МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

**ИВАНОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**ИНСТИТУТ МАТЕМАТИКИ, ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ЕСТЕСТВЕННЫХ НАУК**

|  |  |
| --- | --- |
| «Рекомендовать к защите»  Заведующий кафедрой информационных технологий и прикладной математики, доцент  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Данилова С.В.  протокол заседания кафедры № \_\_\_\_  от «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2023 г. |  |

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

(БАКАЛАВРСКАЯ РАБОТА)

**зависимость качества распознавания mnist от архитектуры нейросети**

|  |  |
| --- | --- |
| Направление подготовки: | 02.03.02 Фундаментальная информатика и информационные технологии |
| Направленность образовательной программы: | Фундаментальная информатика и информационные технологии |
| Выпускную квалификационную работу выполнил: | студент 4 курса очной формы обучения \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Аннаев Рустам |
| Руководитель выпускной квалификационной работы: | доцент кафедры информационных технологий и прикладной математики,  канд. физ.-мат. наук,. \_\_\_\_\_\_\_\_Хашин Сергей Иванович |

Иваново, 2023

Министерство образования и науки Российской Федерации

ФГБОУ ВО «Ивановский государственный университет»

Институт математики, информационных технологий и естественных наук

Кафедра информационных технологий

и прикладной математики

|  |
| --- |
| **Утверждаю:**  Зав. кафедрой ИТ иПМ  канд.экон.наук, доцент\_\_\_\_\_\_\_\_\_Данилова С.В. |
| «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2023год |

**ЗАДАНИЕ**

по подготовке выпускной квалификационной работы

студента 4 курса бакалавриата

института математики, информационных технологий и естественных наук

Аннаева Рустама

(фамилия, имя и отчество студента)

Тема работы: «Зависимость качества распознавания MNIST от архитектуры нейросети»

(утверждена приказом ИвГУ № 2п/275 от «2» декабря 2022 г.)

Срок сдачи студентом законченной работы: «15» июня 2023 г.

Исходные данные к работе:

методические указания по выполнению выпускной квалификационной работы для бакалавров по направлению 02.03.02 Фундаментальная информатика и информационные технологии.

Краткое содержание работы:

В работе исследуется, как различные архитектуры нейросетей влияют на точность распознавания цифр из набора данных MNIST. Результаты исследования могут помочь выбрать оптимальную архитектуру нейросети для решения задачи распознавания цифр.

|  |  |  |  |
| --- | --- | --- | --- |
| Дата выдачи задания: | «3» мая 2023 г. | | |
| Научный руководитель,  канд. физ.-мат. наук, доцент | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) |  | Хашин С.И. |
| Задание принял к исполнению,  студент | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) |  | Аннаев Р. |

Министерство образования и науки Российской Федерации

ФГБОУ ВО «Ивановский государственный университет»

Институт математики, информационных технологий и естественных наук

Кафедра информационных технологий

и прикладной математики

|  |
| --- |
| **Утверждаю:**  Зав. кафедрой ИТ и ПМ  канд.экон.наук, доцент\_\_\_\_\_\_\_\_\_Данилова С.В. |
| «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2023год |

**КАЛЕНДАРНЫЙ ГРАФИК**

выполнения выпускной квалификационной работы

(бакалаврской работы)

студента института математики, информационных технологий и естественных наук Аннаева Рустама

(фамилия, имя и отчество студента)

Тема работы: «Зависимость качества распознавания MNIST от архитектуры нейросети»

(утверждена приказом ИвГУ № 2п/275 от «2» декабря 2022г.)

|  |  |  |
| --- | --- | --- |
| Характер и объем работ | Плановый срок выполнения | Отметки  о выполнении |
| 1. Сбор информации и изучение  литературы | 15.05.2023 г. |  |
| 2. Обработка и анализ материалов  для написания ВКР | 20.05.2023 г. |  |
| 3. Написание аналитического раздела | 25.05.2023 г. |  |
| 4. Написание проектного раздела | 31.05.2023 г. |  |
| 6. Прохождение нормоконтроля | 6.06.2023 г. |  |
| 7. Предоставление ВКР на рецензию | 8.06.2023 г. |  |
| 8. Подготовка презентационного материала | 12.06.2023 г. |  |
| 9. Предоставление ВКР на кафедру  для допуска к защите | 15.06.2023 г. |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Дата выдачи календарного графика: | «2» мая 2023 г. | | |
| Научный руководитель,  канд. физ.-мат. наук, доцент | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) |  | Хашин С.И. |
| Календарный график принял к исполнению,  студент | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) |  | Аннаев Р. |

Оглавление

[Введение 5](#_Toc136714478)

[1. НЕЙРОСЕТЬ 6](#_Toc136714479)

[2. KERAS И TENSORFLOW 11](#_Toc136714480)

[3. МЕТОД ГЛАВНЫХ КОМПОНЕНТ 19](#_Toc136714481)

[4. ВЛИЯНИЕ МЕТОДА ГЛАВНЫХ КОМПОНЕНТ НА ТОЧНОСТЬ РАСПОЗНАВАНИЯ MNIST И CIFAR-10 23](#_Toc136714482)

[5. СРАВНЕНИЕ РЕЗУЛЬТАТОВ 28](#_Toc136714483)

[Заключение 38](#_Toc136714484)

[Список литературы 39](#_Toc136714485)

# Введение

Данная работа посвящена исследованию зависимости качества распознавания MNIST от архитектуры нейросети, а в особенности влиянию метода главных компонент на точность распознавания. Вначале мы рассмотрим, что представляет собой нейронная сеть, далее, рассмотрим инструменты для работы с нейронными сетями, затем ознакомимся с методом главных компонент, и как данный метод влияет на работу нейросети. В конце работы сравним результаты между работой нейронных сетей, где к исходным данным был применен метод главных компонент и где исходные данные не были изменены.

Актуальность данной темы обусловлена широким применением распознавания изображений в различных областях, таких как компьютерное зрение, автоматическая обработка документов, биометрическая идентификация и многие другие. Точность распознавания играет важную роль в этих областях, и поэтому важно изучать различные методы и архитектуры нейросетей для повышения точности распознавания. Кроме того, исследование влияния архитектуры нейросети на качество распознавания MNIST имеет большое практическое значение, как в учебных, так и в индустриальных целях. Это позволяет разработчикам создавать более эффективные системы распознавания изображений, а студентам и исследователям - лучше понимать основы глубокого обучения и их применения.

Таким образом, данная работа представляет собой важное исследование в области глубокого обучения и распознавания изображений с использованием набора данных MNIST и CIFAR-10. Мы представим методику и результаты экспериментов, а также дадим рекомендации по выбору оптимальной архитектуры нейросети и применению метода главных компонент для улучшения точности распознавания.

# 1. **НЕЙРОСЕТЬ**

Нейронная сеть — это метод в искусственном интеллекте, который учит компьютеры обрабатывать данные таким же способом, как и человеческий мозг. Это тип процесса машинного обучения, называемый глубоким обучением, который использует взаимосвязанные узлы или нейроны в слоистой структуре, напоминающей человеческий мозг. Он создает адаптивную систему, с помощью которой компьютеры учатся на своих ошибках и постоянно совершенствуются [1].

Термин "нейронная сеть" появился в середине XX века. Первые работы, в которых были получены основные результаты в данном направлении, были проделаны Уорреном Мак-Каллоком и Уолтером Питтсом. В 1943 году ими была разработана компьютерная модель нейронной сети на основе математических алгоритмов и теории деятельности головного мозга. Они выдвинули предположение, что нейроны можно упрощённо рассматривать как устройства, оперирующие двоичными числами, и назвали эту модель "пороговой логикой". Подобно своему биологическому прототипу нейроны Мак-Каллока-Питтса были способны обучаться путём подстройки параметров, описывающих синаптическую проводимость. Исследователи предложили конструкцию сети из электронных нейронов и показали, что подобная сеть может выполнять практически любые вообразимые числовые или логические операции. Мак-Каллок и Питтс предположили, что такая сеть в состоянии также обучаться, распознавать образы, обобщать, т. е. обладает всеми чертами интеллекта.

Интерес к исследованию нейронных сетей угас после публикации работы по машинному обучению Минского и Пейперта в 1969 году. Ими были обнаружены основные вычислительные проблемы, возникающие при компьютерной реализации искусственных нейронных сетей. Первая проблема состояла в том, что однослойные нейронные сети не могли совершать "сложение по модулю 2", то есть реализовать функцию "Исключающее ИЛИ". Второй важной проблемой было то, что компьютеры не обладали достаточной вычислительной мощностью, чтобы эффективно обрабатывать огромный объём вычислений, необходимый для больших нейронных сетей. Исследования нейронных сетей замедлились до того времени, когда компьютеры достигли больших вычислительных мощностей. Одним из важных шагов, стимулировавших дальнейшие исследования, стала разработка в 1975 году Вербосом метода обратного распространения ошибки, который позволил эффективно решать задачу обучения многослойных сетей и решить проблему со "сложением по модулю 2".

В 1975 году Фукусимой был разработан когнитрон, который стал одной из первых многослойных нейронных сетей. Фактическая структура сети и методы, используемые в когнитроне для настройки относительных весов связей, варьировались от одной стратегии к другой. Каждая из стратегий имела свои преимущества и недостатки. Сети могли распространять информацию только в одном направлении или перебрасывать информацию из одного конца в другой, пока не активировались все узлы, и сеть не приходила в конечное состояние. Достичь двусторонней передачи информации между нейронами удалось лишь в сети Хопфилда (1982), и специализация этих узлов для конкретных целей была введена в первых гибридных сетях. Алгоритм параллельной распределённой обработки данных в середине 1980 годов стал популярен под названием коннективизм. В 1986 году в работе Руммельхарта и Мак-Клелланда коннективизм был использован для компьютерного моделирования нейронных процессов.

В 2006 году было предложено несколько неконтролируемых процедур обучения нейронных сетей с одним или несколькими слоями с использованием, так называемых алгоритмов глубокого обучения. Эти алгоритмы могут быть использованы для изучения промежуточных представлений, как с выходным сигналом, так и без него, чтобы понять основные особенности распределения сенсорных сигналов, поступающих на каждый слой нейронной сети.

Как и во многих других случаях, задачи высокой сложности требуют применения не одного, а нескольких методов решения или их синтеза. Не исключение и искусственные нейронные сети. С самого начала нынешнего столетия в работах различных исследователей активно описываются нейро-нечёткие сети, ячеечно-нейросетевые модели [2].

Технологии нейронных сетей продолжали развиваться, но только в последние годы они достигли своего максимального потенциала, благодаря доступности вычислительных мощностей и большому количеству данных. Нейросети позволяют решать и прогнозировать сложные задачи, такие как распознавание образов, обработка естественного языка и анализ больших объёмов данных, что делает их незаменимым инструментом в современных технологиях. Однако развитие нейросетей не останавливается и предоставляет множество новых возможностей для исследования и применения в различных сферах жизни.

Базовая нейронная сеть содержит три слоя искусственных нейронов:

1. входной — обрабатывает информацию извне, анализирует или классифицирует ее и передает на следующий слой;
2. скрытый (их может быть несколько) — анализирует выходные данные предыдущего слоя, обрабатывает их и передает на следующий;
3. выходной — выдает окончательный результат после обработки всех данных.

Глубокие нейронные сети отличаются тем, что искусственные нейроны в них связаны друг с другом, а каждой такой связи присваивается определенный вес, который отражает ее значимость. Кроме того, связь между нейронами может быть «упреждающей». Это означает, что данные проходят через них только в одном направлении. Такое происходит, если значение «веса» соединения ниже заданного. При обучении нейронной сети все ее «веса» изначально задаются случайными значениями. Обучающие данные подаются на нижний, или входной, слой. Затем они проходят через последующие слои, пока не достигают выходного. Во время обучения «веса» и пороговые значения постоянно корректируются до тех пор, пока данные обучения не будут постоянно давать одинаковые результаты. Эти «веса» помогают определить важность той или иной переменной во входных данных. При прохождении каждого слоя входные данные умножаются на их «веса», а затем суммируются. Если получившееся значение выше заданного порога, то нейрон активируется и передает данные на следующий уровень.

В зависимости от архитектуры нейросети делятся на типы:

* прямого распространения — обрабатывают входные данные и сразу выдают результат. Чаще всего применяются для распознавания образов и текста, а также классификации данных;
* рекуррентные — перенаправляют информацию туда и обратно по слоям, пока не получат конечный результат. Этот тип обычно используется для прогнозирования;
* сверточные — обрабатывают каждый признак в отдельном слое. Такой тип применяется в классификации изображений, обработке языка и т.д.

Помимо основных типов встречаются десятки подтипов нейросетей. Например, модульные — это, по сути, совокупность нейросетей, которые работают независимо друг от друга, чтобы ускорить вычисления. Нейронные сети широко используются в самых разных областях — от медицины и до сферы развлечений. Они выполняют несколько основных типов задач:

1. Автоматическую генерацию контента:

* Чат-бот ChatGPT от OpenA может отвечать на вопросы, поддерживать диалог, искать ошибки в коде, сочинять стихи, писать сценарии и т.д.
* Нейросети DALL-E, Midjourney, Stable Diffusion генерируют картинки по текстовому описанию.
* Нейросеть Imaginary soundscape создает фоновую музыку по изображению.
* VALL-E озвучивает текст, точно имитируя голос человека и его интонации.

1. Распознают и обрабатывают естественный язык:

* Чат-бот ChatGPT способен вести диалог, искать ошибки в коде, сочинять стихи, писать сценарии и даже давать медицинские советы. Нейросеть можно встраивать в любые сайты и сервисы, а Microsoft уже внедрила чат-бота в свой поисковик Bing для получения более точных результатов поиска.
* Чат-бот LaMDA может вести диалог на любую тему и отвечать на любые вопросы пользователя. В перспективе его внедрение позволит создавать новые категории приложений.
* Нейросеть Yandex Cloud распознает более десяти языков одновременно и переводит ответы на вопросы. На ее базе можно создавать многоязычных голосовых помощников и роботов для call-центров.

1. Классифицируют объекты

* Нейросеть российской компании NTechLab, которая работает с городской сетью камер в Москве, позволяет выявлять нарушителей. Также ее можно оптимизировать для идентификации по лицу в общественном транспорте для оплаты и организации безопасной системы пропусков на предприятиях.
* Нейросеть в основе системы помощи водителю Ford BlueCruise распознает объекты на дороге, что позволяет контролировать работу автопилота.
* Российский сервис YouScan с помощью нейросетей отслеживает упоминание брендов в соцсетях и сегментирует их для аналитики.

Кроме того, нейросети применяются для более узкоспециализированных задач:

* Регрессии. Нейронные сети помогают прогнозировать биржевые курсы, оценивать стоимость имущества и так далее;
* Прогнозирования. Например, нейросети широко применяются в банкинге, чтобы оценить платежеспособность новых заемщиков;
* Кластеризации. Нейросети помогают исследователям и специалистам big data сортировать большие объемы данных в условиях, когда конечное количество классов неизвестно. Например, кластеризация применяется для выявления классов картинок. На простейшем уровне такая функция реализована в любом современном смартфоне, она позволяет автоматически сортировать фото и видео по папкам и выдавать пользователям тематические подборки [3].

Чтобы создавать и обучать такие нейросети, нужно использовать специальные инструменты и библиотеки.

# 2. KERAS И TENSORFLOW

Одним из наиболее популярных и широко используемых инструментов для разработки нейросетей является Keras.

Keras — это библиотека для языка программирования Python, которая предназначена для глубокого машинного обучения. Она позволяет быстрее создавать и настраивать модели — схемы, по которым распространяется и подсчитывается информация при обучении. Но сложных математических вычислений Keras не выполняет и используется как надстройка над другими библиотеками. Keras с версии 2.3 — это надстройка над библиотекой TensorFlow, которая нужна для машинного обучения. TensorFlow выполняет все низкоуровневые вычисления и преобразования и служит своеобразным движком, математическим ядром. Keras же управляет моделями, по которым проходят вычисления. До версии 2.3 Keras мог использовать в качестве движка вычислительные библиотеки Theano или CNTK. Но в новых версиях поддержка прекратилась, теперь библиотека работает только с TensorFlow. Keras создавалась как гибкая модульная библиотека, которую легко настраивать и модифицировать. Она бесплатная, у нее открытый исходный код, который может посмотреть любой желающий.

Keras имеет узкую специализацию. Это инструмент для специалистов по машинному обучению, которые работают с языком Python: именно его чаще всего используют благодаря удобству математических вычислений. Keras применяют разработчики, которые создают, настраивают и тестируют системы машинного обучения и искусственного интеллекта, в первую очередь нейронные сети.

Keras предоставляет удобный инструмент для построения и обучения нейронных сетей. С его помощью можно легко создавать модели и настраивать слои в них для достижения более точных результатов. Keras также обеспечивает обработку входных и выходных данных, преобразует данные в нужный формат, подбирает датасеты и визуализирует модели для анализа ее работы. Кроме того, в Keras можно определить функции ошибки и оптимизаторы для подготовки модели к работе, а также провести обучение и тестирование. Он также предлагает возможность сборки и запуска программ машинного обучения. Все перечисленное можно делать и без Keras, но дольше и сложнее. Keras выступает как программный интерфейс, который упрощает действия.

Машинное обучение — по сути, обработка информации с помощью запрограммированной сети, где на основе определенных данных делаются те или иные выводы [9, c.28]. Структура сети называется моделью. Часто ее представляют в виде графа, схемы или таблицы.

Глубокое обучение — это метод машинного обучения, который позволяет предсказывать результат по набору входных данных — например, распознавать объекты. Обычно используются нейронные сети с множеством уровней. Особенность глубокого обучения в том, что при этом методе используются многослойные сети — это значит, что у них большое количество уровней. На каждом уровне, или слое, находится определенное количество нейронов. Разные слои могут выполнять разные функции [8, c.31].

Например, первый слой получает на вход данные, обрабатывает их определенным образом и отправляет полученный результат на второй слой. Второй слой получит уже не исходные данные, а результат вычислений на первом слое — это обычно набор числовых значений, которые, так или иначе, отражают характеристики входных данных. Затем процесс повторяется с третьим и последующими слоями вплоть до выхода. Чаще всего выходные данные — это вероятность, высчитанная нейронной сетью.

Самостоятельные расчеты по такой модели невозможны, а создавать слои вручную — долгий процесс. В современных сетях могут использоваться десятки слоев и тысячи нейронов, и их будет сложно описать самостоятельно. Поэтому для работы с нейронными сетями и глубоким обучением применяют различные инструменты, позволяющие делать это в несколько строчек. Один из таких инструментов — Keras, который быстро создает, визуализирует и запускает сети с заданными параметрами. Например, вместо того чтобы вручную описывать свойства каждого нейрона, а потом создавать сотню таких элементов в цикле и повторять это для каждого слоя, можно один раз написать инициализацию в Keras. Это несколько строк, которые описывают свойства целевой модели: ее глубину, плотность, механику расчетов и другие параметры. После этого модель можно обучать и собирать [4].

В Keras присутствуют различные наборы данных, которые можно импортировать и загрузить. Например: классификация небольших изображений CIFAR10, определение тональности рецензий IMDB, классификация новостных тем Reuters, рукописные цифры MNIST и так далее [7].

Для загрузки набора MNIST нужны следующие команды:

from keras.datasets import mnist

(X\_train, y\_train), (X\_test, y\_test) = mnist.load\_data()

Keras содержит многочисленные слои и параметры, такие как функции потерь, оптимизаторы, метрики оценки. Они используются для создания, настройки, тренировки и оценки нейронных сетей.

Загрузим эти требуемые слои для построения классификатора цифр:

from keras.models import Sequential

from keras.layers import Dense

from keras.layers import Dropout

from keras.layers import Flatten

from keras.optimizers import Adam

from keras.utils import np\_utils

Keras также поддерживает одномерные и двухмерные сверточные и рекуррентные нейронные сети. Для классификатора воспользуемся сверточной нейронной сетью (слой Conv2D):

from keras.layers.convolutional import Conv2D

from keras.layers.convolutional import MaxPooling2D

В Keras также масса методов для предварительной обработки. Применим метод keras.np\_utils.to\_categorical() для унитарной кодировки y\_train и y\_test. Перед этим изменим форму и нормализуем данные в соответствии с требованиями:

# изменение размерности

X\_train = X\_train.reshape(X\_train.shape[0], X\_train.shape[1], X\_train.shape[2], 1).astype('float32')

X\_test = X\_test.reshape(X\_test.shape[0], X\_test.shape[1], X\_test.shape[2], 1).astype('float32')

# нормализация, чтобы получить данные в диапазоне 0-1

X\_train/=255

X\_test/=255

number\_of\_classes = 10

y\_train = np\_utils.to\_categorical(y\_train, number\_of\_classes)

y\_test = np\_utils.to\_categorical(y\_test, number\_of\_classes)

Для добавления импортированных слоев используется метод add() с дополнительными параметрами:

model = Sequential()

model.add(Conv2D(32, (5, 5), input\_shape=(X\_train.shape[1], X\_train.shape[2], 1), activation='relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Conv2D(32, (3, 3), activation='relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Dropout(0.5))

model.add(Flatten())

model.add(Dense(128, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(number\_of\_classes, activation='softmax')

Перед тренировкой нужно настроить процесс обучения, что делается с помощью метода compile().

model.compile(loss='categorical\_crossentropy',

optimizer=Adam(), metrics=['accuracy'])

Тренировка моделей Keras происходит с помощью массивов и метода fit():

model.fit(X\_train, y\_train, validation\_data=(X\_test, y\_test), epochs=5, batch\_size=200)

Тренировка занимает некоторое время. В данном случае прошло всего 5 epocs, но их число можно увеличить.

Тренировка выглядит вот так:

*Epoch 1/5*

*300/300 [==============================] - 60s 182ms/step - loss: 0.4877 - accuracy: 0.8436 - val\_loss: 0.0809 - val\_accuracy: 0.9747*

*Epoch 2/5*

*300/300 [==============================] - 56s 188ms/step - loss: 0.1596 - accuracy: 0.9528 - val\_loss: 0.0539 - val\_accuracy: 0.9830*

*Epoch 3/5*

*300/300 [==============================] - 55s 183ms/step - loss: 0.1187 - accuracy: 0.9632 - val\_loss: 0.0427 - val\_accuracy: 0.9864*

*Epoch 4/5*

*300/300 [==============================] - 53s 177ms/step - loss: 0.0996 - accuracy: 0.9698 - val\_loss: 0.0360 - val\_accuracy: 0.9877*

*Epoch 5/5*

*300/300 [==============================] - 52s 173ms/step - loss: 0.0901 - accuracy: 0.9732 - val\_loss: 0.0320 - val\_accuracy: 0.9893*

После тренировки модели результаты проверяются с помощью метода evaluate():

model.evaluate(X\_test, y\_test)

Вывод результата:

*313/313 [==============================] - 4s 11ms/step - loss: 0.0320 - accuracy: 0.9893*

Согласно результату, точность распознавания на данной модели равна 98,93%.

TensorFlow – это открытая система глубокого обучения, разработанная компанией Google. Она предназначена для создания и обучения нейронных сетей, которые могут выполнять различные задачи машинного обучения, такие как классификация, распознавание образов, прогнозирование и т.д. [10, c.46].

Возможности TensorFlow:

* Создание и обучение различных типов нейронных сетей, таких как сверточные, рекуррентные и т.д.
* Работу с большими наборами данных, включая поддержку параллельного обучения на нескольких устройствах.
* Использование графической обработки для ускорения вычислений.
* Интеграцию с другими инструментами машинного обучения и фреймворками.
* Создание и развертывание моделей на различных устройствах, включая мобильные устройства и сервера.

TensorFlow также предоставляет различные инструменты и ресурсы для обучения и развертывания моделей, включая библиотеки для работы с изображениями, звуком и текстом, а также готовые модели для выполнения различных задач. Одним из ключевых преимуществ TensorFlow является его открытость и активное сообщество разработчиков, которые постоянно работают над улучшением системы и созданием новых инструментов для работы с ней.

TensorFlow предоставляет широкий спектр инструментов и возможностей, которые помогают в разработке систем глубокого обучения:

* Создание графа вычислений: TensorFlow позволяет разработчикам создавать граф вычислений, который представляет нейронную сеть в виде набора математических операций, выполняемых над данными. Это позволяет управлять процессом обучения и оптимизировать вычисления для получения наилучших результатов.
* Оптимизация производительности: TensorFlow использует оптимизацию производительности, такую как вычисления на GPU и TPU, чтобы ускорить обучение нейронных сетей и уменьшить время, необходимое для обучения модели.
* Богатый выбор модулей и библиотек: TensorFlow содержит множество модулей и библиотек, которые упрощают процесс разработки систем глубокого обучения. Они включают в себя библиотеки для работы с изображениями, звуком, текстом и другими типами данных, а также инструменты для визуализации данных и результатов.
* Поддержка развертывания: TensorFlow позволяет разработчикам развертывать свои модели на различных устройствах, включая мобильные устройства и серверы, что делает его удобным инструментом для создания реальных продуктов на основе глубокого обучения.
* Активное сообщество разработчиков: TensorFlow имеет широкое сообщество разработчиков, которые создают новые инструменты, расширения и документацию для улучшения системы. Это позволяет разработчикам быстро получать помощь и решения проблем, а также обмениваться знаниями и опытом.

В целом, TensorFlow является мощным инструментом для разработки систем глубокого обучения, который упрощает процесс создания и оптимизации нейронных сетей, а также облегчает их развертывание в реальных продуктах. TensorFlow является одной из наиболее распространенных систем глубокого обучения, но существуют и другие системы, такие как PyTorch, Caffe, Theano и другие [5].

В данной работе TensorFlow используется для того, чтобы получить доступ к Keras. Установить TensorFlow можно через командную строку (cmd.exe) введя следующую команду:

pip3.11 install tensorflow, где:

1. pip – это менеджер пакетов Python;
2. 3.11 – версия Python (необязательный аргумент);
3. install – команда установки нового пакета;
4. tensorflow – название пакета которое требуется установить.

## 3. МЕТОД ГЛАВНЫХ КОМПОНЕНТ

Метод главных компонент — это технология многомерного статистического анализа, используемая для сокращения размерности пространства признаков с минимальной потерей полезной информации. Предложен K. Пирсоном в 1901 г., а затем детально разработан американским экономистом и статистиком Г. Хоттелингом.

С математической точки зрения метод главных компонент представляет собой ортогональное линейное преобразование, которое отображает данные из исходного пространства признаков в новое пространство меньшей размерности. При этом первая ось новой системы координат строится таким образом, чтобы дисперсия данных вдоль неё была бы максимальна. Вторая ось строится ортогонально первой так, чтобы дисперсия данных вдоль неё, была бы максимальной их оставшихся возможных и т.д. Первая ось называется первой главной компонентой, вторая — второй и т.д.

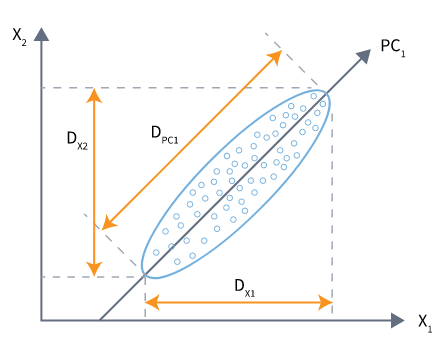


Рисунок 1 – Снижение размерности пространства

На рисунке 1 показано снижение размерности исходного 2-мерного пространства (X1, X2) с помощью метода главных компонент до 1-мерного. Первая главная компонента PC1 ориентирована вдоль направления наибольшей вытянутости эллипсоида рассеяния точек объектов исходного набора данных в пространстве признаков, т.е. с ней связана наибольшая дисперсия. На рисунке, также, несложно увидеть, что проекция дисперсии данных на ось первой главной компоненты DPC1, больше, чем её проекции на исходные оси DX1 и DX2, но меньше их суммы. Т.е. с помощью первой главной компоненты выразить всю дисперсию данных не удалось. Поэтому строят вторую, третью и т.д. главные компоненты, пока они суммарно не отразят всю дисперсию.

Таким образом, смысл метода заключается в том, что с каждой главной компонентой связана определённая доля общей дисперсии исходного набора данных (её называют нагрузкой). В свою очередь, дисперсия, являющаяся мерой изменчивости данных, может отражать уровень их информативности.

Действительно, вдоль некоторых осей исходного пространства признаков изменчивость может быть большой, вдоль других — малой, а вдоль третьих вообще отсутствовать. Предполагается, что чем меньше дисперсия данных вдоль оси, тем менее значим вклад переменной, связанной с данной осью и, следовательно, исключив эту ось из пространства (т.е. переменную из модели), можно уменьшить размерность задачи, почти не проиграв в информативности данных.

Следовательно, задача метода главных компонент заключается в том, чтобы построить новое пространство признаков меньшей размерности, дисперсия между осями которой будет перераспределена так, чтобы максимизировать дисперсию по каждой из них. Для этого выполняется последовательность следующих действий:

1. Вычисляется общая дисперсия исходного пространства признаков. Это нельзя сделать простым суммированием дисперсий по каждой переменной, поскольку они, в большинстве случаев, не являются независимыми. Поэтому суммировать нужно взаимные дисперсии переменных, которые определяются из ковариационной матрицы.
2. Вычисляются собственные векторы и собственные значения ковариационной матрицы, определяющие направления главных компонент и величину связанной с ними дисперсии.
3. Производится снижение размерности. Диагональные элементы ковариационной матрицы показывают дисперсию по исходной системе координат, а её собственные значения — по новой. Тогда разделив дисперсию, связанную с каждой главной компонентой на сумму дисперсий по всем компонентам, получаем долю дисперсии, связанную с каждой компонентой. После этого отбрасывается столько главных компонент, чтобы доля оставшихся составляла 80-90%.

Следует отметить, что директивный подход к выбору числа компонент не всегда даёт хорошие результаты. Это связано с тем, что часть дисперсии данных может быть обусловлена шумами, а не информативностью компонент. Тогда, задав порог, скажем, 80% может оказаться, что в них только 60% дисперсии связаны с информативностью, а 20% с шумом. Поэтому на практике часто используют различные специальные критерии для определения числа компонент, такие как критерий Кайзера, критерий сломанной трости и т.д.

Основными ограничениями метода главных компонент являются:

* невозможность смысловой интерпретации компонент, поскольку они «вбирают» в себя дисперсию от нескольких исходных переменных;
* метод может работать только с непрерывными данными.

Метод главных компонент включается в состав большинства аналитических платформ и широко используется для снижения размерности входных данных на этапе их предобработки. Метод иногда рассматривают как часть более общего подхода к снижению размерности данных — факторного анализа. В аналитических платформах в модулях факторного анализа часто практически реализован именно метод главных компонент [6].

Рассмотрим применение данного метода на датасетах MNIST и CIFAR-10.

# ВЛИЯНИЕ МЕТОДА ГЛАВНЫХ КОМПОНЕНТ НА ТОЧНОСТЬ РАСПОЗНАВАНИЯ MNIST И CIFAR-10

MNIST и CIFAR-10 являются широко известными наборами данных для задач распознавания изображений. MNIST содержит изображения рукописных цифр, а CIFAR-10 - изображения объектов различных классов.

Воспользуемся методом главных компонент на данных датасетах чтобы уменьшить размерность и проверим, как изменится точность и скорость распознавания по сравнению с исходной размерностью. Напишем программу на Python 3.11.

Для начала импортируем необходимые библиотеки:

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

from keras.layers import Dropout

from sklearn.decomposition import PCA

from keras.utils import np\_utils

TensorFlow понадобится чтобы воспользоваться Keras, а Keras в свою очередь – для работы с нейросетями, PCA – для применения метода главных компонент, np\_utils – для того чтобы воспользоваться функцией to\_categorical, которая преобразует вектор классов в бинарную матрицу классов. Dropout потребуется, чтобы избежать переобучения в модели нейросети.

Выполним загрузку датасета СIFAR-10:

(x\_train,y\_train),(x\_test,y\_test) = keras.datasets.cifar10.load\_data()

В переменных x\_train, x\_test, y\_train и y\_test присваиваются данные из датасета CIFAR-10. Сам датасет состоит из 60 000 изображений размером 32x32 пикселя, разделенных на 10 классов, по 6 000 изображений в каждом классе. Обучающие данные содержат 75% строк набора данных, а тестовые данные содержат оставшиеся 25%.В переменных x\_train и x\_test будут находиться изображения, а в y\_train и y\_test – метки классов, соответствующие каждому изображению. Обучающие данные будут использоваться для обучения модели, а тестовые данные – для проверки ее точности.

Выполним нормализацию данных в переменных x\_train и x\_test:

x\_train = x\_train.astype("float32") / 255.0

x\_test = x\_test.astype("float32") / 255.0

Каждое значение пикселя в изображении представлено в виде целого числа от 0 до 255, где 0 - это черный цвет, а 255 - белый цвет. Нормализация данных заключается в преобразовании каждого значения пикселя в вещественное число в диапазоне от 0 до 1, делением каждого значения на 255. Это упрощает обучение модели и улучшает ее производительность.

Выполним преобразование изображений в векторы:

x\_train = x\_train.reshape((len(x\_train), -1))

x\_test = x\_test.reshape((len(x\_test), -1))

Эти строки кода используются для изменения формы массивов x\_train и x\_test. Каждое изображение в датасете CIFAR-10 представляет собой трехмерный массив, где первое измерение соответствует количеству изображений, а два других измерения соответствуют высоте и ширине изображения. Однако для обучения модели нейронной сети требуется двумерный массив, где каждое изображение представлено в виде одномерного массива. Поэтому эти строки кода используются для изменения формы массивов x\_train и x\_test из трехмерного массива в двумерный массив, где каждое изображение представлено в виде одномерного массива.

Применим метод главных компонент:

pca = PCA(n\_components=n)

x\_train = pca.fit\_transform(x\_train)

x\_test = pca.transform(x\_test)

PCA используется для уменьшения размерности изображений в наборе данных CIFAR-10. После применения PCA каждое изображение будет представлено в виде одномерного массива с n признаками. fit\_transform() и transform() - это методы из библиотеки scikit-learn, которые используются для предобработки данных перед обучением модели. Метод fit\_transform() используется для вычисления параметров преобразования данных и применения их к обучающим данным, а метод transform() используется для применения тех же параметров преобразования к тестовым данным. Таким образом, метод fit\_transform() используется только на обучающих данных, а метод transform() используется на тестовых данных после обучения модели на обучающих данных. Это позволяет применять те же параметры преобразования к тестовым данным, что и к обучающим данным, что улучшает качество модели.

Преобразуем метки классов:

num\_classes = 10

y\_train = np\_utils.to\_categorical(y\_train, num\_classes)

y\_test = np\_utils.to\_categorical(y\_test, num\_classes)

Код использует функцию to\_categorical() из библиотеки np\_utils в Keras, чтобы преобразовать метки классов в бинарные матрицы в формате one-hot encoding. Это необходимо для обучения нейронной сети с категориальной кросс-энтропией в качестве функции потерь. Число столбцов в бинарной матрице соответствует числу классов в данных.

Создадим модель нейросети:

model = keras.Sequential(

[

layers.Dense(x1, activation="relu", input\_shape=(n,)),

layers.Dense(x2, activation="sigmoid"),

Dropout(0.2),

layers.Dense(x3, activation="tanh"),

layers.Dense(10, activation="softmax"),

]

)

Данная модель состоит из последовательности слоев, каждый из которых определяется с помощью функции Dense. Dense - это слой нейронной сети, который полностью связывает каждый нейрон в предыдущем слое с каждым нейроном в текущем слое. Каждый нейрон в Dense слое получает входные данные от всех нейронов в предыдущем слое и вычисляет свой выход на основе взвешенной суммы входных данных и соответствующих весов. В Keras, Dense слой создается с помощью функции Dense(), которая принимает на вход количество нейронов в слое и функцию активации. Первый слой имеет x1 нейронов, второй – x2 нейронов, третий – x3 нейронов, а последний слой имеет 10 нейронов и использует функцию активации softmax. Слой Dropout(0.2) добавлен между вторым и третьим слоями, чтобы случайным образом исключать 20% нейронов во время обучения и предотвращать переобучение. Функция активации relu используется в первом слое, во втором слое используется функция sigmoid, в третьем tanh. Функция активации ReLu представляет собой простую математическую функцию, которая возвращает входное значение, если оно положительное, в противном случае вернет ноль. Функция активации sigmoid используется для ограничения выхода нейрона в диапазоне от 0 до 1. Она часто используется в задачах бинарной классификации, где необходимо определить, принадлежит ли объект к определенному классу или нет. Функция активации tanh (гиперболический тангенс) возвращает значение, которое находится между -1 и 1. Она подходит для работы с данными, которые содержат как положительные, так и отрицательные значения. Функция tanh позволяет учесть взаимодействие между разными слоями нейронной сети и помогает сгладить данные, что улучшает качество обучения. Последний слой нейронной сети состоит из 10 нейронов и использует функцию активации softmax. Функция softmax преобразует выходные данные нейронов в вероятности, сумма которых равна 1. В итоге данная модель будет применена для решения задачи классификации на 10 классов.

Произведем компиляцию модели:

model.compile(optimizer="adam", loss=tf.keras.losses.CategoricalCrossentropy(from\_logits=False), metrics=["accuracy"])

Здесь используется функция активации "adam" в качестве оптимизатора. Оптимизаторы используются для обновления весов в процессе обучения нейронной сети. В данной работе будут использованы несколько оптимизаторов помимо Adam: Adagrad, SGD и Nadam. В качестве функции потерь используется CategoricalCrossentropy, которая используется для многоклассовой классификации. В данном случае, параметр from\_logits установлен на False, что означает, что выходные значения модели не являются нормализованными. В качестве метрики используется "accuracy", которая является мерой точности модели в предсказании правильного класса.

И наконец, выполним обучение модели:

model.fit(x\_train, y\_train, epochs=y1, batch\_size=y2, validation\_split=0.1)

Модель обучается на x\_train и y\_train в течение y1 эпох с размером пакета y2 и валидационным разделением 0.1. Аргументы метода fit() включают x - входные данные, y - целевые данные, batch\_size - размер пакета, epochs - количество эпох, validation\_split - доля данных, которые будут использоваться для валидации. В данном случае, x\_train и y\_train используются в качестве входных и целевых данных соответственно, размер пакета равен y2, обучение происходит в течение y1 эпох, а 10% данных используется для валидации.

Оценим модель на тестовых данных:

test\_loss, test\_acc = model.evaluate(x\_test, y\_test, verbose=2)

Функция evaluate() в Keras применяется, чтобы оценить производительность обученной модели на тестовом наборе данных. Она возвращает значение потерь и метрики для модели на тестовом наборе данных. Параметр verbose управляет подробностью вывода оценки. Значение 0 означает отсутствие вывода, 1 - вывод прогресс-бара, а 2 - вывод одной строки на каждую эпоху.

Выведем эти значения на экран:

print('Test loss:', test\_loss)

print("Test accuracy:", test\_acc)

Где переменная test\_loss – значение потерь, а test\_acc - значение точности.

Программа написана полностью. Она также подходит и для датасета MNIST, нужно только загрузить данные датасета MNIST вместо CIFAR-10:

(x\_train, y\_train), (x\_test, y\_test) = keras.datasets.mnist.load\_data()

Теперь, меняя значения переменных n, x1, x2, x3, y1, y2, а также используя различные оптимизаторы, определим, как метод главных компонент влияет на точность распознавания.

## СРАВНЕНИЕ РЕЗУЛЬТАТОВ

Для сравнения сначала выясним, какова точность распознавания в датасетах в исходном виде, без применения метода главных компонент. Переделаем программу, которую рассматривали в предыдущей главе, а точнее построим другую модель нейросети. Начнем с CIFAR-10:

model = keras.Sequential(

[

layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(32, 32, 3)),

layers.MaxPooling2D(pool\_size=(2, 2)),

layers.Dropout(0.25),

layers.Flatten(),

layers.Dense(x, activation='relu'),

layers.Dropout(0.5),

layers.Dense(num\_classes, activation='softmax')

]

)

Опишем каждый слой данной модели:

1. Conv2D - слой свертки с 32 фильтрами размера 3x3 с функцией активации ReLU. Он принимает на вход изображения размера 32x32x3, то есть 32 пикселя в ширину, 32 пикселя в высоту и 3 цветовых канала (RGB).
2. MaxPooling2D - слой пулинга с окном размера 2x2. Он сжимает каждый пул в 2 раза, что помогает уменьшить размерность входных данных.
3. Dropout - исключает каждый нейрон из обучения с заданной вероятностью (в данном случае 0,25).
4. Flatten - слой, который преобразует признаки из двумерного формата в одномерный для последующего использования в полносвязных слоях.
5. Dense - полносвязный слой с x нейронами и функцией активации ReLU.
6. Dropout - еще один слой Dropout для уменьшения переобучения с такой же вероятностью исключения как и ранее.
7. Dense - последний выходной слой с количеством нейронов равным числу классов и функцией активации Softmax для предсказания вероятностей для каждого класса.

Conv2D - как уже упоминалось, это слой свертки. В задаче компьютерного зрения изображение представляется в виде трехмерного массива (высота, ширина, количество цветовых каналов), где каждый элемент это пиксель изображения. Сверточный слой применяет фильтры (также называемые ядрами или фильтрами свертки) ко всем возможным областям входного изображения. Фильтр это матрица размера фиксированного размера (обычно 3x3 или 5x5), которая перемещается по всему входному изображению и вычисляет сверточное произведение между элементами матрицы фильтра и соответствующими пикселями на изображении. По результатам свертки формируется карта признаков (feature map).

MaxPooling2D - это слой пулинга. Его цель - уменьшение размерности карта признаков. После свертки изображения можно заметить, что некоторые пиксели имеют большое значение, тогда как другие - маленькое или даже нулевое значание. MaxPooling2D работает следующим образом: изображение разбивается на квадратики заданного размера (обычно 2x2), и для каждого квадратика выбирается значение максимального пикселя. Полученные значения объединяются в новый массив, который и будет выходом из слоя пулинга. Таким образом, размерность карта признаков уменьшается примерно в два раза с каждым слоем MaxPooling2D.

Flatten - слой, который преобразует двумерный массив (карта признаков) в одномерный вектор. Этот слой не выполняет никаких вычислений - он просто объединяет значения каждого элемента карта признаков в аккуратный одномерный массив. Обычно этот слой ставится перед полносвязным слоем, который будет обрабатывать вектор признаков (после того, когда изображение было проанализировано наличием структур в нём).

Запустим программу с различными значениями x, y1, y2 и различными оптимизаторами:

Таблица 1

Зависимость точности распознавания CIFAR-10 от различных параметров

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| x | y1 | y2 | общее кол-во параметров | Точность | Оптимизатор |
| 270 | 5 | 136 | 1947876 | 60,72% | adam |
| 427 | 18 | 115 | 3080003 | 68,25% | adam |
| 825 | 7 | 202 | 5949981 | 63,85% | adam |
| 488 | 17 | 65 | 3519874 | 43,98% | adagrad |
| 870 | 10 | 228 | 6274476 | 35,57% | adagrad |
| 389 | 6 | 50 | 2805985 | 39,16% | adagrad |
| 407 | 12 | 195 | 2935783 | 67,26% | nadam |
| 785 | 15 | 249 | 5661541 | 68,10% | nadam |
| 391 | 14 | 197 | 2820407 | 66,49% | nadam |
| 402 | 18 | 98 | 2899728 | 56,51% | SGD |
| 214 | 9 | 33 | 1544060 | 59% | SGD |
| 107 | 18 | 102 | 772483 | 55,78% | SGD |

Точность распознавания колеблется в пределах 35-68%. Наихудший результат показывает модель с оптимизатором Adagrad. Наилучший результат показывают оптимизаторы Adam и Nadam. Для окончательного вывода, рассмотрим, как обстоят дела с MNIST.

Напишем модель для датасета MNIST:

model = keras.Sequential([

Flatten(input\_shape=(28, 28, 1)),

Dense(x, activation='relu'),

Dense(10, activation='softmax')

])

Данная модель состоит из трех слоёв:

1. Первый слой преобразует изображение размером 28x28 пикселей в вектор длины 784;
2. Второй слой содержит x нейронов с функцией активации ReLU;
3. Последний выходной слой с функцией активации Softmax, который содержит 10 нейронов, соответствующих 10 классам (цифрам от 0 до 9)

Запустим программу с различными значениями x, y1, y2:

Таблица 2

Зависимость точности распознавания MNIST от различных параметров

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| x | y1 | y2 | общее кол-во параметров | Точность | Оптимизатор |
| 383 | 20 | 68 | 304495 | 97,90% | adam |
| 754 | 9 | 121 | 599440 | 97,97% | adam |
| 922 | 17 | 199 | 733000 | 97,83% | adam |
| 562 | 15 | 61 | 446800 | 91,97% | adagrad |
| 857 | 6 | 125 | 681325 | 88,79% | adagrad |
| 165 | 17 | 179 | 131185 | 89,71% | adagrad |
| 905 | 9 | 192 | 719485 | 98,16% | nadam |
| 852 | 12 | 100 | 677350 | 98,15% | nadam |
| 708 | 20 | 221 | 562870 | 98,07% | nadam |
| 636 | 15 | 78 | 505630 | 94,62% | SGD |
| 919 | 12 | 147 | 730615 | 92,80% | SGD |
| 422 | 10 | 246 | 335500 | 91,29% | SGD |

И вновь наихудший результат показал оптимизатор Adagrad. Вероятнее всего, причина в том, что эти наборы данных содержат сложные изображения, что приводит к повышенной переменной обучаемости, что затрудняет сходимость модели (модель не может правильно обучаться) и может приводить к недообучению или переобучению модели. Adagrad также может иметь проблемы с вычислительными затратами и неравномерностью векторов градиента. Оптимизаторы Adam и Nadam показывают наилучшие результаты. Adam показывает хорошие результаты на функциях, для которых градиенты имеют разный масштаб и/или неоднородно распределены. Nadam - это оптимизатор глубоких нейронных сетей, который комбинирует метод Nesterov Momentum и метод Adam. Он позволяет быстрее находить минимум функции ошибки и ускоряет обучение модели. Nadam также уменьшает шум в градиентах и улучшает общую точность модели. SGD (Stochastic Gradient Descent) - это простой и часто используемый метод оптимизации для машинного обучения. Он используется, чтобы минимизировать функцию ошибки при обучении модели, путём изменения весовых коэффициентов. В данном случае выдает средние результаты между Adam, Nadam и Adagrad.

Теперь проверим, какие результаты получаются в датасетах MNIST и CIFAR-10 с применением метода главных компонент:

Таблица 3

Зависимость точности распознавания CIFAR-10 от различных параметров

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| n | x1 | x2 | x3 | y1 | y2 | общее кол-во параметров | Точность | Оптимизатор |
| 128 | 968 | 319 | 881 | 12 | 251 | 724723 | 55,95% | nadam |
| 128 | 574 | 958 | 478 | 16 | 201 | 1088088 | 55,70% | adam |
| 64 | 728 | 155 | 263 | 13 | 70 | 203983 | 55,65% | nadam |
| 64 | 953 | 721 | 428 | 10 | 254 | 1063085 | 55,62% | adam |
| 128 | 897 | 939 | 476 | 18 | 153 | 1411145 | 55,59% | adam |
| 64 | 563 | 164 | 256 | 18 | 245 | 173901 | 55,49% | nadam |
| 256 | 479 | 250 | 901 | 5 | 242 | 478274 | 54,84% | adam |
| 256 | 717 | 123 | 213 | 14 | 186 | 301135 | 54,79% | nadam |
| 128 | 136 | 922 | 669 | 7 | 41 | 768045 | 54,74% | nadam |
| 256 | 980 | 661 | 960 | 16 | 230 | 1545431 | 54,55% | nadam |
| 512 | 787 | 316 | 384 | 17 | 41 | 778317 | 54,15% | nadam |
| 64 | 130 | 773 | 510 | 16 | 161 | 509563 | 53,93% | adam |
| 32 | 438 | 850 | 388 | 14 | 130 | 721682 | 53,90% | adam |
| 512 | 879 | 780 | 891 | 14 | 169 | 1842118 | 53,45% | nadam |
| 512 | 601 | 999 | 354 | 16 | 233 | 1267261 | 53,15% | adam |

Продолжение таблицы 3

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| n | x1 | x2 | x3 | y1 | y2 | общее кол-во параметров | Точность | Оптимизатор |
| 32 | 500 | 496 | 694 | 13 | 125 | 616864 | 52,90% | nadam |
| 32 | 958 | 880 | 848 | 8 | 140 | 1631112 | 52,76% | adam |
| 256 | 101 | 143 | 593 | 11 | 225 | 131875 | 52,70% | adam |
| 1024 | 687 | 821 | 303 | 20 | 93 | 1521129 | 52,30% | adam |
| 32 | 807 | 570 | 731 | 16 | 56 | 911912 | 52,25% | nadam |
| 512 | 475 | 384 | 689 | 16 | 225 | 698624 | 52,13% | adam |
| 1024 | 522 | 647 | 980 | 17 | 42 | 1518281 | 51,89% | nadam |
| 1024 | 334 | 982 | 974 | 16 | 39 | 1638512 | 51,67% | nadam |
| 1024 | 591 | 703 | 274 | 19 | 232 | 1217597 | 50,85% | adam |
| 16 | 284 | 454 | 176 | 16 | 142 | 216068 | 46,38% | nadam |
| 64 | 294 | 435 | 298 | 17 | 158 | 280353 | 42,50% | SGD |
| 512 | 858 | 569 | 902 | 15 | 154 | 1452095 | 42,50% | SGD |
| 1024 | 630 | 940 | 360 | 14 | 152 | 1581260 | 40,82% | SGD |
| 256 | 809 | 667 | 604 | 18 | 243 | 1157705 | 40,67% | SGD |
| 1024 | 321 | 805 | 571 | 18 | 169 | 1054181 | 40,37% | SGD |
| 32 | 471 | 870 | 952 | 15 | 222 | 1264905 | 40,00% | SGD |
| 128 | 291 | 671 | 303 | 8 | 110 | 440127 | 39,18% | SGD |
| 16 | 356 | 980 | 580 | 17 | 115 | 930702 | 38,75% | SGD |
| 512 | 929 | 582 | 141 | 17 | 115 | 1101460 | 37,96% | adagrad |
| 1024 | 797 | 710 | 375 | 10 | 52 | 1653890 | 37,84% | adagrad |
| 16 | 256 | 339 | 890 | 16 | 49 | 402985 | 36,83% | adagrad |
| 8 | 243 | 703 | 611 | 14 | 52 | 609983 | 36,70% | adam |
| 8 | 972 | 314 | 112 | 5 | 225 | 350680 | 36,63% | nadam |
| 8 | 290 | 537 | 897 | 16 | 54 | 650443 | 36,44% | nadam |
| 16 | 406 | 110 | 300 | 7 | 105 | 87982 | 36,00% | SGD |
| 8 | 156 | 269 | 551 | 8 | 140 | 197927 | 35,88% | adam |
| 128 | 159 | 809 | 183 | 5 | 106 | 300021 | 35,74% | SGD |
| 512 | 180 | 570 | 195 | 18 | 124 | 308815 | 34,40% | adagrad |
| 8 | 839 | 601 | 704 | 18 | 94 | 943249 | 34,35% | SGD |
| 256 | 601 | 830 | 951 | 12 | 165 | 1453918 | 34,27% | adagrad |
| 32 | 965 | 775 | 443 | 8 | 159 | 1128703 | 34,20% | SGD |
| 256 | 824 | 919 | 416 | 5 | 71 | 1356833 | 34,18% | adagrad |
| 8 | 102 | 182 | 640 | 17 | 134 | 143194 | 33,71% | SGD |
| 128 | 692 | 430 | 782 | 13 | 228 | 732130 | 33,65% | adagrad |
| 128 | 103 | 633 | 376 | 17 | 203 | 321273 | 32,87% | adagrad |
| 32 | 527 | 115 | 800 | 11 | 141 | 178921 | 32,68% | adagrad |
| 8 | 356 | 487 | 386 | 16 | 90 | 369301 | 32,18% | adagrad |
| 64 | 184 | 925 | 863 | 10 | 236 | 990863 | 32,11% | adagrad |
| 64 | 492 | 180 | 340 | 6 | 179 | 185670 | 31,25% | adagrad |
| 8 | 113 | 871 | 481 | 11 | 120 | 524563 | 30,78% | adagrad |
| 32 | 709 | 159 | 725 | 8 | 206 | 259547 | 30,23% | adagrad |
| 1024 | 493 | 643 | 609 | 6 | 98 | 1221263 | 29,63% | adagrad |

Продолжение таблицы 3

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| n | x1 | x2 | x3 | y1 | y2 | общее кол-во параметров | Точность | Оптимизатор |
| 4 | 608 | 703 | 113 | 16 | 152 | 511859 | 26,33% | adagrad |
| 4 | 530 | 895 | 271 | 17 | 93 | 723431 | 28,27% | nadam |
| 4 | 786 | 642 | 603 | 14 | 55 | 902953 | 28,23% | adam |
| 4 | 566 | 962 | 103 | 14 | 48 | 648513 | 27,94% | adam |
| 16 | 727 | 900 | 122 | 5 | 251 | 778711 | 27,94% | adagrad |
| 4 | 721 | 837 | 730 | 5 | 107 | 1226969 | 27,03% | nadam |
| 4 | 278 | 951 | 393 | 16 | 200 | 644795 | 26,82% | SGD |
| 4 | 193 | 300 | 943 | 11 | 52 | 352448 | 26,77% | adagrad |
| 4 | 760 | 626 | 289 | 5 | 113 | 664289 | 26,51% | SGD |

Таблица 4

Зависимость точности распознавания MNIST от различных параметров

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| n | x1 | x2 | x3 | y1 | y2 | общее кол-во параметров | Точность | Оптимизатор |
| 49 | 885 | 172 | 667 | 16 | 112 | 318713 | 98,34% | adam |
| 24 | 600 | 142 | 724 | 16 | 106 | 211124 | 98,12% | nadam |
| 49 | 937 | 411 | 774 | 18 | 230 | 759006 | 98,11% | nadam |
| 24 | 918 | 292 | 297 | 19 | 41 | 381299 | 98,10% | adam |
| 49 | 808 | 725 | 468 | 10 | 207 | 971383 | 98,06% | nadam |
| 49 | 853 | 716 | 952 | 10 | 107 | 1346228 | 98,01% | adam |
| 98 | 675 | 576 | 825 | 8 | 199 | 940486 | 97,96% | nadam |
| 196 | 270 | 709 | 411 | 20 | 166 | 541259 | 97,92% | nadam |
| 98 | 494 | 662 | 911 | 8 | 181 | 989709 | 97,82% | adam |
| 98 | 983 | 534 | 724 | 13 | 156 | 1017363 | 97,81% | nadam |
| 196 | 340 | 177 | 283 | 18 | 223 | 180551 | 97,73% | adam |
| 196 | 995 | 155 | 596 | 7 | 34 | 449341 | 97,68% | nadam |
| 196 | 545 | 262 | 750 | 9 | 166 | 455177 | 97,65% | adam |
| 98 | 280 | 949 | 691 | 9 | 98 | 957759 | 97,65% | adam |
| 24 | 272 | 248 | 115 | 8 | 115 | 104299 | 97,53% | adam |
| 24 | 370 | 655 | 604 | 7 | 103 | 654529 | 97,39% | nadam |
| 392 | 562 | 624 | 858 | 12 | 125 | 1117018 | 97,36% | adam |
| 392 | 393 | 107 | 154 | 8 | 59 | 214789 | 97,35% | adam |
| 392 | 131 | 118 | 521 | 15 | 89 | 134278 | 97,22% | nadam |
| 392 | 113 | 906 | 700 | 16 | 255 | 789603 | 96,99% | nadam |
| 12 | 879 | 446 | 220 | 11 | 168 | 504457 | 95,79% | adam |
| 12 | 928 | 795 | 539 | 18 | 120 | 1185063 | 95,64% | nadam |
| 392 | 504 | 672 | 171 | 13 | 45 | 654235 | 95,32% | SGD |
| 196 | 753 | 266 | 946 | 13 | 53 | 610957 | 95,29% | SGD |
| 24 | 669 | 545 | 194 | 15 | 60 | 489749 | 94,82% | SGD |
| 12 | 640 | 752 | 597 | 6 | 36 | 945873 | 94,78% | adam |

Продолжение таблицы 4

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| n | x1 | x2 | x3 | y1 | y2 | общее кол-во параметров | Точность | Оптимизатор |
| 196 | 994 | 890 | 808 | 19 | 114 | 1809386 | 94,47% | SGD |
| 49 | 512 | 181 | 553 | 13 | 76 | 224639 | 94,35% | SGD |
| 392 | 747 | 574 | 116 | 19 | 102 | 790793 | 94,23% | SGD |
| 12 | 247 | 257 | 857 | 5 | 200 | 296633 | 94,17% | nadam |
| 98 | 617 | 558 | 695 | 20 | 137 | 801392 | 93,84% | SGD |
| 24 | 696 | 590 | 876 | 16 | 91 | 955116 | 93,57% | SGD |
| 98 | 934 | 268 | 845 | 11 | 93 | 578811 | 93,31% | SGD |
| 98 | 980 | 583 | 783 | 20 | 110 | 1134055 | 90,49% | adagrad |
| 49 | 792 | 760 | 495 | 14 | 95 | 1023935 | 89,48% | adagrad |
| 12 | 492 | 381 | 348 | 11 | 100 | 330655 | 88,94% | SGD |
| 49 | 922 | 706 | 811 | 20 | 182 | 1279235 | 88,33% | adagrad |
| 49 | 493 | 965 | 868 | 5 | 180 | 1348538 | 87,68% | SGD |
| 196 | 427 | 249 | 395 | 19 | 173 | 293401 | 86,88% | adagrad |
| 24 | 684 | 467 | 434 | 12 | 140 | 544457 | 85,66% | adagrad |
| 392 | 761 | 489 | 694 | 14 | 158 | 1018701 | 85,17% | adagrad |
| 6 | 737 | 497 | 829 | 10 | 169 | 793087 | 84,49% | nadam |
| 6 | 399 | 805 | 397 | 16 | 209 | 648755 | 84,38% | nadam |
| 6 | 111 | 994 | 953 | 14 | 55 | 1069880 | 84,17% | adam |
| 24 | 629 | 905 | 953 | 11 | 179 | 1458833 | 83,82% | adagrad |
| 12 | 232 | 659 | 229 | 13 | 81 | 310003 | 83,18% | adagrad |
| 6 | 127 | 759 | 341 | 5 | 140 | 360621 | 82,57% | adam |
| 12 | 198 | 850 | 656 | 6 | 185 | 736550 | 82,52% | SGD |
| 6 | 860 | 441 | 740 | 19 | 54 | 720211 | 82,08% | SGD |
| 98 | 321 | 824 | 670 | 13 | 237 | 856567 | 81,79% | adagrad |
| 392 | 505 | 436 | 402 | 6 | 110 | 598785 | 79,80% | adagrad |
| 12 | 741 | 764 | 100 | 12 | 205 | 654031 | 79,28% | adagrad |
| 6 | 204 | 886 | 337 | 12 | 249 | 485357 | 75,23% | SGD |
| 196 | 625 | 759 | 495 | 7 | 243 | 979419 | 74,59% | adagrad |
| 6 | 962 | 765 | 833 | 12 | 81 | 1389847 | 74,57% | adagrad |
| 6 | 603 | 342 | 119 | 9 | 51 | 252806 | 74,01% | adagrad |
| 3 | 634 | 866 | 384 | 8 | 222 | 889224 | 53,79% | adam |
| 3 | 141 | 895 | 475 | 10 | 110 | 558014 | 53,12% | nadam |
| 3 | 815 | 476 | 616 | 17 | 208 | 691678 | 52,82% | nadam |
| 3 | 639 | 913 | 955 | 5 | 217 | 1469306 | 52,30% | adam |
| 3 | 643 | 590 | 308 | 13 | 163 | 567650 | 49,38% | SGD |
| 3 | 928 | 437 | 563 | 11 | 33 | 661919 | 48,73% | adagrad |
| 3 | 712 | 143 | 355 | 8 | 192 | 159487 | 47,86% | SGD |
| 3 | 334 | 607 | 124 | 8 | 72 | 281323 | 47,21% | adagrad |

В таблице 3 используется датасет CIFAR-10, а в таблице 4 используется MNIST. Наилучший результат с 56% точности в распознавании CIFAR-10 получился при векторах, уменьшенных до 128 с оптимизаторами Adam и Nadam. В среднем вектора, уменьшенные до размеров 64, 256 и 512 показывают примерно одинаковые результаты точности на уровне 46%. Точность начинает ухудшаться, когда размер векторов опускается ниже 16. Наилучший результат точности в датасете CIFAR-10 с применением метода главных компонент сопоставим с наихудшим результатом в исходных данных. В MNIST же наилучший результат с 98% точности получился у векторов размерностью 98, 49, 24 и 196. В среднем с результатом точности в 90-94% показывают вектора с размерностью 12, 24, 49, 98, 196 и 392. Результат ухудшается при размерности меньше 10. Можно заметить, что результаты точности с применением метода главных компонент в некоторых случаях лучше, чем в исходных данных, следовательно, данный метод может быть эффективен в MNIST при правильном подборе параметров и построения модели нейросети, которая в полной мере продемонстрирует возможности метода главных компонент.

# Заключение

В данной работе мы исследовали влияние архитектуры нейросети и метода главных компонент на точность распознавания изображений в наборе данных MNIST и CIFAR-10. В первой главе мы рассмотрели основы нейросетей и их архитектуры, во второй - TensorFlow и Keras, два наиболее популярных фреймворка для создания нейросетей. В третьей главе мы представили метод главных компонент и разбирались, как он работает. В четвертой главе мы провели эксперименты, чтобы проверить, как метод главных компонент влияет на точность распознавания. Мы сравнили точность распознавания с использованием уменьшенных размерностей, полученных из метода главных компонент, с точностью распознавания без использования данного метода. В пятой главе мы обсудили результаты наших экспериментов и сделали выводы.

В заключении мы можем подытожить, что результаты нашей работы показали, что выбор определенной архитектуры нейронной сети может существенно повлиять на качество распознавания рукописных цифр. Кроме того, использование метода главных компонент для уменьшения размерности данных может улучшить точность распознавания. Поэтому, выбор оптимальной архитектуры и применение метода главных компонент могут обеспечить высокую точность распознавания изображений и повысить эффективность системы. В целом, результаты нашей работы имеют практическое применение в области компьютерного зрения и могут стать полезным вкладом в развитие исследований в этой области.

# Список литературы

1. Что такое нейронная сеть? // Amazon Web Services : [сайт]. – 2022. – URL: <https://aws.amazon.com/ru/what-is/neural-network/> (Дата обращения 15.05.23)
2. История возникновения нейронных сетей // Интерфейс : [сайт]. – 2015. – URL: <http://www.interface.ru/home.asp?artId=37476> (Дата обращения 16.05.23)
3. Что такое нейросеть: как устроен человеческий мозг «в цифре» // РБК Тренды : [сайт]. – 2023. – URL: <https://trends.rbc.ru/trends/industry/641157be9a7947d3401fa3e8> (Дата обращения 17.05.23)
4. Keras // Skillfactory media : [сайт]. – 2023. – URL: <https://blog.skillfactory.ru/glossary/keras/> (Дата обращения 18.05.23)
5. Обзор на TensorFlow от Google: лидирующая в отрасли система глубокого обучения // Дзен : [сайт]. – 2023. – URL: <https://dzen.ru/a/ZCUSA5sgdGEZoSnt> (Дата обращения 19.05.23)
6. Метод главных компонент // Loginom Wiki : [сайт]. – 2018. – URL: <https://wiki.loginom.ru/articles/principal-component-analysis.html> (Дата обращения 20.05.23)
7. Особенности Keras // PythonRu : [сайт]. – 2020. – URL: <https://pythonru.com/biblioteki/osobennosti-keras-keras-8> (Дата обращения 21.05.23)
8. Гафаров Ф.М Искусственные нейронные сети и приложения: учеб. пособие / Ф.М. Гафаров, А.Ф. Галимянов. – Казань: Изд-во Казан. ун-та, 2018. – 121 с.
9. Шолле Ф. Глубокое обучение на Python. — СПб.: Питер, 2018. — 400 с.: ил. — (Серия «Библиотека программиста»).
10. Машинное обучение и TensorFlow. – СПб.: Питер, 2019. – 336 с.: ил. – (Серия «Библиотека программиста»).