Реферат

Пояснительная записка содержит 61 страниц (из них 18 страниц приложений). Количество использованных источников — 9. Количество приложений — 4.

Ключевые слова: кластеризация, нейронная сеть, дисперсия, градиент, ковариационная матрица, матрица Гессе, метод k -ближайших соседей, метод главных компонент, алгоритмы поска объектов на изображении, математическое ожидание, python.

Целью данной работы является проектирование и реализация программных средств для определения географического положения по фотографии.

В первом разделе производится обзор и анализ существующих методов и подходов для определения местоположения по фотографии.

Во втором разделе разобраны теоретические основы построения модели для определения географических координат по фотографии.

В третьем разделе приведена программная реализация модели для распознавания координат по фотографии.

В четвертом разделе разобраны практическое применение и экспериментальное исследование модели.

В приложении 1 представлена реализация скрипта, для описания моделей БД.

В приложении 2 представлена реализация модели классификатора, основанного на методе k -ближайших соседей.

В приложении 3 представлена реализация модели классификатора, основанного на методе сопоставления объектов на изображении.

В приложении 4 представлена реализация скрипта для вычисления координат изображения на основе его класса.

Содержание

Введение

Человечество всегда стремилось к неизведанным территориям, ведь в процессе своей эволюции наш вид покорял всё новые и новые места необъятного мира. Именно поэтому проблема позиционирования объектов и субъектов является неотъемлемой на протяжении всего пути развития нашей цивилизации.

Система координат – гениальное изобретение человеческого ума. Начало было положено ученым Гиппархом, предложившим ввести географические координаты. Благодаря такой удобной модели на сегодняшний день можно довольно точно определить положение почти всех объектов на нашей планете.

Но прогресс не стоит на месте. Люди всё время стремятся улучшить качество своей жизни, поэтому на протяжении всей своей истории мозг человека претерпевал ряд колоссальных изменений. Именно благодаря способности мыслить человечество смогло занять высшую ступень эволюции. В современном мире автоматизация и «человекозамещение» играют огромную роль, и во многом именно попытки человечества воссоздать мыслительный процесс дали толчок к такому вектору развития. Эти попытки увенчались успехом, когда Уоррен Мак-Калок и Уолтер Питтс смогли теоретически описать математическую модель — нейронную сеть, имитирующую функции мозга человека. Затем последовала реализация этой модели, и в настоящее время мы уже не можем представить нашу жизнь без нейронных сетей. Ведь почти в любой отрасли найдётся применение данной модели, которую можно обучить выполнять почти любую задачу.

В связи с этим актуальным и перспективным направлением в настоящее время является разработка нейросетевых моделей, решающих ряд различных проблем. Одной из них является проблема географического позиционирования. Сложность данной задачи состоит в том, что на нашей планете находится огромное количество различных объектов, и для сопоставления координат каждому требуются большие объёмы запоминающих устройств. Именно тут и приходит на помощь нейросетевая модель, которая способна обрабатывать такое количество информации.

Проблема географического позиционирования объектов в основном связана с автоматизацией различных процессов. Ведь если с помощью фотографии можно вычислить местоположение, то эти данные можно использовать для анализа текущей ситуации и планирования ряда последующих действий.

Таким образом, целью данной работы является создание программных средств с использованием нейросетевой модели для выявления географических координат по фото.

В первом разделе производится обзор и анализ существующих методов и подходов для определения местоположения по фотографии.

Во втором разделе разобраны теоретические основы построения модели для определения географических координат по фотографии.

В третьем разделе приведена программная реализация модели для распознавания координат по фотографии.

В четвертом разделе разобраны практическое применение и экспериментальное исследование модели.

Раздел I. Обзор и анализ существующих методов и подходов для определения местоположения по фотографии

В данном разделе приводятся основные методы географического позиционирования по фотографии.

1.1. Основные понятия и идеи в географическом позиционировании, основанные на анализе пикселей изображения

В целом данная задача является довольно сложной, ведь изображения часто содержат негативные информативные сигналы, которые только ухудшают восприятие. Однако некоторые ориентиры, такие как погодные условия, растительность, дорожная разметка, архитектурные сооружения и другие опознавательные знаки, помогают определить приблизительное местоположение.

Для упрощения поиска выгодно составить некую карту, на основе которой будет производится сравнивание изображений и сопоставление метки каждому из них. Основываясь на этой карте модель делает вывод о том к какому классу (месту на карте) принадлежит данное изображение и определяет наиболее вероятные координаты. Этот метод во многом похож на работу модели PlaNet, которая использует 3D карту поверхности земли разбитую на сектора в зависимости от расположения изображений. Результатом работы данной модели является вероятностное распределение координат изображения на данной карте. В основе данной модели лежит CNN — свёрточная нейронная сеть (о принципах работы которой будет сказано ниже), позволяющая представить изображение в виде некоторого вектора признаков. Затем в дальнейшем производится классификация данного вектора по существующим областям на карте, который также описываются некоторыми векторами.

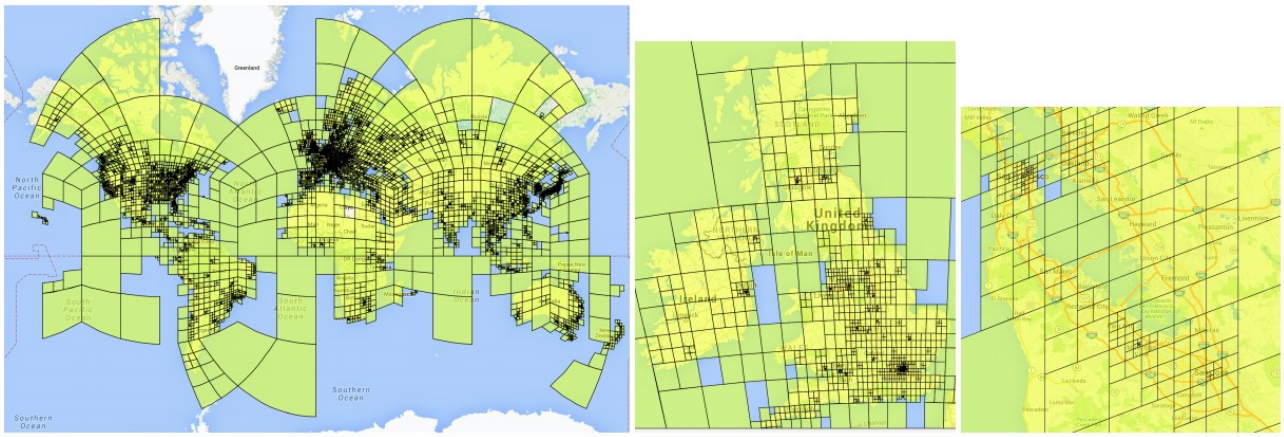


Рис. 1.1 — карта покрытия модели PlaNet

Понятие классификации изображений включает в себя процесс извлечения классов (свойственных признаков) из самого изображения путём анализа принадлежности к определенной группе. Зачастую данный метод применяется именно к растровым изображениям, которые представляют собой сетку пикселей (цветных точек). В зависимости от характера взаимодействия аналитика с компьютером в процессе классификации различают два подхода: классификацию с обучением и классификацию без обучения (так называемая «классификация без учителя»).

Классификация с обучением применяется к уже размеченной выборке данных, когда известны метки классов конкретных объектов и достаточно лишь распределить новые по этим классам. Такой подход является более точным по сравнению с классификацией без обучения, ведь в процессе разметки наблюдается обратная связь, которая контролирует качество самой классификации.

Однако на практике также применяется подход классификации без обучения. Он эффективен в том случае, когда заранее неизвестны метки классов. Ведь зачастую информации о полученных данных очень мало, и приходится делать свои предположения и догадки о составе данной выборки. В данном случае применим только этот метод классификации.

1.2. Обзор и анализ подходов представления изображений в виде вектора признаков

Во многом, как уже было сказано выше, классификация сводится к построению вектора признаков для каждого элемента набора данных и анализу этих векторов. Эта стратегия находит своё применение во многих областях, где необходимо выделить конкретную информацию о многогранном объекте. Ведь если рассматривать изображение без вектора признаков, то информацию будет нести каждый пиксель, а это в среднем $1920\,x\,1080\,x\,3$ признаков, так как зачастую используются различные цветовые модели (например, RGB), которые в математическом виде можно представить, как матрицу размерности (n, m, k), где n — количество пикселей по горизонтали изображения, m — количество пикселей по вертикали изображения, k — количество каналов изображения (для модели RGB их 3). В этом случае каждый элемент матрицы будет являться своего рода признаком. Однако существует решение данной проблемы: каждое изображение можно представлять в качестве вектора определенных признаков, свойственных для данного класса. Существует большое количество алгоритмов, позволяющих выделить определенные свойства того или иного объекта. В данной работе будут рассмотрены два подхода: перцептивное хеширование и кодирование с помощью нейросетевой модели.

◆ Алгоритм перцептивного хеширования.

Данный алгоритм описывает ряд функций для генерации индивидуального (но не уникального) вектора признаков для изображения. Таким образом, близкие (визуально) изображения будут близкими геометрическими векторами. Рассмотрим один из простейших алгоритмов — отображение средних значений низких частот. Данный алгоритм состоит из нескольких шагов:

- Уменьшается размер изображения посредством его сжатия (например, до размера $8x8$, таким образом общее количество пикселей будет равно 64);
- Убирается цвет изображения путём перевода в градации серого (чтобы уменьшить размер хеша втрое);
- Вычисляется среднее значение для каждого пикселя;
- Строится цепочка битов для каждого пикселя, путём сопоставления каждому значению 0 или 1 (в зависимости от того, больше или меньше оно среднего);
- Полученная строчка переводится в шестнадцатеричную систему счисления, эта строка и будет хешем изображения (например, $45bca12de975c2d5$).

Сравнение данных строк проводится с помощью расстояния Хемминга, то есть подсчета количества различных битов ($d_{ij}=\sum_{k=1}^{p}\left|x_{ik}-x_{jk}\right|$, где p — длина хеша в двоичной системе счисления). Такой алгоритм позволяет сравнивать изображения, используя минимальное количество признаков. Однако он является простейшим, и не особо подходит для задачи классификации погоды по изображению.

◆ Алгоритм кодирования с помощью нейронной сети.

Принцип работы и математическое обоснование структуры нейрона и сети в целом будут приведены далее, а сейчас остановимся на конкретных аспектах данного алгоритма. Существует разновидность нейронной сети, которая называется энкодер. Суть её работы заключается в следующем: на вход подается некоторое количество признаков, связанных с изображением (например, изображение размером $n x m$ сжимается до размера $250x250$) , которое кодируется в некий вектор, на основе которого происходит грцппировка по классам (классы зачастую выбираются чисто эмпирическим путём, основываясь на результате работы и её цели). Получается, что информации в данном векторе достаточно, чтобы полностью закодировать изображение, таким образом, сеть сама учится выявлять некие признаки, которые важны для описания данного изображения. На выходе получается готовый вектор

признаков, для которого можно применять различные алгоритмы классификации, которые будут рассмотрены позже.

В итоге эти два подхода на выходе получают вектор признаков, однако их время работы не совсем сопоставимо. В среднем для того, чтобы обучить на фиксированном количестве данных сеть, необходимо потратить промежуток времени длиной n (n зависит от вычислительной мощность оборудования, на котором проводилось обучение), однако кодирование уже на обученной сети в дальнейшем не будет занимать особо много времени (что является довольно весомым доводом в пользу нейронной сети). А для алгоритма перцептивного хеширования, чтобы закодировать одно изображение необходим промежуток времени k (k зависит от вычислительной мощность оборудования на котором проводилось обучение). Стоит заметить, что $k \sim n$. Тогда для кодирования 100 изображений алгоритм перцептивного хеширования потратит время $100 \cdot k$, а энкодер (часть автоэнкодера, кодирующая изображение) вместе с обучением потратит время $n+c$, где c — время кодирования 100 изображений, при чём $c \ll k$. Таким образом, сложность алгоритма перцептивного хеширования равна $O(kn)$, где n — количество данных, а k — некая постоянная, зависящая от вычислительной мощность оборудования на котором проводилось обучение. А сложность нейронной сети — $O(n)$, где n — количество данных в обучающей выборке. Поэтому по временным затратам нейронная сеть эффективнее алгоритма хеширования. Также стоит отметить, что сеть сама учится выбирать признаки для конкретной задачи, а перцептивный хешер работает по заданному алгоритму, поэтому при добавлении новых данных нейросетевую модель достаточно просто «дообучить» на них.

1.3. Обзор и анализ подходов кластеризации данных

Любая классификация строится на разбиении данных на множества, обладающие определенными свойствами. Такой подход очень близок к кластеризации данных, когда множества представляют собой определенный кластер данных. Существует огромное количество алгоритмов, позволяющих разбить данные на кластеры, но применение каждого из них в общем виде сводится к следующим этапам:

- Отбор выборки объектов для кластеризации;
- Определение множества переменных, по которым будут оцениваться объекты в выборке. При необходимости – нормализация значений переменных;
- Вычисление значений меры сходства между объектами;
- Применение метода кластерного анализа для создания групп сходных объектов (кластеров);
- Представление результатов анализа.

Рассмотрим некоторые из существующих алгоритмов. Основной идеей каждого является вычисление расстояний (меры сходства) между объектами разных классов. Так так в качестве входных данных удобнее рассматривать вектора признаков, то вычислить расстояние между ними будет не очень сложно. Однако сначала необходимо провести нормализацию данных, чтобы все компоненты давали одинаковый вклад при расчёте расстояний (обычно все значения приводят к некоторому диапазону, например, $[-1;1]$или $[0;1]$). Уже после нормализации можно использовать одну из следующих метрик для нахождения расстояния:

- Евклидово расстояние (представляет собой геометрическое расстояние в многомерном пространстве):

  $$p(x,\tilde{x})=\sqrt{\sum_{i}^{n}(x_i-\tilde{x}_i)^2}$$

- Квадрат Евклидова расстояния (применяется для придания большего веса более отдаленным друг от друга объектам):

  $$p(x,\tilde{x})=\sum_{i}^{n}(x_i-\tilde{x}_i)^2$$

- Расстояние городских кварталов (манхэттенское расстояние) (это среднее значение разностей по координатам):

  $$p(x,\tilde{x})=\sum_{i}^{n}|x_i-\tilde{x}_i|$$

- Расстояние Чебышева (это расстояние используется, когда нужно определить два объекта как «различные», если они различаются по какой-либо одной координате):

  $$p(x,\tilde{x})=max(|x_i-\tilde{x}_i|)$$

- Степенное расстояние (применяется в случае, когда необходимо увеличить или уменьшить вес, относящийся к размерности, для которой соответствующие объекты сильно отличаются):

  $p(x,\tilde{x})=\sqrt[r]{\sum_{i}^{n}(x_i-\tilde{x}_i)^p}$, где r, p — параметры, определяемые пользователем.

- Расстояние Махаланобиса (это расстояние между заданной точкой и центром тяжести, деленное на ширину эллипсоида в направлении заданной точки. Если матрица ковариации является единичной, то расстояние Махаланобиса становится равным расстоянию Евклида. Если матрица ковариации диагональная (но необязательно единичная), то получившаяся мера расстояния равна нормализованному евклидовому расстоянию):

$p_m(x)=\sqrt{(x-\mu)^T S^{-1}(x-\mu)}$, где μ — средний вектор, до которого вычисляется расстояние, S — ковариационная матрица.

Существует несколько типов алгоритмов кластеризации, но в данной работе будут рассмотрены лишь несколько из них:

- Алгоритм квадратичной ошибки (метод k- средних).

Задачу кластеризации можно рассматривать как построение оптимального разбиения объектов на группы. При этом оптимальность может быть определена как требование минимизации среднеквадратичной ошибки разбиения:

$e^2(X,L)=\sum_{j=1}^{K}\sum_{i=1}^{n_j}\left\|x_i^{(j)}-c_j\right\|^2$, где c_j — «центр масс» кластера, j — точка со средними значениями характеристик для данного кластера.

Алгоритмы квадратичной ошибки относятся к типу плоских алгоритмов. Самым распространенным алгоритмом этой категории является метод k- средних. Этот алгоритм строит заданное число кластеров, расположенных как можно дальше друг от друга. Работа алгоритма делится на несколько этапов:

1. Случайно выбрать k точек, являющихся начальными «центрами масс» кластеров;
2. Отнести каждый объект к кластеру с ближайшим «центром масс»;
3. Пересчитать «центры масс» кластеров согласно их текущему составу;
4. Если критерий остановки алгоритма не удовлетворен, вернуться к пункту 2.

В качестве критерия остановки работы алгоритма обычно выбирают минимальное изменение среднеквадратичной ошибки. Так же возможно останавливать работу алгоритма, если на шаге 2 не было объектов, переместившихся из кластера в кластер.

К недостаткам данного алгоритма можно отнести необходимость задавать количество кластеров для разбиения.

- Алгоритм минимального покрывающего дерева.

Этот алгоритм сначала строит на графе минимальное покрывающее дерево, а затем последовательно удаляет ребра с наибольшим весом. Таким образом, данные можно разделять на произвольное количество кластеров. При этом на выходе мы получаем древовидную структуру кластеров, которую в дальнейшем можно изучать.

- Алгоритм иерархической кластеризации.

Данный алгоритм основан на принципе объединения классов в более обширные, пока все объекты выборки не будут содержатся в одном кластере, при этом используется так называемый восходящий принцип построения кластеров, при котором всем объектам выборки изначально ставятся в соответствия различные кластеры, и объединение происходит

с помощью выбранной метрики. При этом критерием остановки является пороговое расстояние или число кластеров. Таким образом, результат алгоритмы представляется в виде некого дерева разбиений - дендрограммы.

К недостаткам данного алгоритмы можно отнести излишнее разбиение исходных данных, которое может быть лишним в контексте конкретной задачи.

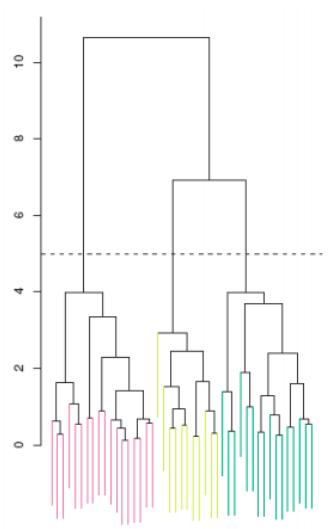


Рис. 1.2 — пример дендрограммы

- Алгоритм выделения главных компонент.

Данный алгоритм основан на способе уменьшить размерность исходных данных, потеряв при этом минимум информации. На основе полученных данных строятся базисные вектора, на которые проецируется вся исходная выборка. Таким образом достигается уменьшение размерности данных.

Более подробно некоторые алгоритмы будут рассмотрены ниже.

Оценка сложности данных алгоритмов даёт следующий результат: алгоритм квадратичной ошибки (метод k - средних) имеет сложность $O(k \cdot l \cdot n)$, где $\sigma(z)=z$ — число кластеров, l — количество итераций, $\sigma(z)=\frac{1}{(1+\exp(-z))}$ — объём данных; алгоритм минимального покрывающего дерева — $O(n^2 \cdot \log(n))$, где $\sigma(z)=\tanh(z)$ — объём данных; алгоритм иерархической кластеризации — $O(n^2)$, где $\sigma(z)=max(0,z)$ — объём данных; алгоритм выделения главных компонент — $O(n^2 \cdot k)$, где n — объём данных, k — сложность математических вычислений.

1.4. Обзор и анализ подходов классификации с обучением

В области классификации также выделяют подход, который происходит с обучением. Он отличается от вышеизложенного тем, что часть данных является уже отсортированной по классам, и остаётся лишь построить разбиение для остальной части по уже имеющимся классам. Перед непосредственно самой классификацией данные необходимо нормализовать (это процесс аналогичен нормализации данных для классификации без обучения).

Рассмотрим несколько подходов при данной классификации: наивный байесовский классификатор, нейросетевая модель прямого распростанения, метод опорных векторов (SVM) и метод выделения особых точек (ORB).

- Наивный байесовский классификатор.

Суть данного алгоритма заключается в применении теоремы Байеса. Это не отдельный алгоритм, а некая группа, которая придерживается общего принципа: каждая классифицируемая пара признаков не зависит друг от друга. В итоге получается модель, которая с помощью вероятностного подхода определяет принадлежность объекта к тому или иному классу.

Предположение наивного байесовского классификатора состоит в том, что каждый признак создаёт независимый и равный вклад в результат, а сама теорема выглядит так: $P(x \vee y) = \frac{P(y \vee x)P(x)}{P(y)}$, где $P(x \vee y)$— условная вероятность (событие x при условии события y, $P(x), P(y)$— вероятности каждого события по отдельности. Тогда в применении к задаче классификации y— параметры модели, которые необходимо обучить, а x— данные, которые уже размечены.

- Нейросетевая модель прямого распространения.

Данный подход использует понятие нейронной сети, то есть модели, которая по неким входным данным может построить выходные, учитывая данные, для которых уже известен результат. В общем случае данная модель учится воспроизводить результат по уже известным параметрам. В применении к классификации с обучением такой подход используют очень часто. Ведь по уже размеченным данным нейронная сеть сможет провести классификацию довольно точно.

- Метод опорных векторов (SVM).

Данный алгоритм имеет широкое применение на практике, он используется в задачах классификации и регрессии. Реализация данного метода сводится к геометрическому представлению данных, при этом основной задачей алгоритма является поиск правильной линии или гиперплоскости разделяющей данные на два класса.

Однако при построении разбиения возникает ситуация, когда возможно несколько правильных решений. Это связано с тем, что количество линий (плоскостей), разделяющих данные, есть бесконечно большое число (рис. 1.3).

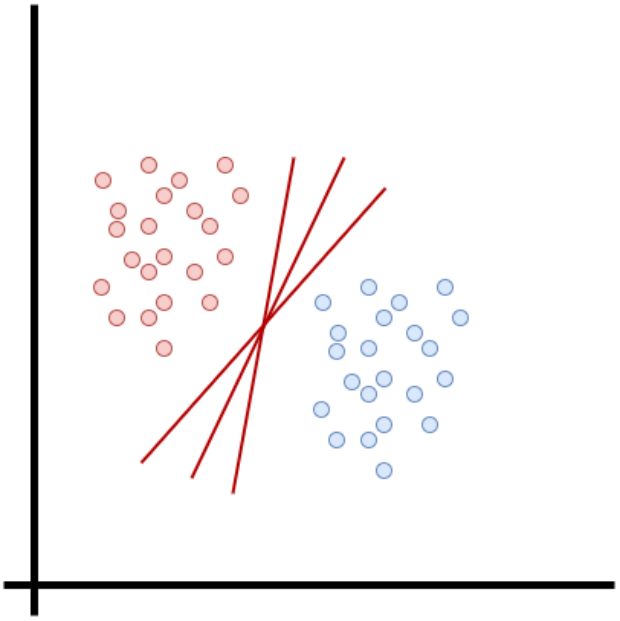


Рис. 1.3 — варианты расположения разделяющей прямой

Решением в данной ситуации является поиск точек, которые расположены к линии разделения ближе всего. Данные точки называются опорными векторами (рис. 1.4).

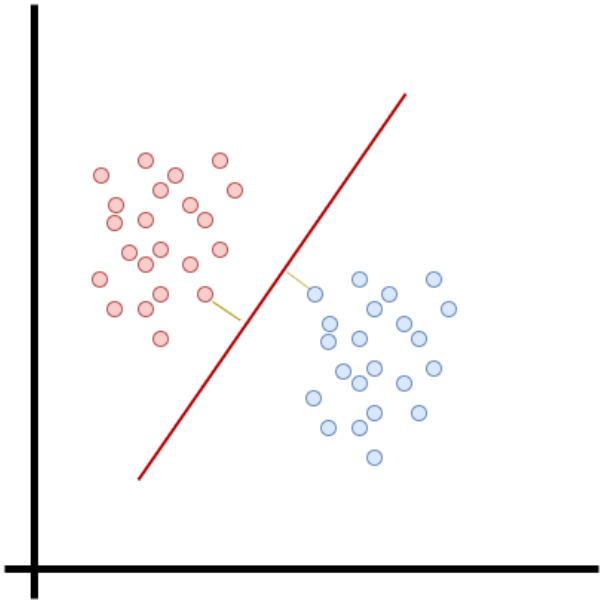


Рис. 1.4 — построение опорных векторов

В результате при оптимальном построении опорных векторов, алгоритм находит единственно верное разбиение данных на два класса. При обобщении данного метода на многомерное пространство получается универсальный классификатор.

- Метод выделения особых точек (ORB).

Данный метод основан на выделении особых точек для изображения, которое является представителем известного (размеченного) класса. В последствии для каждого конкретного класса имеется определённое количество особых точек, характеризующее его в целом, при анализе входное изображение также анализируется на наличие особых точек, которые сравниваются с уже имеющимися в конкретном классе. Так происходит опознавание объекта на изображении, соответствующего определённому классу. При этом предоставляется возможность узнать ориентацию объекта на изображении.

Оценивая сложность данных алгоритмов можно прийти к выводу, что временные затраты наивного байесовского классификатора и нейронной сети прямого распространения примерно равны, и сложность каждого из них составляет $O(n)$, где n— объём данных. Однако сложность метода опорных векторов составляет $O(m^3 \cdot n)$, где m— объем данных, а n — количество признаков, поэтому данный метод идеален для сложных, но небольших или средних наборов данных. Сложность метода выделения особых точек можно оценить, как $O(n \cdot m \cdot k)$ в самом плохом исходе, где n — среднее количество особых точек на изображении конкретного класса, m — количество особых точек на исходномм изображении, а k — количество классов. Также стоит учесть, что основные положения байесовского классификатора противоречат действительности, ведь зачастую признаки объекта напрямую зависят друг от друга.

1.5. Выводы

В данном разделе проведён разбор и сравнительный анализ основных методов классификации. Исходя из этого, можно сделать выводы, что следует попробовать реализовать несколько готовых решений, на основе которых выбрать лучший подход. При главным фактором является масштабируемость, быстрота работы и универсальность.

Раздел II. Теоретические основы построения модели для определения географических координат по фотографии

В данном разделе приводятся теоретические основы и архитектура модели, определяющей местоположение по фотографии.

2.1. Теоретическая основа нейронной сети

Нейронная сеть — это последовательность нейронов, которые соединены синапсами. Это понятие пришло из биологии. Ведь благодаря такой структуре появляется возможность хранить, обрабатывать поступающую информацию, а также воспроизводить её. И всё это можно реализовать программно. Именно поэтому нейронные сети обрели такую популярность.

В основном нейронные сети используются для решения сложных аналитических задач, которые нельзя обработать традиционными алгоритмами. Самыми распространёнными применениями являются:

- классификация — распределения данных по параметрам;
- предсказание — возможность предсказывать следующий шаг;
- распознавание — определение положения объекта и его свойств.

Однако это не все сферы применения нейронных сетей, ведь почти в каждой области существую проблемы, которые решаются с помощью них.

Ключевым понятием в этой модели является нейрон, который представляет собой вычислительную единицу, получающую на вход информацию и преобразующею её на выходе с помощью различных математических операций. Нейрон по сути реализует следующие действия: он вычисляет взвешенную сумму всех элементов входного вектора $\vec{w}$, используя вектор весов $\vec{x}$(а также аддитивную составляющую смещения w_0, а затем к результату может применяться функция активации σ (рис. 2.1). Таким образом, получается, что нейрон отбирает наиболее весомы признак из входного вектора.

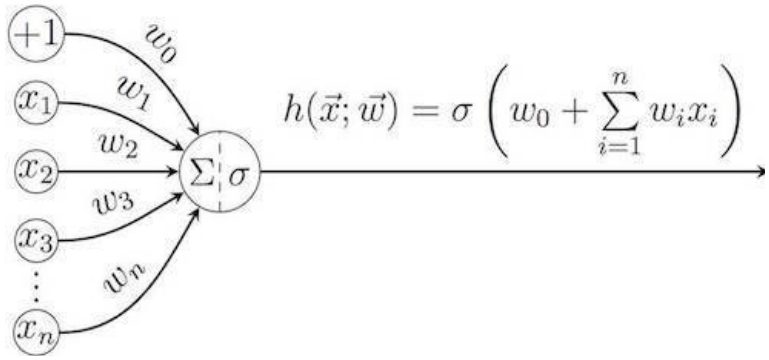


Рис. 2.1 — принцип работы нейрона

Функция активации представляет собой некую математическую функцию, которая используется для подключения несвязанных входных данных с выводом, у которого простая и предсказуемая форма. Существует несколько различных функций активации:

- Функция тождества (Identity): $\sigma(z)=z$;
- Сигмоидальная, логистическая функция (Logistic): $\sigma(z)=\frac{1}{(1+\exp(-z))}$;
- Гиперболический тангенс (Tanh): $\sigma(z)=\tanh(z)$;
- Полулинейная функция (Rectified linear, ReLU): $\sigma(z)=max(0,z)$.

Графики данных функций представлены ниже (рис. 2.2).

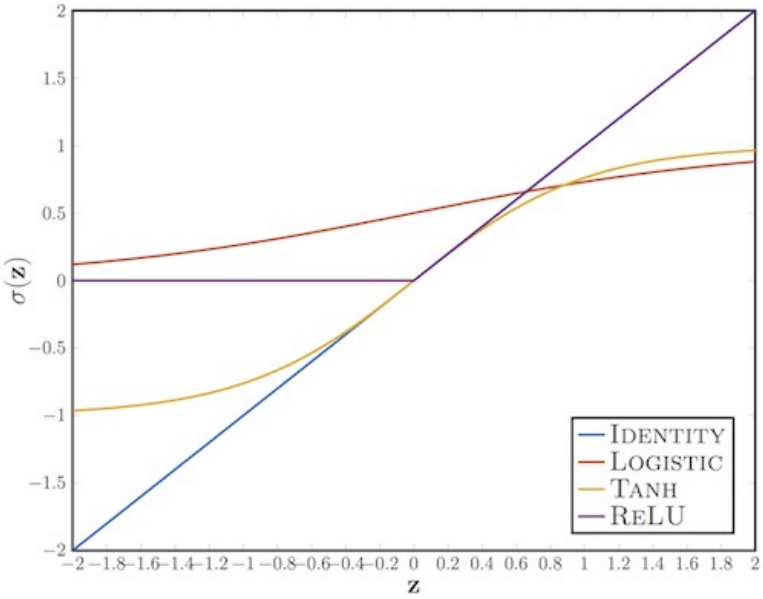


Рис. 2.2 — графики функций активации

Существует ряд параметров нейронной сети, которые невозможно обосновать математически, то есть выбор их значения определяется чисто эмпирическим путём. Они носят название — гиперпараметры, и включают в себя следующие величины:

- количество скрытых слоёв и их размер;
- скорость обучения;
- выбор нейронов со смещением.

Их определение напрямую зависит от конкретной задачи. Зачастую результат работы модели может очень сильно колебаться в зависимости от выбора этих параметров.

Главная задача обучаемых моделей — это их способность обобщаться, то есть хорошо работать на новых данных. Поскольку сразу во время обучения проверить модель на новых данных не представляется возможным, то необходимо пожертвовать часть размеченных данных, чтобы на них проверять качество модели.

Для такой проверки чаще всего используются два способа:

- Отложенная выборка.

  При таком подходе оставляется некоторая доля обучающей выборки (как правило от 20% до 40%), а модель обучается на остальных данных (60 − 80% исходной выборки), при этом считается некоторая метрика качества модели (например, самое простое – доля правильных ответов в задаче классификации) на отложенной выборке.

- Кросс-валидация данных.

  При данном способе исходные данные разбиваются на k подвыборок. При этом модель обучается k раз на разных $k-1$ подвыборках исходной выборки, а проверяется на одной подвыборке (каждый раз на разной). Получаются k оценок качества модели, которые обычно усредняются, выдавая среднюю оценку качества классификации на кросс-валидации.

Кросс-валидация дает лучшую по сравнению с отложенной выборкой оценку качества модели на новых данных. Но кросс-валидация вычислительно дорогостоящая, если данных достаточно много.

После того как сеть была обучена (то есть были найдены веса и смещения, при которых функция потерь минимальна) можно делать предсказания, то есть для неразмеченных данных узнавать результат работы сети.

2.2 Теоретические основы метода ближайших соседей (KNN (k nearest neighbor) классификатор)

Данный алгоритм при $k=1$ относит классифицируемый объект k к тому классу, которому принадлежит ближайший обучающий объект $\alpha(x;X^n)=y_x^{(1)}$. При этом обучение данного классификатора сводится к запоминанию выборки X^n . Однако при $k>1$ чтобы сгладить шумовое влияние выбросов, необходимо классифицировать объекты путём сопоставления их k ближайшим соседям. Таким образом, каждый из соседей $u_x^{(i)}, i=1,..,k$ сопоставляется с объектом x , при удачном сопоставлении объект x относится к классу соседа $y_x^{(i)}$.

Алгоритм относит объект x к тому классу, сопоставление с которым пройдёт наилучшим образом:

$$\alpha(x;X^n;k)=\arg\max_{y\in Y}\sum_{i=1}^{k}[y_x^{(i)}=y]$$

При этом сопоставление может включать в себя различные расстояния приведённые в обзоре подходов к кластеризации данных.

При классификации для каждого объекта выполняются следующие операции:

- Вычисляется расстояние до каждого из объектов обучающей выборки;
- Отбирается k объектов обучающей выборки, расстояние до которых минимально (результат сопоставления).
- При этом класс классифицируемого объекта — это класс, соседа с которым наиболее удачно прошло сопоставление.

Для обучения классификатора задаётся начальный параметр — k, затем выбирается k случайных объектов из выборки, для них вычисляется расстояние до остальных объектов, и все объекты относятся к соответствующему соседу, после этого в каждом классе происходит пересчёт «центра» (путём устреднения значение объектов). Эта процедура повторяется до того, как изменение центра классов будут меньше, чем фиксированное δ. В случае с k-medoids классификатором (применяемым в работе) «центры» классов выбираются из существующих объектов данных, путём вычисления расстояний между каждыми объектами и нахождения равноудалённого от всех.

2.3. Теоретические основы метода главных компонент (PCA)

Основное предположение данного метода состоит в том, что любые данные в n — мерном пространстве можно представить в виде некого эллипсоида в подпространстве исходного пространства, при этом новый базис в этом подпространстве будет совпадать с осями данного эллипсоида. Это предположение позволяет объединить (избавиться) от сильно скоррелированных (зависящих друг от друга) признаков, так так при проекции на новый базис, являющийся ортогональным, основная часть данных останется неизменной.

Для того чтобы процесс проецирования на вектор никак не влиял на значение средних векторов, необходимо прохождение данного вектора через центр исходных данных. Для этого данные нужно центрировать, то есть линейно сдвинуть так, чтобы средние значения признаков равнялись 0. При этом оператор, обратный сдвигу, при применении к центрированным данным даст изначальные значения в исходной размерности. При этом математическое ожидание для исходных никак не изменится, так как оно задаёт положение случайной величины. А вот дисперсия сильно зависит от порядков значений случайной величины, поэтому если данные сильно разнятся в порядках своих значений, то их необходимо стандартизировать.

В данном случае под математическим ожиданием понимается среднее (взвешенное по вероятностям возможных значение) значение случайной величины:

$E(X)=\int_{-\infty}^{\infty} x\, f_X(x)dx$, где $E(X)$ — математическое ожидание абсолютно непрерывной случайной величины, $f_X(x)$ — плотность распределения. Или же для дискретной случайной величины:

$E(X)=\sum_{i=1}^{\infty} x_i\, p_i$, где X — дискретная случайная величина, имеющая распределение $P(X=x_i)=p_i,\ \sum_{i=1}^{\infty} p_i=1$.

Под дисперсией случайной величины понимается мера разброса значений случайной величины относительно её математического:

$D(X)=Var(X)=E[(X-E(X))^2]$, где X — случайная величина, E — математическое ожидание, а в случае, когда X — дискретная случайная величина:

$Var(X)=\sum_{i=1}^{n}\sum_{j=i+1}^{n} p_i p_j (x_i-x_j)^2$, где x_i — i -ое значение случайной величины, $p_i=P(X=x_i)$ — вероятность того, что случайная величина принимает значение x_i , n — количество значений, которые принимает случайная величина.

Для описания формы случайного вектора необходима ковариационная матрица — это матрица, у которой (i,j) -элемент является корреляцией признаков (X_i, X_j) . Согласно формуле:

$Cov(X_i,X_j)=E[(X_i-E(X_i))\cdot(X_j-E(X_j))]$. В нашем случае она равна $Cov(X_i,X_j)=E(X_i X_j)$, так как $E(X_i)=E(X_j)=0$. При этом если $X_i=X_j$, то $Cov(X_i,X_j)=Var(X_i)$. Поэтому в ковариационной матрице по диагонали стоят дисперсии признаков, а в остальных ячейках — ковариации соответствующих признаков. Заметим также, что матрица ковариации будет симметричной, так как ковариации являются симметричными. Можно увидеть, что ковариационная матрица является обобщением дисперсии на многомерный случай случайных величин, поэтому она также описывает разброс данной величины.

Для каждой пары значений в исходных данных необходимо сформировать ковариационную матрицу Σ . После этого необходимо найти такие вектора, при которых максимизировалась бы дисперсия (ковариационная матрица) при проекции данных на них. Пусть имеются v_i — вектора на которые мы будем проецировать наши данные, тогда проекция одного из векторов будет равна: $\widetilde{X}=\sum_i v_i^T X$, дисперсия проекции на вектор

соответственно будет равна $Var(\sum_i v_i^T X)$. В общем виде для центрированных величин дисперсия выражается так: $Var(X)=\Sigma=E(X\cdot X^T)$, соответственно, дисперсия проекции:

$$Var(\widetilde{X})=\widetilde{\Sigma}=E(\widetilde{X}\,\widetilde{X^T})=E((\sum_i v_i^T X)\cdot(\sum_i v_i^T X)^T)=E(\sum_i v_i^T X X^T \sum_i v_i)=\sum_i v_i^T E(X X^T)\sum_i v_i=\sum_i v_i^T \Sigma \sum_i v_i$$

Можно заметить, что дисперсия максимизируется при максимальном значении $\sum_i v_i^T \Sigma \sum_i v_i$. Здесь можно использовать отношение Рэлея: $R(E,x)=\frac{x^T E x}{x^T x}=\lambda\frac{x^T x}{x^T x}=\lambda$ и $E x=\lambda x$. Таким образом, направление максимальной дисперсии у проекции всегда совпадает с собственным вектором, имеющим максимальное собственное значение, равное величине этой дисперсии. Наибольший вектор имеет направление, схожее с линией регрессии и, спроецировав на него нашу выборку, мы потеряем информацию, сравнимую с суммой остаточных членов регрессии. При этом потеря при проецировании — это дисперсия по неучтённым при проецировании собственным векторам. Используя проецирование в направлении определённого собственного вектора можно сделать вывод о кластерах исходных данных (рис. 2.3).

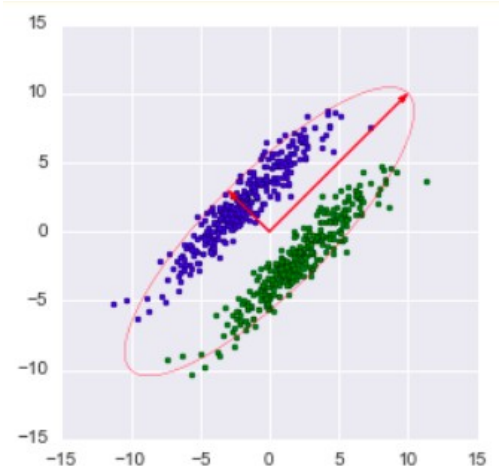


Рис. 2.3 — базисные вектора некоторого набора данных

2.4. Теоретические основы построения дескрипторов изображения (метод выделения особых точек)

Каждое изображение характеризуется набором точек — пикселей, которые задают цветовой восприятие изображения в целом, однако при распознавании изображения глаз человека выделяет особые сектора, которые характерны тому или иному объекту. Именно поэтому при разработке модели важно учесть эту особенность, так как анализ всех точек может привести к совершенно иному результату, ведь на изображении помимо полезной информации есть шум, который будет мешать работе модели.

Таким образом, необходимо на изображении выделить ряд точек, вклад в которых был бы максимальным. Согласно этим точкам следует упростить анализ, и представлять изображение в качестве совокупности этих точек. Помимо максимального вклада эти точки должны обязательно присутствовать на схожем изображении, а детектор (модель анализирующая изображение) должен распознавать их вне зависимости от многих факторов (то есть обеспечить инвариантность нахождения этих точек).

После поиска особых точек, каждая из них характеризуется положением на изображении, но при сравнении двух таких точек разных изображений данные о положении не оказывают особого влияния на анализ и сравнение. Именно поэтому необходимо описать точку в виде некого вектора параметров (дескриптора), на аснове которого можно было бы проводить сравнительный анализ, при этом дескрипторы должны обеспечивать инвариантность нахождения соответствия между особыми точками относительн о преобразований изображения. В итоге данная модель должна решать ряд следующих задач:

- Поиск ключевых точек на изображении и создание уникального дескриптора для каждой;
- Проверка совпадения ключевых точек (матчинг) на различных изображениях;
- Анализ совпавших точек и предоставление результата сравнивания.

Основным методом выделения особых точек и их дескрипторов считается SIFT (Scale Invariant Feature Transform). С помощью него можно выбрать точки инвариантные относительно смещения, поворота, масштабирования, изменения яркости и положения камеры изображения.

Данный метод для детектирования особых точек строит пирамиду гауссианов и разностей гауссианов, при этом гауссианом изображения называют: $L(x,y,\sigma)=G(x,y,\sigma)*I(x,y)$, где L — значение гауссиана в точке с координатами (x,y) , σ — радиус размытия, G — гауссово ядро ($G(x,y,\sigma)=e^{\frac{-\|x-y\|^2}{\sigma^2}}$), I — значение исходного изображения, $*$ — операция свёртки.

Разностью гауссианов называют изображение, полученное путём попиксельного вычитания одного гауссиана исходного изображения из гауссиана с другим радиусом размытия:

$D(x,y,\sigma)=(G(x,y,k\sigma)-G(x,y,\sigma))*I(x,y)=L(x,y,k\sigma)-L(x,y,\sigma)$. При этом масштабируемым пространством изображения является набор всевозможных, сглаженных некоторым фильтром, версий исходных изображений, таким образом, гауссово пространство

является масштабируемым, линейным и инвариантным относительно сдвигов, вращений, масштаба, не смещающим локальные экстремумы, и обладает свойством полугрупп.

В общем, инвариантность относительно масштаба достигается за счёт нашождения ключевых точек для исходного изображения, взятого в разных масштабах. Для этого строится пирамида гауссианов: все масштабируемое пространство разбивается на некоторые участки — октавы, причем часть масштабируемого пространства, занимаемого следующей октавой, в два раза больше части, занимаемой предыдущей. К тому же, при переходе от одной октавы к другой делается ресэмплинг изображения, его размеры уменьшаются вдвое. Естественно, что каждая октава охватывает бесконечное множество гауссианов изображения, поэтому строится только некоторое их количество N, с определенным шагом по радиусу размытия. С тем же шагом достраиваются два дополнительных гауссиана (всего получается $N+2$), выходящие за пределы октавы. Далее будет видно, зачем это нужно. Масштаб первого изображения следующей октавы равен масштабу изображения из предыдущей октавы с номером N.

Параллельно с построением пирамиды гауссианов, строится пирамида разностей гауссианов, состоящая из разностей соседних изображений в пирамиде гауссианов. Соответственно, количество изображений в этой пирамиде будет $N+1$ (рис. 2.4).

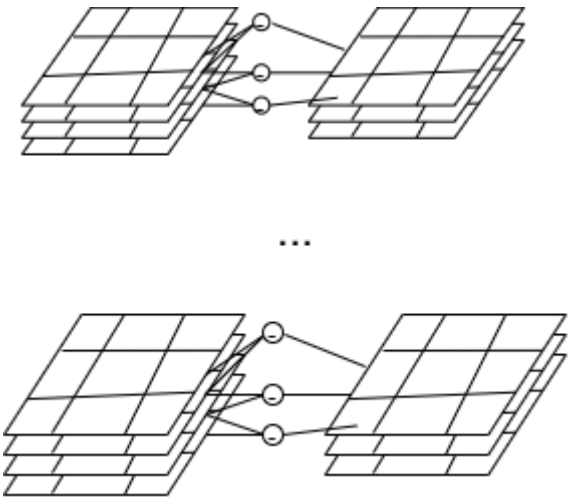


Рис. 2.4 — пирамида гауссиан и пирамида разности гауссиан

После того, как пирамиды были построены, необходимо посчитать локальный экстремум разности гауссиан, им будет являться особая точка. При этом в каждом изображении пирамиды ищутся локальные экстремумы с помощью сравнения каждой точки с её соседями на текущем уровне и на уровне выше и ниже в пирамиде.

После поиск точек экстремума необходимо уточнить всё ли найденные точки удовлетворяют критерию «особенности». Для начала необходимо уточнить координаты особой точки с субпиксельной точностью, этого можно достичь с помощью аппроксимации функции разности гауссиан многочленом Тейлора:

$D(X)=D_0+\frac{\partial D^T}{\partial X}X+\frac{1}{2}X^T\frac{\partial^2 D}{\partial X^2}X$, где D — функция разности гауссиан, $X=(x,y,\sigma)$ — вектор смещения относительно точки разложения.

Экстремум многочлена Тейлора можно найти путём вычисления производной и приравнивая её к нулю: $\widetilde{X}=\frac{-\partial^2 D^{-1}}{\partial X^2}\frac{\partial D}{\partial X}$. Все производные вычисляются по формулам конечных разностей, в итоге получаем СЛАУ размерности $3x3$ относительно компонент вектора $\widetilde{X}$. Если одна из компонент вектора больше чем половина шага сетки в этом направлении, то это означает, что точка экстремума была вычислена неверно и нужно сдвинутся к соседней точке в направлении указанных компонент. И повторить процедуру уточнения снова. Если мы в процессе выбора вышли за пределы октавы, то следует исключить данную точку из рассмотрения.

После вычисления положения экстремума необходимо проверить на малость разность гауссиан в этой точке:

$D(\widetilde{X})=D_0+\frac{1}{2}\frac{\partial D^T}{\partial X}\widetilde{X}$. При неудачной проверки точка исключается, как точка с малым контрастом.

Также найденную точку необходимо проверить на освещённость, если особая точка лежит на границе какого-то объекта или плохо освещена, то её необходимо исключить из рассмотрения. Такие точки имеют большой изгиб (одна из компонент второй производной) вдоль границы и малый в перпендикулярном направлении. Этот большой изгиб определяется матрицей Гессе — симметричной квадратичной формы, описывающей поведение функции во втором порядке: $H=\begin{pmatrix}D_{xx} & D_{xy}\\ D_{yx} & D_{yy}\end{pmatrix}$. Пусть $Tr(H)=D_{xx}+D_{yy}=\alpha+\beta$ — след матрицы, а $Det(H)=D_{xx}D_{yy}-(D_{xy})^2=\alpha\beta$ — определитель, $r=\frac{\alpha}{\beta}$ — отношение большего изгиба к

меньшему, тогда $\frac{Tr(H)^2}{Det(H)}=\frac{(\alpha+\beta)^2}{\alpha\beta}=\frac{(r+1)^2}{r}$ и точка рассматривается дальше, если $\frac{Tr(H)^2}{Det(H)}<\frac{(r+1)^2}{r}$.

Для составления дескриптора точки необходимо вычислить её ориентацию, она вычисляется исходя из градиентов соседних точек. Все вычисления производятся на изображении в пирамиде гауссианов, с масштабом наиболее близким к масштабу ключевой точки. При этом величина m и направление θ градиента в точке (x, y) вычисляются по формулам:

$$m(x,y)=\sqrt{(L(x+1,y)-L(x-1,y))^2+(L(x,y+1)-L(x,y-1))^2}$$

$\theta(x,y)=\arctan(\frac{L(x,y+1)-L(x,y-1)}{L(x,y+1)-L(x,y-1)})$. Направление ключевой точки можно найти из гистограммы направлений O . Гистограмма состоит из 36 компонент, которые равномерно покрывают промежуток в 360 градусов, при этом она формируется из компонент, в которые каждая точка вносит вклад $m(x,y)\cdot G(x,y,\sigma)$ и которые покрывают промежуток, содержащий направление градиента $\theta(x,y)$. При этом дескриптор ключевой точки состоит из всех полученных гистограмм.

На практике часто применяют дескриптор BRIEF, который строится из 256 бинарных сравнений между яркостями пикселов на размытом изображении, при этом также учитывается ориентация. Полученный бинарный дескриптор оказывается устойчив к сменам освещения, перспективному искажению, быстро вычисляется и сравнивается.

Для вычисления особых точек используется алгоритм FAST-ER, который строит оптимальную последовательность точек и оптимизирует её с помощью дерева решений.

2.5. Выводы

В данном разделе был проведен теоретический разбор моделей, которые будут использоваться для построения системы географического позиционирования по фотографии.

Раздел III. Программная реализация модели для распознавания координат по фотографии

В данном разделе приведена реализация модели для определения местоположения по фотографии.

3.1 Выбор средств для программной реализации

В качестве основного языка программирования при реализации модели будет использоваться Python версии 3.8. Для удобства разработки будет использоваться среда PyCharm, которая обеспечивает богатый функционал для разработки. В качестве основных библиотек и модулей будет использован следующий набор (таблица 3.1).

Название библиотеки/модуля	*Область применения*	*Источник*
Keras	Использование основных моделей, применяемых в машинном обучении	Библиотека python
Matplotlib	Построение графиков для анализа полученных результатов	Библиотека python
CSV	Запись вектора признаков в csv файл	Библиотека python
numpy	Использование для работы с математическими объектами	Библиотека python
json	Запись структуры данных в json формате	Библиотека python
sklearn	Использование основных алгоритмов кластеризации данных	Библиотека python
peewee	Взаимодействие с sql базой посредством ORM	Библиотека python
cv2	Использование алгоритмов image processing	Библиотека python
os	Взаимодействие с операционной системой	Библиотека python

Таблица 3.1 — источники и области применения основных библиотек и модулей

3.2. Общая структура и логика работы модели

На основе проведённых исследований и анализа теоретического материала была выработана модель действий, которые должны привести к результату. В рамках данной модели предусмотрены следующие шаги (рис. 3.1):

1. На вход модели подаётся изображение;
2. С помощью первых (свёрточных) слоёв остаточной сети ResNet формируется вектор признаков, описывающий данное изображение;

3. Сформированный вектор признаков (дескриптор) подаётся на вход полносвязной сети, которая определяет является ли это изображение мировой достопримечательностью;
4. При успешном определении класса мировой достопримечательности определяются m ближайших дескрипторов в рамках данного класса, и с помощью определённой метрики (Евклидовой или Махаланобиса) вычисляется взвешенное среднее по координатам этих дескрипторов. Данное взвешенное и будет результатом работы модели;
5. Если же не удалось определить класс достопримечательности, то с помощью полносвязной сети определяется тип достопримечательности (здание, собор памятник и так далее.);
6. При успешном определении типа производится кластерный анализ данного дескриптора путём выбора ближайшего соседа с известными координатами;
7. Если же тип определить не удалось, то изображение анализируется на предмет уличных вывесок, дорожных знаков, номеров домов и других объектов с текстом, при нахождении таких объектов производится анализ текста и последующий поиск по базе.

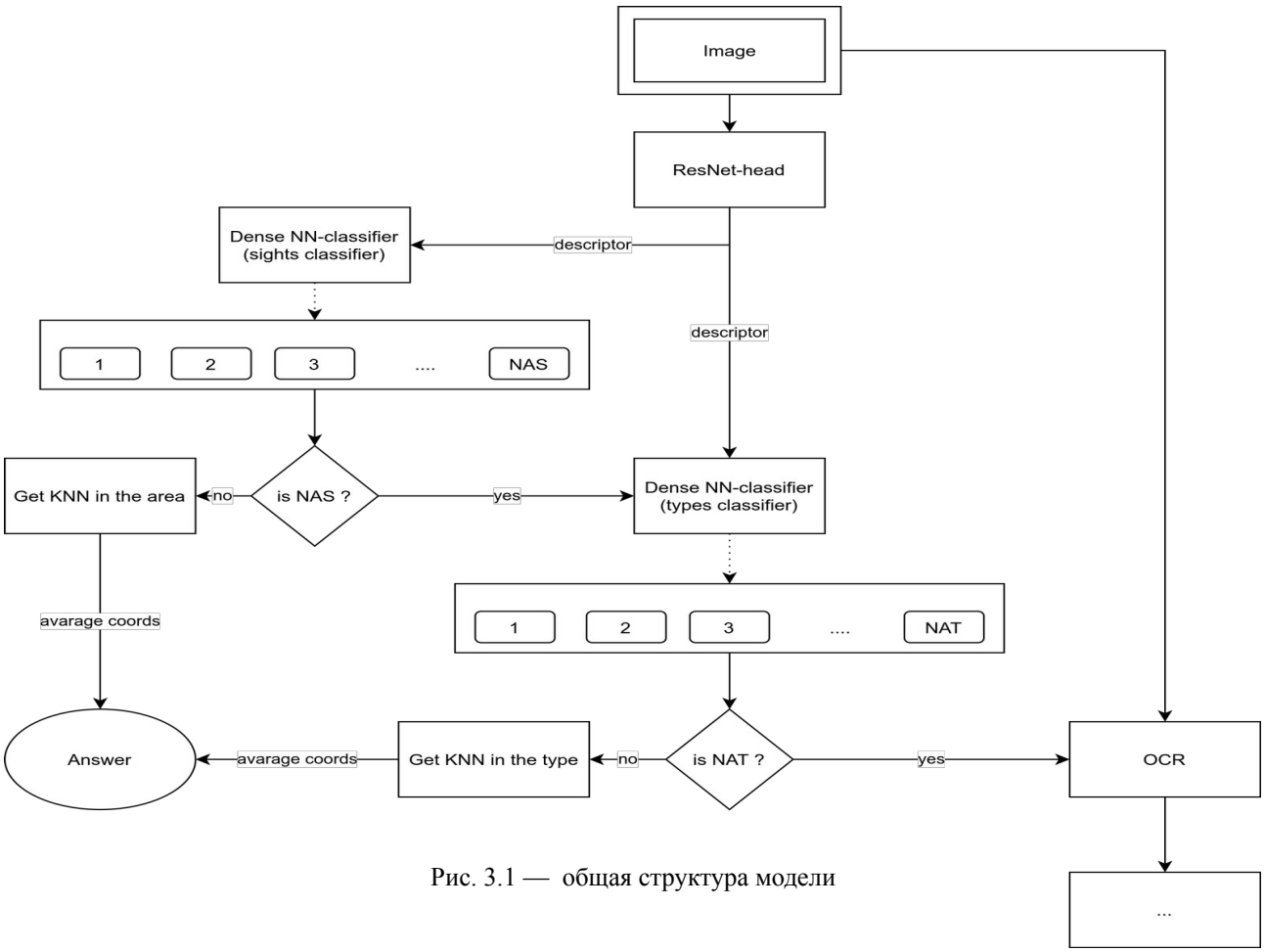


Рис. 3.1 — общая структура модели

3.3. Создание дескрипторов изображения

В рамках поставленной задачи было принято решение использовать уже обученную нейронную сеть подобную ResNet (MobileNet) без последних полносвязных слоёв, производящих классификацию. Данный выбор обоснован тем, что модель MobileNet уже была обучена на датасете ImageNet, тем самым при формировании дескрипторов использовались признаки уже отобранные при обучении.

В данной работе подробное рассмотрение аспектов создания дескрипторов и их классификации с помощью нейронной сети прямого распространения рассматриваться не будут.

3.4. Подготовка и хранение данных

В качестве обучающей выборки для нейронной сети прямого распространения была взята база фотографий путешественников (ссылку на базу можно найти в списке литературы). Данная выборка содержит фотографии и геометки, где они были сделаны сделаны. В рамках первой задачи были отобраны фотографии относящиеся к мировым достопримечательностям, путём ограничения радиуса каждой достопримечательности. Таким образом все фотографии совершённые в радиусе 50 метров попадали в класс к одной достопримечательности.

С помощью MobileNet были сформированы дескрипторы для каждой фотографии относящейся к достопримечательности, а те фотографии, которые не попали в область, больше не рассматривались (рис. 3.2).

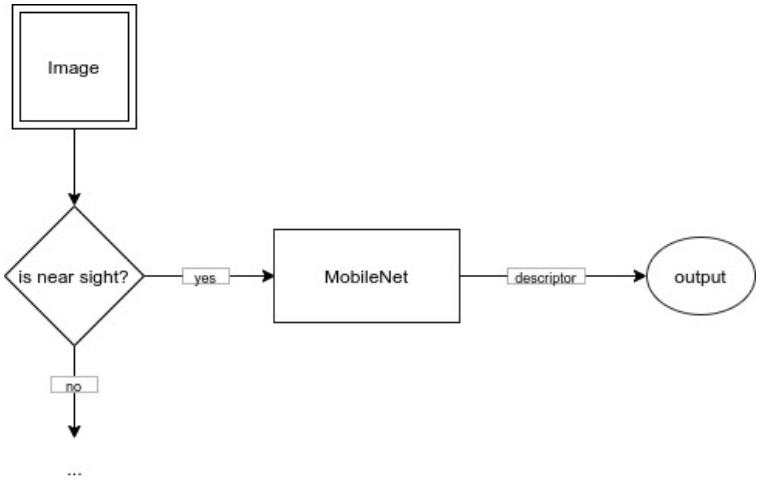


Рис. 3.2 — схема создания дескриптора изображения

Сформированные дескрипторы были записаны в csv файл и в дальнейшем перенесены базу данных, для последующего анализа. В базе данных имеются две таблицы (рис. 3.3) Sights и Descriptors.

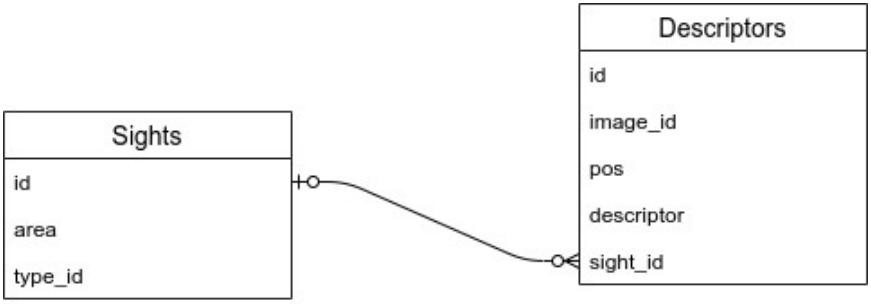


Рис. 3.3 — схема базы данных с дескрипторами

Для описания информации о дескрипторах в таблице Descriptors предусмотрены колонки image_id (идентификатор изображения в базе), pos (географические координирнаты изображения), descriptor (дескриптор изображения) и sight_id (идентификатор достопримечательности, к который относится данной изображение), а в таблице Sight — area (географическая область достопримечательности), type_id (идентификатор типа достопримечательности).

3.5. Классификация с использованием kneighbors classifier

После подготовки дескрипторов необходимо обучить классификатор различать изображения по классам достопримечательностей. Для этого использовалось три подхода: классификация с помощью нейронной сети, с помощью kneighbors классификатора и с помощью алгоритма поиска схожих объектов. В этой работе будут рассмотрены последние два.

Для создания классификатора на основе метода k -ближайших соседей использовалась библиотека sklearn. Основные функции и классы данного скрипта представлены в таблице 3.2.

Название функции/класса	*Основной функционал*
`split_data(data, percent, count)`	Разделение данных на обучающие и тестовые с определённым процентом (`percent`)
`get_data(data_path)`	Получение списка дескрипторов из файла (`data_path`) и соответствующих им меток
`show_data(data_path)`	Использование метода главных компонента (`PCA`) для визуализации и анализа данных из файла (`data_path`)
`AutoClassifiers`	Класс для классификации данных с использованием различных метрик и представлением результата классификации, а также его точности
`mahalanobis_dist(a, b)`	Нахождение расстояния Махаланобиса между векторами `a` и `b`
`collect_data(data_output_path)`	Сбор данных в формате `json` (в файл `data_output_path`) в виде списка идентификаторов дескрипторов, относящихся к определённому классу и расстояний между ними

Перед самим процессом классификации необходимо провести анализ данных, на основе которого можно подобрать метрику, количесво классов и другие гиперпараметры. В этом может помочь метод главных компонент PCA.

После проведённого анализа необходимо провести обучение и тестирование классификатора, благодаря взаимосвязи функций и классов получается достичь функционирование модели согласно её назначению (рис. 3.4)

Таблица 3.2 — основные функции и классы классификатора

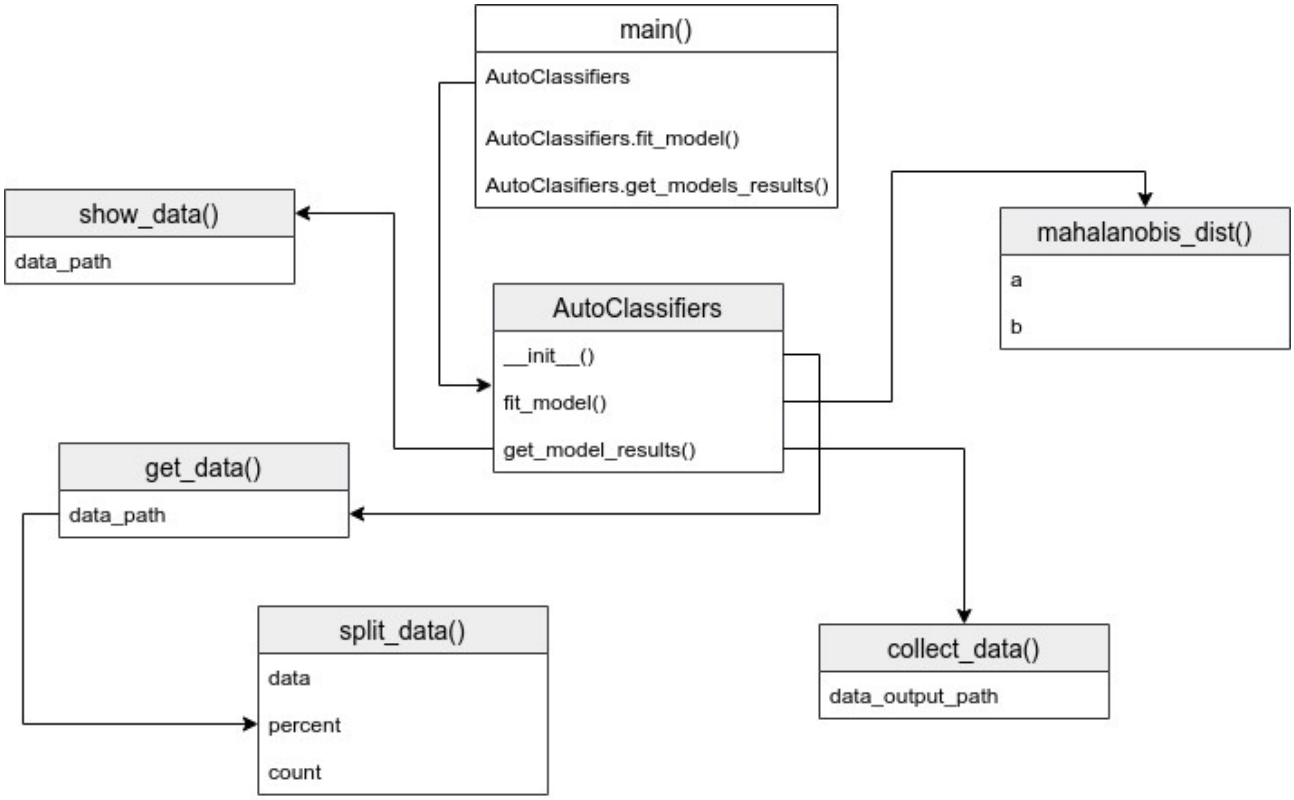


Рис. 3.4 — взаимосвязь между функциями и классами в модели классификатора

Программная реализация данной модели на языке python приведена в приложении 2. Данная модель работает согласно алгоритму, приведённому на блок-схеме (рис. 3.5).

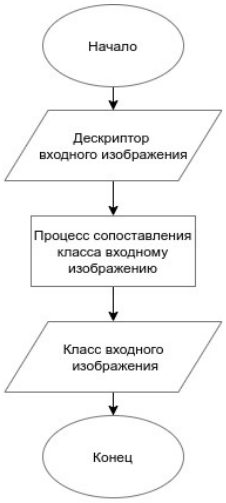


Рис. 3.5 — блок-схемы обучения и работы классификатора

3.6 Вычисление координат изображения на основе его класса

После того как был получен результат принадлежности к классу определённой достопримечательности, необходимо найти координаты, где было сделано входное изображение. Это достигается путём поиска m схожих дескрипторов из данного класса с входным. После этого вычисляется расстояние (Махаланобиса или Евклидово) от исходного вектора до схожих с ним, тем самым формируя некие коэффициенты, которые нормируются, и при подсчёте расстояния берётся взвешенное среднее $pos = \frac{\sum_{i=1}^{m} pos_i \cdot dist_i}{m}$, где pos_i — место положение i -го схожего дескриптора, а $dist_i$ — коэффициент обратный нормированному расстоянию.

Основные функции данного скрипта представлены в таблице 3.3.

Название функции/класса	*Основной функционал*
`convert_str_desc(s)`	Преобразование строки дескриптора `s` в вектор признаков
`avg_val_of_vec(vec1, vec2)`	Вычисление среднего значение двух векторов `vec1` и `vec2`
`get_manhattan_dist(descr1, descr2)`	Вычисление манхэттенского расстояния между двумя дескрипторами `descr1` и `descr2`
`get_sq_of_euclid_dist(descr1, descr2)`	Вычисление квадрата Евклидового расстояние между двумя дескрипторами `descr1` и `descr2`
`split_by_labels(data, labels)`	Разбиение данных (`data`) по меткам (`labels`)
`unite_by_labels(data)`	Объединение данных (`data`) вместе с метками
`get_nearest_descr_from_all(limit, cur_descr, dist_metric)`	Получение `limit` ближайших дескрипторов из всех к `cur_descr` с использованием метрики `dist_metric`
`get_avg_dist_pos(data, metadata)`	Получение усреднённых координат изображения на основе данных о нём (`data`) и данных о ближайших дескрипторах (`metadata`)

Таблица 3.3 — назначение функций в программной реализации скрипта, который вычисляет координаты на основе класса изображения

Взаимосвязь данных функций представлена на рисунке 3.6.

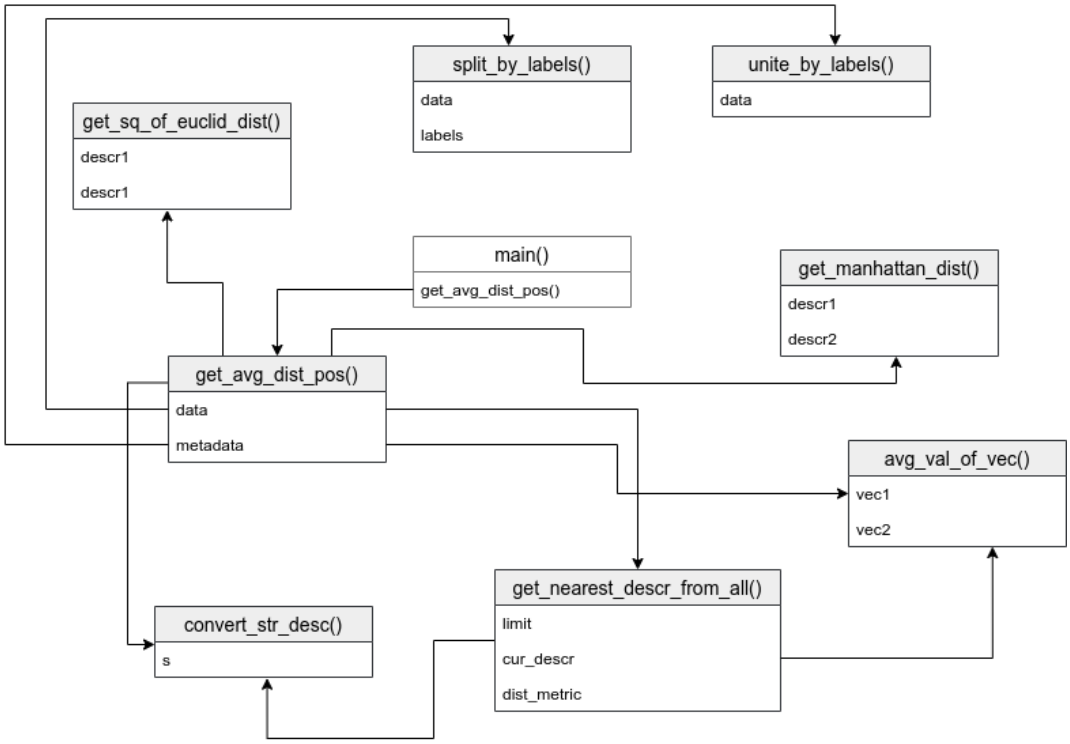


Рис. 3.6 — взаимосвязь между функциями и в программной реализации скрипта, который вычисляет координаты на основе класса изображения

Программная реализация данного скрипта на языке python будет приведена в приложении 3. Данный скрипт работает согласно алгоритму, приведённому на блок-схеме (рис. 3.6).

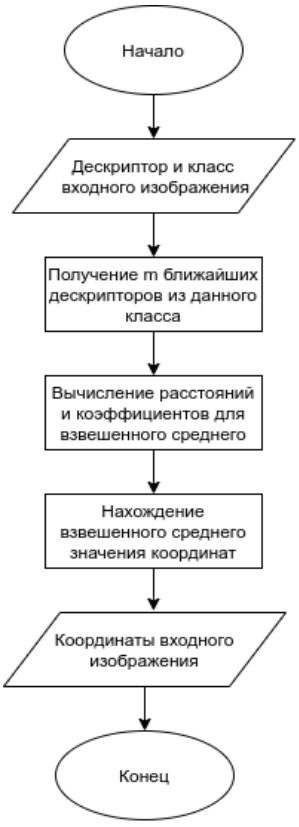


Рис. 3.6 — блок-схема работы скрипта

3.7 Классификация изображений с использованием метода обнаружения объектов

Для того чтобы получить класс входного изображения можно также использовать метод обнаружения определённых объектов, которые характеризуют конкретный класс в целом. Это можно реализовать с помощью алгоритма ORB. Но для этого необходимо сформировать базу ключевых объектов для каждого класса, чтобы охарактеризовать их ключевыми точкам. При этом база объектов должна включать многогранное покрытие своего класса без ощутимых помех.

Алгоритм для поиска объектов включен в библиотеку CV2. Основываясь на нём была разработана модель. Основные функции и классы данной модели представлены в таблице 3.4.

Название функции/класса	*Основной функционал*
`get_base_of_img(imgs_path)`	Получение базы объектов для каждого класса по пути `imgs_path`
`orb_predict(imgs_path, target_img_path, k)`	Получение результата работы классификатора основанного на сравнивании `k` базовых изображений (`imgs_path`), характерны для данного класса с исходным (`target_img_path`)

Таблица 3.4 — назначение функций в программной реализации моде классификации по средством сравнивания наличия объектов, характерных для данного класса

Взаимосвязь данных функций представлена на рисунке 3.7.

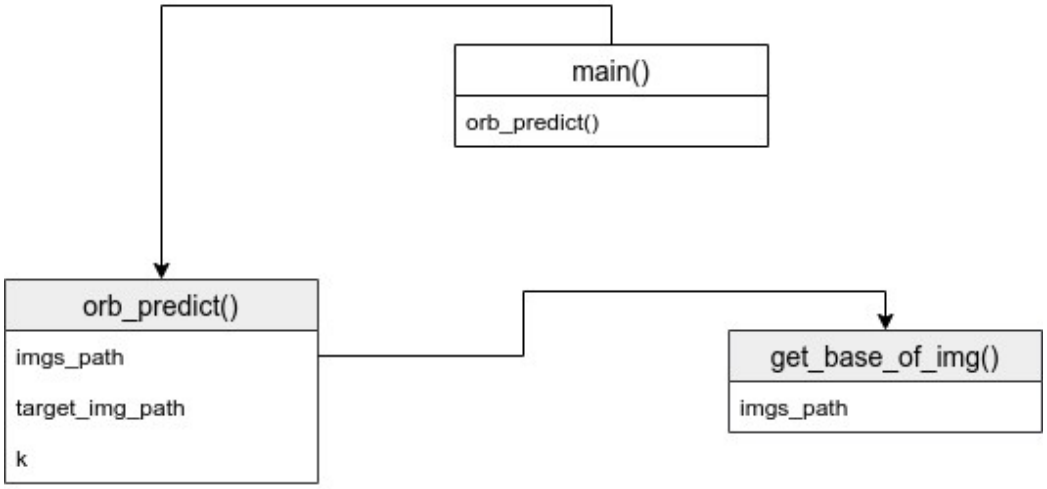


Рис. 3.7 — взаимосвязь между функциями и в программной реализации

Программная реализация данной модели на языке python будет приведена в приложении 4. Данная модель работает согласно алгоритму, приведённому на блок-схеме (рис. 3.8).

Рис. 3.8 — блок-схема, описывающая алгоритм работы модели

3.8 Выводы

В данном разделе была спроектирована архитектура приложения и его реализация. В начале были разобраны модели, которые составляют классификатор в целом. Описана структура программной реализации, включая модули и их назначение, а также приведен алгоритм работы каждой модели.

Раздел IV. Практическое применение и экспериментальное исследование модели

В данном разделе рассматривает применение модели распознавания координат по фотографии, а также проводится обзор точности работы алгоритма.

4.1. Экспериментальное исследование работы моделей классификаторов

Данные модели, принцип работы которых был описан выше, должны выдавать координаты мировой достопримечательности по фотографии. В процессе работы были разработаны три классификатора (на основе нейронной сети прямого распространения, на основе метода k -ближайших соседей и на основе алгоритма поиска схожих объектов на изображении).

Можно заметить, что модель основанная на нейронной сети довольно успешно справляется с достопримечательностями, которые присутствуют в базе, однако при попытке подать на вход изображение отсутствующее в базе происходит ошибочное предсказание. Это связанно с тем, что усреднение координат ведётся по уже существующим в базе изображения.

Намного хуже показывает себя модель основанная на алгоритме сравнения схожих объектов на изображении. (рис. 4.1 — 4.3)

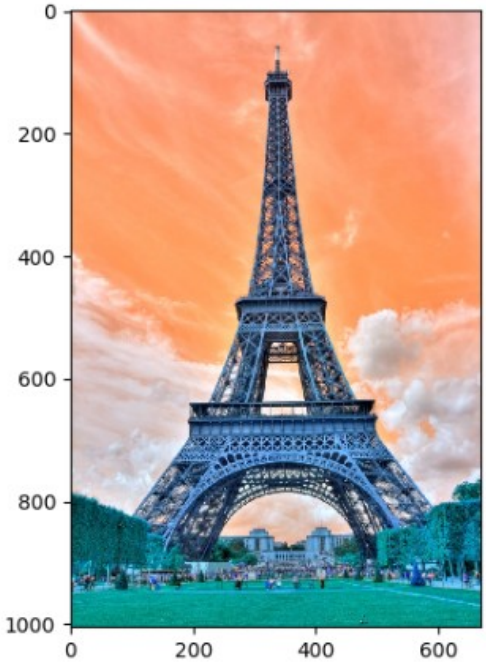


Рис. 4.1 — изображение подаваемое на вход модели

Рис. 4.2 — самое похожее изображение по содержанию ключевых объектов

```
(58.36666666666667, '26519195', '3')
```

Рис. 4.3 — результат работы модели: «расстояние» между ключевыми объектами класса и ключевыми объектами изображения, идентификатор изображения, метка класса

Как видно, что результаты работы очень далеки от ожидаемого. Это можно объяснить наличием большого числа уникальных точек для каждого изображения в определённом классе. Более подробно описание данных результатов будет приведено в анализе точности моделей.

Результат работы модели классификатора основанного на методе k -ближайших соседей очень схож с результатом работы нестроевой модели, однако нейронная сеть способна быстро масштабироваться в отличие от kneighbors классификатора.

4.2. Анализ точности работы классификаторов

В экспериментальном исследовании модель основанная на на алгоритме сравнения схожих объектов на изображении показала себя очень плохо. Так как при попытки выделить особые точки на изображениях, для их дальнейшего сравнивания , получилось так, что некоторые особые точки идентичны почти всем. И так, например, изображение на рисунке 4.5 содержит достаточное кол-во особых точек, чтобы совпасть почти со всеми объектами. Однако при сравнивании похожих объектов модель показывает хорошие результаты (рис. 4.4 — 4.6).

Рис. 4.4 — результаты «матчинга» (сравнения) особых точек

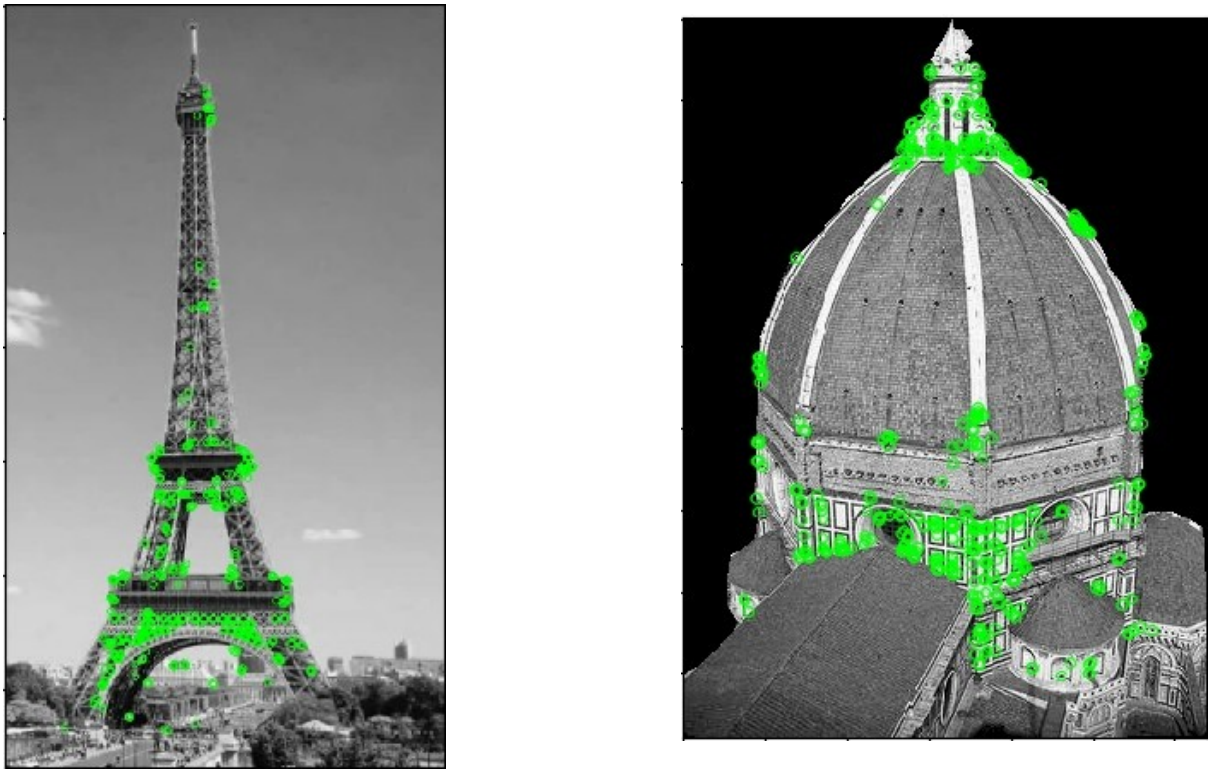


Рис. 4.5 — результаты выделения особых точек

Рис. 4.6 — результаты «матчинга» (сравнения) особых точек

Изображения берутся черно-белыми, так как цвет только мешает выделению особых точек.

При анализе результатов модели основанной на методе k -ближайших соседей следует учесть попытку «визуализировать» данный с помощью метода главных компонент (PCA) (рис. 4.7).

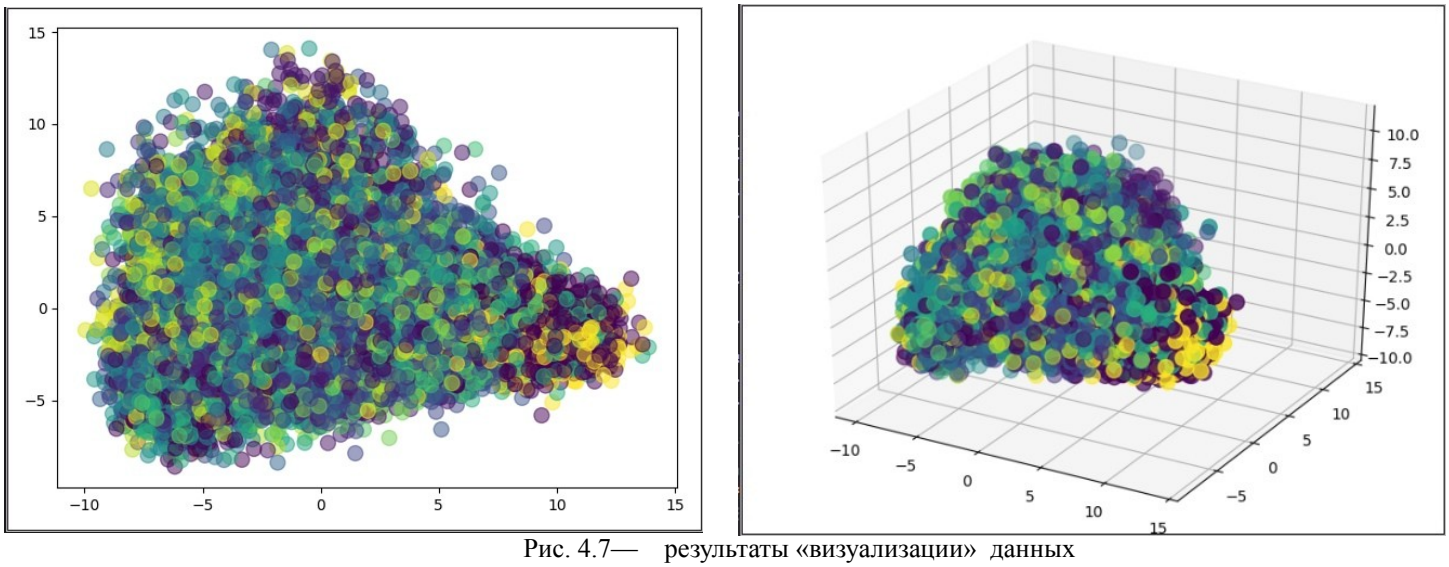


Рис. 4.7— результаты «визуализации» данных

Конечно нельзя избежать потерю части данных при попытке снизить размерность, однако можно сделать вывод, что качество не является эталонным.

По графику дисперсии можно сделать вывод, что большая часть весовых коэффициентов стремится к левой части графика, что свидетельствует о нормальном разбросе случайной величины (рис. 4.8).

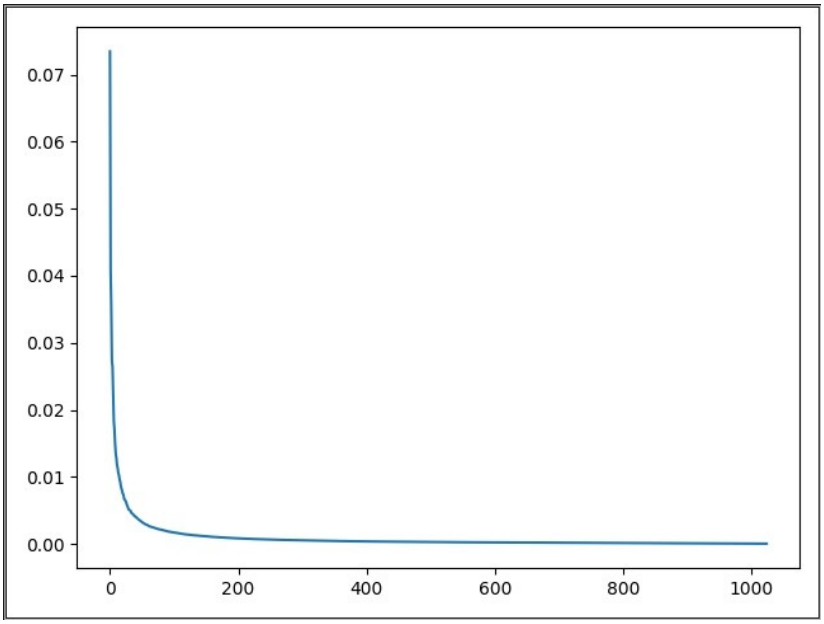


Рис. 4.8 — график дисперсии данных

Из графика накопительной дисперсии видно, что при 1000 компонентах график стремится к 1, таким образом, 1000 базисных компонент почти не дают потерю данных, следовательно размерность данных можно уменьшить до 1000 (рис. 4.9).

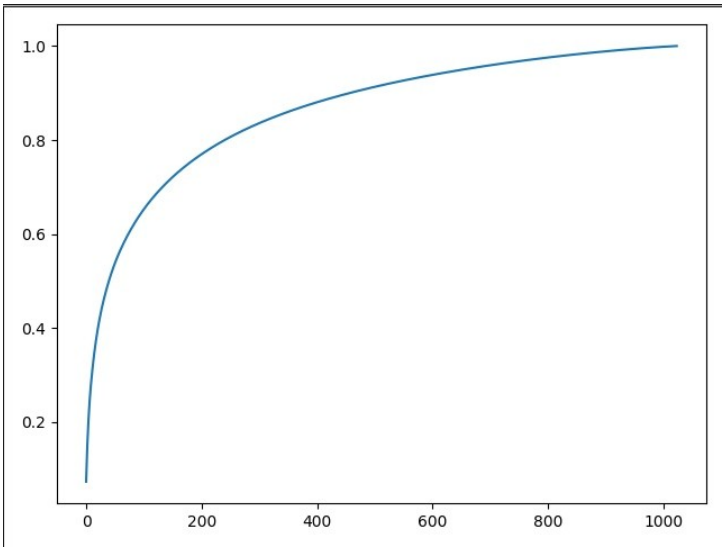


Рис. 4.9 — график накопительной дисперсии данных

Однако если отобрать 5 топовых классов по количеству изображений в них, то распределение будет немного лучше (рис. 4.10).

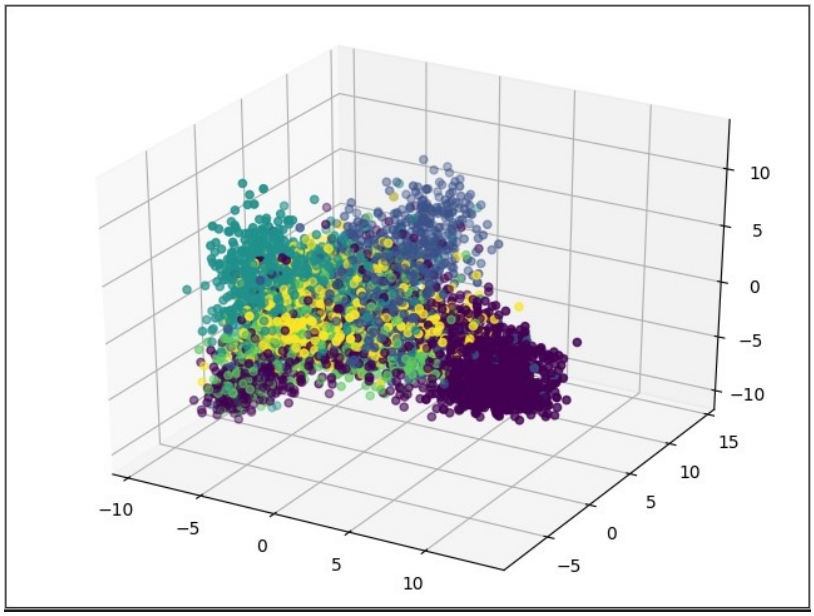
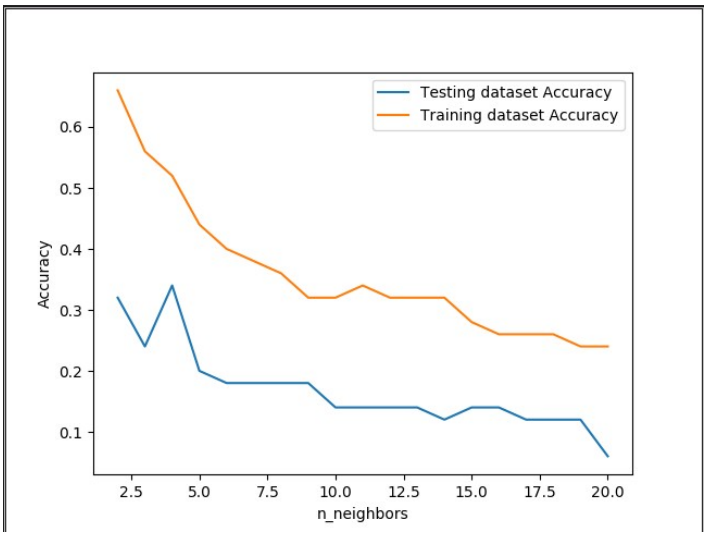


Рис. 4.10 — результаты «визуализации» данных

Исходя из анализа данных можно подобрать параметры для модели основанной на методе k -ближайших соседей и попытаться сравнить точность данной модели при разных значениях k (рис. 4.11).



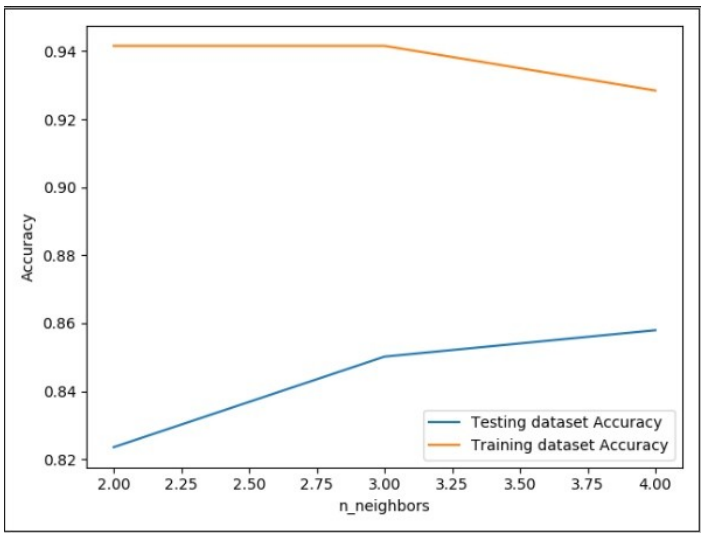


Рис. 4.11 — точность модели при разных параметрах k (n_neighbors)

Исходя из этих графиков, можно сделать вывод, что при отборе 5 топовых классов по количествам изображение точность классификатора достаточно большая, однако при большем количестве классов точность падает.

Анализ результатов и точности работы модели основанной на нейронной сети прямого распространения в данной работе не рассматривается.

4.3. Выводы

В данном пункте проведен анализ работы моделей с исследованиями точности выполнения поставленной задачи, на основании которых можно сделать вывод о корректной работе в целом.

Заключение

Результатом данной работы является проектирование и реализация модели для её использования в определении местоположения по фотографии.

В процессе достижения поставленной цели были решены следующие задачи:

- Проведено исследование проблемной области с целью получения необходимых знаний для дальнейшего проектирования системы и реализации алгоритмов;
- Проанализированы существующие методы определения местоположения по фотографии, с целью изучения границ применимости, а также гибкости различных алгоритмов;
- Построены модели, которые позволяют определять местоположение популярных достопримечательностей по фото;
- Реализована программная часть данной модели.

Практическая значимость данных программных средств заключается в том, что в современном мире многие средства автоматизации зависят от положения в пространстве, поэтому применение данной модели может улучшить данные средства.

В перспективе предлагается проводить дальнейшие исследования в данной области и решить следующие задачи:

- Определение местоположения по фотографии, не содержащей достопримечательностей;
- Определение местоположения по фотографии содержащей дорожные знаки, вывески, разметку, номера домов, название улиц и так далее;
- Повышение точности работы модели.

Список литературы

1. Анализ данных и регрессия: второй курс в статистике / Фредерик Мостеллер (Mosteller, Frederick) и Джон У. Тьюки (John W. Tukey) ;
2. Image processing with neural networks — a review / Egmont-Petersen, M., de Ridder, D., Handels ;
3. Прикладная статистика: классификация и снижение размерности. / С.А. Айвазян, В.М. Бухштабер ;
4. Data Clustering: Algorithms and Applications / Charu C. AggarwalChandan K. Reddy ;
5. Кластеризация данных / Котов А., Красильников Н. (2006) ;
6. Лекции по методу опорных векторов / К. В. Воронцов (21 декабря 2007 г.) ;
7. Глубокое обучение на Python / Франсуа Шолле ;
8. http://www.deeplearningbook.org/contents/autoencoders.html ;
9. https://www.cs.ccu.edu.tw/~wtchu/projects/Weather/index.html .

Приложение 1. Реализация скрипта, для описания моделей БД

```
from peewee import *


user = 'geotagging_db_user'
password = '123456'
db_name = 'geotagging_db'
db_host = '212.24.111.5'

db_handler = MySQLDatabase(db_name,
                           user=user,
                           password = password,
                           host=db_host)

class BaseModel(Model):
class Meta:
     database = db_handler

class Sight(BaseModel):
id = PrimaryKeyField(null=False)
area = CharField(max_length=30)
type_id = IntegerField()
db_table = "sights"

class Meta:
     db_table = "sights"

class Descriptor(BaseModel):
id = PrimaryKeyField(null=False)
image_id = CharField(max_length=30, unique=True)
descriptor = CharField(max_length=40000)
sight_id = ForeignKeyField(Sight, backref='descrs')
db_table = "descriptors"

class Meta:
     db_table = "descriptors"

class Streets(BaseModel):
id = PrimaryKeyField(null=False)
pos = CharField(max_length=40, unique=True)
house_number = CharField(max_length=50)
street = CharField(max_length=15000)
city = CharField(max_length=50)
class Meta:
     db_table = "streets"
```

Приложение 2. Реализация модели классификатора, основанного на методе k-ближайших соседей

```
from sklearn.neighbors import KNeighborsClassifier
from tqdm import tqdm
from sklearn.decomposition import PCA
from mpl_toolkits.mplot3d import Axes3D

import numpy as np
import matplotlib.pyplot as plt
import argparse
import pylab as pl
import json


classes_count = 20


def split_data(data, percent=10, count=-1):
if count != -1:
     data = data[:count]
train_cnt = int(len(data[0]) * ((100 - percent) / 100))
return    data[0][:train_cnt],    data[0][train_cnt:],    data[1]
[:train_cnt], data[1][train_cnt:]


def get_data(data_path):
with open(data_path, "r") as read_file:
     data = json.load(read_file)
x = list()
y = list()
for i in data:
     x.append(data[i]['descr'])
     y.append(data[i]['label'])
return x, y
```

```
def collect_data(data_output_path):
with open(data_output_path, "w") as write_file:
     data = dict()
     x, y = get_data(args.csv_path)
     for i in range(len(x)):
          data[i] = {'descr': list(x[i]), 'label': int(y[i])}
     json.dump(data, write_file)


def show_data(n_classes=classes_count, d_data=None):
if d_data is None:
     d_data = list()
x_train = d_data[0]
y_train = d_data[1]
pca = PCA()
reduced = pca.fit_transform(x_train)
fig = plt.figure()
ax = Axes3D(fig)
ax.scatter(reduced[:, 0], reduced[:, 1], reduced[:, 2], c=y_train)
pl.show()


def mahalanobis_dist(a, b):
xx = a.T
yy = b.T
e = xx - yy
X = np.vstack([xx, yy])
V = np.cov(X.T)
if np.linalg.det(V) == 0:
     p = np.eye(V.shape[0])
else:
     p = np.linalg.inv(V)
```

```
D = np.sqrt(np.sum(np.dot(e, p) * e))
return D


class AutoClassifiers:

def __init__(self, data_path, graph_path, json_output_path,
metric=None, limit=None):
     self.data_path = data_path
     self.graph_path = graph_path
     self.json_output_path = json_output_path
     self.metric = metric
     self.classes_count = classes_count
     self.limit = limit
     self.classification_results = dict()

def fit_models(self):
     data = get_data(self.data_path)
     x_train, x_test, y_train, y_test = split_data(data,
percent=15, count=10000)
     if self.limit is not None:
          x_train = x_train[:self.limit]
          x_test = x_test[:self.limit]
          y_train = y_train[:self.limit]
          y_test = y_test[:self.limit]
          data = data[0][:self.limit]
     else:
          data = data[0]
     neighbors = np.arange(2, self.classes_count + 1)
     train_accuracy = np.empty(len(neighbors))
     test_accuracy = np.empty(len(neighbors))
     for i, k in tqdm(enumerate(neighbors)):
          if self.metric is None:
```

```
                knn = KNeighborsClassifier(n_neighbors=k, n_jobs=-
1)
          elif self.metric == 'mahalanobis':
                knn      =      KNeighborsClassifier(n_neighbors=k,
algorithm='brute',    metric='mahalanobis',    metric_params={'V':
np.cov(x_train)}, n_jobs=-1)
          else:
                knn      =      KNeighborsClassifier(n_neighbors=k,
metric=self.metric, n_jobs=-1)
          knn.fit(x_train, y_train)
          train_accuracy[i] = knn.score(x_train, y_train)
          test_accuracy[i] = knn.score(x_test, y_test)
          neighbors_res = knn.kneighbors(data, n_neighbors=k)
          self.classification_results[str(k)]                    =
list(zip(list(map(lambda x: list(x), neighbors_res[0])), list(
                map(lambda x: list(map(lambda y: int(y), list(x))),
neighbors_res[1]))))

     plt.plot(neighbors,  test_accuracy,  label='Testing  dataset
Accuracy')
     plt.plot(neighbors,  train_accuracy,  label='Training  dataset
Accuracy')
     plt.legend()
     plt.xlabel('n_neighbors')
     plt.ylabel('Accuracy')
     plt.savefig(self.graph_path)

def get_models_results(self):
     with open(self.json_output_path, "w") as write_file:
          json.dump(self.classification_results, write_file)


if __name__ == '__main__':
parser = argparse.ArgumentParser()
```

```
parser.add_argument('--data_path', type=str, dest='data_path')
parser.add_argument('--graph_path', type=str, dest='g_path')
parser.add_argument('--json_output_path',                type=str,
dest='json_output_path')

args = parser.parse_args()

clustering_model    =    AutoClassifiers(data_path=args.data_path,
json_output_path=args.json_output_path,    graph_path=args.g_path,
limit=50)
clustering_model.fit_models()
clustering_model.get_models_results()
```

Приложение 3. Реализация модели классификатора, основанного на методе сопоставления объектов на изображении

```
import argparse
import sys
import cv2
import os

from matplotlib import pyplot as plt


def get_base_of_img(imgs_path):
imgs_folders = dict()
for folder in os.listdir(imgs_path):
     imgs_folders[folder]    =    os.listdir(os.path.join(imgs_path,
folder))
return imgs_folders


def orb_predict(imgs_path, target_img_path, k):
imgs_folders = get_base_of_img(imgs_path)
t_img = cv2.imread(target_img_path, cv2.IMREAD_GRAYSCALE)
sights_matches = dict()
orb = cv2.ORB_create()
bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
kp2, des2 = orb.detectAndCompute(t_img, None)
for sight_id, imgs_loc in imgs_folders.items():
     sights_matches[sight_id] = dict()
     for img_loc in imgs_loc:
          img    =    cv2.imread(os.path.join(imgs_path,    sight_id,
img_loc), cv2.IMREAD_GRAYSCALE)
          kp1, des1 = orb.detectAndCompute(img, None)
          matches = bf.match(des1, des2)
          img_id = img_loc.split('.')[0]
```

```
          sights_matches[sight_id][img_id]     =     sorted(matches,
key=lambda x: x.distance)
min_avg_dist = sys.maxsize
sim_img_id = ''
sim_sight_id = ''
for sight_id in sights_matches.keys():
     for img_id in sights_matches[sight_id].keys():
          avg_dist = 0
          for i in range(k):
               avg_dist     +=     sights_matches[sight_id][img_id]
[i].distance
          avg_dist /= k
          if avg_dist < min_avg_dist:
               min_avg_dist = avg_dist
               sim_img_id = img_id
               sim_sight_id = sight_id
sim_img    =    cv2.imread(os.path.join(imgs_path,    sim_sight_id,
sim_img_id + '.png'))
plt.imshow(sim_img)
plt.show()
return (min_avg_dist, sim_img_id, sim_sight_id)


if __name__ == '__main__':
parser = argparse.ArgumentParser()
parser.add_argument('--imgs_path', type=str, dest='imgs_path')

args = parser.parse_args()
some_img                                                              =
cv2.imread('/media/qunity/Workspace/Python_projects/qwe-test-
project/geotagging_project/imgs/paris1.jpg')
plt.imshow(some_img)
plt.show()
res = orb_predict(args.imgs_path,
```

```
                    '/media/qunity/Workspace/Python_projects/qwe-
test-project/geotagging_project/imgs/paris1.jpg',
                60)
print(res)
```

Приложение 4. Реализация скрипта для вычисления координат изображения на основе его класса

```
import numpy as np
import json
from sql_models import *
from tqdm import tqdm
from sklearn.cluster import OPTICS, MeanShift, KMeans


def convert_str_desc(s):
desc = []
buff = ''
for i in range(len(s)):
     if s[i] == ',':
          desc.append(float(buff))
          buff = ''
     else:
          buff += s[i]
desc.append(float(buff))
return desc


def avg_val_of_vec(vec1, vec2):
n = min(len(vec1), len(vec2))
vec3 = []
for i in range(n):
     vec3.append((vec1[i] + vec2[i]) / 2)
return vec3


def convert_str_sight(s):
return int(str(s))
```

```
def get_manhattan_dist(descr1, descr2):
dist = 0
for i in range(min(len(descr1), len(descr2))):
     dist += abs(descr1[i] - descr2[i])
return dist


def get_sq_of_euclid_dist(descr1, descr2):
dist = 0
for i in range(min(len(descr1), len(descr2))):
     dist += (descr1[i] - descr2[i]) ** 2
return dist


def split_by_labels(data, labels):
count = int(np.max(labels))
result = [[] for i in range(count + 1)]
for i, x in enumerate(data):
     result[int(labels[i])].append(x)
return result


def unite_by_labels(data):
labels = list()
result = list()
for i, cluster in enumerate(data):
     for x in cluster:
          labels.append(i)
          result.append(x)

return result, labels


def score_clustering(x, labels, proccess_Din=np.max):
```

```
clusters = split_by_labels(x, labels)
Din = list()
avgs = list()
for cluster in clusters:
     avgs.append(np.average(cluster))
     Din.append(np.var(cluster))

Dout = np.var(avgs)
return proccess_Din(Din) / Dout



def get_nearest_descr_from_all(limit, cur_descr, dist_metric):
if limit == -1:
     descrs = Descriptor.select()
else:
     descrs = Descriptor.select().limit(limit)
if dist_metric == 'manhattan':
     dist                                                    =
get_manhattan_dist(convert_str_desc(descrs[0].descriptor),
cur_descr)
     nearest_descr = convert_str_desc(descrs[0].descriptor)
     sight = convert_str_sight(descrs[0].sight_id)
     for row in descrs:
          desc = convert_str_desc(row.descriptor)
          if get_manhattan_dist(desc, cur_descr) < dist:
               dist = get_manhattan_dist(desc, cur_descr)
               nearest_descr = desc
               sight = convert_str_sight(row.sight_id)
elif dist_metric == 'square_of_euclid':
     dist                                                    =
get_sq_of_euclid_dist(convert_str_desc(descrs[0].descriptor),
cur_descr)
     nearest_descr = convert_str_desc(descrs[0].descriptor)
     sight = convert_str_sight(descrs[0].sight_id)
```

```
     for row in descrs:
          desc = convert_str_desc(row.descriptor)
          if get_sq_of_euclid_dist(desc, cur_descr) < dist:
               dist = get_sq_of_euclid_dist(desc, cur_descr)
               nearest_descr = desc
               sight = convert_str_sight(row.sight_id)
return (nearest_descr, sight)


def get_unic_sigth_id(limits):
sights_req = Sight.select(Sight.id).limit(limits)
sights = list()
for row in sights_req:
     sights.append(convert_str_sight(row.sight_id))
return sights


def get_all_descr_from_sight(sight_id):
descrs_req   =   Descriptor.select().where(Descriptor.sight_id   ==
sight_id)
descrs = list()
imgs_id = list()
for row in descrs_req:
     descrs.append(convert_str_desc(row.descriptor))
     imgs_id.append(int(row.image_id))
return (descrs, imgs_id)


def sum_descr(descr1, descr2):
n = len(descr1)
s_descr = [0] * n
for i in range(n):
     s_descr[i] = descr1[i] + descr2[i]
return s_descr
```

```
def get_similar_pos(data, metadata):
sim_descrs = data[0]
dist = data[1]
imgs_id = data[2]
coeffs = list()
for i in range(len(dist)):
     if dist[i] == 0:
          avg_vec = sim_descrs[i]
          tdist = get_manhattan_dist(avg_vec, sim_descrs[0])
          img_id = imgs_id[0]
          for i in range(1, len(sim_descrs)):
               if   get_manhattan_dist(sim_descrs[i],   avg_vec)   <
tdist:
                    tdist    =    get_manhattan_dist(sim_descrs[i],
avg_vec)
                    img_id = imgs_id[i]
          with open(metadata, "r") as read_file:
               mdata = json.load(read_file)
          for i in range(len(mdata)):
               if int(mdata[i]['id']) == img_id:
                    return mdata[i]['loc']
     else:
          coeffs.append(1 / dist[i])
sum_coeff = sum(coeffs)
avg_vec = [0] * len(sim_descrs[0])
for i in range(len(sim_descrs)):
     avg_vec   =   sum_descr(avg_vec,   list(map(lambda   x:   x   *
coeffs[i], sim_descrs[i])))
avg_vec = list(map(lambda x: x * (sum_coeff ** -1), avg_vec))
tdist = get_manhattan_dist(avg_vec, sim_descrs[0])
img_id = imgs_id[0]
for i in range(1, len(sim_descrs)):
```

```
     if get_manhattan_dist(sim_descrs[i], avg_vec) < tdist:
          tdist = get_manhattan_dist(sim_descrs[i], avg_vec)
          img_id = imgs_id[i]
with open(metadata, "r") as read_file:
     mdata = json.load(read_file)
for i in range(len(mdata)):
     if int(mdata[i]['id']) == img_id:
          return mdata[i]['loc']


def get_avg_dist_pos(data, metadata):
sim_descrs = data[0]
dists = data[1]
imgs_id = data[2]
with open(metadata, "r") as read_file:
     mdata = json.load(read_file)
coordinates = list()
for j in range(len(imgs_id)):
     for i in range(len(mdata)):
          if int(mdata[i]['id']) == imgs_id[j]:
               coordinates.append([mdata[i]['loc']['lat'],
mdata[i]['loc']['lng']])
               break
if len(coordinates) == 1:
     return coordinates[0]
coeffs = list()
for i in range(len(dists)):
     if dists[i] == 0:
          avg_pos = dists[i]
          sim_pos = coordinates[0]
          sim_img_id = imgs_id[0]
          for i in range(1, len(coordinates)):
               if  get_manhattan_dist(coordinates[i],  avg_pos)  <
get_manhattan_dist(sim_pos, avg_pos):
```

```
                    sim_pos = coordinates[i]
                    sim_img_id = imgs_id[i]
          return avg_pos
     else:
          coeffs.append(1 / dists[i])
avg_pos = [0] * len(coordinates[0])
for i in range(len(coordinates)):
     avg_pos = sum_descr(avg_pos, list(map(lambda x: x *
coeffs[i], coordinates[i])))
coeff_sum = sum(coeffs)
avg_pos = list(map(lambda x: (coeff_sum ** -1) * x, avg_pos))
sim_pos = coordinates[0]
sim_img_id = imgs_id[0]
for i in range(1, len(coordinates)):
     if get_manhattan_dist(coordinates[i], avg_pos) <
get_manhattan_dist(sim_pos, avg_pos):
          sim_pos = coordinates[i]
          sim_img_id = imgs_id[i]
return avg_pos


def get_data_about_similar_descrs(sigth_id, c_desc, n):
descrs, b_imgs_id = get_all_descr_from_sight(sigth_id)
sim_descr = list()
dist = list()
imgs_id = list()
j = 0
for desc in descrs:
     if len(sim_descr) < n:
          sim_descr.append(desc)
          dist.append(get_manhattan_dist(desc, c_desc))
          imgs_id.append(b_imgs_id[j])
     else:
          for i in range(len(sim_descr)):
```

```
                if get_manhattan_dist(desc, c_desc) < dist[i]:
                    dist[i] = get_manhattan_dist(desc, c_desc)
                    sim_descr[i] = desc
                    imgs_id[i] = b_imgs_id[j]
                    break
    j += 1
return (sim_descr, dist, imgs_id)


def get_similar_sight_id(descr, avgs_file):
with open(avgs_file, "r") as read_file:
    avg_val_of_vecs = json.load(read_file)
t_sight_id = next(iter(avg_val_of_vecs))
dist = get_manhattan_dist(avg_val_of_vecs[t_sight_id][0], descr)
for sight_id in avg_val_of_vecs.keys():
    if  get_manhattan_dist(avg_val_of_vecs[sight_id][0],  descr)  <
dist:
        dist  =  get_manhattan_dist(avg_val_of_vecs[sight_id][0],
descr)
        t_sight_id = sight_id
return t_sight_id


def get_avg_val_of_descrs(limits, fout):
if limits == -1:
    descrs_req = Descriptor.select()
else:
    descrs_req = Descriptor.select().limit(limits)
base_table = {}
for row in tqdm(descrs_req):
    desc = convert_str_desc(row.descriptor)
    sight_id = convert_str_sight(row.sight_id)
    if not sight_id in base_table:
        base_table[sight_id] = []
```

```
            base_table[sight_id].append(desc)
        else:
            avg_desc = avg_val_of_vec(base_table[sight_id][0], desc)
            base_table[sight_id].pop(0)
            base_table[sight_id].append(avg_desc)
with open(fout, "w") as fp:
        json.dump(base_table, fp)


def sight_cluster(x, algo):
clust = None
if algo == 'optics':
        clust       =       OPTICS(metric='manhattan',       n_jobs=-1,
min_cluster_size=50)
if algo == 'meanshift':
        clust = MeanShift(cluster_all=True, n_jobs=-1)
if algo == 'kmeans':
        clust = KMeans(n_clusters=4, n_jobs=-1)

return clust.fit_predict(x)
```