# ВВЕДЕНИЕ

Классификация и архивирование инженерных работ облегчает повторное использование деталей в дальнейшем. Было подсчитано, что 75% проектных работ состоит из базового дизайна корпуса, который является вариациями и усовершенствованиями каких-либо предыдущих проектов.

Текущие каталоги описывают детали по различным атрибутам, таким как производитель, серия продуктов и просто номера моделей. Эти атрибуты требуют ручной записи и уточнения, и никак не описывают строение детали. Именно поэтому работы в областях компьютерного зрения, графики и др. стремятся автоматизировать процесс классификации деталей.

В этой работе исследуются методы распознавания и классификации паттернов для последующей идентификации и определения в базах данных САПР. Основная задача – улучшить текущие методы классификации и облегчить для инженеров поиск и повторное использование, тем самым улучшая управление жизненным циклом продукта, а также ускоряет разработку и проектирование продукта.

Для ее достижения определено несколько этапов – необходимо определить объект, над которым будут производиться операции, в данном случае – файл 3D модели определенного формата. Также необходимо определить, к каким классам будет привязан алгоритм, и в конце определить, на какой именно системе будет базироваться алгоритм. В результате необходимо получить решение, посредством которого можно разработать алгоритм, определяющий класс детали.

Оглавление

[ВВЕДЕНИЕ 1](#_Toc71558598)

[1. Аналитический анализ алгоритма 3](#_Toc71558599)

[1.1. Форматы 3D моделей 3](#_Toc71558600)

[1.1.1. Содержимое 3D моделей 3](#_Toc71558601)

[1.1.2. Хранение 3D моделей 6](#_Toc71558602)

[1.1.3. Основные форматы 9](#_Toc71558603)

[1.1.4. Вывод по подразделу 13](#_Toc71558604)

[1.2. Классификация изделий 14](#_Toc71558605)

[1.3. Алгоритмы классификации 17](#_Toc71558606)

[1.3.1. Классификация на основе пошагового построения 17](#_Toc71558607)

[1.3.2. Классификация на основе методов компьютерного зрения 18](#_Toc71558608)

[1.3.3. Классификация на основе графа Риба 21](#_Toc71558609)

[1.3.4. Классификация на основе вокселей 22](#_Toc71558610)

[1.4. Выбор нейросети и ее параметров 24](#_Toc71558611)

[1.4.1. Выбор входных и выходных данных 24](#_Toc71558612)

[1.4.2. Архитектуры нейросетей 24](#_Toc71558613)

[1.4.3. Вывод по разделу 30](#_Toc71558614)

[2. Алгоритм? 31](#_Toc71558615)

[2.1. Выбор алгоритма вокселизации 31](#_Toc71558616)

[2.2. Датасет 34](#_Toc71558617)

[2.3. Выбор нейросети 37](#_Toc71558618)

[2.4. Анализ ее работы/анализ результата 37](#_Toc71558619)

[2.5. Вывод 37](#_Toc71558620)

[ЗАКЛЮЧЕНИЕ 38](#_Toc71558621)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 40](#_Toc71558622)

# Аналитический анализ алгоритма

## Форматы 3D моделей

Изначально стоит определить, что именно подразумевается под понятием “3D модель”. 3D-модель – это объемная фигура в пространстве, создаваемая в специальной программе. За основу, как правило, принимаются чертежи, фотографии, рисунки и подробные описания, опираясь на которые, специалисты и создают виртуальную модель.

Прежде чем анализировать способы классификации и варианты алгоритмов, стоит определить, 3D модели какого именно формата мы будем использовать в дальнейшем. В этом разделе рассматриваются основные форматы представления 3D моделей, их достоинства и недостатки.

### Содержимое 3D моделей

Прежде чем говорит о кодировании форматов и их принципиальных достоинствах, стоит определить особенности 3D форматов. В этом разделе описываются общие особенности и содержимое форматов.

1. Кодирование геометрии моделей

Основное, что существует у 3D модели любого формата – ее геометрия, т. е. общее строение фигуры, положение ее граней и вершин [1]. Это является базисом, который в дальнейшем уточняется с помощью различных элементов.

Существует три принципиальных метода кодирования информации о геометрии поверхности, каждый из которых имеет свои преимущества и недостатки. Это аппроксимирующая сетка (англ. approximate mesh), точная сетка (англ. precise mesh) и конструктивная блочная геометрия (англ. constructive solid geometry (CSG)).

В случае кодировки аппроксимирующей сеткой поверхность модели представляется сеткой многоугольников (или полигонов), чаще всего это треугольники, так как это наименьшая 2D фигура, с которой можно смоделировать форму (рисунок 1.1).

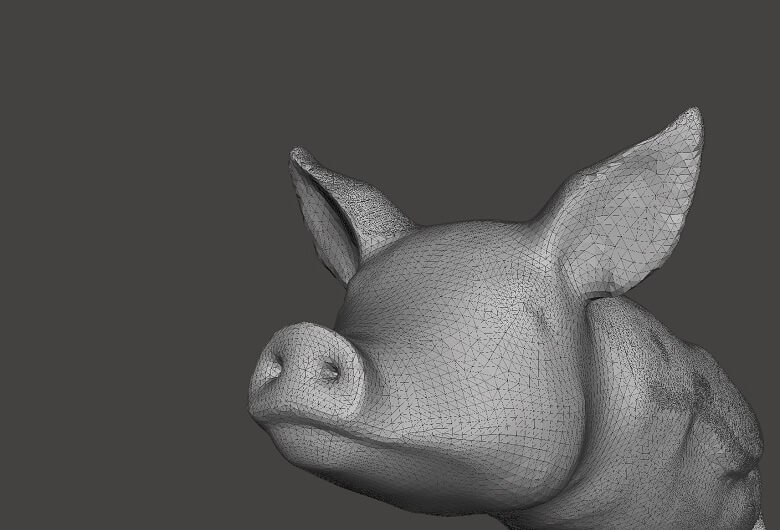


Рисунок 1.1 – Пример представления 3D модели

Вершины покрывающих многоугольников и вектор нормали сохраняются в файле, это и есть аппроксимация геометрии поверхности модели (рисунок 1.2)

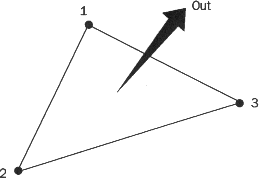
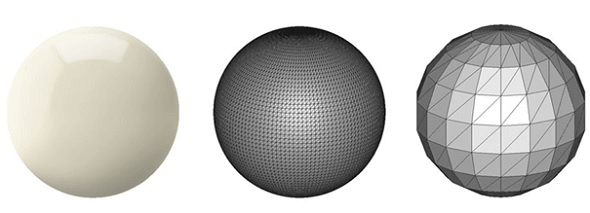


Рисунок 1.2 – Единица аппроксимирующей сетки

Процесс покрытия поверхности многоугольниками называется тесселяция, а форматы файлов – тесселированными. Чем больше многоугольников используется для представления фигуры, тем точнее получается аппроксимированная модель (рисунок 1.3).

Рисунок 1.3 – Аппроксимированная модель при различных уровнях точности

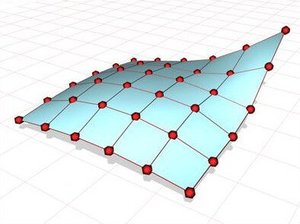
Вариант представления моделей в виде мелкой сетки позволяет добиться максимальной точности, используя функции Non-Uniform Rational B-Spline (или NURBS) вместо полигонов (рисунок 1.4). Эти параметрические поверхности состоят из небольшого количества взвешенных контрольных точек и набора параметров, называемых узлами. На основе узлов поверхность может быть вычислена математически путем плавной интерполяции по контрольным точкам.

Рисунок 1.4 – Пример патчей NURBS

Эти поверхности выглядят гладкими в любом масштабе и могут воспроизводить геометрию поверхности небольшой части 3D модели. Однако всегда есть компромисс. Хотя мелкая сетка точна при любом разрешении, они отрисовываются медленнее, и их следует избегать в приложениях, где важна быстрая визуализация.

Последний вариант представления моделей – конструктивная блочная геометрия. Для этого варианта не используются сетки, а формы создаются путем простейших операций (сложение и вычитание) примитивных форм (кубы, сферы и т. д.). Большинство САПР используют этот принцип.

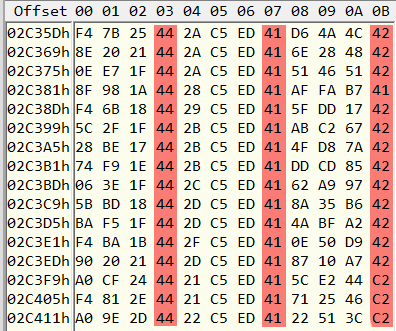
Конструктивная твердотельная геометрия подходит для описания проектируемых 3D моделей и очень удобна для пользователя. Еще одним большим преимуществом является то, что каждый отдельный шаг редактирования (сложение, вычитание, преобразования примитивных форм) сохраняется в этом формате файла 3D. Таким образом, любой шаг можно отменить и повторить в любой момент.

Очевидно, что при конвертации этого формата в формат на основе сетки теряется информация об отдельных этапах редактирования.

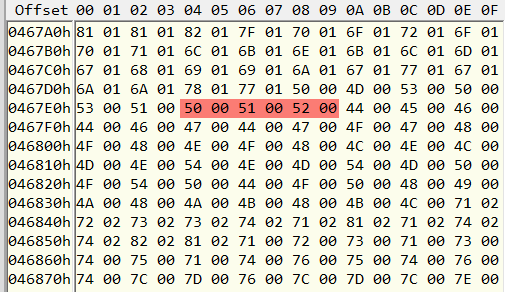
### Хранение 3D моделей

Если взять за основу наиболее часто встречающийся вариант с аппроксимирующей сеткой, то основной объём любого файла с моделью составляют несколько больших таблиц с данными о вершинах, о том, как они соединяются и как на них натягиваются текстуры.

Начать стоит с вершин. Простой список с координатами x, y, z в шестнадцатеричном формате показан на рисунке 1.7.

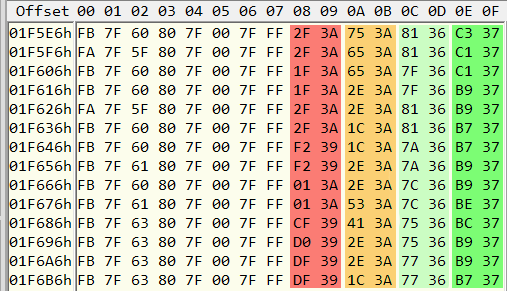
Рис 1.7 – Таблица координат x, y, z вершин

Так как чаще всего координаты хранятся в виде 32-битных плавающих чисел, их легко распознать внутри файла по повторяющимся через 4 байта цифрам в диапазоне 40–45, или для отрицательных чисел C0-C5. Конечно, могут встречаться и иные байты, но вероятность такого невелика. Так происходит потому, что диапазон координат 3D модели небольшой с точки зрения порядка, а порядок как раз хранится в старшем байте. Далее, нужна таблица, где указано, в каком порядке вершины соединяются в треугольники. Пример такой таблицы показан на рисунке 1.8.

Рисунок 1.8 – Пример хранения треугольников, моделирующих форму

Это 16-битные номера вершин, группами по три. Так как в моделях обычно не более нескольких сотен вершин, числа эти маленькие, и такую таблицу тоже легко видно визуально. В данном примере выделен один из треугольников, состоящий из вершин с номерами 50, 51 и 52.

И третья – таблица текстурных координат, чтобы привязать вершины к плоской текстуре, которую нужно на них натянуть, показана на рисунке 1.9.

Рисунок 1.9 – Таблица соответствия текстуры и вершин

Координаты x и y в пределах текстуры приводятся к диапазону от 0 до 1, и, если текстура имеет размер 2048x2048 или 4096x4096, нет смысла в высокой точности. Поэтому они чаще всего хранятся как плавающие числа с половинной точностью, 16 битные. Старший байт у них получается чуть больше 0x30, изредка доходя до 0x40 или чуть больше. Здесь красным и оранжевым выделены текстурные координаты, зеленым и светло-зеленым – координаты на карте освещения.

### Основные форматы

Форматы можно разделить по применимости на 2 группы – универсальные, которые используются во всех САПР, и собственные форматы, которые используются только в определенных САПР (к примеру, DWG, использующийся программой Autodesk).

Универсальные форматы служат «промежуточным звеном», позволяющим различным САПР использовать одну и ту же модель. Тогда, после создания модели, она преобразовывается в универсальный формат, возможно, как будет рассмотрено далее, с другим методом хранения/кодирования, который позволяет читать и редактировать модель в различных САПР.

В этом разделе рассматриваются наиболее часто встречающиеся форматы и также определяется, насколько хорошо они подходят для задачи классификации моделей. Основные критерии для определения подходящих форматов можно определить так: во-первых, они должны кодировать значения в удобном виде (в идеале – текстовом). Это позволит удобно их читать, что может понадобиться при разработке алгоритма. Также они должны читаться, а главное, создаваться в различных САПР, и хранить минимально необходимую информацию – нет нужды в информации о текстуре или о среде, так как для определения класса эти параметры значения не имеют.

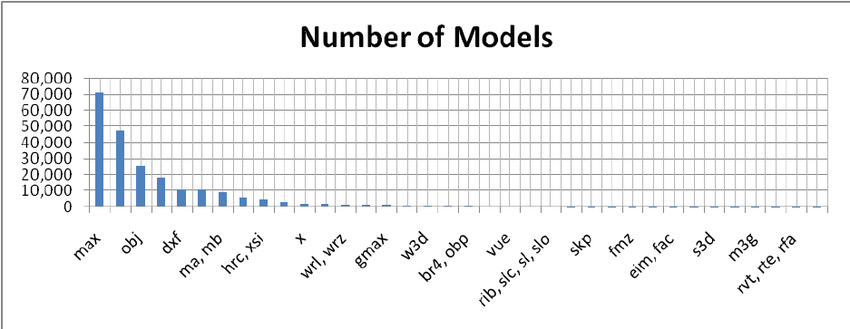
Сначала необходимо определить, какие именно форматы надо рассматривать. В этой работе выборка будет основана на общедоступных статистических данных загрузок для каждого формата. На рисунке 1.10 показана статистика количества моделей для основных форматов.

Рисунок 1.10 – Статистика количества моделей [2]

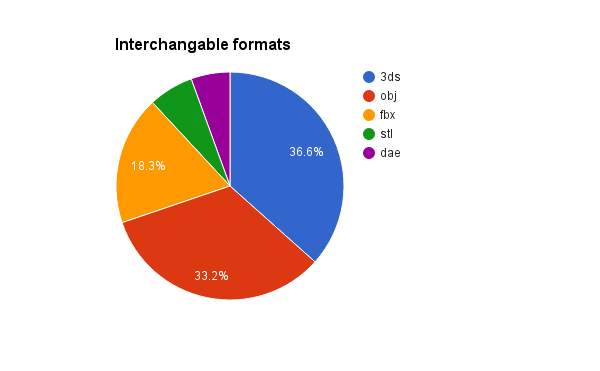
Далее необходимо исключить редко встречающиеся и не-универсальные форматы, получим диаграмму, представленную на рисунке 1.11.

Рисунок 1.11 – Статистика скачиваний для универсальных форматов

Далее рассмотрим каждый формат и определим, насколько он подходит для решения поставленной задачи, исходя из ранее обозначенных критериев.

1. 3ds

Формат файла 3ds является основным форматом программного обеспечения AutoDesk 3ds Max. Это двоичный формат, состоящий из блоков, содержащих различную информацию. Блоки содержат идентификацию, указывающую, какая информация там сохраняется и смещение до следующего фрагмента [3]. Таким образом, программное обеспечение, которое не поддерживает определенные свойства рендеринга, может просто игнорировать их.

Формат файла 3ds поддерживает: геометрию в форме вершин / граней и параметрических поверхностей, текстуры, физические свойства материала, преобразования, информацию о камере и источники света.

1. Obj

Формат файла obj – это текстовый открытый формат файла, разработанный Wavefront Technologies (сейчас Alias ​| Wavefront). Формат был принят другими поставщиками приложений для 3D графики, и некоторые из них могут импортировать / экспортировать. Формат файла obj состоит из ряда строк, каждая из которых содержит ключ и различные значения. Ключ в каждой строке указывает информацию о типе, которой следует следовать. По этой причине формат файла obj не требует заголовка. В таблице 1.1 показаны основные ключи, используемые при кодировании.

Таблица 1.1. Файловые ключи obj

|  |  |
| --- | --- |
| Ключ | Описание |
| # | Комментарий |
| v | Вершина |
| l | Линия |
| f | Грань |
| vt | Координата текстуры |
| vn | Нормаль |
| g | Группа |
| … | … |

Простой пример, содержащий одно лицо, будет выглядеть так:

v 0.0 0.0 0.0

v 0.0 1.0 0.0

v 1.0 0.0 0.0

f 1 2 3

Формат файла obj поддерживает: геометрию в форме вершин / ребер / граней и параметрические поверхности, нормали вершин, текстуры, свойства материалов и группы.

1. fbx

FBX может храниться на диске в виде бинарных или ASCII данных, FBX SDK поддерживает оба формата. Оба формата не задокументированы, однако формат ASCII имеет древовидную структуру с чёткими обозначениями идентификаторов, в то время как бинарный формат не задокументирован, но есть неофициальная спецификация, представленная Blender Foundation.

Формат файла fbx поддерживает: геометрию в форме вершин / граней и параметрических поверхностей, текстуры, физические свойства материала, преобразования, информацию о камере и источники света.

1. stl

STL (от англ. stereolithography) – формат файла, широко используемый для хранения трёхмерных моделей объектов для использования в аддитивных технологиях. Информация об объекте хранится как список треугольных граней, которые описывают его поверхность, и их нормалей. STL-файл может быть текстовым (ASCII) или двоичным [4]. Свое название получил от сокращения термина «Stereolithography», поскольку изначально применялся именно в этой технологии трехмерной печати.

Файл типа ASCII STL начинается со строки:

*solid name*

где name – необязательная строка (но, если name опущено, всё равно должен быть пробел после solid). Файл продолжается произвольным числом треугольников, описываемых следующим способом:

*facet normal ni nj nk*

*outer loop*

*vertex v1x v1y v1z*

*vertex v2x v2y v2z*

*vertex v3x v3y v3z*

*endloop*

*endfacet*

где каждое n и v – число с плавающей точкой в формате: знак, мантисса, «e», знак, экспонента, например «-2.648000e-002». Файл завершается строкой:

*endsolid name*

Недостатки формата:

* Невысокая точность геометрии (в версии ASCII можно задавать произвольную точность).
* Большой объём файла для сложных моделей.

1. dae

COLLADA – это формат, разработанный для обмена между 3D приложениями. Использует открытый стандарт XML для обмена форматами, которые в противном случае были бы несовместимы.

COLLADA был задуман как промежуточный формат для переноса файлов. Реализована поддержка таких программ, как Maya (используя ColladaMaya); 3ds Max (при помощи ColladaMax); Poser (v.7.0); Lightwave 3D (version 9.5); Cinema 4D (MAXON); Softimage|XSI; Houdini; MeshLab; CityScape, CityEngine, SketchUp, Blender, modo и Strata 3D. Adobe Photoshop с версии CS3.

Игровые движки, такие как Unreal engine, Unity и Torque 3D, также поддерживают этот формат. Библиотека Assimp также поддерживает формат COLLADA.

Формат по принципам использования и возможностям близок к fbx, который уже был рассмотрен ранее.

### Вывод по подразделу

Исходя из поставленных задач и критериев, можно выделить несколько форматов, которые подходят для дальнейшей работы. Во-первых, это 3ds, который является универсальным и достаточно удобным, но хранит излишнюю информацию, в том числе преобразования, информацию о камере и источниках света.

Формат fbx и dae, в свою очередь, также хранят излишнюю информацию, но имеют идентификаторы, которые достаточно удобны при использовании. Главный недостаток форматов – закрытый исходный код.

Формат stl крайне удобен тем, что имеется возможность задавать точность моделей, что позволяет уменьшить размер файла, но для алгоритма метод хранения – набор треугольников – может быть неудобен.

Формат obj имеет достаточно удобное строение файла, идентификаторы для вершин, граней и т. д., а также хранит минимально необходимую информацию об объекте.

В итоге можно сказать, что лучшими форматами для работы являются obj, как основной, и stl, как альтернативный.

## Классификация изделий

Далее после того, как определен формат хранения модели, необходимо определить, чем руководствоваться при их классификации. В этой работе была выбрана Единая система конструкторской документации (далее ЕСКД), а именно «Общероссийский классификатор изделий и конструкторских документов ОК 012–93» [5].

ЕСКД, согласно ГОСТ 2.001–2013 – комплекс государственных стандартов, устанавливающих взаимосвязанные правила, требования и нормы по разработке, оформлению и обращению конструкторской документации, разрабатываемой и применяемой на всех стадиях жизненного цикла изделия (при проектировании, разработке, изготовлении, контроле, приёмке, эксплуатации, ремонте, утилизации) [6]. Основное назначение ЕСКД – установление единых опциональных правил, требований и норм выполнения, оформления и обращения конструкторской документации, именно поэтому для задачи классификации была выбрана именно эта система.

Классификатор ЕСКД представляет собой систематизированный свод наименований классификационных группировок объектов классификации – изделий основного и вспомогательного производства всех отраслей народного хозяйства, общетехнических документов и их кодов и является составной частью Единой системы классификации и кодирования технико-экономической информации (ЕСКК ТЭИ).

В Классификатор ЕСКД включены классификационные характеристики изделий – деталей, сборочных единиц, комплектов, комплексов (ГОСТ 2.101-68 ""ЕСКД. Виды изделий""), на которые разработана и разрабатывается конструкторская документация по ЕСКД, в том числе стандартные изделия, а также общетехнические документы (нормы, правила, требования, методы и т. д.) на изделия, входящие в Классификатор ЕСКД.

Обозначение изделий и конструкторских документов устанавливается по ГОСТ 2.201–80 "ЕСКД. Обозначение изделий и конструкторских документов". Обозначение основного конструкторского документа (чертежа детали или спецификации) включает: код организации-разработчика (четыре знака), код классификационной характеристики (шесть знаков), код порядкового регистрационного номера (три знака) [7].

Классификационная характеристика является основной частью обозначения изделия и его конструкторского документа. Код классификационной характеристики изделия присваивается по Классификатору ЕСКД и представляет собой шестизначное число, последовательно обозначающее класс (первые два знака), подкласс, группу, подгруппу, вид (по одному знаку). Структура обозначения кода классификационной характеристики имеет вид:

XX X X X X

1,2 цифры – Класс

3 цифра – Подкласс

4 цифра – Группа

5 цифра – Подгруппа

6 цифра – Вид

Для данной работы стоит заострить внимание на классах 71-75, и их производных. Эти классы представлены в таблице 2.1.

Таблица 2.1. Используемые классы изделий.

|  |  |
| --- | --- |
| Обозначение класса | Подробная информация |
| 71 | Детали – тела вращения типа колец, дисков, шкивов, блоков, стержней, втулок, стаканов, колонок, валов, осей, штоков, шпинделей и др. |
| 72 | Детали – тела вращения с элементами зубчатого зацепления; трубы, шланги, проволочки, разрезные, сектора, сегменты; изогнутые из листов, полос и лент; аэрогидродинамические; корпусные, опорные; емкостные; подшипников. |
| 73 | Детали – не тела вращения корпусные, опорные, емкостные. |
| 74 | Детали – не тела вращения плоскостные; рычажные, грузовые, тяговые; аэрогидродинамические; изогнутые из листов, полос и лент; профильные; трубы. |
| 75 | Детали – тела вращения и (или) не тела вращения, кулачковые, карданные, с элементами зацепления, арматуры, санитарно-технические, разветвленные, пружинные, ручки, уплотнительные, отсчетные, пояснительные, маркировочные, защитные, посуды, оптические, электрорадиоэлектронные, крепежные. |

Очевидно, что чем точнее необходимо классифицировать изделие, тем сложнее и точнее необходимо сделать любой алгоритм классификации, и тем больше вероятность ошибки. Поэтому было принято решения для первых версий алгоритма определять только класс изделия.

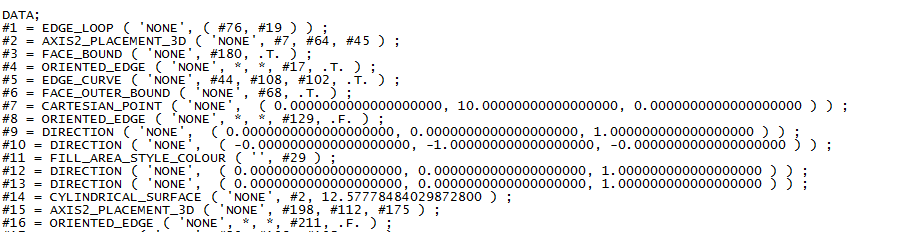
## Алгоритмы классификации

Для классификации моделей применяется большое количество алгоритмов, и далеко не все подходят для задачи классификации инженерных изделий по ГОСТ. Ниже рассмотрены возможные алгоритмы и их применимость.

### Классификация на основе пошагового построения

Как уже было сказано ранее, в большинстве САПР для создания изделий используются конструктивная твердотельная геометрия, которая позволяет построить модель сложной формы за счет действий с моделями простейших форм. Первый вариант классификации моделей – определять шаги построения моделей, а затем на их основе выделять характеризующие для каждого класса черты (к примеру, для деталей вращения – построение окружности и выдавливание по ней).

Этот подход выделяется тем, что не требует никаких особенных вычислительных мощностей, а сами пошаговые операции вполне возможно получить, к примеру, из САПР SolidWorks с помощь файла STEP (Рисунок 3.1)

Рисунок 3.1 – Представление файла STEP в текстовом редакторе

Но этот подход имеет несколько существенных минусов. Во-первых, он не является универсальным, так как для большинства классов невозможно определить какие-либо характерные шаги при построении. Во-вторых, высока вероятность ошибок, так как при большом количестве шагов необходимо учитывать не просто какой-либо один шаг, а их набор.

В итоге было принято решения разрабатывать алгоритм, не используя этот подход из-за предположительно невысокой точности и крайне высокой сложности разработки.

### Классификация на основе методов компьютерного зрения

Для извлечения информации об объекте можно воспользоваться методами, активно использующимися в компьютерном зрении. Это позволит уменьшить количество информации об объекте до минимально необходимого, и соотносить только информацию, полученную после обработки 3D моделей.

Сначала стоит выделить основные понятия, и далее перейти к алгоритму. В компьютерном зрении существует несколько методов извлечения информации из модели. Первый – использование контуров, по которым в дальнейшем будут определяться параметры. Для тех же целей используются шаблоны.

В первом случае контур составляется как место быстрого изменения функции интенсивности изображения. Но у этого метода есть существенные ограничения, которые не позволяют использовать их в этом случае. Из ограничений контурного анализа стоит выделить два – неоднозначность определения контура, к примеру, при наличии помех или одинаковой яркости объекта и фона.

Для шаблонов также стоит выделить ограничения, которые, ввиду поставленной задачи, являются критическими: шаблон зависит от масштаба, углов обзора, поворота картинки (для 2D изображений) и наличия физических помех. Также возможны ложные срабатывания алгоритма, когда искомого объекта на нет, но имеются общие детали у шаблона и области на входном изображении.

Тогда выделим третий метод, которым технически можно воспользоваться для решения поставленной задачи – дескрипторы и особые точки.

Особая точка – это точка изображения, окрестность которой *o(m)* можно отличить от окрестности любой другой точки изображения *o(n)* в некоторой другой окрестности особой точки *o2(m)*. В качестве окрестности точки изображения для большинства алгоритмов берётся прямоугольное окно, составляющее размер *KxK* пикселей. Процесс определения особых точек достигается путем использования детектора и дескриптора.

В 1992 *Haralick* и *Shapiro* выделили следующие требования к особым точкам в виде следующих свойств:

•        *Отличимость (distinctness)* – особая точка должна явно выделяться на фоне и быть отличимой (уникальной) в своей окрестности.

•        *Инвариантность (invariance)* – определение особой точки должно быть независимо к аффинным преобразованиям.

•        *Стабильность (stability)* – определение особой точки должно быть устойчиво к шумам и ошибкам.

•        *Уникальность (uniqueness)* – кроме локальной отличимости, особая точка должна обладать глобальной уникальностью для улучшения различимости повторяющихся паттернов.

•        *Интерпретируемость (interpretability)* – особые точки должны определяться так, чтобы их можно было использовать для анализа соответствий и выявления интерпретируемой информации из изображения.

Детектор – это метод извлечения особых точек из изображения. Детектор обеспечивает инвариантность нахождения одних и тех же особых точек относительно преобразований изображений.

Дескриптор – производит описание найденных ключевых точек, оценивая их позиции через описание окружающих областей. В свою очередь, дескрипторы должны обеспечивать инвариантность нахождения соответствия между особыми точками относительно преобразований изображений. Ищутся особенности, которые более устойчивые к повороту, масштабированию и т. д.

Дескриптор – вектор, описывающий структуру окрестностей особых точек. Как правило, эти вектора формируются на основе набора значений первых и вторых производных изображения в точке.

Дескрипторы должны быть: специфичны (отличаем разные точки), локальны (зависеть только от небольшой окрестности), инвариантны (к искажениям\изменениям освещенности), просты в вычислении.

Дескрипторы, основанные на градиенте яркости, инвариантны к сдвигу яркости.

При описании дескриптора в простой окрестности имеются следующие недостатки: небольшой сдвиг приводит к существенному изменению, не инвариантен к повороту.

Для представления изображения можно использовать гистограммы. Это распределение каких-то признаков (цвет, текстура, глубина). Чаще всего используются гистограммы ориентированных градиентов.

*SIFT:* для детекции используется детектор *DOG* (разница гауссиана). Ориентация идет по градиентам (что позволяет построить гистограмму). Устойчив к изменению освещенности и небольшим сдвигам.

*SURF* – быстрая аппроксимация идей *SIFT*. Решает две задачи – поиск особых точек и создание дескриптора. Обнаружение особых точек основано на вычислении детерминанта матрицы Гессе. Особые точки представляют из себя круги (диаметр показывает масштаб точки). Показаны линии градиента яркости. Дескриптор имеет 64 или 128 чисел описания ключевой точки.

Для нахождения особой точки *SURF* пробегает по пикселям изображения и ищет максимум гессиана (матрица Гессе). Гессиан зависит только от перепада яркости и инвариантен относительно вращения, но не инвариантен к масштабу. Поэтому нужно перебирать разные масштабы фильтров и поочередно применять их к пикселю.

*SURF* разбивает все множество масштабов на октавы. Каждая октава содержит в себе 4 фильтра. Первая октава применяется для каждого второго пикселя, вторая – для каждого четвертого и т. д. (чтобы вычисления производились быстрее).

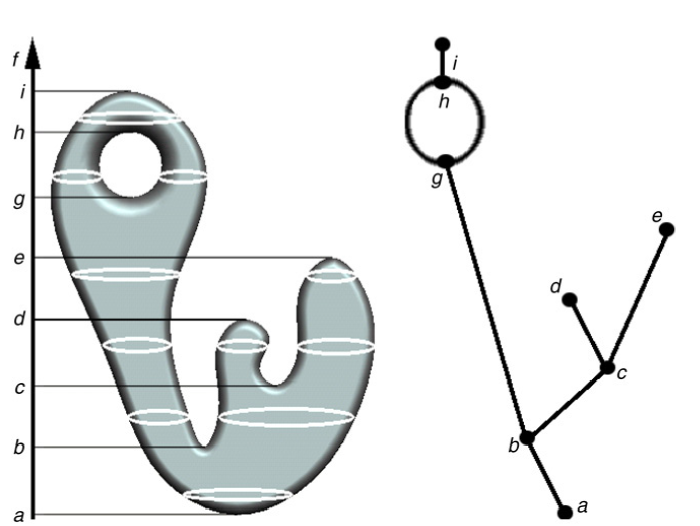
Алгоритм *HOG:* рассматриваем каждый пиксель и его окрестность. Находим направление, в котором цвет становится темнее (градиент), таким образом изображение заменяется на набор векторов, показывающих поток от света к темноте. Далее выделяются области и в них находятся средние направления.

К сожалению, применение типовых методов к 3D моделям достаточно сложно относительно других, специальных методов, описанных далее, из-за чего от их применения решено было отказаться.

### Классификация на основе графа Риба

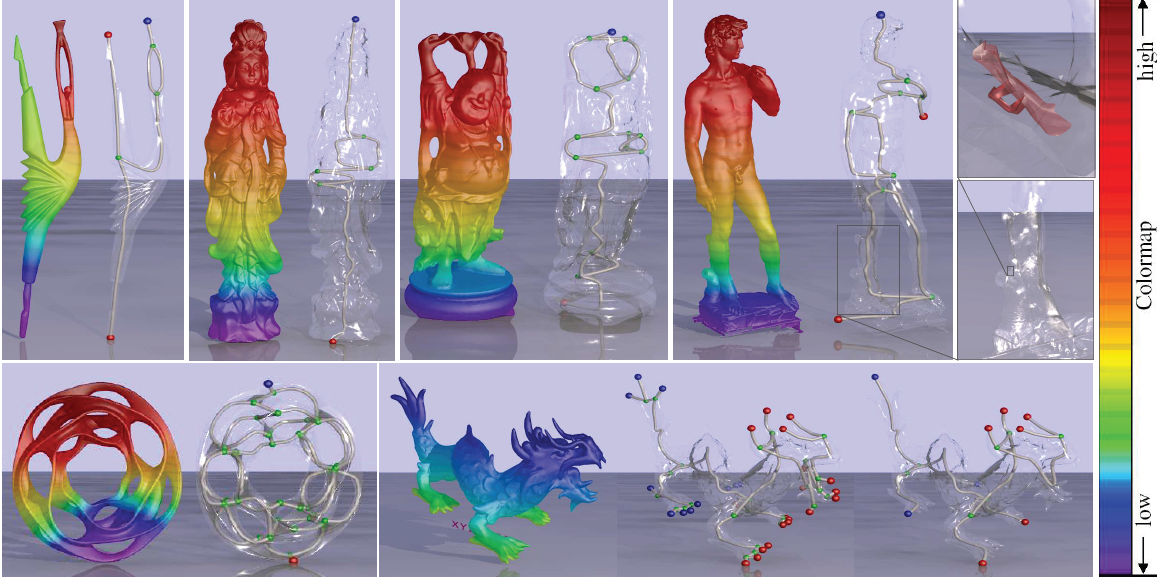
Для анализа форм и получения основной топологической информации об объекте также можно использовать дескриптор формы, который был изобретен Жоржем Рибом в его работе, датированной 1946 годом.

Граф Риба передает топологическую информацию, относящуюся к наборам уровней функции, определенной на форме. Пример представления графа Риба показан на рисунке 3.2. Буквы от a до f обозначают соответствие между критическими значениями и узлами графа [8].

Рисунок 3.2 – Представление Графа Риба относительно высоты.

Вершиной графа Риба называют точку, отвечающую особому слою, содержащему критическую точку (т. е. точку функции, дифференциал в которой равен нулю). Вершину графа Риба назовём концевой, если она является концом ровно одного ребра графа. Все остальные вершины назовём внутренними.

Для построения графа Риба существуют эффективные алгоритмы [9], [10], и данный метод позволяет достаточно точно описывать различные сложные модели (пример показан в работе Valerio Pascucci и др., рисунок 3.3).

Рисунок 3.3 – Пример использования графа Риба для представления объектов сложной формы

Но без методов соотношения таких графов данный способ классификации для решения данной задачи неприменим. И, как будет сказано далее, использование графов Риба для представления модели является избыточным, поэтому от этого метода решено было отказаться.

### Классификация на основе вокселей

Исходя из описанной выше проблемы – необходимости соотносить объекты, можно подобрать другой метод представления объектов. Для этого объекты можно поделить на участки, каждый из которых далее будет сравниваться с исходным и будет определяться, к какому классу принадлежит объект с данной 3D моделью. Для этой задачи лучше всего подойдет вокселизация.

Воксель (объёмный (англ. volumetric) и пиксель (англ. pixel)) – элемент объёмного изображения, содержащий значение элемента растра в трёхмерном пространстве [11]. Воксели являются аналогами двумерных пикселей для трёхмерного пространства.

Воксельные модели часто используются для визуализации и анализа медицинской и научной информации, так как они позволяют получить упрощенной представление модели.

Для поставленной задачи обрабатываемые модели можно поделить на равное число вокселей, что позволить избавиться от проблемы с масштабом. Далее необходимо с помощью алгоритмов соотнести каждый воксель и по совокупному результату определить класс модели. Для поставленной задачи лучше всего подойдет нейросеть, так как входные данные заданы четко, известно точное количество объектов на входе (т. к. мы сами делим модель на воксели), и известен необходимый результат – вероятность принадлежности объекта к тому или иному классу деталей. Как пример можно воспользоваться нейросетью, представленной пользователем *EJ Shim* (см. рисунок 3.3, [12]). Данная нейросеть использует библиотеку моделей, по которым определяется предмет.

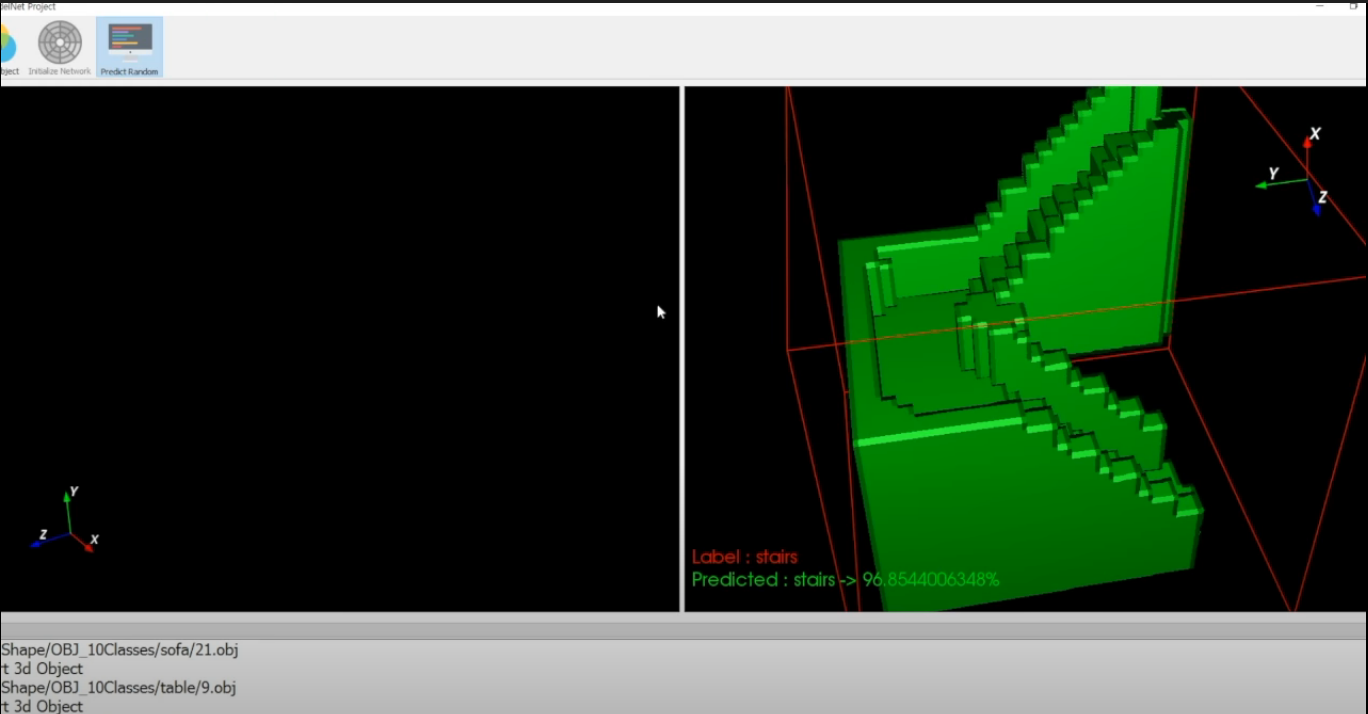


Рисунок 3.3 – Классификация моделей с использованием вокселей

Следующим шагом будет определение алгоритма создания воксель-моделей. Для этого также найдены существуют алгоритмы [13], более подробно создание воксель-моделей в этой работе рассматриваться не будет.

## Выбор нейросети и ее параметров

Далее необходим еще один шаг – нужно определить структуру нейросети. Для этого определим параметры и задачу нейросети, и исходя из них выберем лучший вариант реализации.

### Выбор входных и выходных данных

Первое, от чего отталкиваются при создании нейросети – выбор и определение количества входных данных, а также необходимый результат. Как уже было сказано ранее, в результате необходимо получить вероятность принадлежности объекта к некоторому определенному классу (к которому вероятность максимальна). Определить это необходимо исходя из некоторого набора вокселей, их количество будет неизменно, так как разделение будет производиться автоматически на нужное число по каждой из осей.

Проблема, которую можно встретить при разработке нейросети – необходимо ли определять ориентацию детали в пространстве, и без экспериментальной проверки узнать это невозможно.

### Архитектуры нейросетей

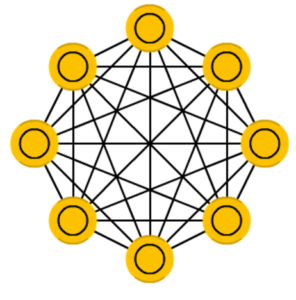
Существует несколько элементарных конфигураций нейросетей, каждая из который выполняет свои задачи.

1. **Нейронные сети прямого распространения** (feed forward neural networks, FF или FFNN) **и перцептроны** (perceptrons, P)- наиболее простой вариант нейросетей, они передают информацию сразу от входа к выходу (рисунок 4.1) [14].

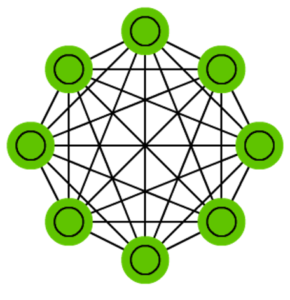
Рисунок 4.1 – *FFNN*

Обычно нейронные сети описываются как системы из нескольких слоев, в которых нейроны одного слоя никак не связаны между собой, а соседние слои полностью связаны.

1. Сети радиально-базисных функций (radial basis function, RBF) – достаточно схожи с НСПР, но функциями активации являются радиально-базисные функции.
2. **Нейронная сеть Хопфилда** (Hopfield network, HN) – это полносвязная нейронная сеть с симметричной матрицей связей. Во время получения входных данных каждый узел является входом, в процессе обучения он становится скрытым, а затем становится выходом (рисунок 4.2) [15].

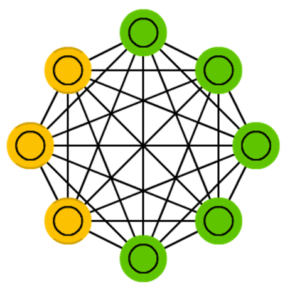
Рисунок 4.2 – HN

1. **Цепи Маркова** (Markov chains, MC или discrete time Markov Chains, DTMC) – это предшественники машин Больцмана (*BM*) и сетей Хопфилда (*HN*) (рисунок 4.3) [16].

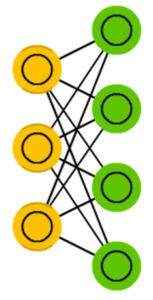
Рисунок 4.3 – DTMC

Каждое следующее состояние зависит только от предыдущего. Хотя на самом деле цепи Маркова не являются НС, они весьма похожи. Также цепи Маркова не обязательно полносвязны.

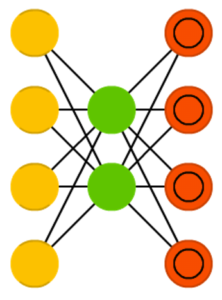
1. **Машина Больцмана** (Boltzmann machine, BM) очень похожа на сеть Хопфилда, но в ней некоторые нейроны помечены как входные, а некоторые – как скрытые (рисунок 4.4). Входные нейроны в дальнейшем становятся выходными [17].

Рисунок 4.4 – BM 

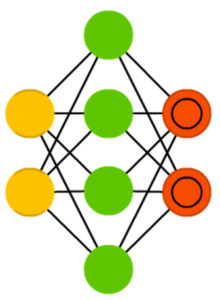
1. **Ограниченная машина Больцмана** (restricted Boltzmann machine, RBM) похожа на машину Больцмана и, следовательно, на сеть Хопфилда. Единственной разницей является её ограниченность. В ней нейроны одного типа не связаны между собой (рисунок 4.5).

Рисунок 4.5 – RBM

1. **Автокодировщик** (autoencoder, AE) похож на *FFNN*, так как это скорее другой способ использования *FFNN*, нежели фундаментально другая архитектура. Основной идеей является автоматическое информации [18]. По форме структура напоминает песочные часы, причем входные и выходные слои симметричны (рисунок 4.6).

Рисунок 4.6 – AE

1. **Разреженный автокодировщик** (sparse autoencoder, SAE) по своей логике является противоположностью обычного. Вместо того, чтобы обучать сеть отображать информацию в меньшем «объёме» узлов, мы увеличиваем их количество. Вместо того, чтобы сужаться к центру, сеть там раздувается (рисунок 4.7) [19]. Сети такого типа полезны для работы с большим количеством мелких свойств набора данных.

Рисунок 4.7 – SAE

1. **Вариационные автокодировщики**(variational autoencoder, VAE) обладают схожей с *AE* архитектурой, но обучают их иному: приближению вероятностного распределения входных образцов [20]. В общем случае нейросеть учитывает влияние нейронов, т. е. если что-то изменилось на одном участке и на другом, то эти события не обязательно должны быть связаны (рисунок 4.8).

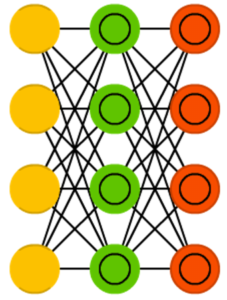
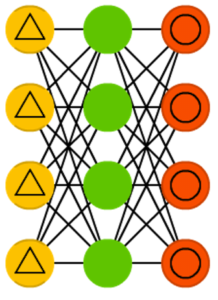


Рисунок 4.8 – *V*AE

1. Шумоподавляющие автокодировщики*(denoising autoencoder, DAE)* – это *AE*, в которые входные данные подаются в зашумленном состоянии. Ошибку мы вычисляем при сравнении выходных данных с зашумленными. Благодаря этому сеть учится обращать внимание на более широкие свойства, поскольку маленькие могут изменяться вместе с шумом [21].

 Рисунок 4.9 – *DAE*

1. **Сеть типа «deep belief»** (deep belief networks, DBN) –это тип архитектуры, представляющий собой соединение RBM или VAE. Такие сети обучаются поблочно, причём каждому блоку требуется лишь уметь закодировать предыдущий (рисунок 4.10) [22]. Такая техника называется «жадным обучением», и заключается в выборе локальных оптимальных решений, не гарантирующих оптимальный конечный результат.

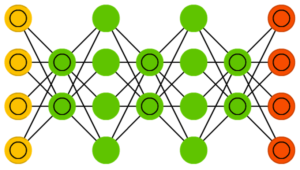


Рисунок 4.10 – DBN

1. **Свёрточные нейронные сети** (convolutional neural networks, CNN)**и глубинные свёрточные нейронные сети** (deep convolutional neural networks, DCNN) обычно они используются для обработки изображений, реже для аудио [23], типичным способом применения *CNN* является классификация изображений. Пример работы нейросети можно описать так: на изображении 200×200 сеть считает квадрат размера 20 x 20 (обычно из левого верхнего угла), затем сдвинется на 1 пиксель и считает новый квадрат, и т. д. Эти входные данные затем передаются через свёрточные слои, в которых не все узлы соединены между собой. Эти слои имеют свойство сжиматься с глубиной, причём часто используются степени двойки: 32, 16, 8, 4, 2, 1. На практике к концу *CNN* прикрепляют *FFNN* для дальнейшей обработки данных (рисунок 4.11). Такие сети называются глубинными (*DCNN*).

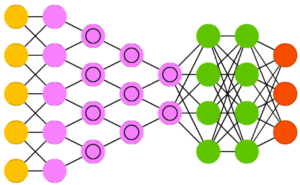


Рисунок 4.11 – *CNN* + *FFNN*

1. **Развёртывающие нейронные сети** (deconvolutional networks, DN), также называемые обратными графическими сетями, являются обратным к свёрточным нейронным сетям (рисунок 4.12).

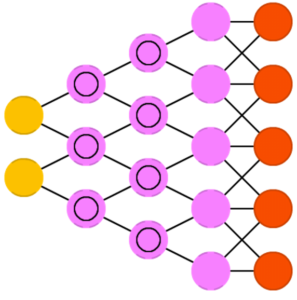


Рисунок 4.12 – DN

### Вывод по разделу

Достаточно очевидно, что для поставленной задачи лучше всего выбрать CNN, или сверточную нейросеть. Действительно, задача определения и классификации изображения очень схожа с задачей классификации 3D моделей, тем более, когда входные данные схожи – вместо пикселей используются воксели.

# Алгоритм?

## Выбор алгоритма вокселизации

Первая задача при обработке моделей- подготовка входных данных для нейросетей. Проблема возникает при неоднородности входных данных, а конкретно- при отличающихся размерах подаваемых моделей. Чтобы избежать ошибок, решено было подготовить алгоритм, который будет вокселизировать модели, ограничиваясь настраиваемым размером, приводя все модели к единообразной матрице значений. Для разработки в качестве основы были выбраны пять уже готовых алгоритмов, и из них выбран подходящий для текущей задачи. Далее будут описаны минусы и плюсы каждого варианта и обоснование выбора итогового. Также будут приведены результаты сравнения скоростей.

Прежде чем переходит к алгоритмам, стоит определить, что из себя представляют воксели и как они хранятся в памяти. Воксели, также как и пиксели, не содержат в себе информации о положении в пространстве. Их координаты вычисляются из трехмерной матрицы, которая моделирует модель. Но, при преобразовании из воксельной модели в .obj, воксели преобразуются в типовую для этого формата модель (к примеру, вершины и грани), что увеличивает общее время работы программы, так как ресурсы тратятся на хранение объекта в памяти и преобразование. Формат .vox разработан для использования в программе *magicavoxel* и предназначен для хранения вокселей, то есть реализует трехмерную матрицу.

Алгоритмы ниже были выбраны с учетом применимости, на вход они принимают файлы в форматах .obj или .stl.

Изначально все найденные алгоритмы можно поделить на 3 группы: первая- на входе получает модель в формате .obj, результат на выходе может варьироваться. Для работы выбран алгоритм *CUDA C mesh voxelizer*. Ключевым преимуществом этого алгоритма стоит выделить использование архитектуры параллельных вычислений CUDA, что в теории должно ускорить обработку. На выходе создается файл в формате .obj или .objvox.

Второй вариант- алгоритмы, использующие в качестве входа файлы в формате .stl, что упрощает их практическое использование, так как файлы в этом формате занимают значительно меньше памяти и проще в строении, а также их можно получить в различных САПР. В этой работе тестировались 2 алгоритма: *VoxSurf* и *Mesh voxelizer.* Основными плюсами были простота реализации и настройки, так как алгоритмы используют один файл и стороннюю библиотеку в первом варианте и один файл во втором. На выходе в первом варианте получается файл .vox, во втором- матрица координат вокселей, которую можно в дальнейшем использовать для создания любых файлов.

Третий вариант- использование библиотеки в программе Matlab. Таких алгоритмов было найдено 2: в первом случае на вход подается модель в формате .obj, на выходе получается трехмерная матрица из bool-переменных. Во втором случае на вход подается .stl модель, а выход- также трехмерная матрица, но в формате int\_8, т. е. целые числа размером не больше 8 бит.

Для выбора итогового варианта использовались следующие критерии: во-первых, их должно быть удобно подстраивать под текущую задачу обработки большого числа моделей. Также, на выходе должны получаться удобные для обработки модели, которые занимают минимум места и хранят минимально необходимый объем информации. Третьим критерием является скорость работы.

Алгоритм, на выходе которого получается матрица вокселей, дает возможность встраивать его в различные программы, но в этой задаче получение сетки недостаточно и его необходимо дорабатывать, в дальнейшем он рассматриваться не будет.

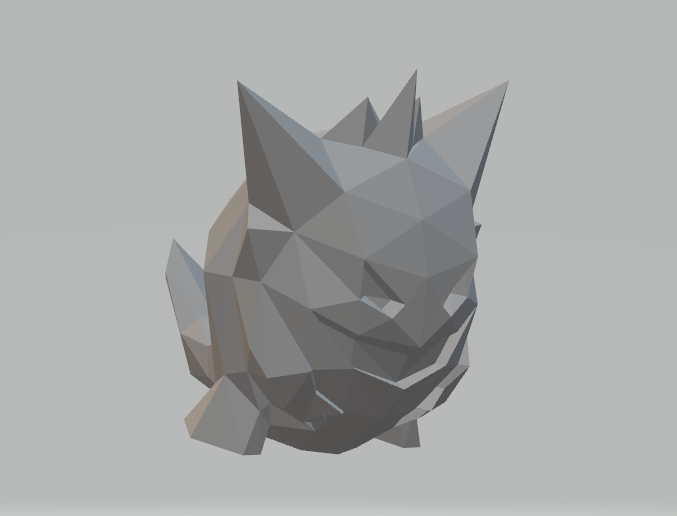
Скорость работы алгоритмов была протестирована на модели, показанной на рисунке 1. Она была выбрана из-за своей сложной структуры, что позволит точнее определить разницу в скорости алгоритмов, которая будет увеличиваться с увеличением сложности модели.

Рисунок 1- Модель для тестирования скорости

Результаты тестирования представлены в таблице 1.

Таблица 1- Сравнение скоростей обработки модели

|  |  |
| --- | --- |
| Алгоритм | Скорость обработки одной модели |
| *CUDA C mesh voxelizer* | 1436 мс |
| *VoxSurf* | 508 мс |
| *Matlab (3D Voxelizer)* | >>1 c |
| *Matlab (Mesh voxelizator)* | 539 мс |

Очевидно преимущество в скорости алгоритмов, основанных на .stl моделях. Также эти алгоритмы удобно использовать для составления датасета, методика подготовки которого будет описана далее.

За основы был взять алгоритм из *Matlab* ввиду легкой реализуемости и изменения, а также из-за понятного кода. Также был подготовлен альтернативный вариант- алгоритм *VoxSurf*.

Для составления датасета оба алгоритма были изменены. Далее улучшение алгоритма будет показано на примере алгоритма для *Matlab*, улучшение *VoxSurf* производилось аналогично.

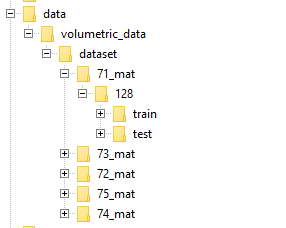
На вход подается папка с файлами (*dirData*), и далее для каждого файла в ней проводится операция вокселизации и записи в файл. В итоге для каждого класса получается набор файлов, представляющих собой сохраненную переменную *Matlab*, из которых в дальнейшем формируется датасет.

|  |
| --- |
| dirData = dir('75/\*.stl'); |
| for index = 1:58 |
| filename1 = convertStringsToChars(string(dirData(index).folder) + '\' + string(dirData(index).name)); |
| [OUTPUTgrid] = VOXELISE(32,32,32,filename1,'xyz'); |
| OUTPUTgrid = int8(OUTPUTgrid); |
| filename = '75\_mat/' + string(index) + '.mat'; |
| save(filename, 'OUTPUTgrid'); |
| end |

## Датасет

После разработки вокселизатора, необходимо подготовить датасет, на основе которого нейросеть будет обучаться и будут определены отличительные черты для каждого класса, по которым в дальнейшем и будет рассчитываться итоговый результат.

Сначала, для проверки работы нейросети, был взят открытый датасет ModelNet10. Он состоит из 10 классов, также существует его улучшенная версия на 40 классов. В каждом классе выделяется две группы, по 32 и 64 вокселя на сторону, а для каждой группы- две выборки- test и train. На первой нейросеть вырабатывает алгоритмы классификации, т. е. обучается, а на второй- проверяет результат, высчитывая предполагаемый класс и сравнивая результат с названием папки, в которой модель хранится. Все модели хранятся в формате, аналогичном получаемому в Matlab- переменная в виде трехмерной матрицы int8. Методика хранения моделей представлена на рисунке 12345.

Рисунок 12345- Методика хранения моделей для основного датасета

Первый датасет был основан на вышеописанном датасете, было выделено 5 папок-классов, в каждом одна группа- по 128 вокселей на сторону, и две выборки, по 50 моделей в каждой.

Для второго датасета, полученного в результате алгоритма VoxSurf, в качестве основы было взято 300 моделей для 5 классов, по 60 моделей на каждый класс. Далее они были преобразованы в файлы формата .vox. На этом шаге были исключены несколько тонкостенных моделей, так как из-за особенностей алгоритма вокселизации получить для них модели .vox было невозможно, именно поэтому итоговое количество моделей для разных датасетов отличается.

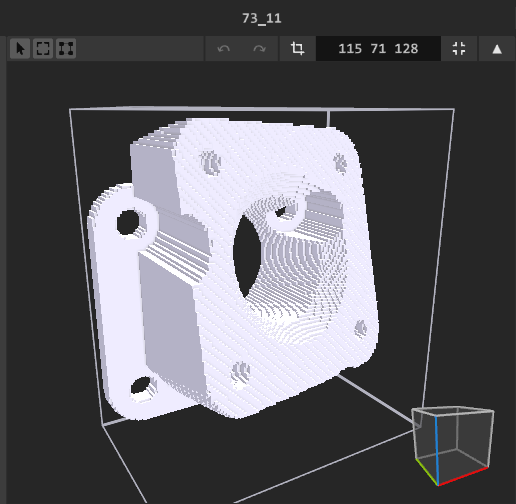
В итоге были получены 280 файлов, разделенных по классам, и названных по формуле НОМЕРКЛАССА\_ПОРЯДКОВЫЙ НОМЕР. Пример такой модели показан на рисунке 12345. На рисунке видно номер модели (73\_11), а также ее размеры (115х71х128). После формирования .vox модели для ее сохранения берется разрешение (в данном случае 128), а затем модель вписывается в куб с гранями, равными этому разрешению, пустое место очищается.

Рисунок 12345- Модель для первичного обучения нейросети

После формирования датасета также были подготовлены несколько моделей для тестирования эффективности нейросети, сформированы они аналогично.

## Выбор нейросети

Для первой проверки методики было решено воспользоваться готовой архитектурой нейросети, для чего в открытом доступе были найдены наиболее подходящие варианты и проведены тестовые запуски с проверкой точности.

Проверены были 3 наиболее подходящие нейросети, которые аналогично выполняли роль классификаторов вокселизированных моделей, но на гораздо более простых моделях, к примеру, лестницах и машинах. Чаще всего в качестве датасетов они использовали датасет ModelNet10, описанный ранее. Стоит уточнить, что нейросетей, работающих с вокселями, достаточно много, но не все удалось проверить, и путем исключения были выбраны 3 нейросети, которые будут описаны ниже.

Изображение выглядит как текст

Автоматически созданное описаниеПервой проверенной нейросетью была EJModelNet [26]. Ее основным преимуществом было удобство представления результата. В данном случае при запуске нейросети она сначала обучается на подготовленной модели, а потом через окно выбора файла подается модель, она автоматически воксеклизируется и выдается наибольшая вероятность принадлежности к какому-либо классу. Окно результата показано на рисунке 1.

Рисунок 1- Окно показа результата работы нейросети. Справа- поданная на вход модель, слева- вокселизированная, ниже- результат классификации

Основным недостатком этой нейросети была невозможность переобучения предоставленной нейросети, так как в открытом доступе отсутствовали необходимые для этого алгоритмы, поэтому для дальнейшей работы она не подходила.

Следующим вариантом была выбрана нейросеть VoxNet [27]. Ее задачей было распознавание объектов в реальном времени, на вход подается облако точек, на котором в ходе работы будут классифицированы объекты, полученные путем вокселизации. К сожалению, блок вокселизации и классификации не удалось выбрать и использовать, поэтому этот вариант тоже пришлось исключить.

Третим вариантом была нейросеть 3DDescriptorNet. Эта нейросеть позволяет делать множество операций над 3Д моделями, а именно восстановление поврежденных объектов, синтез и изменение разрешения объектов, и главное- классификация.

За основу взят датасет ModelNet10, заявленная точность работы с ним- 92,4%, что является преимуществом для ее выбора. Эта точность определяется, используя правило «один против всех». Основная идея- обучение и тестирование классификатора как набора бинарных классификаторов, определяющих принадлежность объекта к классу или к остальному набору классов. Очевидным недостатком этого метода является необходимость построения модели для каждого класса, что, как будет показано далее, заметно скажется на времени тестирования, но позволяет максимально объективно оценить точность работы нейросети для каждого класса, что в условиях поставленной задачи классификации изделий играет первоочередную роль. Заявленное сравнение точностей нейросетей представлено в таблице 1 [27]. Очевидно, что заявленная точность будет характерна для датасетов с сильно различающимися по форме моделей, и точность классификации изделий, внешне достаточно схожих, будет заметно меньше.

Таблица 1. Сравнение точностей методов классификации.

|  |  |
| --- | --- |
| Метод | Точность классификации, % |
| 3D DescriptorNet | 92,4 |
| 3D-GAN | 91,0 |
| VConv-DAE | 80,5 |
| SPH | 79,8 |
| DeepPana | 85,5 |
| 3D ShapeNet | 83,5 |
| ECC | 90,0 |
| Panorama-NN | 91,1 |
| Geometry Image | 88,4 |

Перед началом обучения берется предварительно обученная модель, которая в дальнейшем будет настраиваться под датасет. В ходе работы каждые 10 итераций получаемая модель сохраняется в отдельную папку. В ходе работы нейросеть строит модель на основе каждого объекта из выборки для обучения, а далее проверяет точность на тестовой выборке конкретного класса.

Для каждой итерации выводится следующий набор данных, связанный с точность определения:

*train loss: 133.4321, train acc: 0.6000, test loss: 554.5170, test acc: 0.4750,*

здесь *train loss*- среднеквадратичная точность классификации конкретно выбранного заранее класса для обучения, чем она меньше, тем более подстроенное будет итоговая формула определения

*train acc*- точность в процентах для обучения

*test loss* и *test acc*- аналогично для тестирования.

Далее для каждого класса была проведена проверка точности, ее метод и результаты описаны в следующем разделе.

## Анализ ее работы/анализ результата

## Вывод

# ЗАКЛЮЧЕНИЕ

Классификация изделий облегчает повторное использование деталей в дальнейшем, а также позволяет систематизировать изделия, которые в целом являются слабоизмененными вариациями одной модели. Именно поэтому задача классификации изделий является востребованной и достаточно актуальной.

Стоит отметить основные выведенные аспекты алгоритма классификации изделий. Во-первых, был определен лучший формат, в котором стоит подавать на вход модели, является obj, и, как альтернатива, fbx. Эти форматы предоставляют минимально необходимый набор данных, а именно необходимое в дальнейшем положения вершин и граней, и также удобный доступ к этим данным посредством ключей, что облегчит дальнейшую обработку. Также эти форматы являются универсальными, что позволит при необходимости преобразовывать другие форматы и получать универсальный алгоритм для изделий, созданных в различных САПР.

Также были определены классы изделий, с классификации которых достаточно начать тестировать возможный алгоритм. Это классы 71–75 по классификатору, а именно различные детали вращения, трубы и корпуса. У данных классов есть подклассы, классификация которые является дополнительной задачей, точность результатов которой еще предстоит определить.

В качестве алгоритма классификации, как наиболее удобный и быстрый, был выбран следующий алгоритм: сначала необходимо разделить модель на фиксированное количество вокселей, что позволит четко обозначить входные данные, которые далее подаются в CN-нейросеть, которая на основе предоставленных базисных, или шаблонных, объектов определит вероятность принадлежности данного изделия к конкретному классу.

Так как в данной работе предоставлены только исследовательские наблюдения, в качестве продолжения можно выполнить непосредственную разработку и тестирование алгоритма для различных основных классов, и, как было описано ранее, выбор возможных для определения подклассов, для которых точность работы будет больше необходимой.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Разбор форматов: 3d-модели изнутри // [habr.com]. URL: habr.com/ru/post/263009/ (дата обращения: 15.11.2020).
2. An overview of 3D data content, file formats and viewers // [researchgate.net]. URL: researchgate.net/publication/228976876\_An\_overview\_of\_3D\_data\_content\_file\_formats\_and\_viewers (дата обращения: 29.11.2020).
3. Исследование 3d форматов хранения данных в интеллектуальных системах виртуальной реальности // [Международный журнал экспериментального образования]. URL: expeducation.ru/ru/article/view?id=7186 (дата обращения: 02.12.2020).
4. The Most Common 3D File Formats // [all3dp.com]. / URL: all3dp.com/3d-file-format-3d-files-3d-printer-3d-cad-vrml-stl-obj/ (дата обращения: 02.12.2020).
5. Классификатор ЕСКД // [КлассИнформ]. URL: classinform.ru/ok-eskd/kod.html (дата обращения: 03.12.2020).
6. Единая система конструкторской документации (ЕСКД). Общие положения [Текст]: ГОСТ 2.001–2013. Введ. 2013-11-22.  М.: Стандартинформ, 2011.
7. Единая система конструкторской документации (ЕСКД). Обозначение изделий и конструкторских документов [Текст]:   
   ГОСТ 2.201–80. Введ. 1980-03-21. М.: Стандартинформ, 2011.
8. S. Biasotti, Giorgi D., Spagnuolo M., Falcidieno B. Reeb graphs for shape analysis and applications // DOI 10.1016/j.tcs.2007.10.018 28.02.2008.
9. Doraiswamy H., Natarajan V. Efficient algorithms for computing Reeb graphs // DOI 10.1016/j.comgeo.2008.12.003 07.08.2009.
10. Pascucci V., Scorzelli G., Bremer P., Mascarenhas A. Robust On-line Computation of Reeb Graphs: Simplicity and Speed // DOI 10.1145/1276377.1276449 09.2007.
11. Cohen-Or D., Kaufman A. Fundamentals of Surface Voxelization // DOI 10.1006/gmip.1995.1039. 11.1995.
12. EJModelNet // [github.com]. URL: https://github.com/EJShim/EJModelNet (дата обращения: 15.12.2020).
13. Voxelizer and SDF // [github.com]. URL: https://github.com/rFalque/voxelization\_and\_sdf (дата обращения: 25.12.2020).
14. Rosenblatt, F. The perceptron: A probabilistic model for information storage and organization in the brain // DOI [10.1037/h0042519](https://psycnet.apa.org/doi/10.1037/h0042519). 1958.
15. Hopfield J. Neural networks and physical systems with emergent collective computational abilities // DOI [10.1073/pnas.79.8.2554](https://doi.org/10.1073/pnas.79.8.2554). 01.10.1982.
16. Hayes B. First Links in the Markov Chain // American Scientist. –2013. –№101. –С. 92.
17. Michael I., Terrence J. Learning and Relearning in Boltzmann Machines // Graphical Models: Foundations of Neural Computation. –2001.
18. Bourlard, H., Y. Kamp Auto-association by multilayer perceptrons and singular value decomposition // Biological Cybernetics. –2004. –№59. –С. 291-294.
19. M. Ranzato Efficient Learning of Sparse Representations with an Energy-Based Model // NIPS. –2006. –№19.
20. Diederik P., M. Welling Auto-Encoding Variational Bayes // CoRR. –2014.
21. P. Vinsent Extracting and composing robust features with denoising autoencoders // ICML. –2006. –№08.
22. Y. Bengio Greedy Layer-Wise Training of Deep Networks // NIPS. –2006.
23. LeCun, Y. Gradient-based learning applied to document recognition // Proceedings of the IEEE. –1998. –№86.

Вокселизация

1. [Fundamentals of Voxelization](http://www.cs.sunysb.edu/~vislab/projects/volume/Papers/Voxel/index.html) (англ.). IEEE Computer, Vol. 26, No. 7 (July 1993). Дата обращения: 9 июля 2010. [Архивировано](https://www.webcitation.org/65XTHidqM?url=http://www.cs.sunysb.edu/vislab/wordpress//projects/volume/Papers/Voxel/index.html) 18 февраля 2012 года.
2. /ссылки на репозитории?/

Нейросети

1. https://github.com/EJShim/EJModelNet
2. https://www.ri.cmu.edu/pub\_files/2015/9/voxnet\_maturana\_scherer\_iros15.pdf
3. http://www.stat.ucla.edu/~jxie/3DDescriptorNet/3DDescriptorNet.html