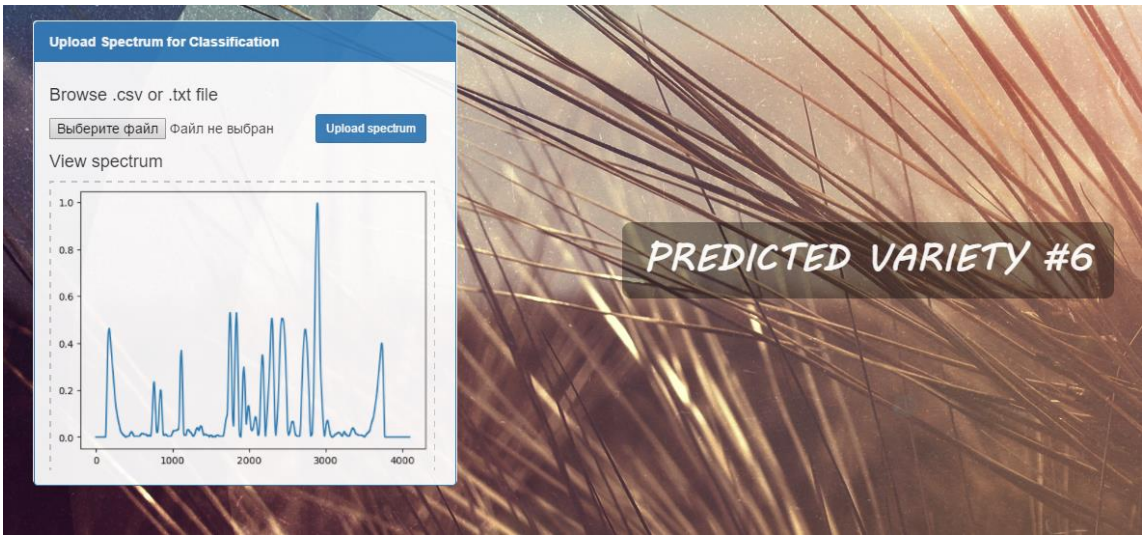


Рисунок 4.5 – Результат классификации спектра сорта под номером 11

Ответ сети соответствует действительности, обработка заняла порядка 2 секунд. Если же ввести неверный файл – файл другого расширения, файл с размером больше 200 Кбайт или файл с неверными данными внутри (верными считаются данные, содержащие 4096 значений целочисленного или типа с плавающей точкой), то файл не будет загружен, а пользователь увидит сообщение об ошибке, представленное на рисунке 4.6.

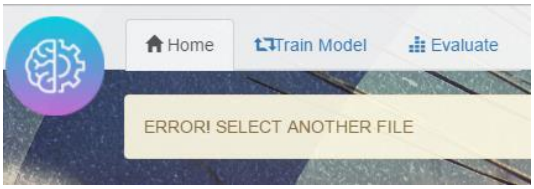


Рисунок 4.6 – Сообщение об ошибке при выборе неверного файла

Однако проверить эффективность классификации для большого числа сортов не представляется возможным, если загружать по одному файлу – это будет очень долго. Специально для этого микросервис, отвечающий за классификацию, был протестирован отдельно в облаке на большой тестовой

подвыборке. На рисунке 4.7 изображен график снижения показателей эффективности распознавания с ростом числа сортов. Этот график представлен на третьей странице разработанного веб-приложения.

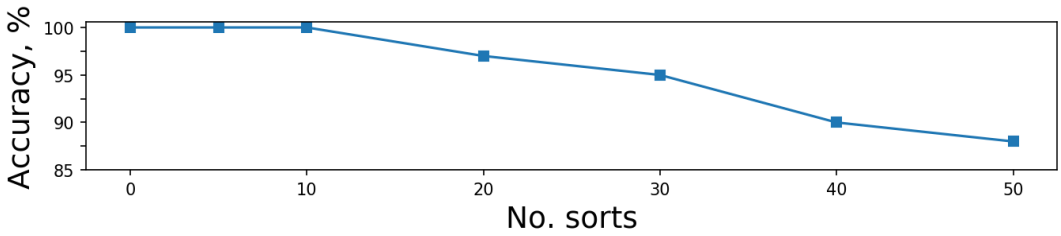


Рисунок 4.7 – График, иллюстрирующий эффективность для разного числа сортов

Как можно видеть из рисунка выше, эффективность распознавания для количества сортов менее десяти составила 100%. Авторы работы [5] добились таких же показателей, но их эксперименты показали, что эффективность резко падает после добавления новых сортов, в то время как данная модель ведет себя достаточно стабильно и обеспечивает высокий уровень верно распознанных спектров даже для 40 сортов.

Можно сказать, что разработанный программный комплекс отвечает поставленным требованиям:

– имеет высокий результат эффективности распознавания, превосходящий представленные в литературе аналоги;

– обеспечивает высокую скорость обработки входных данных;

– позволяет обучать классификатор на новых данных;

– имеет удобный веб-интерфейс;

– разработанное приложение является кроссплатформенным;

– предусмотрена система обработки ошибок.

Скриншоты разработанного программного комплекса находятся в приложении Ж.

5 ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ДИПЛОМНОЙ РАБОТЫ

5.1 Технико-экономическое обоснование целесообразности разработки программного комплекса для анализа и классификации сортов твердой пшеницы

Разработанный программный комплекс посвящен решению важной и весьма актуальной задачи – автоматизации трудоемкого процесса генетической классификации белков. Данная программа предназначена для научного применения в узкой сфере классификации белковых спектров, полученных путем применения метода электрофореза.

Главными критериями оценки приложения являются эффективность распознавания как можно большего числа сортов и скорость обработки входных данных – именно эти критерии повлияют на решение научных организаций при покупке данного программного обеспечения. Преимущество данной разработки заключается в использовании новейших алгоритмов искусственного интеллекта, глубоких нейронных и сверточных сетей, предложен новый подход, включающий в себя предварительную компрессию входных данных. Все это обеспечивает сверхбыструю скорость определения сорта и высокие результаты эффективности при большом числе сортов.

Единственными ресурсами для развития приложения являются вычислительные мощности, позволяющие быстро обучать и хранить модели нейронных сетей, а также новые базы данных. Функционал программы позволяет обучаться на других видах электрофореграмм, не только белковых спектрах твердой пшеницы, что расширяет круг возможных потребителей.

Ключевыми партнерами будут являться лаборатории генетики, организации, занимающиеся исследованиями в области хранения и выращивания новых культур и сортов растений.

В приложении К, таблицах К.1 – К.3, проведены расчеты конкурентно-способности разработанного программного комплекса, сделано сравнение с базовым аналогом (прямого аналога нет, ведется сравнение с методом, представленным в литературе и решающим аналогичную задачу). Как было сказано ранее, разработанная программа показывает наивысшие результаты по точности распознавания и скорости обработки данных. Научные программы имеют пресный ничем не выдающийся интерфейс с множеством кнопок и окон, в которых сложно разобраться, не прочитав многостраничный мануал, в то время как внешний вид разработанной программы удобен и понятен пользователю.

Продукт несет в первую очередь научную пользу, а значит основной идеей развития является доступ к возможностям приложения по платной подписке.

5.2 Оценка трудоемкости работ по созданию программного обеспечения

В качестве единицы измерения объема ПО может быть использована строка исходного кода (*LOC*). Общий объем ПО ($V_{\text{о}}$) определяется исходя из количества и объема функций, реализуемых программой, по каталогу функций ПО по формуле (5.1):

$$V_{\text{о}} = \sum_{i=1}^{n} V_i, \tag{5.1}$$

где n – общее число функций;

V_i – объём отдельной функции ПО.

Уточненный объем ПО ($V_{\text{у}}$) определяется по формуле (5.2):

$$V_{\text{у}} = \sum_{i=1}^{n} V_{\text{у}i}, \tag{5.2}$$

где $V_{\text{у}i}$ – уточненный объем отдельной функции ПО в строках исходного кода.

Результаты расчетов представлены в приложении К, таблица К.4.

Разработанное программное обеспечение по своим характеристикам относится ко второй категории сложности.

На основании принятого к расчету (уточненного) объема ($V_{\text{у}}$) и категории сложности ПО принимаем нормативную трудоемкость ПО выполняемых работ $T_{\text{н}} = 73$ чел.-дн.

Дополнительные затраты труда, связанные с повышением сложности разрабатываемого ПО, учитываются посредством коэффициента повышения сложности ПО ($K_{\text{с}}$), который определяем по формуле (5.3):

$$K_{\text{с}} = 1 + \sum_{i=1}^{n} K_i, \tag{5.3}$$

где n – количество учитываемых характеристик;

K_i – коэффициент, соответствующий степени повышения сложности.

Принимаем коэффициент повышения сложности ПО равным 1,26.

Влияние фактора новизны на трудоемкость учитывается путем умножения на соответствующий коэффициент, учитывающий новизну ПО ($K_{\text{н}}$). Разработанная программа имеет категорию новизны Б, а значение $K_{\text{н}} = 0{,}72$.

Степень использования в разрабатываемом ПО стандартным модулям определяется их удельным весом в общем объеме ПО. Коэффициент, учитывающий степень использования стандартных модулей в разработанном приложении равен $K_т = 0{,}65$.

Значение коэффициентов удельных весов трудоемкости стадий разработки ПО определяются с учетом установленной категории новизны ПО и приведены в таблице Д.5.

Общая трудоемкость разработки ПО ($Т_о$) определяется суммированием нормативной (скорректированной) трудоемкости ПО по стадиям разработки.

По результатам расчетов $Т_о = 76$ чел.-дн.

Результаты расчетов по определению нормативной и скорректированной трудоемкости программного обеспечения по стадиям разработки и общую трудоемкость разработки ПО ($Т_о$) представлены в приложении К, таблица К.3.

5.3 Расчет затрат на разработку программного продукта

Для расчета затрат на разработку (себестоимости) программного продукта определим необходимые параметры: тарифная ставка, ставка арендных платежей, стоимость ПК, стоимость кВт-час и занесем их в таблицу К.7.

Суммарные затраты на разработку ПО ($З_р$) определяются по формуле (5.4):

$$З_р = З_{тр} + З_{эт} + З_{тех} + З_{м.в} + З_{мат} + З_{общ.пр} + З_{непр}. \qquad (5.4)$$

Расходы на оплату труда разработчиков с отчислениями ($З_{тр}$) определяются по формуле (5.5):

$$З_{тр} = ЗП_{осн} + ЗП_{доп} + ОТЧ_{зп}, \qquad (5.5)$$

где $ЗП_{осн}$ – основная заработная плата разработчиков, руб.;

$ЗП_{доп}$ – дополнительная заработная плата разработчиков, руб.;

$ОТЧ_{зп}$ – сумма отчислений от заработной платы (социальные нужды, страхование от несчастных случаев), руб.

Основная ЗП разработчиков рассчитывается по формуле (5.6):

$$ЗП_{осн} = С_{ср.час} \cdot Т_о \cdot K_{ув}, \qquad (5.6)$$

где $С_{ср.час}$ – средняя часовая тарифная ставка, руб./час;

$Т_о$ – общая трудоемкость разработки, чел.-час, таблица Д.3;

$К_{ув}$ – коэффициент, учитывающий доплаты стимулирующего характера, примем $К_{ув} = 1{,}5$.

Средняя часовая тарифная ставка определяется по формуле (5.7):

$$С_{ср.час} = \frac{\sum_i С_{чi} \cdot n_i}{\sum_i n_i}, \quad (5.7)$$

где n_i – количество разработчиков *i*-й категории;

$С_{чi}$ – часовая тарифная ставка разработчика *i*-й категории, руб./час.

Часовая тарифная ставка определяется путем деления месячной тарифной ставки на установленный при восьмичасовом рабочем дне фонд рабочего времени ($F_{мес} = 168$) (5.8):

$$С_ч = \frac{С_{м1} \cdot Т_{к1}}{F_{мес}} \quad (5.8)$$

где $С_{м1}$ – тарифная ставка 1-го разряда;

$Т_{к1}$ – тарифный коэффициент.

$$С_{ср.час} = С_ч = \frac{31 \cdot 5{,}68}{168} = 1{,}05 \text{ руб./ч.}$$

$$ЗП_{осн} = 1{,}05 \cdot 76 \cdot 8 \cdot 1{,}5 = 957{,}6 \text{ руб. за 76 дней.}$$

Дополнительная заработная плата рассчитывается по формуле (5.9):

$$ЗП_{доп} = \frac{ЗП_{осн} \cdot Н_{доп}}{100\%}, \quad (5.9)$$

где $Н_{доп}$ – норматив на дополнительную заработную плату разработчиков (принимается в размере 7 – 15 %).

$$ЗП_{доп} = \frac{957{,}6 \cdot 10}{100} = 95{,}76 \text{ руб. за 76 дней.}$$

Отчисления от основной и дополнительной заработной платы (отчисления на социальные нужды и обязательное страхование) рассчитываются по формуле (5.10):

$$\text{ОТЧ}_{\text{с.н}} = \frac{(\text{ЗП}_{\text{осн}} + \text{ЗП}_{\text{доп}}) \cdot \text{Н}_{\text{з.п}}}{100\%}, \qquad (5.10)$$

где $\text{Н}_{\text{з.п}}$ – процент отчислений на социальные нужды и обязательное страхование от суммы основной и дополнительной заработной платы ($\text{Н}_{\text{з.п}}$ = 34%).

$$\text{ОТЧ}_{\text{с.н}} = \frac{(957{,}6 + 95{,}76) \cdot 34}{100} = 358{,}14 \text{ руб. за 76 дней.}$$

$$\text{З}_{\text{тр}} = 957{,}6 + 95{,}76 + 358{,}14 = 1411{,}5 \text{ руб. за 76 дней.}$$

Затраты машинного времени ($\text{З}_{\text{м.в}}$) определяются по формуле (5.11):

$$\text{З}_{\text{м.в}} = \text{С}_{\text{ч}} \cdot K_{\text{т}} \cdot t_{\text{эвм}}, \qquad (5.11)$$

где $\text{С}_{\text{ч}}$ – стоимость 1 часа машинного времени, руб./ч;

$K_{\text{т}}$ – коэффициент мультипрограммности, показывающий распределение времени работы ЭВМ в зависимости от кол-ва пользователей ЭВМ; $K_{\text{т}}$ = 1;

$t_{\text{эвм}}$ – машинное время ЭВМ, необходимое для разработки и отладки проекта, ч.

Стоимость машино-часа определяется по формуле (5.12):

$$\text{С}_{\text{ч}} = \frac{\text{ЗП}_{\text{об}} + \text{З}_{\text{ар}} + \text{З}_{\text{ам}} + \text{З}_{\text{э.п}} + \text{З}_{\text{в.м}} + \text{З}_{\text{т.р}} + \text{З}_{\text{пр}}}{F_{\text{эвм}}}, \qquad (5.12)$$

где $\text{ЗП}_{\text{об}}$ – затраты на заработную плату обслуживающего персонала с учетом всех отчислений, руб./год;

$\text{З}_{\text{ар}}$ – стоимость аренды помещения под размещение вычислительной техники, руб./год;

$\text{З}_{\text{ам}}$ – амортизационные отчисления за год, руб./год;

$\text{З}_{\text{э.п}}$ – затраты на электроэнергию, руб./год;

$\text{З}_{\text{в.м}}$ – затраты на материалы, необходимые для обеспечения нормальной

работы ПЭВМ (вспомогательные), руб./год;

$З_{т.р}$ – затраты на текущий и профилактический ремонт ЭВМ, руб./год;

$З_{пр}$ – прочие затраты, связанные с эксплуатацией ПЭВМ, руб./год;

$F_{эвм}$ – действительный фонд времени работы ЭВМ, час/год.

Все статьи затрат формируются в расчете на единицу ПЭВМ.

Годовые затраты на аренду помещения ($З_{ар}$) определяются по формуле (5.13):

$$З_{ар} = \frac{С_{ар} \cdot S}{Q_{эвм}}, \qquad (5.13)$$

где $С_{ар}$ – средняя годовая ставка арендных платежей, руб./м^2;

S – площадь помещения, м^2.

$$З_{ар} = 15{,}2 * 10 / 1 = 152 \text{руб.}$$

За 76 дней разработки на аренду необходимо потратить 31,56 руб.

Сумма годовых амортизационных отчислений ($З_{ам}$) определяется по формуле (5.14):

$$З_{ам} = \frac{\sum_i З_{пр i}(1 + K_{доп}) m_i \cdot Н_{ам i}}{Q_{эвм}}, \qquad (5.14)$$

где $З_{пр i}$ – затраты на приобретение i-го вида основных фондов, руб;

$K_{доп}$ – коэффициент, характеризующий дополнительные затраты, связанные с доставкой и наладкой оборудования, $K_{доп}$ = 12% от $З_{пр}$;

$З_{пр i} / (1 + K_{доп})$ – балансовая стоимость ЭВМ, руб;

$Н_{ам i}$ – норма амортизации, %;

$Q_{эвм}$ – количество обслуживаемых ПЭВМ, шт.

$$З_{ам} = 700 \cdot (1 + 0{,}12) \cdot 1 \cdot 0{,}2 = 156{,}8 \text{ руб.}$$

За 76 дней разработки амортизационные отчисления составят 32,6 руб.

Стоимость электроэнергии, потребляемой за год, определяется по формуле (5.15):

$$З_{эп} = \frac{М_{сум} \cdot F_{эвм} \cdot С_{эл} \cdot A}{Q_{эвм}}, \quad (5.15)$$

где $М_{сум}$ – паспортная мощность ПЭВМ, кВт; $М_{сум}$ = 0,5 кВт;

$С_{эл}$ – стоимость одного кВт-часа электроэнергии, руб, $С_{эл}$ = 0,1192;

A – коэффициент интенсивного использования мощности, A = 0,9.

Действительный годовой фонд времени работы ПЭВМ ($F_{эвм}$) рассчитывается по формуле (5.16):

$$F_{эвм} = (Д_{г} - Д_{вых} - Д_{пр}) \cdot F_{см} \cdot K_{см}(1 - K_{пот}), \quad (5.16)$$

где $Д_{г}$ – общее количество дней в году, 365 дней;

$Д_{вых}$, $Д_{пр}$ – число выходных и праздничных дней в году, 118 дней;

$F_{см}$ – продолжительность 1 смены, $F_{см}$ = 8 часов;

$K_{см}$ – количество рабочих смен ЭВМ, $K_{см}$ = 1;

$K_{пот}$ – коэффициент, учитывающий потери рабочего времени, связанные с профилактикой и ремонтом ЭВМ, примем $K_{пот}$ = 0,01.

$$F_{эвм} = (365 - 118) \cdot 8 \cdot 1 \cdot (1 - 0{,}1) = 1778{,}4 \text{ ч. в год.}$$

С учётом, что срок разработки программного продукта составляет 76 дней, действительный фонд времени работы ПЭВМ составляет 370,3 ч.

$$З_{эп} = 0{,}5 \cdot 1778{,}4 \cdot 0{,}1192 \cdot 0{,}9 = 95{,}4 \text{ руб.}$$

Следовательно, за 76 дней разработки расходуется 19,86 руб.

Затраты на материалы ($З_{в.м}$), необходимые для обеспечения нормальной работы ПЭВМ составляют около 1% от балансовой стоимости ЭВМ и определяются по формуле (5.17):

$$З_{в.м} = \sum_{i} З_{пр i}(1 + K_{доп}) \cdot m_i \cdot K_{м.з}, \quad (5.17)$$

где $З_{пр}$ – затраты на приобретение (стоимость) ЭВМ, руб.;

$K_{доп}$ – коэффициент, характеризующий доп. Затраты, связанные с доставкой, монтажом и наладкой оборудования, $K_{доп}$ = 12 – 13 % от $З_{пр}$;

$K_{\text{м.з}}$ – коэффициент, характеризующий затраты на вспомогательные материалы ($K_{\text{м.з}}$ = 0,01).

$$\text{Зв.м} = 700 \cdot (1 + 0{,}12) \cdot 0{,}01 = 7{,}84 \text{ руб. за 76 дней.}$$

Затраты на текущий и профилактический ремонт ($З_{\text{т.р}}$) принимаются равными 5% от балансовой стоимости ЭВМ и вычисляются по формуле (5.18):

$$З_{\text{т.р}} = \sum_i З_{\text{пр}i} (1 + K_{\text{доп}}) m_i \cdot K_{\text{т.р}}, \qquad (5.18)$$

где $K_{\text{т.р}}$ – коэффициент, характеризующий затраты на текущий и профилактический ремонт, $K_{\text{т.р}}$ = 0,05.

$$З_{\text{т.р}} = 700 \cdot (1 + 0{,}12) \cdot 0{,}05 = 39{,}2 \text{ руб. за 76 дней.}$$

Прочие затраты на эксплуатацию ПК ($З_{\text{пр}}$) состоят из амортизационных отчислений на здания, стоимости услуг сторонних организаций и составляют 5% от балансовой стоимости. Вычисляются по формуле (5.19):

$$З_{\text{пр}} = \sum_i З_{\text{пр}i} (1 + K_{\text{доп}}) m_i \cdot K_{\text{пр}}, \qquad (5.19)$$

где $K_{\text{пр}}$ – коэффициент размера прочих затрат, связанных с эксплуатацией ЭВМ ($K_{\text{пр}}$ = 0,05).

$$З_{\text{пр}} = 700 \cdot (1 + 0{,}12) \cdot 0{,}05 = 39{,}2 \text{ руб. за 179 дней.}$$

Для расчета машинного времени ЭВМ ($t_{\text{эвм}}$ в часах), необходимого для разработки и отладки проекта, следует использовать формулу (5.20):

$$t_{\text{эвм}} = (t_{\text{р.п}} + t_{\text{вн}}) F_{\text{см}} \cdot K_{\text{см}}, \qquad (5.20)$$

где $t_{\text{р.п}}$ – срок реализации стадии «Рабочий проект» (РП);

$t_{\text{вн}}$ – срок реализации стадии «Ввод в действие» (ВП); $t_{\text{р.п}} + t_{\text{вн}} = 29$;

$F_{\text{см}}$ – продолжительность рабочей смены, ч; $F_{\text{см}}$ = 8 ч;

$K_{\text{см}}$ – количество рабочих смен, $K_{\text{см}}$ = 1.

$$t_{\text{эвм}} = 29 \cdot 8 \cdot 1 = 232 \text{ ч.}$$

$$\text{С}_{\text{ч}} = \frac{152 + 156{,}8 + 95{,}4 + 7{,}84 + 39{,}2 + 39{,}2}{1778{,}44} = 0{,}27 \text{ руб./ч.}$$

$$\text{З}_{\text{м.в.}} = 0{,}27 \cdot 1 \cdot 232 = 62{,}64 \text{ руб.}$$

Расчет затрат на изготовление эталонного экземпляра ($\text{З}_{\text{эт}}$) осуществляется по формуле (5.21):

$$\text{З}_{\text{эт}} = \left(\text{З}_{\text{т.р}} + \text{З}_{\text{тех}} + \text{З}_{\text{м.в}}\right) \cdot K_{\text{эт}}, \qquad (5.21)$$

где $K_{\text{эт}}$ – коэффициент затрат на изготовление эталонного ПП, $K_{\text{эт}}$ = 0,05.

$$\text{З}_{\text{эт}} = (39{,}2 + 0 + 62{,}64) \cdot 0{,}05 = 5 \text{ руб.}$$

Затраты на материалы (носители информации и прочее), необходимые для обеспечения работы ПЭВМ, рассчитываются по формуле (5.22):

$$\text{З}_{\text{мат}} = \text{З}_{\text{приобр}} \cdot \left(1 + K_{\text{доп}}\right) \cdot K_{\text{м.з}}, \qquad (5.22)$$

где $\text{З}_{\text{мат}}$ – затраты на приобретение ЭВМ, руб.;

$K_{\text{доп}}$ – коэффициент, характеризующий доп. Затраты, связанные с доставкой, монтажом и наладкой оборудования, $K_{\text{доп}}$ = 12-13 % от $\text{З}_{\text{приобр}}$;

$K_{\text{м.з}}$ – коэффициент, характеризующий затраты на вспомогательные материалы ($K_{\text{м.з}}$ = 0,01).

$$\text{З}_{\text{мат}} = 700 \cdot (1 + 0{,}12) \cdot 0{,}01 = 7{,}84 \text{ руб.}$$

Общепроизводственные затраты ($\text{З}_{\text{общ.пр}}$) определим по формуле (5.23):

$$\text{З}_{\text{общ.пр}} = \frac{\text{ЗП}_{\text{осн}} \cdot \text{Н}_{\text{общ.пр}}}{100\%}, \qquad (5.23)$$

где $\text{Н}_{\text{общ.пр}}$ – норматив общепроизводственных затрат.

$$\text{З}_{\text{общ.пр}} = 957{,}6 * 0{,}05 = 47{,}88 \text{ руб. за 76 дней.}$$

Непроизводственные затраты рассчитываются по формуле (5.24):

$$З_{непр} = \frac{ЗП_{осн} \cdot Н_{непр}}{100\%}, \quad (5.24)$$

где $Н_{непр}$ – норматив непроизводственных затрат.

$$З_{непр} = 957,6 * 0,05 = 47,88 \text{ руб. за 76 дней.}$$

Итого получаем суммарные затраты на разработку:

$$З_р = 47,88 + 47,88 + 7,84 + 5 + 62,64 + 1411,5 = 1582,58 \text{ руб.}$$

Результаты расчетов приведены в таблице К.8.

5.4 Расчет отпускной цены разрабатываемого программного продукта. Стоимость платной подписки

Оптовая цена ПП ($Ц_{опт}$) определяется по формулам (5.25) – (5.26):

$$Ц_{опт} = З_р + П_р, \quad (5.25)$$

$$П_р = \frac{З_р \cdot У_р}{100}, \quad (5.26)$$

где $З_р$ – себестоимость ПО, руб.;

$П_р$ – прибыль от реализации ПП, руб.;

$У_р$ – уровень рентабельности ПП, % ($У_р$ = 30 %).

$$П_р = \frac{1582,58 \cdot 30}{100} = 474,77 \text{ руб.}$$

$$Ц_{опт} = 1582,58 + 474,77 = 2057,35 \text{ руб.}$$

Прогнозируемая отпускная цена ПП рассчитывается по формуле (5.27):

$$Ц_{отп} = З_р + П_р + Р_{ндс}, \quad (5.27)$$

Налог на добавленную стоимость ($Р_{ндс}$) рассчитывается по формуле (5.28):

$$P_{ндс} = \frac{(З_р + П_р) \cdot Н_{ндс}}{100}, \quad (5.28)$$

где $Н_{ндс}$ – ставка налога на добавленную стоимость, %, $Н_{ндс}$ = 20 %.

$$P_{ндс} = (1582{,}58 + 474{,}77) * 0{,}2 = 411{,}47 \text{ руб}$$

Отпускная цена с НДС составит:

$$Ц_{отп.ндс} = 2057{,}35 + 411{,}47 = 2468{,}82 \text{ руб.}$$

Потенциальными потребителями приложения являются лаборатории генетики, научные институты, медицинские учреждения и др. В Беларуси 4 медицинских университета, 4 института генетики, а число медицинских организаций больше 500. Если учесть, что только в некоторых медицинских организациях производят электрофорез белков мочи, например, то можно сказать, что всего имеется около сотни организаций по Беларуси, которым требуется анализ электрофореграмм.

Если не брать в расчет организации из других стран, то стоимость месячной подписки будет составлять около 2 рублей.

Использование веб-приложения позволяет отказаться от услуг генетика, который выполняет анализ белкового спектра – разбор может быть сделан специалистами, выполняющими процедуру электрофореза.

Средняя зарплата специалиста генетика по стране составляет 700 рублей. Значит каждая из ста организаций сможет сэкономить 698 рублей на зарплате специалиста по анализу электрофоретических спектров белка. В год это – 8376 рублей.

6 ЭЛЕКТРОМАГНИТНОЕ ИЗЛУЧЕНИЕ КОМПЬЮТЕРНОЙ ТЕХНИКИ. МЕТОДЫ СНИЖЕНИЯ УРОВНЯ ИЗЛУЧЕНИЙ

6.1 Электромагнитные излучения компьютера

Исследования ученых за последние 20 лет показали, что электромагнитные поля, созданные техническими системами, даже в сотни раз слабее естественного поля Земли, могут быть опасными для здоровья человека, и если не изменить принципы построения электронных и радиотехнических систем, то тенденция их развития и негативное влияние на биологические системы на уровне действия полей могут привести к катастрофическому за своими последствиями воздействия на биосферу и человеку.

Плоды научно-технического прогресса, которые должны служить на благо человечества, становятся агрессивными по отношению даже к своим создателям. Стремительно растет энергонасыщенность быта людей. Электроника подходит в все ближе к человеку: компьютер, телевизор, видео-системы, микроволновые печи, радиотелефоны - вот далеко не полный перечень технических средств, с которыми человек постоянно взаимодействует. Паутина проводов электроснабжения в домах и в служебных помещениях окружают человека. Человек находится длительное время под действием искусственных полей, созданных электронными системами и системами электроснабжения.

Особенно стремительно в нашу жизнь входят компьютеры и телевизионные системы. Сегодня во всем мире компьютеры занимают важное место в работе, жизни и отдыхе людей. Без них уже невозможно представить наш мир. Одним из вредных аппаратных обеспечений ЭВМ для человеческого организма являются дисплеи. Дисплеи, сконструированные на основе электронно-лучевой трубки, являются источниками электростатического поля, мягкого рентгеновского, ультрафиолетового, инфракрасного, видимого, низкочастотного, сверхнизкочастотных и высокочастотного электромагнитного излучения (ЭМИ). Влияние комплекса ЭМИ или отдельных его видов на возникновения различных заболеваний начали изучать с момента их использования В конце 50-х годов в СССР были введены первые нормативы, ограничивающие радиочастотное влияние. В конце 60-х годов советские ученые составили таблицу воздействия электромагнитных полей, даже очень слабых, на нервную систему человека. В 70-е годы эта проблема стала предметом широких дискуссий и исследований.

Источниками электромагнитных излучений являются сети (частота 50 Гц), система строчной развертки (2-400 кГц), блок модуляции луча (5-10 МГц). Было установлено, что излучение низкой частоты, в первую очередь, негативно влияет на центральную нервную систему, вызывая головные боли, головокружение,

тошноту, депрессию, бессонницу, отсутствие аппетита, возникновения синдрома стресса, причем нервная система реагирует даже на короткие по продолжительности воздействия относительно слабых полей частоты: меняется гормональное состояние организма, нарушаются биотоки в мозгу. Все это отражается на процессах обучения и запоминания.

Низкочастотное электромагнитное поле может стать причиной кожных заболеваний (угревая сыпь, себорейная экзема, розовый лишай и др.), болезней сердечно-сосудистой системы и желудочно-кишечного тракта, оно влияет на белые кровяные тельца, что приводит к возникновению опухолей, в том числе и злокачественных.

Особое внимание медики уделяют исследованиям влияния электромагнитных излучений на женщин в период беременности. Статистические данные свидетельствуют о том, что работа за компьютером нарушает нормальный ход беременности, часто является причиной появления на свет детей с врожденными пороками, из которых наиболее распространенными являются дефекты развития головного мозга. Поэтому необходимо, чтобы руководство своевременно переводило беременных женщин на работу, не связанную с использованием мониторов.

Существуют убедительные доказательства неблагоприятного комплексного влияния мониторов ПК на организм работающих. В таблице 6.1 приведены результаты медико-биологических исследований воздействия ПК на пользователей, проведенные Российским научно-исследовательским институтом охраны труда.

Таблица 6.1 – Результаты воздействия ПК на пользователей

Симптомы воздействия компьютера	Процент операторов, которые сообщили о симптомах			
	Работа за дисплеями, месяцев			
	До 12, смена неполная	До 12, смена полная	Более 12	Более 24
1	2	3	4	5
боль головы, боль в глазах	8	35	51	76
усталость, головокружение	5	32	41	69
нарушение сна	0	8	15	50
сонливость в течение дня	11	22	48	76

Продолжение таблицы 6.1

1	2	3	4	5
изменение настроения	8	24	27	50
повышенная раздражительность	3	11	22	51
депрессия	3	16	22	50
снижение интеллектуальных способностей	0	3	12	40
выпадение волос	0	0	3	5
боль в мышцах	11	14	21	32
боль в области сердца, нарушение ритма	0	5	7	32
снижение половой активности	12	18	34	64

Считается, что основной причиной негативного влияния мониторов ПК, телевизоров, другой бытовой техники на их пользователей является торсионная компонента электромагнитных излучений.

6.2 Электромагнитные излучения портативных компьютеров

В результате научно-технического прогресса был создан портативный компьютер – *Notebook*. Удобство его заключается в том, что мы можем взять *Notebook* в дорогу, на отдых и т.д. Но проблема электромагнитных излучений портативных компьютеров заслуживает серьезного пристального внимания. Электростатическое поле и рентгеновское излучение действительно отсутствуют в жидкокристаллических экранах, и относительно переменных электромагнитных полей, но утверждение о безопасности портативных компьютеров по этим параметрам явно преждевременное.

Часто можно услышать мнение, что портативные компьютеры типа *Notebook* безопасны для пользователей и не нуждаются в таких дополнительных мерах защиты, как приэкранные фильтры: их можно считать устройствами, которые сохраняют здоровье людей и потребляют значительно меньше энергии, чем их электронно-лучевые предшественники. В основе подобных рассуждений лежит тот факт, что в портативных компьютерах используются экраны на основе

жидких кристаллов, не генерирующие вредных излучений, присущих обычным мониторам с электронно-лучевой трубкой. Однако результаты исследований, проведенных в научно-исследовательских центрах, показали, что электромагнитное излучение портативных компьютеров типа *Notebook* значительно превышает экологические нормативы. Учитывая результаты исследований величины электромагнитного излучения *Notebook*, можно прийти к выводу, что информационная торсионная компонента по уровню негативного воздействия на пользователя ничем не отличается от мониторов на основе электронно-лучевой трубки (ЭЛТ).

6.3 Безопастные уровни излучений. Средства защиты от излучений

Степень воздействия электромагнитных излучений на организм человека зависит от диапазона частот, интенсивности воздействия соответствующего фактора, продолжительности облучения, характера излучения (непрерывное или модулированное), режима облучения, размеров облучаемой поверхности тела и индивидуальных особенностей организма. В таблице 6.2 представлены допустимые уровни электромагнитных полей.

Таблица 6.2 – Допустимые уровни электромагнитных полей

Диапазоны частот	0,3-300 кГц	0,3-3 МГц	3-30 МГц	30-300 МГц	0,3-300 ГГц
Допустимые уровни	25 В/м	15 В/м	10 В/м	3 В/м	10 мкВ/см^2

При систематическом воздействии электромагнитных излучений, превышающих допустимые значения, происходят функциональные нарушения нервной, эндокринной и сердечно-сосудистой систем человека, а также некоторые изменения состава крови, особенно выраженные при высокой напряженности электрического поля.

При превышении допустимой напряженности и плотности потока энергии электромагнитного поля необходимо применять основные средства и способы защиты:

– экранирование рабочего места;

– удаление рабочего места от источника электромагнитного поля;

– рациональное размещение в рабочем помещении оборудования, излучающего электромагнитную энергию;

– установление рациональных режимов работы оборудования и обслуживающего персонала;

– применение предупреждающей сигнализации (световой, звуковой);

– применение средств индивидуальной защиты.

Эффективным и часто применяемым методом защиты от низкочастотных и радиоизлучений является экранирование. Для экранов используют главным образом материалы с большой электрической проводимостью (медь, латунь, алюминий и его сплавы, сталь). Экраны должны быть заземлены.

В качестве средств индивидуальной защиты применяется спецодежда, изготовленная из металлизированной ткани в виде комбинезонов, халатов. Используя спецодежду из металлизированной ткани необходимо особо строго соблюдать требования электробезопасности.

7 РЕСУРСО- И ЭНЕРГОСБЕРЕЖЕНИЕ ПРИ ИСПОЛЬЗОВАНИИ ИНФОРМАЦИОННЫХ СИСТЕМ

7.1 Проблемы энергосбережения

Энергоэффективность и энергосбережение – это два понятия давно и прочно вошедших в нашу жизнь.

Энергосбережение – комплекс мер, конечной целью которых является достижение более рационального и эффективного использования топливо-энергетических ресурсов, а также с целью привлечения «освобожденной» энергии для хозяйственных нужд.

В свою очередь, энергоэффективность – это рациональное использование ресурсов энергетики. То есть если меры по энергосбережению направлены преимущественно на снижение потребления этих ресурсов, то энергоэффективность работает в направлении их более эффективного использования.

Вопросы энергосбережения, ставшие весьма актуальными, волнуют, как весь мир в целом, так и каждого индивидуума в отдельности. У каждого – причины свои, одни стараются сэкономить на этом личные средства, другие размышляют на уровне более глобального масштаба. Но пока в министерствах и ведомствах обсуждают и принимают различные законопроекты касательно проблем энергосбережения, можно попытаться изменить ситуацию в своем ведении, повысить энергоэффективность в пределах предприятий, в первую очередь, сэкономить на издержках.

Энергосбережение, кроме материальной выгоды, имеет огромное значение в области сохранения природных ресурсов, поэтому решая вопросы и проблемы энергосбережения сегодня, мы, в первую очередь, заботимся о дне завтрашнем. Бесконтрольное энергопотребление в конечном итоге приведет к дефициту природных ресурсов, ведь в своем большинстве они являются не возобновляемыми, и к экологической катастрофе.

Спрос на энергетические ресурсы постоянно растет, вместе с тем повышаются тарифы на них, сокращаются запасы полезных ископаемых (нефть, газ, уголь), ухудшается экология страны – все это придает особенное значение энергосбережению. Многие страны ведут разработку и реализацию программ по повышению эффективности в использовании энергоресурсов. В связи с этим затронем одну из важнейших тем «Проведение мероприятия по энергосбережению на предприятии».

Одной из причин для проведения мероприятия по энергосбережению на предприятии, является снижение издержек и повышение экономической эффективности на производстве. На данный момент на промышленных

предприятиях процент энергетических затрат в издержках составляет 9 – 12%, и этот процент постоянно растет. Эта проблема связана, в основном, с физическим и моральным износом оборудования, также большие потери энергетических ресурсов возникают и при транспортировке.

Энергосберегающий путь в развитии белорусской экономики допустим только в реализации формирования и применения программ и мероприятий по энергосбережению на предприятиях и производствах. Необходимо установить на промышленных предприятиях максимально рациональное применение и использование энергоресурсов. Отсутствие мероприятия по энергосбережению на предприятии может привести к значительному экономическому ущербу самого предприятия и отрицательному влиянию на экологию окружающей среды.

Если этот процесс не остановить, то рост издержек, сопровождаемый финансовыми потерями, будет задерживать обновление базы производства предприятия, что необходимо для развития производства. Для того чтобы предотвратить этот процесс необходимо не только провести мероприятия по энергосбережению на предприятии, но и вести постоянные разработки, обновление и совершенствование методов энергоаудита, оценки результативности и качества программ по энергосбережению, необходимо чтобы эти программы учитывали многовариантность использования инвестиционных источников, которые предназначены для их осуществления.

Эти меры помогут снизить энергозатраты и издержки, что позволит приобрести дополнительные средства, с помощью которых станет возможным обеспечение производству обновления технологического оборудования. Для того чтобы достигнуть такого результата необходимо прибегнуть к следующим мерам:

– реализовать комплексное обследование предприятия;

– получить рекомендации специалистов по комплектации и необходимому оснащению предприятия современным оборудованием;

– разработать комплексную программу энергосбережения предприятия;

– после чего реализовать все программные мероприятия по энергосбережению на предприятии.

7.2 Расчет сбережения ресурсов при внедрении программы

Внедрение программных комплексов, выполняющих автоматизированную классификацию и анализ белковых последовательностей, позволяет генетикам избежать рутинной работы по разбору формулы аминокислот и определению принадлежности к определенному сорту.

Но нельзя полностью отказаться от работы генетика, так как автоматизированную систему необходимо запускать на исполнение, когда это необходимо и следить за тем, чтобы данные, вводимые в программу были корректными. Также генетик может принять решение в случае, если система не может однозначно определить принадлежность какому-либо сорту. Что бы оценить экономию времени за год ($T_э$), используем формулу (7.1).

$$T_э = (T_{р.у.} - T_{а.у.}) A_{исп.н.}, \quad (7.1)$$

где $T_{р.у.}$ – время выполнения генетиком типовой операции анализа белковой последовательности. Для расчетов примем $T_{р.у.} = 10$ минут;

$T_{а.у.}$ – время выполнения операции определения сорта программой – 2 секунды;

$A_{нj}$ – количество *j*-х операций, выполняемых новым ПО в течение месяца. Примем $A_{нj} = 100$.

$$T_э = (10 * 60 - 2) \cdot 100 = 16{,}6 \text{ часов}.$$

Как видно из расчетов экономия во времени составляет 16,6 часов за месяц. При использовании автоматизированной системы один пользователь мог бы запустить на исполнение еще 29900 операций.

Так же для использования разработанной системы не надо иметь специальных навыков анализа белковых последовательностей, так как программа выполняет все без участия эксперта в предметной области. Поэтому оплата труда будет несколько ниже, чем для высококвалифицированных генетиков.

ЗАКЛЮЧЕНИЕ

В ходе выполнения дипломной работы был создан программный комплекс, предназначенный для анализа и классификации сортов твердой пшеницы по электрофоретическому спектру глиадина. Аналитический обзор существующих методов не выявил наличие прямых аналогов. Как правило, подобная работа по определению принадлежности зерна к какому-либо сорту выполняется генетиками вручную и требует большого количества времени для анализа белков и квалификацию в области определения сорта по электрофореграммам.

В литературе уже были отмечены попытки автоматизировать работу по классификации белковых последовательностей [5,12,17], но авторы смогли добиться высоких показателей эффективности распознавания только для выборки из 10 сортов. Предложенная в настоящей работе модель в ходе тестирования показала точность классификации около 90% даже для 50 сортов. Показатели можно улучшить за счет использования специальных генетических маркеров для каждого сорта, но так как данная информация отсутствует в базе данных, этого не было сделано.

Была проведена работа с исходной базой данных, чтобы отсеять некорректные примеры из обучающей и тестовой выборок и проведена нормировка данных для подачи на вход нейросетям.

Разработано и протестировано веб-приложение для решения поставленной задачи. Веб-приложение имеет микросервисную архитектуру, что обеспечивает модульность разработанной системы и позволяет ядру приложения продолжать работу даже при сбое в каком-то из модулей. Веб-приложение является клиент-серверным: на сервере хранится ядро приложения, отвечающее за всю логику, а клиентская часть представлена веб-интерфейсом. Приложение является кросс-платформенным и требует только наличия браузера.

Все модели нейронных сетей были написаны с помощью открытых библиотек машинного обучения – *Keras, TensorFlow*. Обучение ИНС было ускорено за счет использования бесплатного облачного сервиса *Floydhub*, который позволяет запускать программы в *GPU*-режиме с использованием графических ускорителей модели *Nvidia K*80.

Было показано преимущество сверточных сетей в задачах классификации изображений, так как они обеспечивают низкую потерю данных при сокращении размерности за счет использования фильтров, обучаются быстрее, так как имеют меньшее число параметров.

Приложение является экономически эффективным и отвечает всем требованиям энергосбережения.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Glorot, X. Understanding the difficulty of training deep feedforward neural networks / X. Glorot, Y. Bengio //Aistats. – 2010. – Vol. 9. – P. 249 – 256.

2. Bengio, Y. Scaling learning algorithms towards AI/ Y. Bengio, Y. LeCun// In L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, Large Scale Kernel Machines. MIT Press. – 2007. – 1634 p.

3. Hinton, G. A practical guide to training restricted Boltzmann machines / G. Hinton // Momentum. – 2010. – Vol. 9. – No. 1. – P. 926.

4. He, K. Deep residual learning for image recognition / K. He et al. // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. – 2016. – P. 770 – 778.

5. Ososkov, G. Feature Extraction for Data Input to Neuro-Classifiers/ G. Ososkov, D. Baranov //Mathematical Modeling and Computational Physics (MMCP 2009): Book of Abstract of the International Conference. Dubna. – 2009. – P. 110 – 111.

6. Якубке, Х. Д. Аминокислоты, пептиды, белки/ Х. Якубке, Х. Ешкайт // Пер. с немец. М.:Мир. – 1985. – 456 с.

7. Губарева, Н. К. Идентификация сортов сельскохозяйственных культур по электрофоретическим спектрам запасных белков / Н. К. Губарева, И. П. Гаврилюк, А. В. Конарев // Аграрная Россия. – 2015. – №. 11. – С. 21 – 27.

8. Kohonen, T. The self-organizing map / T. Kohonen //Neurocomputing. – 1998. – Vol. 21. – No. 1. – P. 1 – 6.

9. Self-Organizing Maps with Google’s TensorFlow [Electronic resource] – Mode of access: https://codesachin.wordpress.com/2015/11/28/self-organizing-maps-with-googles-tensorflow/ – Date of access: 01.06.2017

10. Хайкин, С. Нейронные сети: полный курс, 2-е издание / С. Хайкин // Пер. с англ. – М. Издательский дом Вильямс. – Москва, 2006. – 1104 с.

11. Sokal, R. R. A statistical method for evaluating systematic relationships / R. R. Sokal //Univ Kans Sci Bull. – 1958. – Vol. 38. – P. 1409 – 1438.

12. Ruanet, V. V. The use of a self-organizing feature map for the treatment of the results of RAPD and ISSR analyses in studies on the genomic polymorphism in the genus Capsicum L/ V. V. Ruanet, E. Z. Kochieva, N. N. Ryzhova //Russian Journal of Genetics. – 2005. – Vol. 41. – No. 2. – P. 202 – 210.

13. Sonnhammer, E. L. L. A hidden Markov model for predicting transmembrane helices in protein sequences/ E. L. L. Sonnhammer et al. //Ismb. – 1998. – Vol. 6. – P. 175 – 182.

14. Rosenblatt, F. The perceptron, a perceiving and recognizing automaton Project Para/ F. Rosenblatt // Report: Cornell Aeronautical Laboratory, – 1957. – Vol. 85. – No. 460 – 461.

15. Krogh, A. Hidden Markov models for labeled sequences /A. Krogh //Pattern Recognition, 1994. Vol. 2-Conference B: Computer Vision & Image Processing., Proceedings of the 12th IAPR International. Conference on. – IEEE, 1994. – Vol. 2. – P. 140 – 144.

16. Williams, D. Learning representations by back-propagating errors / D. Williams, G. Hinton //Nature. – 1986. – Vol. 323. – No. 6088. – P. 533-538.

17. Ruanet, V. V. The use of artificial neural networks for automatic analysis and genetic identification of gliadin electrophoretic spectra in durum wheat / V. V. Ruanet, A. M. Kudryavtsev, S. Y. Dadashev //Russian Journal of Genetics. – 2001. – Vol. 37. – No. 10. – P. 1207 – 1209.

18. Jensen, K. Classification of potato varieties using isoelectrophoretic focusing patterns, neural nets, and statistical methods / K. Jensen et al. //Journal of agricultural and food chemistry. – 1997. – Vol. 45. – No. 1. – P. 158 – 161.

19. Bateman, A. HMM-based databases in InterPro / A. Bateman, D. H. Haft //Briefings in bioinformatics. – 2002. – Vol. 3. – No. 3. – P. 236 – 245.

20. Mi, H. PANTHER version 11: expanded annotation data from Gene Ontology and Reactome pathways, and data analysis tool enhancements / H. Mi et al. //Nucleic acids research. – 2017. – Vol. 45. – No. D1. – P. D183 – D189.

21. Гончаров, П. В. Классификация сжатых изображений методами глубинного обучения /П. В. Гончаров // Современные проблемы машиноведения: тез. докл. XI Междунар. науч.- техн. конф. (науч. чтения, посвящ. П. О. Сухому), Гомель, 20–21 окт. 2016 г. / М-во образования Респ. Беларусь, Гомел. гос. техн. ун-т им. П. О. Сухого, Филиал ПАО «Компания «Сухой» ОКБ «Сухого»; под общ. ред. С. И. Тимошина. – Гомель: ГГТУ им. П. О. Сухого, 2016. – 226 с.

22. Kramer, M.A. Nonlinear principal component analysis using autoassociative neural networks / M.A. Kramer //AIChE journal. – 1991. – Vol. 37. – No. 2. – P. 233 – 243.

23. Vincent, P. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion / P. Vincent et al. // Journal of Machine Learning Research. – 2010. – Vol. 11. – P. 3371 – 3408.

24. Nair, V. Rectified linear units improve restricted boltzmann machines / V. Nair, G. E. Hinton //Proceedings of the 27th international conference on machine learning (ICML-10). – 2010. – P. 807 – 814.

25. Kingma, D. Adam: A method for stochastic optimization / D. Kingma, J. Ba // arXiv preprint arXiv:1412.6980. – 2014.

26. LeCun, Y. Gradient-based learning applied to document recognition/ Y. LeCun et al. //Proceedings of the IEEE. – 1998. – Vol. 86. – No. 11. – P. 2278 – 2324.

27. An open-source software library for Machine Intelligence [Electronic resource]. – Mode of access: https://www.tensorflow.org/. – Date of access: 15.04.2017

28. Keras: Deep Learning library for Theano and TensorFlow [Electronic resource]. – Mode of access: https://keras.io/. – Date of access: 15.04.2017

29. Flask web development, one drop at a time [Electronic resource]. – Mode of access: https://www.floydhub.com/. – Date of access: 18.05.2017

30. Srivastava, N. Dropout: a simple way to prevent neural networks from overfitting / N. Srivastava et al. // Journal of Machine Learning Research. – 2014. – Vol. 15. – No. 1. – P. 1929 – 1958.

31. Simonyan, K. Very deep convolutional networks for large-scale image recognition / K. Simonyan, A. Zisserman // Preprint arXiv:1409.1556. – 2014. – 438 p.

ПРИЛОЖЕНИЕ А

(справочное)

Блок-схемы алгоритмов

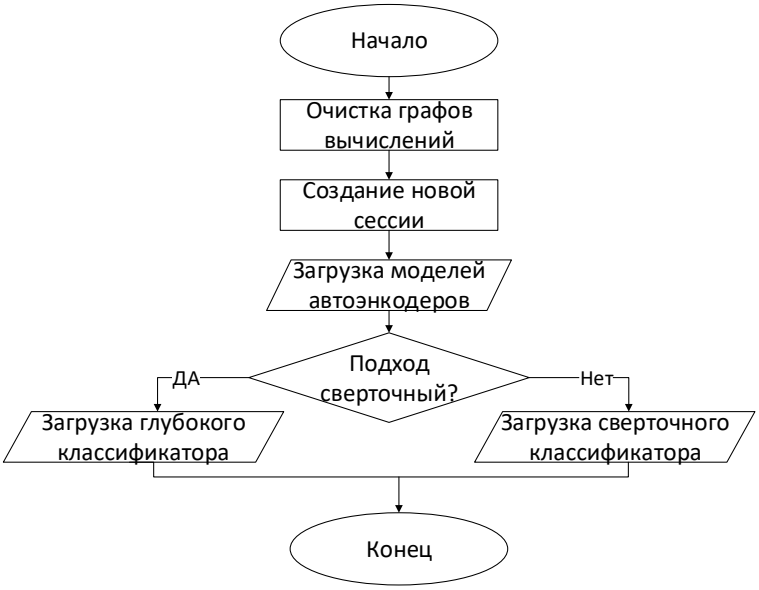


Рисунок А.1 – Блок-схема функции начальной инициализации

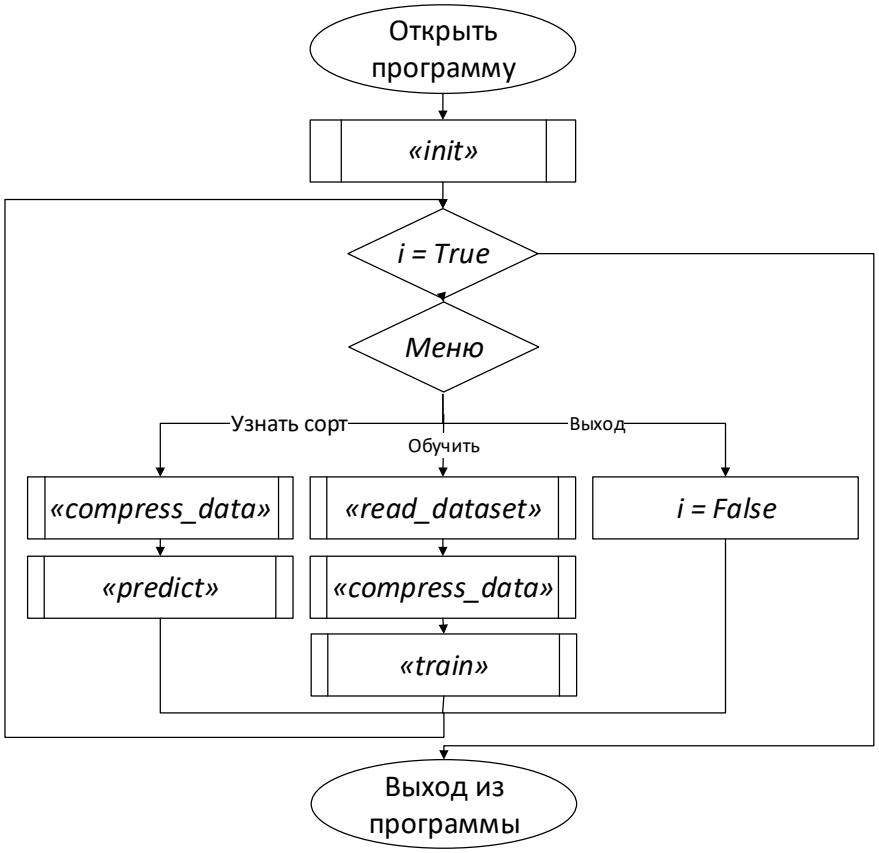


Рисунок А.2 – Функциональная схема приложения

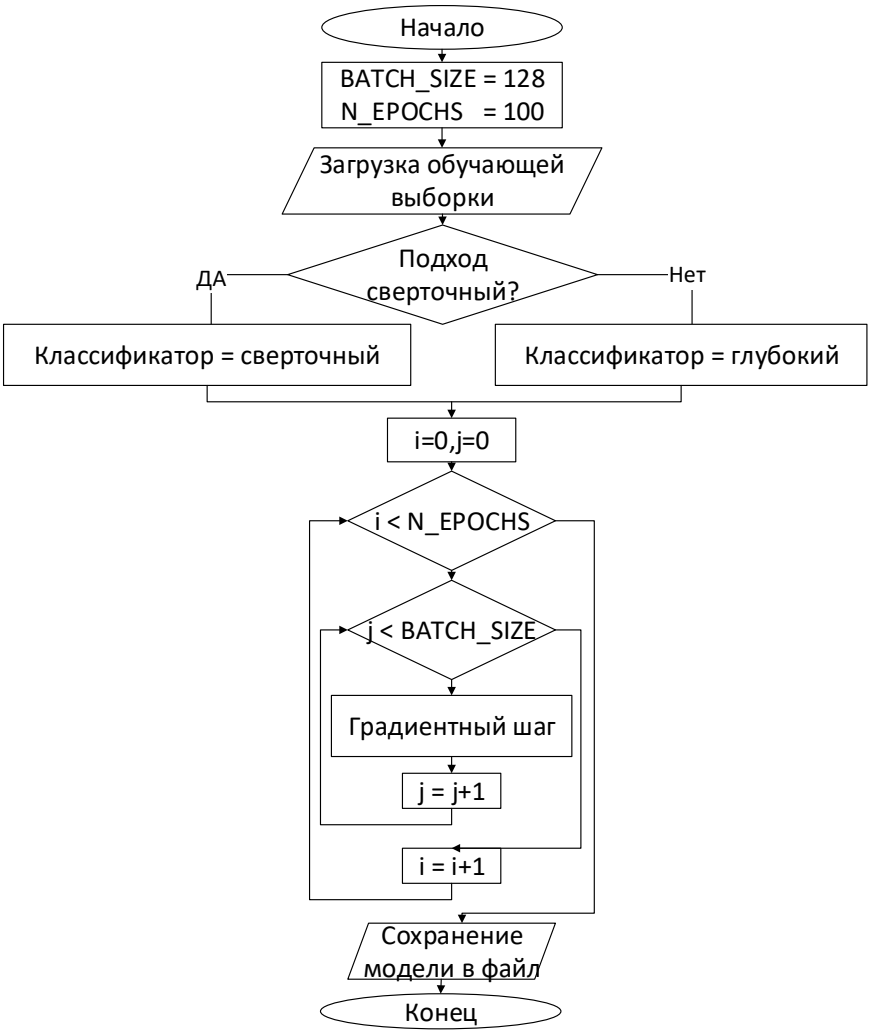


Рисунок А.3 – Блок схема функции обучения сети

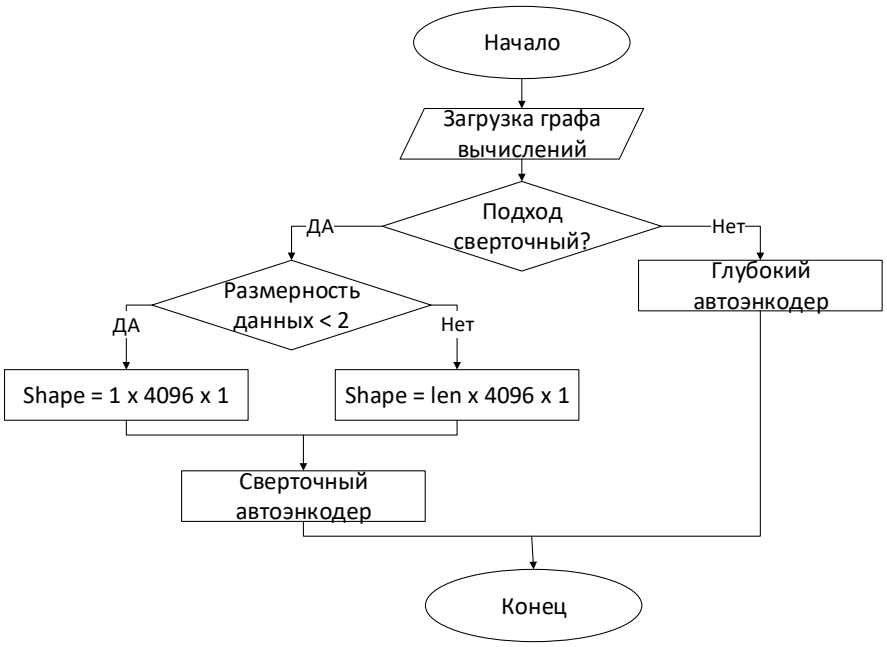


Рисунок А.4 – Блок-схема функции компрессии

Рисунок А.5 – Блок-схема функции предсказания сорта

ПРИЛОЖЕНИЕ Б
(обязательное)
Программный код

Основной *API*

```
# -*- coding: utf-8 -*-
from flask import Flask, url_for, render_template
from flask import request, flash
from werkzeug import secure_filename
import random, threading, webbrowser
import os

import numpy as np
import matplotlib.pyplot as plt
from io import BytesIO
import base64

import subprocess
import sys

import tensorflow as tf
from NN.networks_func import networks_init, predict, compress_data
import pickle

from NN.utils.create_dataset import read_dataset
import time

app = Flask(__name__)

app.config['APP_ROOT']                  = os.path.dirname(os.path.abspath(__file__))
app.config['UPLOAD_FOLDER']     = os.path.join(app.config['APP_ROOT'], "NN/data/")
app.config['TRAIN_CALLBACKS']   = os.path.join(app.config['APP_ROOT'],
"callbacks_train/api.py")
app.config['TRAIN_FUNC']        = os.path.join(app.config['APP_ROOT'], 'NN/train.py')
app.config['ALLOWED_EXTENSIONS'] = set(['txt', 'csv'])
app.config['MAX_FILE_SIZE']     = 200 * 1024  # 200 Kb limit
app.config['NETWORD_MODE']      = 'conv'
app.config['CALLBACKS']         = None
```

```
app.config['SESS'],    \
app.config['CONV_AE'],      \
app.config['DEEP_AE'],      \
app.config['PREDICTOR']              = networks_init(app.config['NETWORD_MODE'])

app.config['TRAIN_PROC']        = None

# or set directly on the app
app.secret_key = b'secret_key'

def validateExtension(filename):
   '''Check if file meets the requirements

   #Arguments
      filename : string, name of the file

   #Returns
      boolean : True if file meets the requirement or False otherwise
   '''
   return '.' in filename and \
         filename.rsplit('.', 1)[-1].lower() in app.config['ALLOWED_EXTENSIONS']


@app.route('/')
def homepage():
         return render_template('main.html')


@app.route("/predict", methods=['GET', 'POST'])
def upload():
         plot_url = False
         sort     = None
         if request.method == 'POST':
                  file     = request.files['file']
                  filename = secure_filename(file.filename)
                  if file and validateExtension(filename):
                           spectra = np.genfromtxt(file, delimiter=',')
                           if spectra.shape != (4096, ): #spectrum length
                                    plot_url = False
```

```
                        flash("ERROR! SELECT ANOTHER FILE")
                else:
                        # prepare plot stream
                        img = BytesIO()

                        fig  = plt.figure()
                        ax   = fig.add_subplot(111)
                        ax.plot(spectra)
                        fig.savefig(img, format='png', bbox_inches='tight')
                        plot_url = base64.b64encode(img.getvalue()).decode('ascii')

                        img.seek(0)

                        #predict sort
                        sort = predict(spectra, \
                                                app.config['SESS'], \
                                                app.config['NETWORD_MODE'], \
                                                app.config['CONV_AE'] if
app.config['NETWORD_MODE'] == 'conv' \
                                                                        else
app.config['DEEP_AE'], \
                                                app.config['PREDICTOR'])

                        print('Spectra loaded! Predicted sort =', sort)
        else:
                plot_url = False
                flash("ERROR! SELECT ANOTHER FILE")

    return render_template("main.html", plot_url=plot_url, sort=sort)

@app.route("/interrupt", methods=['GET', 'POST'])
def interrupt_training():
    if request.method == 'POST':
        if request.form['submit'] == 'stop':
                print('STOP')

                if app.config['TRAIN_PROC']: # if process is not stopped
                        app.config['TRAIN_PROC'].kill()
                        app.config['TRAIN_PROC'] = None
        if request.form['submit'] == 'start':
```

```
                complete_flag = False
                print('START')
                train_option = request.form['train']

                print(train_option)

                if train_option:
                        try:
                                train_dict = read_dataset(request.files.getlist("files"))
                                print('Data compression...')
                                if train_option == 'conv':
                                        train_dict['X_train'] =
compress_data(train_dict['X_train'], app.config['SESS'], train_option, app.config['CONV_AE'])
                                        train_dict['X_test']  =
compress_data(train_dict['X_test'], app.config['SESS'], train_option, app.config['CONV_AE'])
                                elif train_option == 'deep':

                                        train_dict['X_train'] =
compress_data(train_dict['X_train'], app.config['SESS'], train_option, app.config['DEEP_AE'])
                                        train_dict['X_test']  =
compress_data(train_dict['X_test'], app.config['SESS'], train_option, app.config['DEEP_AE'])

                                print('Done')

                                train_dict.update({'mode': train_option})

                                with open(os.path.join(app.config['UPLOAD_FOLDER'],
'train.pickle'), 'wb') as f:
                                        pickle.dump(train_dict, f, 2)

                                app.config['TRAIN_PROC'] = subprocess.Popen(['python',
app.config['TRAIN_FUNC']])
                                complete_flag = True

                        except:
                                flash('TRAINING FOLDER IS NOT VALID! SELECT
ANOTHER')

                else:
                        flash('SELECT APPROACH: CONVOLUTION OR DEEP!')
```

```
                        if not complete_flag:
                                return render_template('main.html')


        return (''), 204


if __name__ == '__main__':

        # start process with training callbacks
        #app.config['CALLBACKS'] = subprocess.Popen(['python',
app.config['TRAIN_CALLBACKS']],
        #
        stdout=subprocess.PIPE,
        #
        stderr=subprocess.STDOUT)

        port = 5000 + random.randint(0, 999)
        url  = 'http://127.0.0.1:{0}'.format(port)

        threading.Timer(0.001, lambda: webbrowser.open(url)).start()
        app.run(port=port, debug=True)

        #app.config['CALLBACKS'].kill() # kill the subprocess
```

Функции нейронных сетей

```
from .models.autoencoders  import get_deep_AE, get_conv_AE
from .models.classifiers   import get_DNN, get_CNN
#from .utils.
from keras.models         import load_model
from keras.utils          import to_categorical
from keras.callbacks      import RemoteMonitor
from keras                import backend as K

import tensorflow as tf
import numpy      as np
import os
```

```
MODELS_PATH = os.path.dirname(os.path.abspath(__file__))
MODELS_PATH = os.path.join(MODELS_PATH, 'data/models/')

def normalize(x):
    """Normalizes the input between 0 and 1

    # Arguments
        x : list or array

    # Returns
        x normalized
    """
    if x.min(0) < 0:
        x -= x.min(0)
    return x / x.ptp(0)


def networks_init(mode):
    """Restores model for prediction

    # Arguments
        mode : string, 'conv' or 'deep'
    """
    ### Clear graphs
    tf.reset_default_graph()
    K.clear_session()

    sess = tf.InteractiveSession()
    K.tensorflow_backend.set_session(sess)

    # Restore autoencoders
    deep_ae = get_deep_AE()

    init    = tf.global_variables_initializer()
    saver   = tf.train.Saver()
    # run session
    sess.run(init)

    saver.restore(sess, os.path.join(MODELS_PATH, 'deep_AE'))
```

```
    conv_ae = load_model(os.path.join(MODELS_PATH, 'conv_encoder_1.h5'))

    if mode == 'conv':
        predictor = load_model(os.path.join(MODELS_PATH, 'cnn.h5'))
    elif mode == 'deep':
        predictor = load_model(os.path.join(MODELS_PATH, 'dnn.h5'))


    return (sess, conv_ae, deep_ae, predictor)


def predict(x, sess, mode, ae, clf):
    """Returns the predicted class

    # Arguments
        x    : numpy ndarray (4096, 0)
        mode : string, 'conv', 'deep'
        ae   : autoencoder for data compression
        clf  : classifier

    # Returns
        predicted class
    """

    with sess.graph.as_default():
        x_compressed = compress_data(x, sess, mode, ae)
        return clf.predict_classes(x_compressed, verbose=0)[0]


def compress_data(x, sess, mode, ae):
    '''Compress the input data with autoencoder

    # Arguments
        x    : numpy ndarray (4096, 0)
        mode : string, 'conv', 'deep'
        ae   : autoencoder for data compression

    # Returns
        x compressed
    '''
```

```
    with sess.graph.as_default():
        if mode == 'conv':
            if len(x.shape) < 2:
                x_compressed = x.reshape(1, 4096, 1)
            else:
                x_compressed = x.reshape(len(x), 4096, 1)
            x_compressed = ae.predict(x_compressed)
        elif mode == 'deep':
            if len(x.shape) < 2:
                x_compressed = sess.run(ae['z'], feed_dict={ae['x']:[x]})
            else:
                x_compressed = sess.run(ae['z'], feed_dict={ae['x']:x})

        return x_compressed
```

Процедура обучения

```
from keras.callbacks import RemoteMonitor
from models.classifiers import get_DNN, get_CNN
from keras.utils import to_categorical
import numpy as np
import pickle
import os

# callbacks
remote = RemoteMonitor(root='http://localhost:9000', headers=None)

BATCH_SIZE    = 128
PATH_TO_DATA  = os.path.join(os.path.dirname(os.path.abspath(__file__)), 'data/train.pickle')
EPOCHS        = 100

with open(PATH_TO_DATA, 'rb') as f:
        train_dict = pickle.load(f)

N_CLASSES     = train_dict['n_classes']

print('Training classifier...')
if train_dict['mode'] == 'deep':
        dnn = get_DNN(num_classes=N_CLASSES)
        try:
```

```
                dnn.fit(train_dict['X_train'], to_categorical(train_dict['y_train'],
num_classes=N_CLASSES), verbose=1, \
                        batch_size=BATCH_SIZE, \
                        epochs=EPOCHS, \
                        shuffle=True, \
                        validation_data=(train_dict['X_test'], to_categorical(train_dict['y_test'],
num_classes=N_CLASSES)), \
                        callbacks = [remote])
        except:
                print('Training interrupted! Model saved.')

        dnn.save(os.path.join(os.path.dirname(os.path.abspath(__file__)),'data/models/dnn_user.h5')
)

        dnn.save(os.path.join(os.path.dirname(os.path.abspath(__file__)),'data/models/dnn_user.h5')
)

elif train_dict['mode'] == 'conv':
        cnn = get_CNN(input_shape=(256,2), num_classes=N_CLASSES)
        try:
                cnn.fit(train_dict['X_train'], to_categorical(train_dict['y_train'],
num_classes=N_CLASSES), verbose=1, \
                        batch_size=BATCH_SIZE, \
                        epochs=EPOCHS, \
                        shuffle=True, \
                        validation_data=(train_dict['X_test'], to_categorical(train_dict['y_test'],
num_classes=N_CLASSES)), \
                        callbacks = [remote])
        except:
                print('Training interrupted! Model saved')

        cnn.save(os.path.join(os.path.dirname(os.path.abspath(__file__)),'data/models/cnn_user.h5')
)

        cnn.save(os.path.join(os.path.dirname(os.path.abspath(__file__)),'data/models/cnn_user.h5')
)

print('Model saved')
```

Вспомогательные функции

```
import os
import pickle
import numpy as np
import operator
from math import ceil
from sklearn.utils import shuffle
from sklearn.cross_validation import train_test_split
import re


def read_file(file):

    if type(file) == str: # filename
        with open(file) as f:
            file_data = f.readlines()
    else:
        file_data = file.readlines()

    sort   = []
    sample = []
    for i,el in enumerate(file_data):
        el = re.sub('[\n\b\r]', '', el.decode('ascii'))
        if el.isdigit():
            sample.append(int(el))
        else:
            sort.append(sample)
            sample = []
    return np.asarray(sort), len(sort)


def create_dataset(path_to_data, n_classes_to_train, path_to_save=None):

    if n_classes_to_train <= 10 and n_classes_to_train > 0:
        classes_range = (25,25+n_classes_to_train)

    elif n_classes_to_train <= 20 and n_classes_to_train > 10:
        classes_range = (15, 15+n_classes_to_train)
    elif n_classes_to_train <= 30 and n_classes_to_train > 20:
        classes_range = (5, 5+n_classes_to_train)
```

```
    else:
        classes_range = (2, 35)

    files = [f for f in os.listdir(path_to_data) if '.txt' in f]
    print('No. data files - %d' % len(files))

    data   = {}   # data
    labels = {}   # No of labels per class
    count = 0     # sum of spectra

    print('Reading files...')
    for i,f in enumerate(files):
        data[i], labels[i] = read_file(os.path.join(path_to_data, f))
        count += labels[i]



    print('Processing dataset with %d classes...' % (classes_range[1] - classes_range[0]))
    labels_toClassify = sorted(labels.items(), key=operator.itemgetter(1),
reverse=True)[classes_range[0]:classes_range[1]] # choose classes with many samples
    labels_toClassify = dict(sorted(labels_toClassify)) # sort with indexes

    # create dataset
    data_size    = sum(labels_toClassify.values())
    test_size    = 0.2#3 # data will be divided in two parts 0.7 for the train data and 0.3 for the test set
    test_shape   = (sum([ceil(el*test_size) for el in labels_toClassify.values()]),4096) # for each class
we compute the test size
    train_shape  = (data_size-test_shape[0], 4096)
    X_train      = np.zeros(train_shape, dtype=np.int32)
    X_test       = np.zeros(test_shape, dtype=np.int32)
    y_train      = np.zeros((train_shape[0],), dtype=np.int32)
    y_test       = np.zeros((test_shape[0],), dtype=np.int32)

    # indexes
    prev_train_i = 0
    prev_test_i  = 0
    curr_train_i = 0
    curr_test_i  = 0

    # class weights
```

```
    class_weight = {}

    print('Split on train and test sets...')
    for i, key in enumerate(labels_toClassify.items()): # key is a pair of values
        test_slice    = ceil(key[1]*test_size)
        curr_test_i   = prev_test_i  + test_slice
        curr_train_i  = prev_train_i + key[1]-test_slice

        X_train[prev_train_i:curr_train_i], X_test[prev_test_i:curr_test_i] =
train_test_split(data[key[0]],
                                                        test_size = test_size,
                                                        random_state = 42)

        y_train[prev_train_i:curr_train_i] = i
        y_test[prev_test_i:curr_test_i]    = i

        # define class weight
        class_weight[i] = round(train_shape[0] / (curr_train_i-prev_train_i))

        prev_train_i = curr_train_i
        prev_test_i  = curr_test_i


    print('Train_shape = ', X_train.shape, y_train.shape)
    print('Test_shape = ', X_test.shape, y_test.shape)

    # do the shuffling
    X_train, y_train = shuffle(X_train, y_train, random_state=42)
    X_test, y_test   = shuffle(X_test, y_test, random_state=42)


    print('Saving to file...')
    save = {'X_train'      : X_train / 255.,
            'X_test'       : X_test / 255.,
            'y_train'      : y_train,
            'y_test'       : y_test,
            'class_weight' : class_weight}


    if path_to_save is not None:
```

```
        with open(os.path.join(path_to_save, 'spectrums_data_user.pickle'), 'wb') as f:
            pickle.dump(save, f, 2)

    print('Done!')
    return save


def read_dataset(files):

    data   = {}   # data
    labels = {}   # No of labels per class
    count = 0     # sum of spectra

    print('Reading files...')
    for i,f in enumerate(files):
        data[i], labels[i] = read_file(f)
        count += labels[i]



    print('Processing dataset with %d classes...' % (len(labels)))

    # create dataset
    data_size    = sum(labels.values())
    test_size    = 0.2#3 # data will be divided in two parts 0.7 for the train data and 0.3 for the test set
    test_shape   = (sum([ceil(el*test_size) for el in labels.values()]),4096) # for each class we
compute the test size
    train_shape  = (data_size-test_shape[0], 4096)
    X_train      = np.zeros(train_shape, dtype=np.int32)
    X_test       = np.zeros(test_shape, dtype=np.int32)
    y_train      = np.zeros((train_shape[0],), dtype=np.int32)
    y_test       = np.zeros((test_shape[0],), dtype=np.int32)

    # indexes
    prev_train_i = 0
    prev_test_i  = 0
    curr_train_i = 0
    curr_test_i  = 0

    # class weights
    class_weight = {}
```

```
    print('Split on train and test sets...')
    for i, key in enumerate(labels.items()): # key is a pair of values
        test_slice    = ceil(key[1]*test_size)
        curr_test_i   = prev_test_i  + test_slice
        curr_train_i  = prev_train_i + key[1]-test_slice

        X_train[prev_train_i:curr_train_i], X_test[prev_test_i:curr_test_i] =
train_test_split(data[key[0]],
                                                        test_size = test_size,
                                                        random_state = 42)

        y_train[prev_train_i:curr_train_i] = i
        y_test[prev_test_i:curr_test_i]    = i

        # define class weight

        prev_train_i = curr_train_i
        prev_test_i  = curr_test_i


    print('Train_shape = ', X_train.shape, y_train.shape)
    print('Test_shape = ', X_test.shape, y_test.shape)

    # do the shuffling
    X_train, y_train = shuffle(X_train, y_train, random_state=42)
    X_test, y_test   = shuffle(X_test, y_test, random_state=42)


    print('Saving to file...')
    save = {'X_train'      : X_train / 255.,
            'X_test'       : X_test / 255.,
            'y_train'      : y_train,
            'y_test'       : y_test,
                             'n_classes'    : len(labels)}#,

    print('Done!')
    return save

    #print(labels_toClassify)
```

Создание автоэнкодеров

```
from keras.layers import Input,  Conv1D, MaxPooling1D, UpSampling1D, Deconv2D
from keras.models import Model
import tensorflow as tf
import math

def get_deep_AE(dimensions=[4096, 1024, 256]):
        """Build a deep autoencoder w/ tied weights.
   Parameters
   ----------
   dimensions : list, optional
      The number of neurons for each layer of the autoencoder.
   Returns
   -------
   x : Tensor
      Input placeholder to the network
   z : Tensor
      Inner-most latent representation
   y : Tensor
      Output reconstruction of the input
   cost : Tensor
      Overall cost to use for training
   """
      # input to the network
         x          = tf.placeholder(tf.float32, [None, dimensions[0]], name='x')                #input of
the network

         current_input = x

         # Build the encoder
         encoder = []

         for layer_i, n_output in enumerate(dimensions[1:]):
                  n_input = int(current_input.get_shape()[1])
                  #define weights
                  W = tf.Variable(tf.random_uniform([n_input, n_output], #dimension
                                    -1.0/math.sqrt(n_input),
                                     1.0/math.sqrt(n_input))) #value bounds
```

```
            b = tf.Variable(tf.zeros([n_output]))
            encoder.append(W)
            output = tf.nn.relu(tf.matmul(current_input, W) + b)
            current_input = output

        # Get the latent representation
        z = current_input

        encoder.reverse() # For the decoder we use the same weights

        # Build the decoder
        for layer_i, n_output in enumerate(dimensions[:-1][::-1]):
            # decoder is a transposed representation of the encoder
            W = tf.transpose(encoder[layer_i])
            b = tf.Variable(tf.zeros([n_output]))
            output = tf.nn.relu(tf.matmul(current_input, W) + b)
            current_input = output

        # Get the reconstruction of the input
        y = current_input

        cost = tf.reduce_sum(tf.square(y-x)) # Sum of squared errors
        return {'x': x, 'z': z, 'y': y, 'cost': cost}


def get_conv_AE():
    '''Returns autoencoder, encoder'''
    # input
    input_img = Input(shape=(4096,1))

    #Mapping layers
    x = Conv1D(8, 3, padding='same', activation='relu')(input_img)
    x = MaxPooling1D(4, padding='same')(x)
    x = Conv1D(4, 3, padding='same', activation='relu')(x)
    x = MaxPooling1D(4, padding='same')(x)

    #Bottle-neck layer
    encoded = Conv1D(2, 3, padding='same', activation='relu')(x)
```

```
    #Demapping layers
    x = Conv1D(4, 3, padding='same', activation='relu')(encoded)
    x = UpSampling1D(4)(x)
    x = Conv1D(8, 3, padding='same', activation='relu')(x)
    x = UpSampling1D(4)(x)

    #Output
    decoded = Conv1D(1, 3, padding='same', activation='sigmoid')(x)

    #compile model
    encoder     = Model(input_img, encoded)
    autoencoder = Model(input_img, decoded)
    autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

    return (autoencoder, encoder)
```

Создание классификаторов

```
from keras.layers import Conv1D, MaxPooling1D, Dense, Flatten, Dropout
from keras.models import Sequential

def get_DNN(input_shape=(256,), num_classes=20):
    dnn = Sequential()

    #layer1
    dnn.add(Dense(512, activation='relu', input_shape=input_shape)) #(256,)
    dnn.add(Dropout(0.3))

    #layer2
    dnn.add(Dense(1024, activation='relu'))
    dnn.add(Dropout(0.3))

    #layer3
    dnn.add(Dense(512, activation='relu'))
    dnn.add(Dropout(0.3))

    #layer4
    dnn.add(Dense(256, activation='relu'))

    #Output
```

```
    dnn.add(Dense(num_classes, activation='softmax'))

    #compile model
    dnn.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

    return dnn

def get_CNN(input_shape=(256,8), num_classes=20):
    cnn = Sequential()

    #layer 1
    cnn.add(Conv1D(8, 3, activation='relu', padding='same', input_shape=input_shape)) #(256,8)
    cnn.add(MaxPooling1D(pool_size=2, padding='same'))

    #layer2
    cnn.add(Conv1D(16, 3, activation='relu', padding='same'))
    cnn.add(MaxPooling1D(pool_size=2, padding='same'))

    #layer3
    cnn.add(Conv1D(32, 3, activation='relu', padding='same'))
    cnn.add(MaxPooling1D(pool_size=2, padding='same'))

    #layer4
    cnn.add(Flatten())
    cnn.add(Dense(256, activation='relu'))
    cnn.add(Dropout(0.5))

    #Output
    cnn.add(Dense(num_classes, activation='softmax'))

    #compile model
    cnn.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

    return cnn
```

ПРИЛОЖЕНИЕ В

(обязательное)

Руководство пользователя

1. Введение.

Данное веб-приложение предназначено для выполнения процедуры классификации сортов твердой пшеницы на основе электрофоретического спектра глиадина. Программа позволяет выполнять классификацию спектров, а также обучать алгоритм распознавать другие данные. Пользователь должен иметь веб-браузер и обладать базовыми навыками работы с компьютером.

2. Назначение и условия применения.

Программный комплекс предназначен для анализа и классификации спектральных данных с помощью искусственных нейронных сетей.

Приложение реализует следующие функции:

– загрузка файла с исходным спектром;

– определение сорта загруженного спектра;

– загрузка новых данных для обучения классификатора;

– выбор модели для обучения;

– просмотр статистики в процессе обучения.

Для функционирования разработанного веб-приложения требуется только наличие браузера.

3. Подготовка к работе.

Для запуска веб-приложения следует открыть в браузере страницу с адресом программы. Если после загрузки, пользователь видит стартовую страницу приложения, значит программный продукт работает корректно. На рисунке В.1 представлена стартовая страница приложения.

4. Описание операций.

После того, как загружено главное окно приложения, можно начать работу. Для того, чтобы просмотреть параметры загруженных в систему моделей, нужно выбрать пункт меню «*Evaluate*» на панели рядом с логотипом. Тогда пользователю откроется страница (Рисунок В.2), на которой расположены две таблицы с описанием качественных характеристик каждого из двух подходов для обучения:

– сверточный подход («*convolutional architecture*»);

– глубинный подход («*deep architecture*»).

Каждая таблица содержит следующие характеристики:

– размер модели («*model size*»);

– общее число итераций, нужных для обучения («*total steps*»);

– время необходимое на выполнение одной эпохи обучения («*epoch time*»);

– время, затраченное на процедуру обучения («*training time*»).

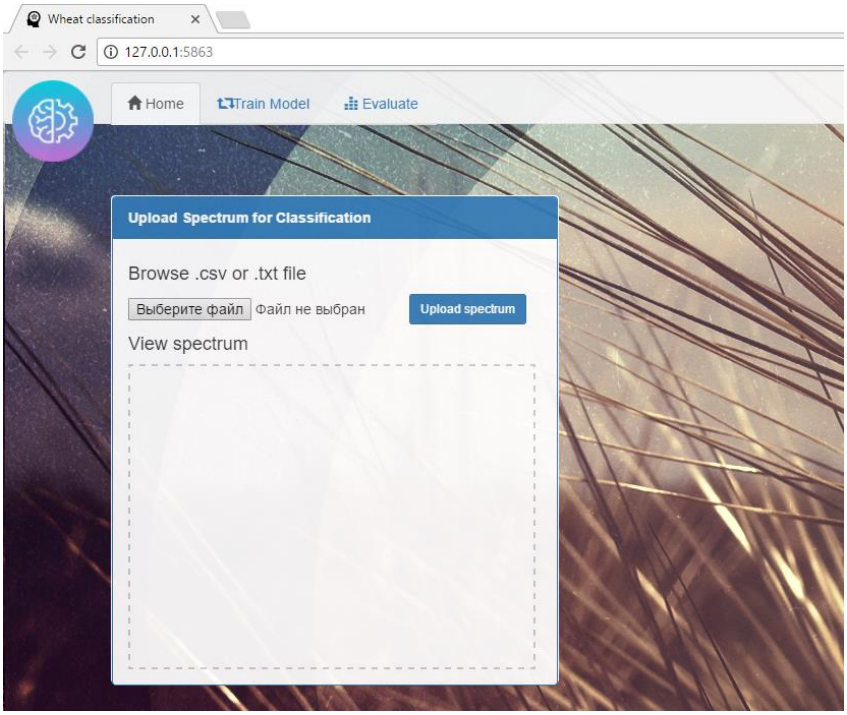


Рисунок В.1 – Главное окно приложения

На графике ниже таблиц отображена кривая падения эффективности распознавания – процент верно распознанных примеров – в зависимости от числа распознаваемых сортов.

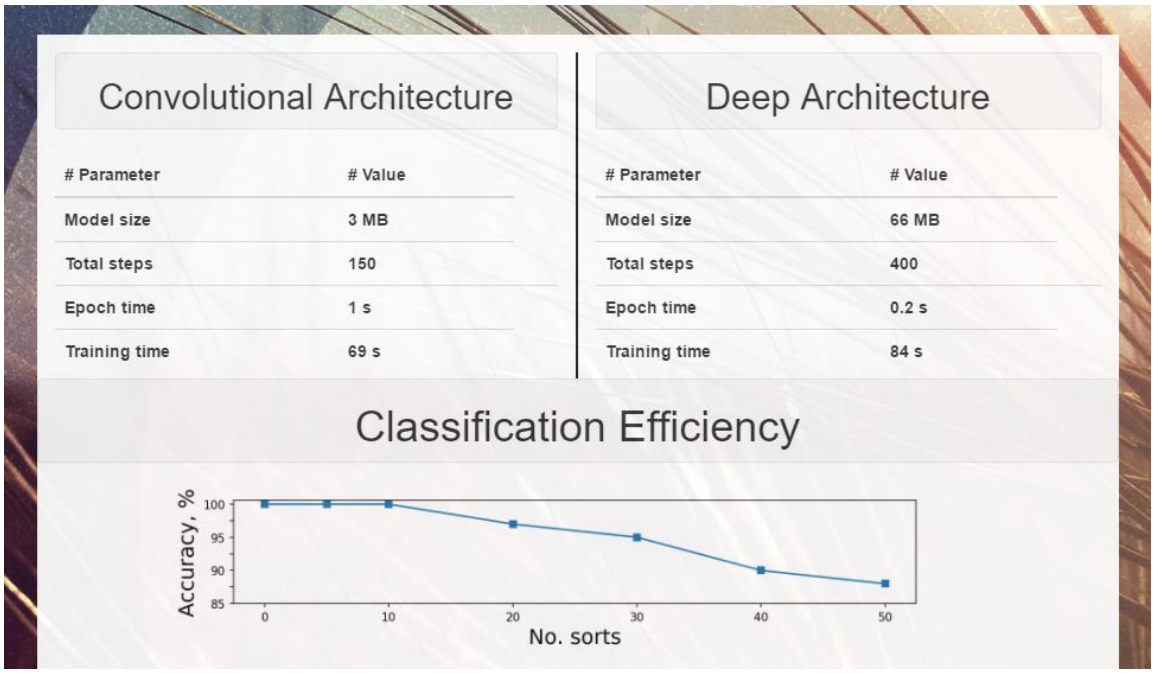


Рисунок В.2 – Страница приложения «*Evaluate*»

Для того, чтобы узнать сорт твердой пшеницы, имея файл с исходным спектром, нужно перейти на страницу «*Home*» – это главное окно приложения и в форме нажать кнопку «Выберите файл», после чего будет предложено выбрать файл с исходным спектром для распознавания, и нажать кнопку «*Upload spectrum*». В случае, если данные были введены корректно, то в окошке формы отобразится спектр из файла, а справа выведет, какой это номер сорта из базы данных (Рисунок В.3).

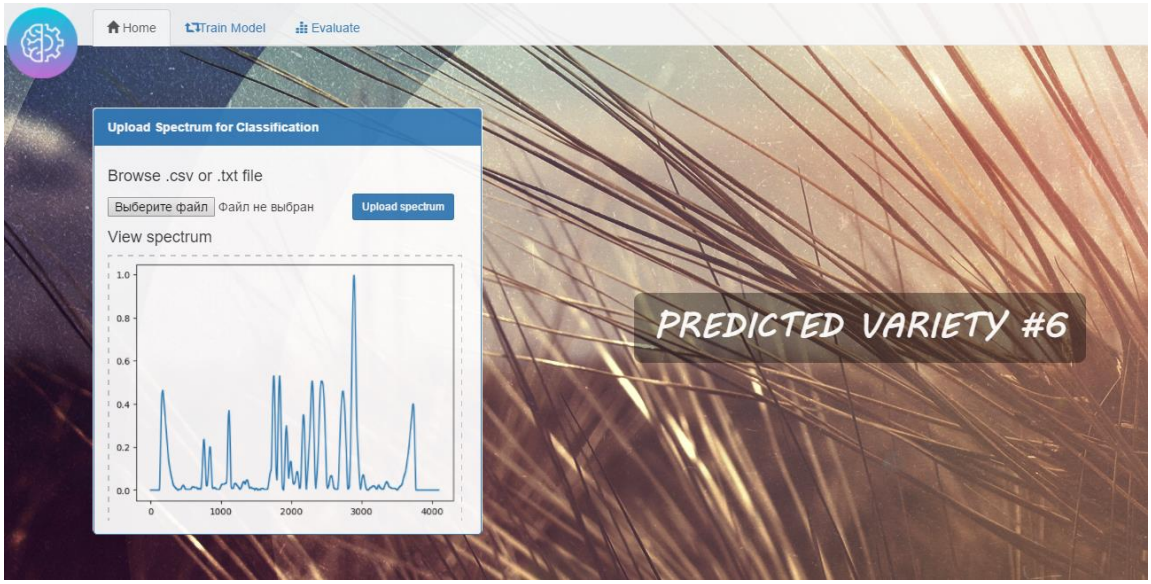


Рисунок В.3 – Классификация спектра

Для того, чтобы обучить классификатор распознавать данные спектров других растений, необходимо щелкнуть курсором мыши на пункт меню «*Train Model*». Тогда пользователь увидит форму для ввода данных обучения модели, представленную на рисунке В.4.

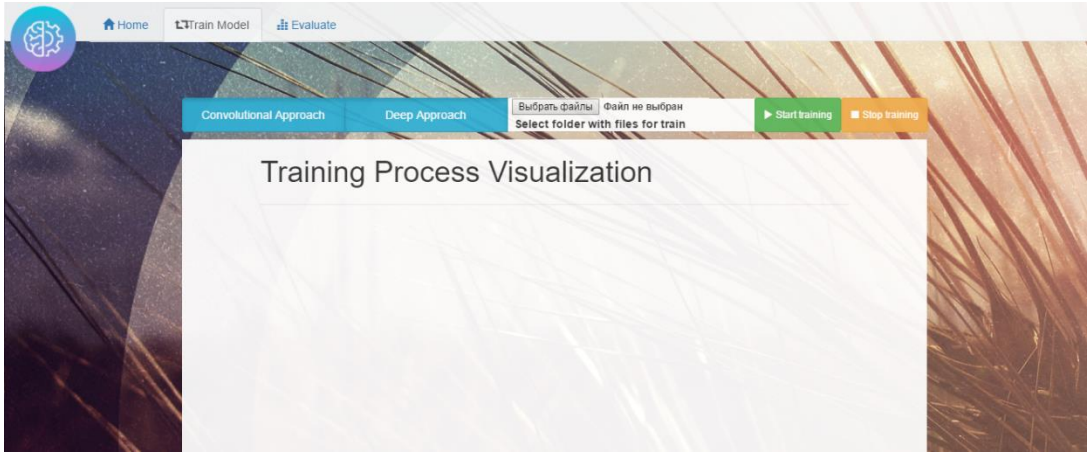


Рисунок В.4 – Страница обучения

На этой странице пользователю нужно обязательно выбрать модель для обучения – это можно сделать, щелкнув мышью один раз на одну из кнопок: «*Deep Approach*» или «*Convolutional Approach*». Далее следует выбрать папку с исходными файлами для обучения. Исходные файлы – это файлы в текстовом формате «*txt*». Каждый файл – это обучающие примеры определенного сорта, разделенные пустой строкой.

Если все данные корректны и выбран подход, можно нажать клавишу «*Start training*» и в пустом окне ниже начнет рисоваться график, представляющий собой кривые обучения (Рисунок В.5).

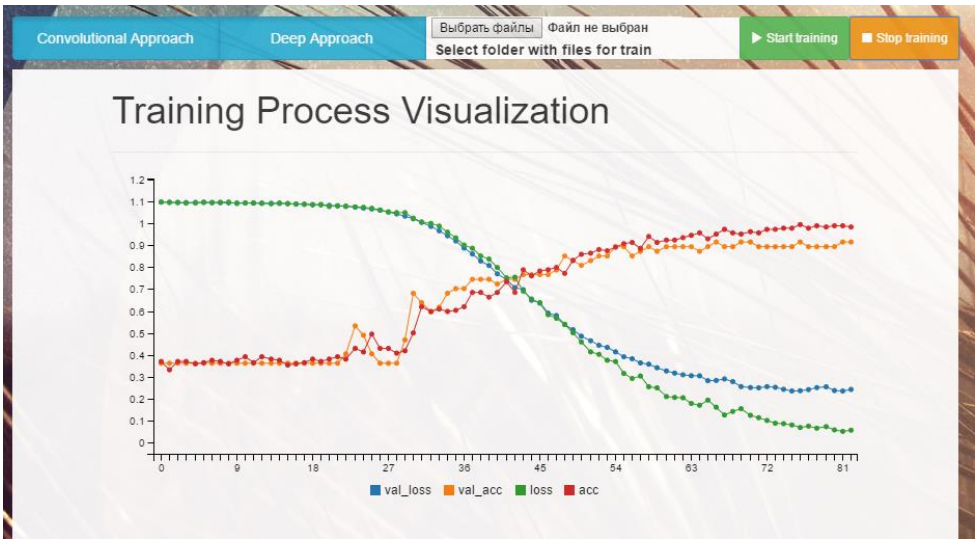


Рисунок В.5 – Визуализация процесса обучения

Можно дождаться процесса завершения обучения или же остановить его вручную, нажав клавишу «*Stop training*».

5. Аварийные ситуации.

Все аварийные ситуации в программе связаны с тем, что пользователь можно выбрать неверный файл, забыть выбрать папку для обучения или не выбрать модель. Тогда на экране появится сообщение (Рисунок В.6) об ошибке и приложение продолжит работу.

Рисунок В.6 – Сообщения об ошибках

ПРИЛОЖЕНИЕ Г
(обязательное)
Руководство программиста

1. Назначение и условия применения программы.

Программный комплекс предназначен для анализа и классификации спектральных данных с помощью искусственных нейронных сетей.

Приложение реализует следующие функции:

– загрузка файла с исходным спектром;

– определение сорта загруженного спектра;

– загрузка новых данных для обучения классификатора;

– выбор модели для обучения;

– просмотр статистики в процессе обучения.

Для функционирования разработанного веб-приложения требуется только наличие браузера.

2. Характеристики программы

Веб-приложение написано на языке программирования *Python* с использованием микрофреймворка *Flask* и имеет микросервисную архитектуру. Сервис, отвечающий за нейросетевые вычисления, реализован функциями библиотек машинного обучения *Keras* и *TensorFlow*.

3. Обращение к приложению

Для изменения исходных кодов можно использовать любой текстовый редактор и интерпретатор *Python* версии 3. Ядро приложения – файл «*project api.py*». Для того, чтобы запустить сервер, нужно в командной строке указать путь к интерпретатору и путь к файлу ядра приложения. Например: «*python project_api.py*».

4. Входные и выходные данные

Входными данными для распознавания являются файлы в текстовом формате с разделителем или без. Файл должен содержать 4096 чисел – значения кривой спектра на каждом отсчете времени.

Для обучения требуется папка с обучающей выборкой. Обучающая выборка – это файлы в текстовом формате «*txt*». Каждый файл – это обучающие примеры определенного сорта, разделенные пустой строкой. То есть если сорт имеет три примера, то файл будет содержать три спектра, разделенных пустой строкой, причем каждый отсчет спектра на отдельной строке, итого: 12290 строк.

Выходными данными является модель обученного классификатора, располагающаяся в папке «*/NN/data/models/*» на сервере.

5. Сообщения

Все виды сообщений описаны в приложении В, рисунок В.6. Программисту требуется просто закрыть окно с ошибкой и продолжить работу.

ПРИЛОЖЕНИЕ Д
(обязательное)
Руководство системного программиста

1. Общие сведения о программе

Программный комплекс предназначен для анализа и классификации спектральных данных с помощью искусственных нейронных сетей.

Приложение реализует следующие функции:

– загрузка файла с исходным спектром;

– определение сорта загруженного спектра;

– загрузка новых данных для обучения классификатора;

– выбор модели для обучения;

– просмотр статистики в процессе обучения.

Для функционирования разработанного веб-приложения требуется только наличие браузера.

2. Структура программы

Программный комплекс, разработанный в ходе выполнения дипломной работы можно разделить на несколько структурных блоков:

– ядро приложения, включающее серверную и клиентскую часть с веб-интерфейсом;

– модуль для обучения модели на новых данных;

– сервис, отвечающий за все вычисления, производимые нейронными сетями;

– программа, отвечающая за отрисовку процесса обучения сети;

– модуль, выполняющий обработку и распознавание входного спектра.

Можно отобразить взаимосвязь всех блоков приложения в виде структурной схемы, представленной на рисунке Д.1

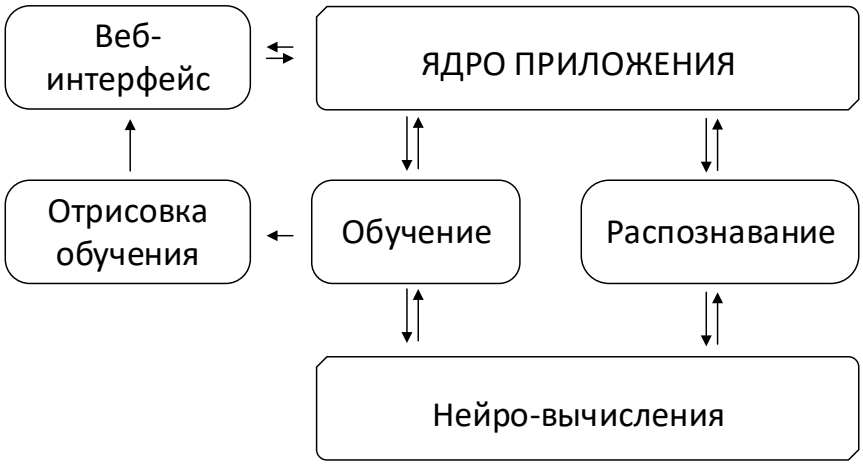


Рисунок Д.1 – Структурная схема приложения. Микросервисы

3. Настройка программы

Само веб-приложение не требует никакой определенной настройки, настраиваемой частью приложения является сервис, отвечающий за нейровычисления. Настройка заключается в изменении параметров моделей нейронных сетей:

– количества слоев нейронов;

– числа нейронов в каждом слое;

– функция ошибки;

– число эпох обучения;

– параметр обучения и т.д.

Файлы, содержащие исходные коды моделей для обучения содержатся в папке «/*NN*/*models/*» на сервере.

Также можно настроить параметры веб-приложения, такие как:

– путь папки для загрузок;

– путь к сервису отрисовки обучения;

– путь к сервисам нейровычислений;

– разрешения файлов для загрузки;

– максимальный размер файла для загрузки;

– метод классификации.

Все параметры являются словарем конфигураций приложения. Для изменения конфигураций, нужно поменять значение переменной в исходном файле ядра приложения «project_api.*py*».

4. Проверка программы

Для проверки работы классификатора в папку «/*NN*/*data/to__predict/*» на сервере добавлены файлы для классификации. Файлы, в названии которых содержится слово «*variety*» и цифра (номер сорта) являются корректными файлами для классификации. Если результат распознавания соответствует числу в названии файла, то значит спектр распознан верно.

Для проверки процедуры обучения в директорию «/*NN*/*data/*» добавлены папки: «*txt_to_train3*», «*txt_to_train10*», «*txt_to_train30*», – содержащие 3, 10 и 30 файлов для обучения соответственно. Если в процессе процедуры обучения, значение «*val_acc*» на графике страницы «*Train model*» растет, а значение «*val loss*» падает, то значит процедура выполняется корректно.

5. Дополнительные возможности

Можно заменить исходную модель классификации на обученную пользователем. Для этого нужно изменить конфигурацию – путь к файлу модели-классификатора.

6. Сообщения

Все виды сообщений описаны в приложении В, рисунок В.6. Требуется просто закрыть окно с ошибкой и продолжить работу.

ПРИЛОЖЕНИЕ Ж

(справочное)

Примеры работы программы

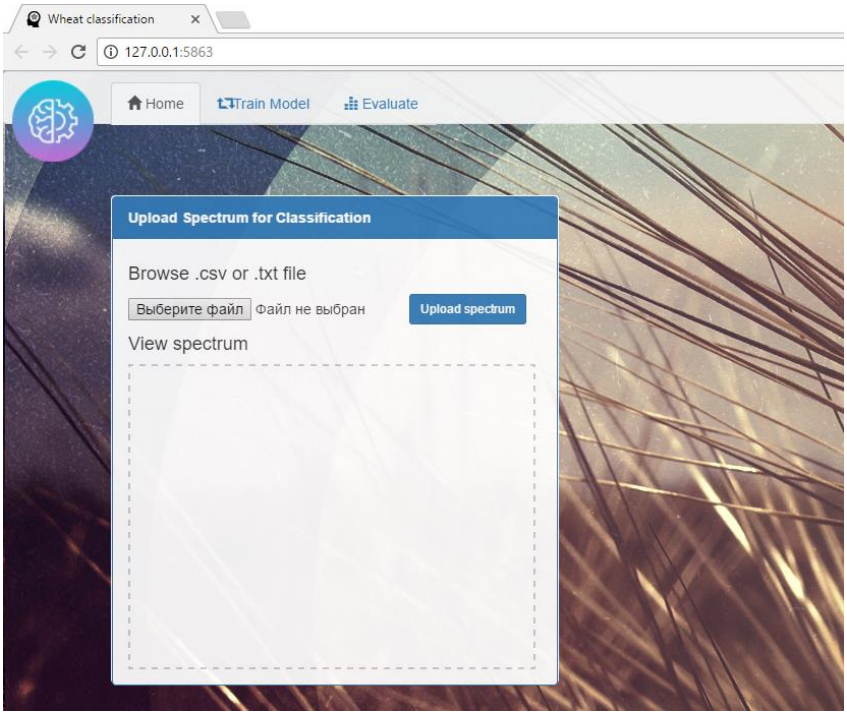


Рисунок Ж.1 – Главное окно приложения

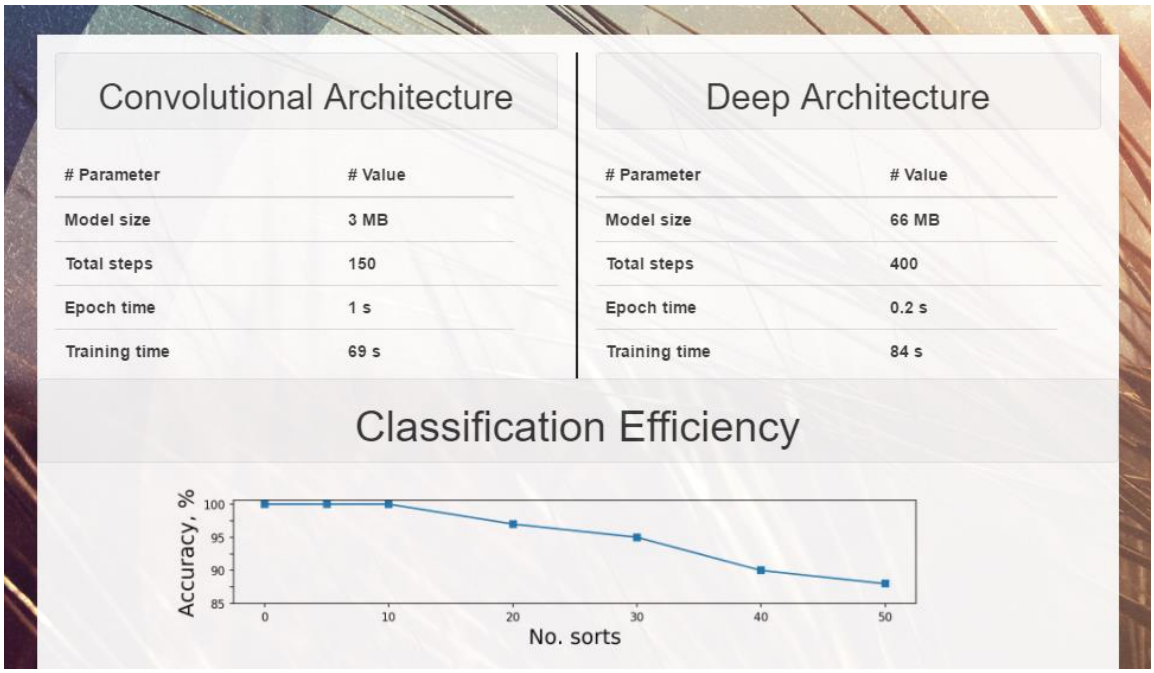


Рисунок Ж.2 – Страница приложения «*Evaluate*»

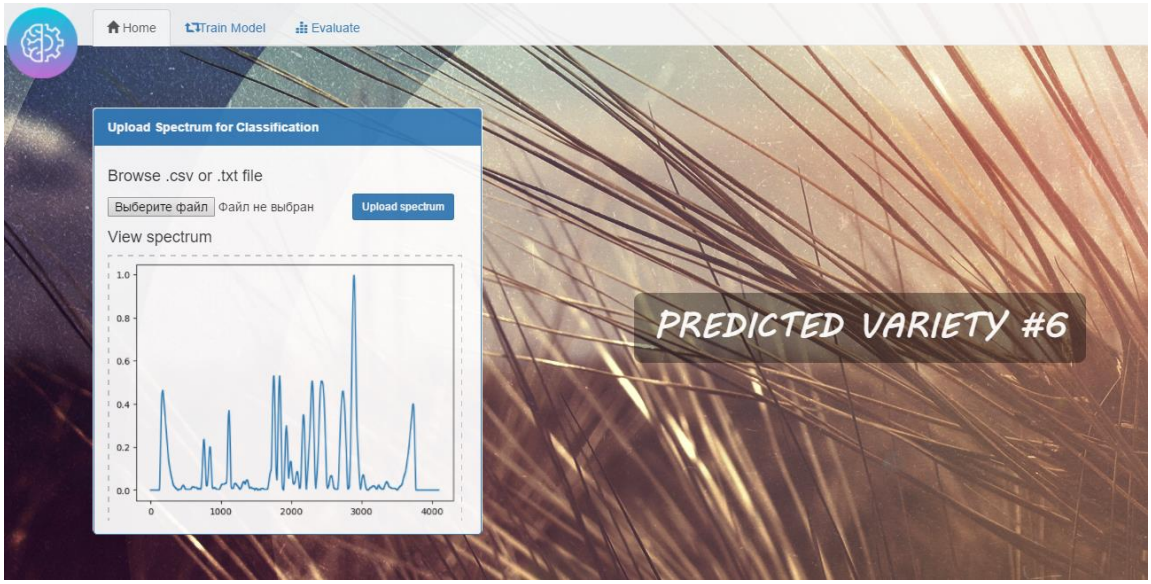


Рисунок Ж.3 – Классификация спектра

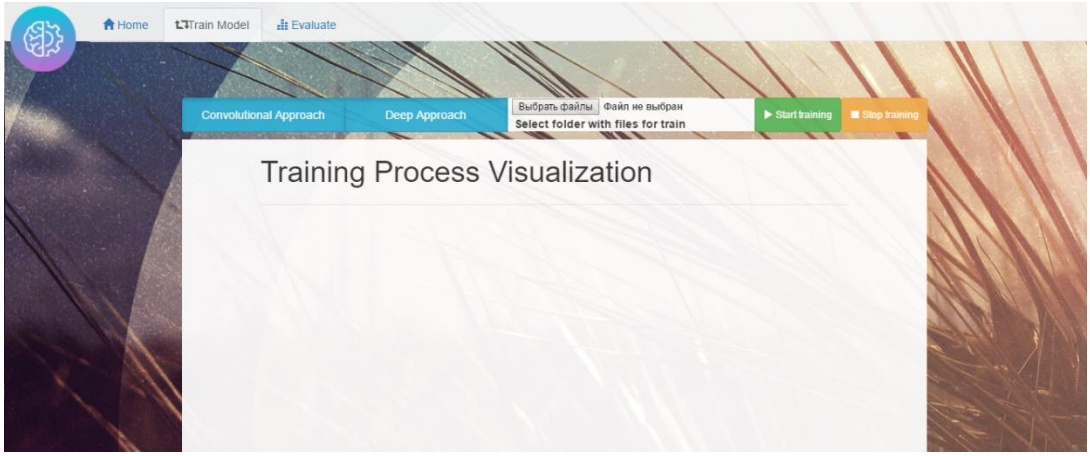


Рисунок Ж.4 – Страница обучения

Рисунок Ж.5 – Сообщения об ошибках

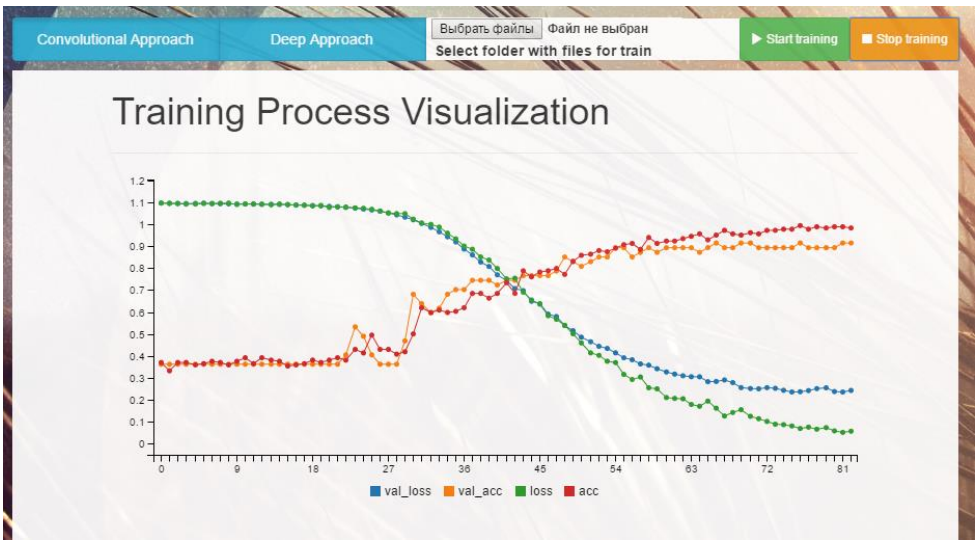


Рисунок Ж.6 – Визуализация процесса обучения

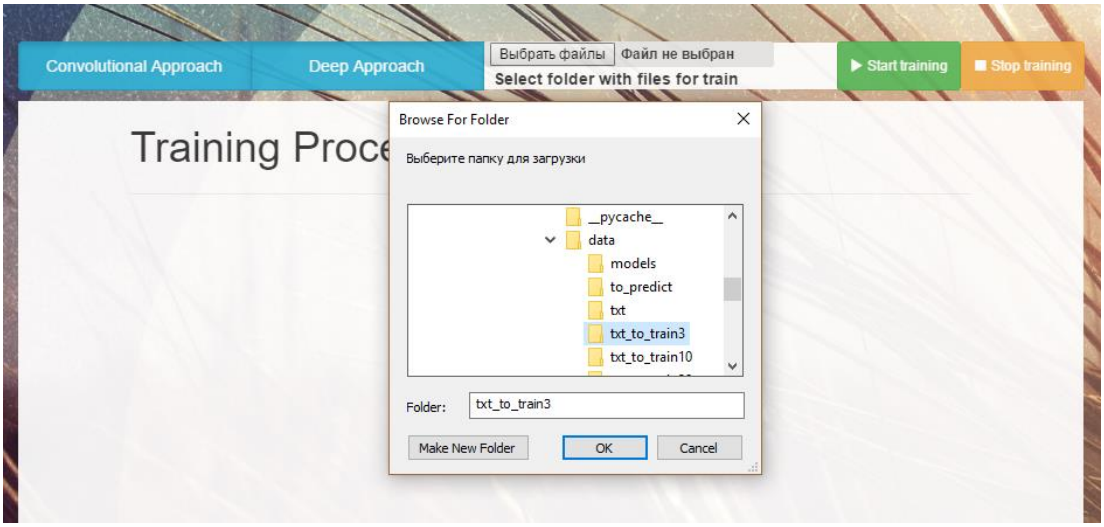


Рисунок Ж.7 – Выбор папки для обучения

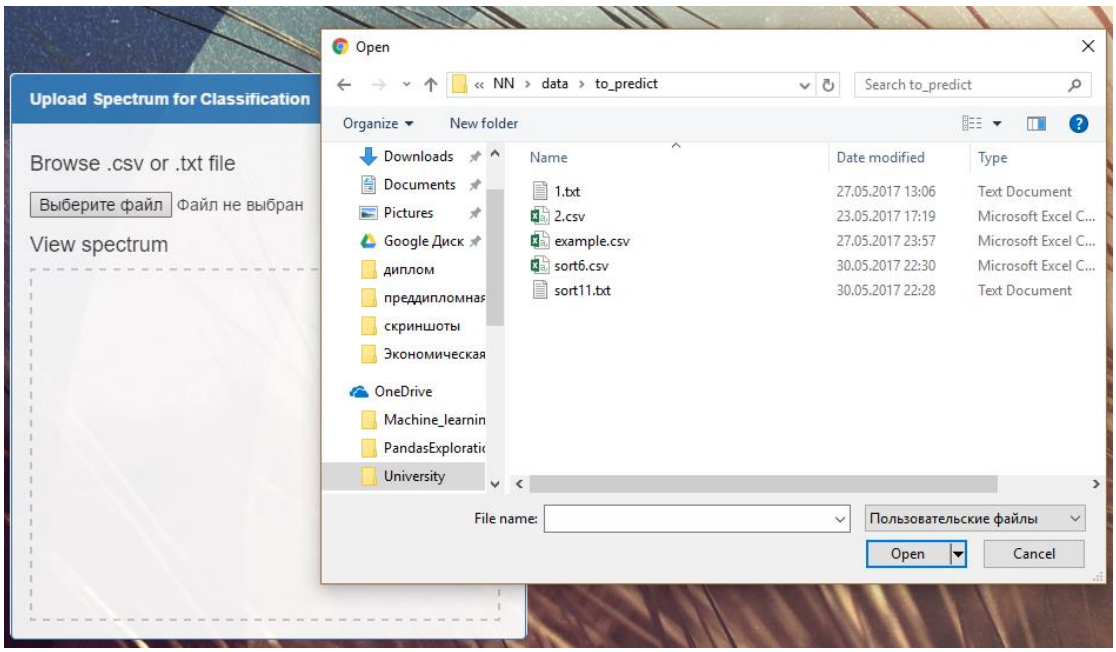


Рисунок Ж.8 – Выбор файла для обучения

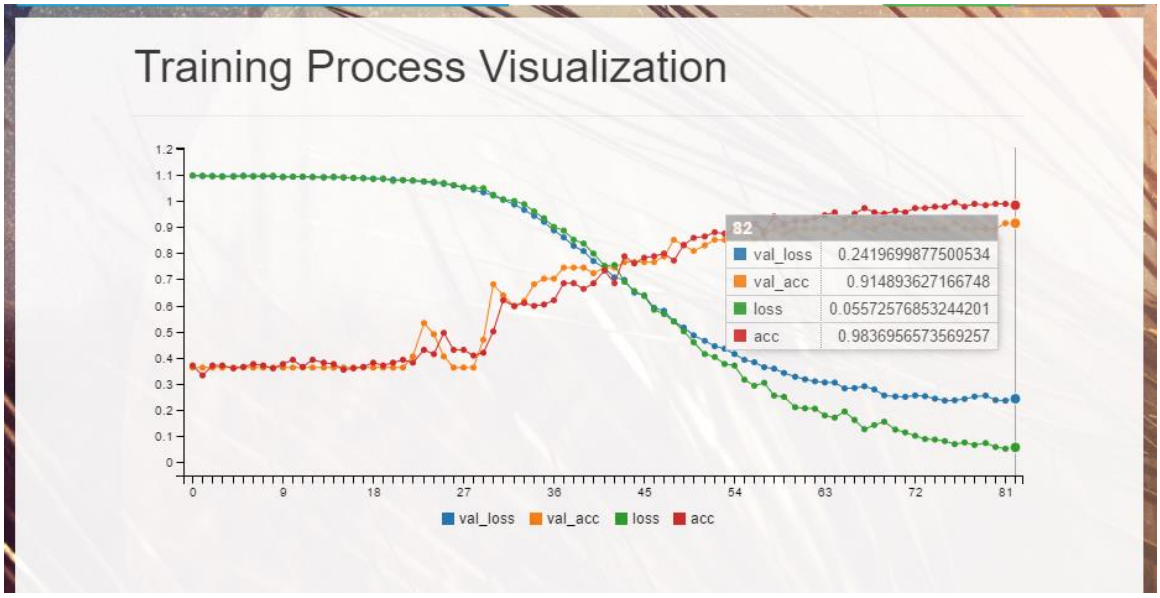


Рисунок Ж.9 – Детализация точки на графике

ПРИЛОЖЕНИЕ К

(справочное)

Результаты расчёта экономического обоснования

Таблица К.1 – Расчет коэффициентов эквивалентности

Наименование параметра	Вес парамет-ра, β	Значения Параметра			$\frac{P_б}{P_э}$	$\frac{P_н}{P_э}$	$\beta \cdot \frac{P_б}{P_э}$	$\beta \cdot \frac{P_н}{P_э}$
		$P_б$	$P_н$	$P_э$				
Объем памяти	0,1	15	5	3	0,2	0,6	0,02	0,06
Время обработки данных	0,2	0,5	0,35	0,2	0,4	0,57	0,08	0,114
Эффективность классификации	0,25	0,7	0,9	1	0,7	0,9	0,175	0,225
Время обучения	0,05	1,6	1,4	1	0,62	0,71	0,031	0,035
Отказы	0,4	2	1	1	0,5	1	0,2	0,4
Итого							0,506	0,834
Коэффициент эквивалентности							0,506/0,834=1,65	

Таблица К.2 – Расчет коэффициента изменения функциональных возможностей

Наименование показателя	Балльная оценка базового ПП	Балльная оценка нового ПП
Объем памяти	2	4
Функциональные возможности	3	5
Быстродействие	3	4
Удобство интерфейса	2	5
Степень утомляемости	3	5
Производительность труда	4	5
Итого	17	28
Коэффициент функциональных возможностей	28/17 = 1,65	

Таблица К.3 – Расчет уровня конкурентоспособности нового ПП

Коэффициенты	Значение
1	2
Коэффициент эквивалентности	1,65
Коэффициент изменения функциональных возможностей	1,65
Коэффициент соответствия нормативам	1
Коэффициент цены потребления	400/480=0,83

Продолжение таблицы К.3

1	2
Интегральный коэффициент конкурентоспособности	(1,65*1,65*1)/0,83 = 3,9

Таблица К.4 – Исходный и уточнённый объём строк исходного кода

Код функции	Наименование (содержание) функций	Объем функции строк исходного кода	
		по каталогу (V_o)	уточненный (V_y)
Ввод, анализ входной информации, генерация кодов и процессор входного языка			
101	Организация ввода информации	120	100
102	Контроль, предварительная обработка и ввод информации	40	10
109	Управление вводом-выводом	45	40
107	Организация ввода-вывода информации в интерактивном режиме	56	50
Формирование и обработка файлов			
303	Обработка файлов	230	220
Управление ПО, компонентами ПО и внешними устройствами			
504	Обработка прерываний	18	10
506	Обработка ошибочных сбойных ситуаций	35	30
Расчетные задачи, формирование и вывод на внешние носители документов сложной формы и файлов			
701	Математическая статистика и прогнозирование	500	240
707	Графический вывод результатов	380	330
709	Изменение состояния ресурсов в интерактивном режиме	310	230
	Итого	1734	1260

Таблица К.5 – Значения коэффициентов удельных весов трудоемкости стадий разработки ПО в общей трудоемкости

Категория новизны ПО	Без применения *CASE*-технологий				
	Стадии разработки ПО				
	ТЗ	ЭП	ТП	РП	ВН
	Значения коэффициентов				
	$K_{т.з}$	$K_{э.п}$	$K_{т.п}$	$K_{р.п}$	$K_{в.н}$
В	0,10	0,20	0,30	0,30	0,10

Таблица К.6 – Расчет общей трудоемкости разработки ПО

Показатели	Стадии разработки					Итого
	ТЗ	ЭП	ТП	РП	ВН	
1	2	3	4	5	6	7
Общий объем ПО ($V_о$), количество строк	–	–	–	–	–	1734
Общий уточненный объем ПО ($V_у$), количество строк	–	–	–	–	–	1260
Категория сложности разрабатываемого ПО	–	–	–	–	–	2
Нормативная трудоемкость разработки ПО ($Т_н$), чел.-дн.	7	15	22	22	7	73
Коэффициент повышения сложности ПО ($K_с$)	1,26	1,26	1,26	1,26	1,26	–
Коэффициент, учитывающий новизну ПО ($K_н$)	0,72	0,72	0,72	0,72	0,72	–
Коэффициент, учитывающий степень использования стандартных модулей ($K_т$)	–	–	–	0,65	–	–
Коэффициент, учитывающий средства разработки ПО ($K_{у.р}$)	1,3	1,3	1,3	1,3	1,3	–
Коэффициенты удельных весов трудоемкости стадий разработки ПО ($K_{т.з}, K_{э.п}, K_{т.п}, K_{р.п}, K_{в.н}$)	0,10	0,20	0,30	0,30	0,10	1,0
Распределение скорректированной (с учетом $K_с$, $K_н$, $K_т$, $K_{у.р}$) трудоемкости ПО по стадиям, чел.-дн.	9	17	26	17	7	76
Общая трудоемкость разработки ($Т_о$), чел.-дн.	–	–	–	–	–	76

Таблица К.7 – Параметры расчета производственных затрат на разработку ПО

Параметр	Единица измерения	Значение
1	2	3
Тарифная ставка 1-го разряда	руб.	31
Разряд разработчика	–	1
Тарифный коэффициент для 15 разряда	–	5,68
Коэффициент $К_{ув}$	–	1,5
Норматив отчислений на доп. зарплату разработчиков ($Н_{доп}$)	%	10
Численность обслуживающего персонала	чел.	–
Разряд обслуживающего персонала	–	–
Тарифный коэффициент	–	–
Средняя годовая ставка арендных платежей ($С_{ар}$) (по результатам мониторинга предложений по аренде помещений)	руб./м^2	15,2
Площадь помещения (S)	м^2	10
Количество ПЭВМ ($Q_{эвм}$)	шт.	1
Затраты на приобретение единицы ПЭВМ	руб.	700
Стоимость одного кВт-часа электроэнергии ($С_{эл}$)	руб.	0,1192
Коэффициент потерь рабочего времени (Кпот)	–	0,1
Затраты на технологию ($З_{тех}$)	руб.	–
Норматив общепроизводственных затрат ($Н_{доп}$)	%	5
Норматив непроизводственных затрат ($Н_{непр}$)	%	5

Таблица К.8 – Результаты расчета суммарных затрат на разработку,руб.

Статья затрат	Итого
1	2
Затраты на оплату труда разработчиков ($З_{тр}$)	1411,5
Основная заработная плата разработчиков	957,6
Дополнительная заработная плата разработчиков	95,76
Отчисления от осн. и доп. заработной платы	358,14
Затраты машинного времени ($З_{м.в}$)	62,64
Стоимость машино-часа, руб./час	0,27
Затраты на заработную плату обслуживающего персонала	0
Годовые затраты на аренду помещения	152
Сумма годовых амортизационных отчислений	156,8
Действительный годовой фонд времени работы ПЭВМ, ч.	1778,4

Продолжение таблицы К.8

1	2
Затраты на текущий и профилактический ремонт	39,2
Прочие затраты, связанные с эксплуатацией ЭВМ	39,2
Машинное время ЭВМ, ч.	232
Затраты на изготовление эталонного экземпляра ($З_{эт}$)	5
Затраты на технологию ($З_{тех}$)	0
Затраты на материалы ($З_{мат}$)	7,84
Общепроизводственные затраты ($З_{общ.пр}$)	47,88
Непроизводственные (коммерческие) затраты ($З_{непр}$)	47,88
Суммарные затраты на разработку ПО ($З_{р}$)	1582,58
Прибыль от реализации ПО ($П_{р}$)	417,77
Отпускная цена ПО без НДС ($Ц_{отп}$)	2057,35
Налог на добавленную стоимость ($Р_{ндс}$)	411,47
Отпускная цена ПО с НДС ($Ц_{отп.ндс}$)	2468,82
Торговая наценка ($Т_{н}$)	370,32
Розничная цена ПО ($Ц_{розн}$)	2839,14