Министерство образования Республики Беларусь

Учреждение образования

«БелорусскиЙ государственный университет

информатики и радиоэлектроники»

Факультет компьютерного проектирования

Кафедра проектирования информационно-компьютерных систем

Дисциплина «Объектно-ориентированное программирование»

|  |
| --- |
| *К защите допустить*: |
| Руководитель курсовой работыассистент |
| \_\_\_\_\_\_\_\_\_\_\_\_\_А.В.Михалькевич\_\_\_.\_\_\_. 20\_\_\_ |

**Пояснительная записка**

к курсовой работе

на тему

**Создание онлайн приложения по распознаванию рукописного текста**

БГУИР КР 1-40 05 01-10 №\_28

|  |  |  |
| --- | --- | --- |
| Студент | (подпись студента) | Ярош. Е.В. |
|  |  | Курсовая работа представлена на проверку \_\_\_.\_\_\_. 20\_\_\_ |
|  |  | (подпись студента) |

Минск 2020

СОДЕРЖАНИЕ

[Введение 4](#_Toc40411378)

1. [Описание проекта 5](#_Toc40411379)

[1.1 Клиентская часть 6](#_Toc40411384)

 [1.2 Серверная часть 6](#_Toc40411387)

[2 Математические методы и алгоритмы анализа изображений 7](#_Toc40411390)

[2.1 Понятие изображения 7](#_Toc40411391)

[2.2 Классификация цветовых систем изображений 8](#_Toc40411392)

[2.3 Растровые и векторные изображения 8](#_Toc40411393)

[2.4 Алгоритмы векторизации изображения 9](#_Toc40411394)

[2.5 Эталонные изображения 10](#_Toc40411395)

[2.6 Математический аппарат аффинных преобразований эталонных изображений 10](#_Toc40411396)

[2.7 Анализ литературных источников и определение корреляционной функции 11](#_Toc40411397)

[3. Объектно-ориентированные технологии программирования 12](#_Toc40411398)

[4. Инструментарий 15](#_Toc40411399)

 [4.1 IDE Microsoft VS Code 15](#_Toc40411400)

[4.2 Использование системы контроля версий GIT 17](#_Toc40411401)

[5. Архитектурный шаблон проектирования MVC 20](#_Toc40411402)

[6. Шаблоны проектирования практических задач 23](#_Toc40411403)

[ЗАКЛЮЧЕНИЕ 26](#_Toc40411404)

[7. Список использованных источников 27](#_Toc40411405)

[8. Приложение А 28](#_Toc40411406)

[9. Приложение Б 31](#_Toc40411407)

# Введение

Вычислительные системы существуют не один десяток лет и главной целью их создания являлась возможность замены человека, выполнения за него трудоемкой работы.

Алгоритм распознавания текста используется в различных сферах: требуется для оцифровки старых книг, перевода текста изображения в электронный вид для более удобной работы с ним.

Распознавание текста – задача сложная для реализации. Человек для этого задействует комплекс знаний и опыта: выделяет текст из совокупности сигналов органов чувств, характерные признаки символов и на основании своего опыта делает вывод о значении символа и всего текста.

Разработчики программ распознавания текста сталкиваются с проблемами наложения символов друг на друга, их похожести в различных языках, низкого качества изображения, а также шума на изображении. До сих пор не было разработано программы, которая обеспечивала бы полную достоверность распознавания, именно поэтому в процесс распознавания символов необходимо вмешательство человека.

В курсовой работе поставлена цель: проектирование графического пользовательского интерфейса для распознавания и анализа текстовой информации на изображениях.

1. **Описание проекта**

Курсовой проект представляет собой онлайн приложение по распознавание рукописных цифр.

Веб-приложение разработано с помощью языков JS и Python, технологии React и Flask.

****

Рисунок 1 –Начальное состояние

****

Рисунок 2 –Результат

**1.1 Клиентская часть**

 Клиентская часть приложения представлена одной страницей. На ней расположены: canvas-окно для ввода изображения, кнопки управления, вывод.

 Клиентская часть приложения выполнена с использованием технологии React JS.

**1.2 Серверная часть**

 Серверная часть приложения выполнена с использованием технологии Flask(Python). По своей сути серверная часть представляет собой API интерфейс принимающий POST запросы с клиентской части.

 Пришедшее с клиентской части изображении обрабатывается, нормализуется и передаётся в обученную нейросеть для анализа. Результат работы НС передаётся обратно в Клиентскую часть.

# 2 Математические методы и алгоритмы анализа изображений

**2.1 Понятие изображения**

Цифровое изображение — массив данных, полученный путем дискретизации (аналого-цифрового преобразования) оригинала. Будучи закодированным с помощью особого алгоритма и записанным на носитель, этот массив данных становится файлом.

## 2.2 Классификация цветовых систем изображений

Большинство графических пакетов оперируют широким кругом цве­товых моделей, часть из которых создана для специальных целей, а другая - для особых типов красок. Примерами таких пакетов являются: CMY, CMYK, RGB, HSB, HLS, Lab, YIQ, YCC.

Перечисленные цветовые модели можно условно разбить на три класса по принципу действия:

• аддитивные (RGB), основанные на сложении цветов;

• субтрактивные (CMY, CMYK), основу которых составляет операция вычита­ния цветов (субтрактивный синтез);

• перцепционные (HSB, HLS, Lab, YCC), базирующиеся на восприятии.

**2.3 Растровые и векторные изображения**

**Векторная графика** – способ представления объектов и изображений в компьютерной графике, основанный на математическом описании элементарных геометрических объектов, таких как точки, линии, сплайны, кривые Безье и многоугольники. Таким образом, объекты векторной графики задаются математическими правилами и являются графическими изображениями математических объектов. Основным преимуществом векторной графики является то, что при изменении размера векторного изображения изображение не превращается в пиксельное, т.е. не теряется его качество. Это значит, что векторное изображение можно увеличивать до любых размеров и при этом не терять четкость. Редактирование векторного изображение также выигрывает перед редактированием растрового, потому что не теряется качество изображения. Векторная графика используется при создании логотипов компаний, шрифтов, в рекламе и полиграфии.
Наиболее распространенные форматы файлов: CDR, AI, EPS, PDF, SVG.

В **растровой графике** изображение представлено в виде графической матрицы, состоящей из пикселей, которые представляют собой объекты прямоугольной или круглой формы фиксированного размера. Каждому пикселю приписан атрибут цвета, поэтому совокупность таких пикселей формирует полное изображение. Изображение будет более детальным и качественным, если на единицу площади будет приходиться больше пикселей. Детализация растрового изображения не может быть увеличена, так как его максимальная составляющая задается при создании. В отличие от векторного изображения при увеличении масштаба изображения пиксели превращаются в крупные зерна. Тем не менее растровые изображения способны передавать реалистичные изображения, которые состоят из тысяч мелких деталей. Растровая графика используется преимущественно при работе с фотографиями, которые в последующем также могут быть использованы совместно с векторной графикой в полиграфии и рекламе.
Наиболее распространенные форматы файлов: JPEG, PNG, TIFF, GIF, BMP.

**2.4 Алгоритмы векторизации изображения**

Векторизация – это процесс конвертирования растрового изображения в векторное, которое впоследствии может использоваться в программах автоматизированного проектирования.

Процесс векторизации состоит из нескольких этапов.

Во-первых, некоторые изображения требуют предварительной настройки – обработки растрового изображения к такому виду, который бы впоследствии с легкостью мог быть приведен к векторному виду алгоритмами. Настройка может быть выполнена несколькими способами: ручным и автоматическим. Ручная настройка проводится в растровом графическом редакторе, а для автоматической настройки используются графические фильтры и методы обработки. Отличительной особенностью векторизации является то, что шум на изображениях отсутствует или игнорируется.

Во-вторых, на изображении выделяются контуры областей и крайних точек, необходимый для выделения точек, описывающих данные области. Трудность данного этапа заключается в том, что результат не дает информации о том, какая группа точек описывает область, так как их может быть сколько угодно. Решение данной проблемы заключается в применении алгоритма, который обходит весь контур, строит выпуклую оболочку и проводит кластеризацию. В результате метода обхода контура получаются списки крайних точек и списки остальных точек контура в порядке направления его обхода. С помощью этих списков точек формируют многоугольники, описывающие области карты. Этот метод считается достаточно эффективным, быстрым и простым в реализации, поэтому он наиболее популярен при векторизации.

В-третьих, проводится формирование векторного формата, на котором решаются дополнительные задачи. На этом этапе решаются и дополнительные задачи: находятся различные статистические характеристики изображения и его отдельных областей.

В-четвертых, полученную векторную информацию, которая включает в себя само векторное изображение и дополнительную информацию, определенную на предыдущем этапе, заносят в определенную структуру данных, которая сохраняется на носителях и используется в графических редакторах или других приложениях.

## ****2.5 Эталонные изображения****

Эталонные изображения применяются в различных методах работы с изображениями, например, при корреляционном методе сравнения двух изображений. В этом случае изображение сдвигается слева направо или сверху вниз по эталонному и последовательно сравниваются каждая из точек. На основе полученных результатов, которые обычно представляются в виде графика, делают выводы о степени похожести двух изображений. В случае распознавания текста в программу загружаются эталонные изображения каждого символа и после сегментирования каждый образ символа сравнивается с загруженным эталонным.

Эталонные изображения $X\_{1}, X\_{2}, …,X\_{m}$ некоторого числа m различных классов образов в n-мерном пространстве задаются в виде точек $ \left(x\_{11},x\_{12},…,x\_{1n}\right), \left(x\_{21},x\_{22},…,x\_{2n}\right), …, \left(x\_{m1},x\_{m2},…,x\_{mn}\right)$. Любое входное изображение $S\_{i}$ также представляется в виде точки $\left(s\_{i1},s\_{i2},…,s\_{in}\right)$ в этом пространстве. Принадлежность входного изображения $S\_{i}$ к одному из m классов определяется с помощью расстояний между точкой $S\_{i}$ и всеми точками $X\_{1}, X\_{2}, …,X\_{m}$ соответствующими эталонным образам. Расстояние и является мерой сходства входного изображения с эталонами классов или образов. Входное изображение относится к тому образу, расстояние до эталонного изображения которого минимально, т.е. решающим правилом является следующее соотношение

$S\_{i}\in X\_{j}, если L\left(S\_{i},X\_{j}\right)=\min\_{j}⁡(L\left(S\_{i},X\_{j}\right))$, (1)

В теории распознавания образов часто используются расстояния по Евклиду (2) и по Минковскому (3):

$L\left(S\_{i},X\_{j}\right)=\sqrt{\sum\_{k=1}^{n}(s\_{ik}-x\_{jk})^{2}}$, (2)

$L\left(S\_{i},X\_{j}\right)=\sqrt[λ]{\sum\_{k=1}^{n}(s\_{ik}-x\_{jk})^{λ}}$, (3)

где $λ$ - целое положительное число, большее двух.

## ****2.6 Математический аппарат аффинных преобразований эталонных изображений****

В целях упрощения анализа изображения считается, что проецируются точки плоской поверхности объекта на плоскую матрицу светочувствительных.

Если проецируемая поверхность объекта параллельна плоскости, содержащей светочувствительные элементы, сформированное изображение объекта, подлежащее дальнейшей обработке и распознаванию, геометрически трансформируется преобразованием подобия, включающим в себя линейные сдвиги вдоль декартовых координатных осей, равномерное масштабирование, вращение в плоскости изображения и зеркальное отражение относительно произвольной прямой, лежащей в плоскости изображения и проходящей в ней через начало координат. Масштабирование, вращение и линейные смещения проекции объекта на светочувствительной среде возникают при соответствующих пространственных перемещениях объекта в поле зрения системы относительно его эталонного пространственного положения. Зеркальное отражение, если не рассматривать тривиальное отражение через зеркало, может возникать при наблюдении прозрачного объекта с обратной (задней) стороны, что в реальных ситуациях встречается нечасто.

Если плоскость, содержащая светочувствительные элементы, становится не параллельна отображаемой поверхности объекта (поворачивается относительно некоторой оси, лежащей в этой плоскости), но объект находится далеко от объектива (линзы), то к указанным выше составляющим преобразования подобия добавляется сжатие изображения вдоль прямой, перпендикулярной указанной оси. Полученное более сложное геометрическое преобразование называется аффинным.



где х, у – абсцисса и ордината точки исходного изображения, х′, у′ – преобразованные координаты этой точки, а1 … а6 – параметры преобразования.

## ****2.7 Анализ литературных источников и определение корреляционной функции****

Корреляционная функция применяется при сравнении двух изображений: чаще всего эталонного и экспериментального.

Нормированная функция корреляции имеет вид:



При распознавании изображений одним из методов является использование корреляционной метрики, а именно – нормированного коэффициента корреляции, который имеет вид:



где f0, g0 – средние значения интенсивности для изображений соответственно f и g.

Пусть дано n эталонных изображений {gi}, i = 1,..,,n, каждое из которых соответствует i-му классу. Обнаружение фрагмента изображения f по методу максимальной корреляционной связи осуществляется тогда по закону:



После этого на основании полученного значения максимальной корреляции, может проверяться достоверность детектирования. Если k(f,gi) ≥ kmin, то обнаружение признается достоверным. В противном случае объект считается нераспознанным. Корреляционный метод считается достаточно эффективным и оптимальным при применении для обнаружения детерминированного сигнала в белом шуме с гауссовым распределением яркостей.

При оптическом распознавании текста используются различные методы, один из которых – метод сопоставления изображений и шаблонов. При непосредственном сравнении этих двух изображений вычисляется степень сходства между образом и каждым из эталонов символов. Классификация тестируемого изображения символа происходит по методу ближайшего соседа.

Многие коммерческие системы OCR используют именно метод корреляции, так как с практической точки зрения его легко реализовать. Тем не менее при недостоверном виде изображения, например, попадении темного пятнышка, результат распознавания может существенно испортиться.

# 3 Объектно-ориентированные технологии программирования

**3.1 Объектно-ориентированное программирование**

До сих пор не существует общепризнанного определения объектно-ориентированного программирования. Можно найти и философское, и научное, и техническое определение. И все они будут правильными.

**Объектно-ориентированное программирование (ООП)** — методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования.

Необходимо обратить внимание на следующие важные части этого определения:

1 объектно-ориентированное программирование использует в качестве основных логических конструктивных элементов объекты, а не алгоритмы;

2 каждый объект является экземпляром определенного класса;

3 классы образуют иерархии.

Программа считается объектно-ориентированной, только если выполнены все три указанных требования.

В соответствии с другими определениям, под ООП можно понимать:

* звено в парадигмах программирования.
* мыслеформа программиста, занятого программными решениями

**3.2 Классы и объекты**

Объект – это абстрактная сущность, наделенная характеристиками объектов окружающего нас реального мира. Так, при объяснении объектов в ООП используют пример жилого дома и, например, дома искусства.

Создание объектов и манипулирование ими – это результат методологии программирования, воплощающей в кодовых конструкциях описания объектов и операции
над ними. Каждый объект программы, как и любой реальный объект, отличается собственными атрибутами и характерным поведением.

Класс – это также абстрактная сущность, которая включает в себя объекты и систематизирует их. Если вернуться к примеру о доме, то классом будет считаться в общем любой дом. При определении каждого из объектов мы будет пользоваться характеристиками данного класса (т.е. его свойствами), а также свойствами, которыми он владеет (т.е. методами).

Классы имеют определенные признаки, среди которых можно выделить то, что каждый класс занимает определенное место в иерархии классов, любой класс определяет некоторую категорию объектов, а всякий объект есть экземпляр некоторого класса.

Класс не имеет физической сущности, его ближайшей аналогией в процедурном программировании является объявление структуры. Память выделяется только тогда, когда класс используется для создания объекта. Этот процесс также называется созданием экземпляра класса.

Любой объект языка C# имеет одинаковые атрибуты
и функциональность с другими объектами того же класса. Это свойство существует не только в данном языке программирования, но и, например, в C++.

Обычно объект находится в некотором уникальном состоянии, определяемом текущими значениями его атрибутов. Функциональность объектного класса определяется возможными операциями над экземпляром этого класса. Определение класса в языке C# содержит инкапсуляцию членов данных и методов, оперирующих с членами данных и определяющих поведение объекта.

**3.3 Инкапсуляция**

Термин «инкапсуляция» похож на «капсулу», что предполагает под собой разграничение, изолирование объекта от внешнего окружения. Так, в ООП инкапсуляция используется для того, чтобы пользователь при создании объекта класса не мог воспользоваться определенными методами, которые непосредственно определяют функционал объекта.

Инкапсуляция существенно повышает надежность разрабатываемых программ, т.к. локализованные в объекте функции обмениваются с программой сравнительно небольшими объемами данных, причем количество и тип этих данных обычно тщательно контролируются.
В результате замена или модификация функций и данных, инкапсулированных в объект, как правило, не влечет за собой плохо прослеживаемых последствий для программы в целом (в целях повышения защищенности программ в ООП почти не используются глобальные переменные).

Другим немаловажным следствием инкапсуляции является легкость обмена объектами, переноса их из одной программы в другую.

**3.4 Наследование**

Наследование в ООП применяется в случаях, когда в программе используется несколько классов с определенным набором одинаковых свойств. Для того, чтобы сократить сложность написания кода используется наследование.

Механизм строится следующим образом: создается первый класс, считающийся самым главным, ему задаются все необходимые свойства и методы, а далее создается второй класс, который и должен унаследовать все свойства и методы первого.

Выделяют также множественное наследование: возможность одного класса наследовать свойства и методы сразу нескольких.

Наследование позволяет практически без ограничений последовательно строить и расширять классы. Начиная с самых простых классов можно создавать производные классы по возрастающей сложности, которые не только легки в отладке, но и просты по внутренней структуре.

Последовательное проведение в жизнь принципа наследования, особенно при разработке крупных программных проектов, хорошо согласуется с техникой нисходящего структурного программирования
(от общего к частному), и во многом стимулирует такой подход. При этом сложность кода программы в целом существенно сокращается. Производный класс (потомок) наследует все свойства, методы и события своего базового класса (родителя) и всех его предшественников в иерархии классов.

При наследовании базовый класс обрастает новыми атрибутами
и операциями. В производном классе обычно объявляются новые члены данных, свойства и методы. При работе с объектами программист обычно подбирает наиболее подходящий класс для решения конкретной задачи
и создает одного или нескольких потомков от него, которые приобретают способность делать не только то, что заложено в родителе.

Кроме того, производный класс может перегружать наследуемые методы в том случае, когда их работа в базовом классе не подходит потомку.

Использование перегрузки в ООП всячески поощряется, хотя в прямом понимании значения этого слова перегрузок обычно избегают. Говорят, что метод перегружен, если он ассоциируется с более чем одной одноименной функцией. Механизм вызовов перегруженных методов в иерархии классов полностью отличается от вызовов переопределенных функций. Виртуальные методы используются для переопределения функций базового класса.

**3.5 Полиморфизм**

Слово полиморфизм от греческих слов poly (много) и morphos (форма) означает множественность форм. Полиморфизм – это свойство родственных объектов (т.е. объектов, классы которых являются производными от одного родителя) вести себя по-разному в зависимости от ситуации, возникающей
в момент выполнения программы. В рамках ООП программист может влиять на поведение объекта только косвенно, изменяя входящие в него методы
и придавая потомкам отсутствующие у родителя специфические свойства.

Для изменения метода необходимо перегрузить его в потомке, т.е. объявить в потомке одноименный метод и реализовать в нем нужные действия. В результате в объекте-родителе и объекте-потомке будут действовать два одноименных метода, имеющие разную кодовую реализацию и, следовательно, придающие объектам разное поведение.

# 4 ИНСТРУМЕНТАРИЙ

# 4.1 IDE Microsoft VS Code

При реализации курсовой работы я использовал IDE Microsoft VS Code.

IDE(Integrated Drive Electronics или же интегрированная среда разработки) – комплекс программных средств, используемый [программистами](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%81%D1%82) для разработки [программного обеспечения](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5). Среда разработки включает в себя:

* текстовый редактор;
* компилятор;
* отладчик и прочее.

Суть IDE заключается в объединении нескольких компонентов для максимальной концентрации программиста на решении алгоритмических задач, без потерь времени на сохранение файла в текстовом редакторе, затем вызове компилятора и так далее. Таким образом, повышается производительность труда разработчика. Также огромным преимуществом среды разработки является наличие статического анализатора кода, который позволяет выявить ошибки в синтаксисе и другие мелкие ошибки по ходу написания программы.

Microsoft VS Code – это среда разработки от компании Microsoft. Она включает в себя поддержку многих компонентов таких как: C#, Python, .NET и её компоненты, имеет встроенный веб–сервер который может пригодиться при веб–разработке, присутствует интеграция с Unity и Unreal Engine. Можно разрабатывать как консольные приложения, так и приложения с графическим интерфейсом. Visual Studio позволяет создавать и подключать сторонние дополнения для расширения функциональности при помощи NuGet, также присутствует интеграция с системой контроля версий Git.

Основные возможности и преимущества программы:

* Имеются встроенные отладчик и командная строка.
* Поддержка практически всех языков программирования.
* Наличие встроенной библиотеки элементов кода.
* Автозавершение при вводе кода.
* Добавление в библиотеку собственных сниппетов.
* Подсветка синтаксиса.
* Одновременная работы с несколькими проектами.
* Поддержка многооконного и двухпанельного режимов.
* Расширение функционала с помощью плагинов.
* Интеграция с Visual Studio Team Services, GitHub и GIT.
* Наличие встроенных средств для тестирования, сборки, упаковки и развертывания приложений.
* Публикация созданных программных продуктов в Microsoft Azure (через посредство Visual Studio Team Services).
* Интегрированная система подсказок.
* Командная работа над проектами.
* Широкий набор настроек и кроссплатформенность.

**4.2 Использование системы контроля версий GIT**

Система контроля версий(СКВ) – это система, записывающая изменения в файл или набор файлов в течение времени и позволяющая вернуться позже к определённой версии. Она позволяет вернуть файлы к состоянию, в котором они были до изменений, вернуть проект к исходному состоянию, увидеть изменения, увидеть, кто последний менял что-то и вызвал проблему, кто поставил задачу и когда и многое другое.

Многие люди в качестве метода контроля версий применяют копирование файлов в отдельную директорию (возможно даже, директорию с отметкой по времени, если они достаточно сообразительны). Данный подход очень распространён из-за его простоты, однако он невероятно сильно подвержен появлению ошибок. Можно легко забыть, в какой директории вы находитесь, и случайно изменить не тот файл или скопировать не те файлы, которые вы хотели.

Для того, чтобы решить эту проблему, программисты разработали локальные СКВ с простой базой данных, которая хранит записи о всех изменениях в файлах, осуществляя тем самым контроль ревизий.

На рисунке 3 представлена схема локального контроля версий.

 

Рисунок 3 – Локальный контроль версий

Одной из популярных СКВ была система RCS, которая и сегодня распространяется со многими компьютерами. RCS хранит на диске наборы патчей (различий между файлами) в специальном формате, применяя которые она может воссоздавать состояние каждого файла в заданный момент времени.

Следующая серьёзная проблема, с которой сталкиваются люди, – это необходимость взаимодействовать с другими разработчиками. Для того, чтобы разобраться с ней, были разработаны централизованные системы контроля версий (ЦСКВ). Такие системы, как CVS, Subversion и Perforce, используют единственный сервер, содержащий все версии файлов, и некоторое количество клиентов, которые получают файлы из этого централизованного хранилища. Применение ЦСКВ являлось стандартом на протяжении многих лет.

На рисунке 4 представлена схема централизованного контроля версий.

 

Рисунок 4 – Централизованный контроль версий

Такой подход имеет множество преимуществ, особенно перед локальными СКВ. Например, все разработчики проекта в определённой степени знают, чем занимается каждый из них. Администраторы имеют полный контроль над тем, кто и что может делать, и гораздо проще администрировать ЦСКВ, чем оперировать локальными базами данных на каждом клиенте.

Несмотря на это, данный подход тоже имеет серьёзные минусы. Самый очевидный минус – это единая точка отказа, представленная централизованным сервером. Если этот сервер выйдет из строя на час, то в течение этого времени никто не сможет использовать контроль версий для сохранения изменений, над которыми работает, а также никто не сможет обмениваться этими изменениями с другими разработчиками.

Существуют распределённые системы контроля версий (РСКВ). В РСКВ (таких как Git, Mercurial, Bazaar или Darcs) клиенты не просто скачивают снимок всех файлов (состояние файлов на определённый момент времени) – они полностью копируют репозиторий. В этом случае, если один из серверов, через который разработчики обменивались данными, умрёт, любой клиентский репозиторий может быть скопирован на другой сервер для продолжения работы. Каждая копия репозитория является полным бэкапом всех данных.

На рисунке 5 изображена схема распределённого контроля версий.

 

 Рисунок 5 – Распределённый контроль версий

Более того, многие РСКВ могут одновременно взаимодействовать с несколькими удалёнными репозиториями, благодаря этому разработчики могут работать с различными группами людей, применяя различные подходы единовременно в рамках одного проекта. Это позволяет применять сразу несколько подходов в разработке, например, иерархические модели, что совершенно невозможно в централизованных системах.

В нашем курсовом проекте использовался онлайн-репозиторий GitHub. Ссылка на него: https://github.com/dmitrysavitski/Shoes-store

**5. АРХИТЕКТУРНЫЙ ШАБЛОН ПРОЕКТИРОВАНИЯ MVC**

Шаблон проектирования MVC предполагает разделение данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: Модель, Представление и Контроллер – таким образом, что модификация каждого компонента может осуществляться независимо.

Контроллер управляет запросами пользователя (получаемые в виде запросов HTTP GET или POST, когда пользователь нажимает на элементы интерфейса для выполнения различных действий). Его основная функция — вызывать и координировать действие необходимых ресурсов и объектов, нужных для выполнения действий, задаваемых пользователем. Обычно контроллер вызывает соответствующую модель для задачи и выбирает подходящий вид.

Модель - это данные и правила, которые используются для работы с данными, которые представляют концепцию управления приложением. В любом приложении вся структура моделируется как данные, которые обрабатываются определённым образом. Что такое пользователь для приложения — сообщение или книга? Только данные, которые должны быть обработаны в соответствии с правилами (дата не может указывать в будущее, e-mail должен быть в определённом формате, имя не может быть длиннее Х символов, и так далее).

Модель даёт контроллеру представление данных, которые запросил пользователь (сообщение, страницу книги, фотоальбом, и тому подобное). Модель данных будет одинаковой, вне зависимости от того, как мы хотим представлять их пользователю. Поэтому мы выбираем любой доступный вид для отображения данных.

Модель содержит наиболее важную часть логики нашего приложения, логики, которая решает задачу, с которой мы имеем дело (форум, магазин, банк, и тому подобное). Контроллер содержит в основном организационную логику для самого приложения (очень похоже на ведение домашнего хозяйства).

Вид обеспечивает различные способы представления данных, которые получены из модели. Он может быть шаблоном, который заполняется данными. Может быть несколько различных видов, и контроллер выбирает, какой подходит наилучшим образом для текущей ситуации.

Веб приложение обычно состоит из набора контроллеров, моделей и видов. Контроллер может быть устроен как основной, который получает все запросы и вызывает другие контроллеры для выполнения действий в зависимости от ситуации.

Рассмотрим классическую схему веб-приложения, которая отражена на рисунке 6:

Рисунок 6 – Принцип работы контроллера, модели и элементов представления в шаблоне проектирования MVC

На этом и последующем рисунках пунктирными линиями показана управляющая информация (такая, например, как ID запрашиваемой записи блога или товара в магазине), а сплошными – собственно данные приложения (которые могут храниться в БД, или в виде файлов на диске, или даже, возможно, в оперативной памяти – этот вопрос лежит за пределами паттерна MVC). В применении к вебу запрос и ответ ходят по HTTP, поэтому можно условно считать, что на этом рисунке пунктиром обозначены заголовки HTTP-запроса и ответа, а сплошными линиями – их тела.

Вопрос о том, кто должен проверять на валидность и права доступа входные данные (Контроллер или Модель), является предметом достаточно многочисленных споров, поскольку паттерн MVC не описывает таких деталей. Это значит, что в этом вопросе выбор за вами (или за вас его сделали авторы вашего любимого фрейvворка или CMS).

На практике придерживаются такого подхода: контроллер проверяет входные данные на предмет «общей» (т.е. независящей от конкретного запроса) корректности, соответствие требованиям Модели к валидности сохраняемых данных проверяет соответствующий метод Модели, а права доступа – метод Access отдельного класса User.

Для вызова Представления в PHP иногда проектируется специальный класс (а то и несколько классов), например, View (это часто встречается в описаниях MVC в реализации того или иного фреймворка), однако это не является требованием MVC.

Также файлы представлений часто называют шаблонами, а при использовании так называемых шаблонизаторов, роль Представления играет сам шаблонизатор, а шаблоны (т.е. файлы, содержащие непосредственно HTML-разметку) в некоторых фреймворках называют layouts.

1. **ШАБЛОНЫ ПРОЕКТИРОВАНИЯ ПРАКТИЧЕСКИХ ЗАДАЧ**

Паттерн представляет определенный способ построения программного кода для решения часто встречающихся проблем проектирования.

Существует множество различных паттернов, которые решают разные проблемы и выполняют различные задачи. Но по своему действию их можно объединить в ряд групп. В основу классификации основных паттернов положена цель или задачи, которые определенный паттерн выполняет.

Порождающие паттерны — это паттерны, которые абстрагируют процесс инстанцирования или, иными словами, процесс порождения классов и объектов. Среди них выделяются следующие:

* Абстрактная фабрика (Abstract Factory)
* Строитель (Builder)
* Фабричный метод (Factory Method)
* Прототип (Prototype)
* Одиночка (Singleton)
* Простая фабрика (Simple Factory)

Другая группа паттернов - структурные паттерны - рассматривает, как классы и объекты образуют более крупные структуры - более сложные по характеру классы и объекты. К таким шаблонам относятся:

* Адаптер (Adapter);
* Мост (Bridge);
* Компоновщик (Composite);
* Декоратор (Decorator);
* Фасад (Facade);
* Приспособленец (Flyweight);
* Заместитель (Proxy).

Третья группа паттернов называются поведенческими - они определяют алгоритмы и взаимодействие между классами и объектами, то есть их поведение. Среди подобных шаблонов можно выделить следующие:

* Цепочка обязанностей (Chain of responsibility);
* Команда (Command);
* Интерпретатор (Interpreter) и т.д.

Существуют и другие классификации паттернов в зависимости от того, относится паттерн к классам или объектам.

Паттерны классов описывают отношения между классами посредством наследования. Отношения между классами определяются на стадии компиляции. К таким паттернам относятся:

* Фабричный метод (Factory Method);
* Интерпретатор (Interpreter);
* Шаблонный метод (Template Method);
* Адаптер (Adapter).

Другая часть паттернов - паттерны объектов описывают отношения между объектами. Эти отношения возникают на этапе выполнения, поэтому обладают большей гибкостью. К паттернам объектов относят следующие:

* Абстрактная фабрика (Abstract Factory);
* Строитель (Builder);
* Прототип (Prototype) и т.д.

Использование шаблонов проектирования отличает сильного программиста. Использование шаблонов дает возможность тратить меньше времени используя уже проверенные решения, без изобретения костылей. Вы делаете меньше просчетов при проектировании используя стандартные типовые решения. При работе в команде вам не требуется объяснять другим программистам что именно вы сделали так как вы можете просто назвать паттерн.

Однако у паттернов есть и свои минусы. Например, неэффективное использование. При знакомстве с паттернами множество программистов начинает их использовать не особо задумываясь, подходят они под конкретную задачу или нет. Также существует критическая необходимость приспосабливать паттерны к реалиям проекта, а не реализовывать их просто ко книжке.

Каждый из паттернов решает свою проблему, но при этом может разрушать принципы программирования. Рассмотрим по одному паттерну из каждого шаблона.

Одиночка или Singleton – это довольно популярный порождающий паттерн проектирования. Этот паттерн решает сразу две проблемы: гарантирует наличие единственного экземпляра класса, предоставляет глобальную точку доступа. При этом он нарушает принцип единственной ответственности – принцип [ООП](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), обозначающий, что каждый [объект](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29) должен иметь одну ответственность и эта ответственность должна быть полностью [инкапсулирована](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D0%BA%D0%B0%D0%BF%D1%81%D1%83%D0%BB%D1%8F%D1%86%D0%B8%D1%8F_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29) в [класс](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29).

Для реализации одиночки достаточно скрыть конструктор по умолчанию и создать публичный статический метод, который и будет контролировать жизненный цикл объекта.

Применять этот паттерн стоит, когда в программе должен быть единственный экземпляр какого–то класса, доступный всем клиентам. Или, когда вам хочется иметь больше контроля над глобальными переменными.

Стратегия(Strategy) – это поведенческий паттерн. Он предлагает определить семейство схожих алгоритмов, которые часто изменяются или расширяются, и вынести их в собственные классы, называемые стратегиями. Вместо того, чтобы изначальный класс сам выполнял тот или иной алгоритм, он будет играть роль контекста, ссылаясь на одну из стратегий и делегируя ей выполнение работы. Чтобы сменить алгоритм, вам будет достаточно подставить в контекст другой объект-стратегию. Важно, чтобы все стратегии имели общий интерфейс. Используя этот интерфейс, контекст будет независимым от конкретных классов стратегий. С другой стороны, вы сможете изменять и добавлять новые виды алгоритмов, не трогая код контекста.

Использовать этот класс стоит, когда вам нужно использовать разные вариации какого-то алгоритма внутри одного объекта, когда у вас есть множество похожих классов, отличающихся только некоторым поведением, когда вы не хотите показывать детали реализации алгоритмов для других классов. Однако использование этого паттерна связано с некоторыми неудобствами. Он усложняет программу за счет дополнительных классов. Помимо этого, для использования этого паттерна клиент должен знать в чем состоит разница между стратегиями чтобы выбрать подходящую.

Адаптер – это структурный паттерн проектирования. С помощью этого паттерна вы можете создать объект-переводчик, который трансформирует интерфейс или данные одного объекта в такой вид, чтобы он стал понятен другому объекту. При этом адаптер оборачивает один из объектов, так что другой объект даже не знает о наличии первого. Например, вы можете обернуть объект, работающий в метрах, адаптером, который бы конвертировал данные в футы. Адаптеры могут не только переводить данные из одного формата в другой, но и помогать объектам с разными интерфейсами работать сообща. Это работает так:

1. Адаптер имеет интерфейс, который совместим с одним из объектов.
2. Поэтому этот объект может свободно вызывать методы адаптера.
3. Адаптер получает эти вызовы и перенаправляет их второму объекту, но уже в том формате и последовательности, которые понятны второму объекту.

Применять этот паттерн стоит когда вы хотите использовать сторонний класс, но его интерфейс не соответствует остальному коду приложения и Когда вам нужно использовать несколько существующих подклассов, но в них не хватает какой–то общей функциональности, причём расширить суперкласс вы не можете.

**ЗАКЛЮЧЕНИЕ**

Целью курсового проекта являлась разработка онлайн приложения по распознаванию рукописного текста.

В ходе выполнения проекта было изучено большое количество технологий, таких как Python Flask, Bootstrap, шаблон проектирования MVC. При работе над курсовой работой я изучил большое количество материалов по теме для грамотного и качественного решения задач, поставленных вначале разработки.

Убедившись в корректности работы программы, можно сказать, что результат курсового проекта «Gриложения по распознаванию рукописного текста» полностью соответствует всем требованиям, поставленным вначале разработки программы.

# 7. Список использованных источников

[1] Обработка и анализ цифровых изображений с примерами на LabVIEW IMAQ Vision/ Визильтер, Ю. В. [и др.]. / М.: ДМК Пресс, 2007. – 464 с.

 [2] Волкова, М. А. Методы обработки и распознавания изображений: учебно-методическое пособие по лабораторному практикуму/ М. А. Волкова, В. Р. Луцив. – СПб : Университет ИТМО, 2016. – 40 с.

[3] Студеникина Л. И. Метод наименьших квадратов: методические указания и индивидуальные задания по выполнению лабораторной работы №15/ Л. И. Студеникина, Т. В. Шевцова – Курск: Юго-Зап. гос. ун-т, 2011. - 52 с.

[4] Балахонцева А. Система распознавания символов на изображениях со сложным фоном/ А. Балахонцева, А. Годоба, Н. Тьен. - Институт компьютерных систем ОНПУ, Одесса, Украина, 2013. - 4 с.

[5] Git. Википедия – свободная энциклопедия [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Git

[6] Model-View-Controller [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Model-View-Controller

[7] Основы паттернов проектирования [Электронный ресурс]. – Режим доступа: https://metanit.com/sharp/patterns/1.1.php

 [8] Cochran, D. Bootstrap Site Blueprints / D. Cochran, I. Whitley. – Packt, 2014. – 304 p.

# Приложение А

Листинг кода файла neural\_network.py

import numpy as np

import pickle

@np.vectorize

def sigmoid(x):

 return 1 / (1 + np.e \*\* -x)

activation\_function = sigmoid

from scipy.stats import truncnorm

def truncated\_normal(mean=0, sd=1, low=0, upp=10):

 return truncnorm((low - mean) / sd,

 (upp - mean) / sd,

 loc=mean,

 scale=sd)

class NeuralNetwork:

 def \_\_init\_\_(self,

 no\_of\_in\_nodes,

 no\_of\_out\_nodes,

 no\_of\_hidden\_nodes,

 learning\_rate):

 self.no\_of\_in\_nodes = no\_of\_in\_nodes

 self.no\_of\_out\_nodes = no\_of\_out\_nodes

 self.no\_of\_hidden\_nodes = no\_of\_hidden\_nodes

 self.learning\_rate = learning\_rate

 self.create\_weight\_matrices()

 def create\_weight\_matrices(self):

 """

 A method to initialize the weight

 matrices of the neural network

 """

 rad = 1 / np.sqrt(self.no\_of\_in\_nodes)

 X = truncated\_normal(mean=0,

 sd=1,

 low=-rad,

 upp=rad)

 self.wih = X.rvs((self.no\_of\_hidden\_nodes,

 self.no\_of\_in\_nodes))

 rad = 1 / np.sqrt(self.no\_of\_hidden\_nodes)

 X = truncated\_normal(mean=0, sd=1, low=-rad, upp=rad)

 self.who = X.rvs((self.no\_of\_out\_nodes,

 self.no\_of\_hidden\_nodes))

 def train(self, input\_vector, target\_vector):

 """

 input\_vector and target\_vector can

 be tuple, list or ndarray

 """

 input\_vector = np.array(input\_vector, ndmin=2).T

 target\_vector = np.array(target\_vector, ndmin=2).T

 output\_vector1 = np.dot(self.wih,

 input\_vector)

 output\_hidden = activation\_function(output\_vector1)

 output\_vector2 = np.dot(self.who,

 output\_hidden)

 output\_network = activation\_function(output\_vector2)

 output\_errors = target\_vector - output\_network

 # update the weights:

 tmp = output\_errors \* output\_network \

 \* (1.0 - output\_network)

 tmp = self.learning\_rate \* np.dot(tmp,

 output\_hidden.T)

 self.who += tmp

 # calculate hidden errors:

 hidden\_errors = np.dot(self.who.T,

 output\_errors)

 # update the weights:

 tmp = hidden\_errors \* output\_hidden \* \

 (1.0 - output\_hidden)

 self.wih += self.learning\_rate \

 \* np.dot(tmp, input\_vector.T)

 def run(self, input\_vector):

 # input\_vector can be tuple, list or ndarray

 input\_vector = np.array(input\_vector, ndmin=2).T

 output\_vector = np.dot(self.wih,

 input\_vector)

 output\_vector = activation\_function(output\_vector)

 output\_vector = np.dot(self.who,

 output\_vector)

 output\_vector = activation\_function(output\_vector)

 return output\_vector

 def confusion\_matrix(self, data\_array, labels):

 cm = np.zeros((10, 10), int)

 for i in range(len(data\_array)):

 res = self.run(data\_array[i])

 res\_max = res.argmax()

 target = labels[i][0]

 cm[res\_max, int(target)] += 1

 return cm

 def precision(self, label, confusion\_matrix):

 col = confusion\_matrix[:, label]

 return confusion\_matrix[label, label] / col.sum()

 def recall(self, label, confusion\_matrix):

 row = confusion\_matrix[label, :]

 return confusion\_matrix[label, label] / row.sum()

 def evaluate(self, data, labels):

 corrects, wrongs = 0, 0

 for i in range(len(data)):

 res = self.run(data[i])

 res\_max = res.argmax()

 if res\_max == labels[i]:

 corrects += 1

 else:

 wrongs += 1

 return corrects, wrongs

 def save\_w(self, path\_to\_save):

 with open(path\_to\_save, "bw") as fh:

 data = (self.wih,

 self.who)

 pickle.dump(data, fh)

 def load\_w(self, path\_to\_load):

 with open( path\_to\_load, "br") as fh:

 data = pickle.load(fh)

 self.wih = data[0]

 self.who = data[1]

# 9. ПРИЛОЖЕНИЕ Б

(Обязательное)

Алгоритм программы

Изм.

Лист

№ документа

Подпись

Дата

Лист

1

ГУИР.507606.016

Разраб.

Ярош

Провер.

Т. Контр.

Михалькевич

Н. Контр.

Утв.

Алгорит обучения НС

Литера

Листов

1

ПИКС, гр. 814303

Реценз.

