МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

«Тверской государственный университет»

Факультет прикладной математики и кибернетики

Кафедра информационных технологий

**ОТЧЕТ**

**по преддипломной практике**

Направление: *02.04.02 «Фундаментальная информатика и*

*информационные технологии»*

Магистерская программа: *«Информационные технологии в*

*управлении и принятии решений»*

Тема: «Сопоставление ключевых точек изображений с помощью нейронных сетей»

|  |  |
| --- | --- |
|  | **Выполнил:** студент 2 курса магистратурыИванов Артём Николаевич**Научный руководитель:**кандидат физ.-мат. наук,Сорокин Сергей Владимирович**Оценка:** |
|  Тверь – 2018 г. |  |

**ОГЛАВЛЕНИЕ**

[Введение 2](#_Toc514840485)

[Цели и задачи 2](#_Toc514840486)

[Актуальность 3](#_Toc514840487)

[Структура работы 4](#_Toc514840488)

[1. Задача сопоставления ключевых точек 5](#_Toc514840489)

[1.1. Основные определения и общее решение задачи 5](#_Toc514840490)

[1.2. Оценка состояния темы исследования 7](#_Toc514840491)

[2. Разработка архитектуры нейронной сети 8](#_Toc514840492)

[2.1. Определение и основные термины 8](#_Toc514840493)

[2.2. Общая структура сети для решения задачи 9](#_Toc514840494)

[2.3. Первоначальные вариации структур базовой нейронной сети 11](#_Toc514840495)

[2.4. Доработка структуры нейронной сети 13](#_Toc514840496)

[3. Доработка архитектуры нейронной сети 16](#_Toc514840497)

[3.1. Подход hard-mining 16](#_Toc514840498)

[3.2. Первая реализация метода hard-mining 17](#_Toc514840499)

[3.3. Результаты первого применения hard-mining 20](#_Toc514840500)

[3.4. Новые изменения сети 21](#_Toc514840501)

[3.5. Вторая реализация метода hard-mining 22](#_Toc514840502)

[3.6. Результаты второго применения метода hard-mining 25](#_Toc514840503)

[4. Генерация обучающей выборки 26](#_Toc514840504)

[4.1. Общий алгоритм получения набора вырезанных областей 26](#_Toc514840505)

[Заключение 28](#_Toc514840506)

[Список использованных источников 29](#_Toc514840507)

# Введение

В наше время очень много внимания уделяется совершенствованию технологий искусственного интеллекта. Большой популярностью пользуются нейронные сети, которые зачастую позволяют добиться лучших результатов[2], чем при использовании традиционных алгоритмов, как, например, в проблеме сопоставления ключевых точек изображений. Данная задача имеет широкий круг применения: создание панорам, создание стереопары и реконструкция трехмерной модели объекта по его двумерным проекциям в принципе, распознавание объектов и поиск по образцу из какой-то базы, слежение за движением объекта по нескольким снимкам, реконструкция аффинных преобразований изображений. Также следует отметить, что область знаний, рассматривающая такого рода задачи (компьютерное зрение) достаточно молода, соответственно еще нет определенного универсального метода, решающего все вышеперечисленные проблемы в полном объеме, т.е. для всех входных изображений. В тоже время существующие алгоритмы для решения данных задач постепенно совершенствуются, в том числе и с применением нейронных сетей.

## Цели и задачи

В данной научно-исследовательской работе поставлена цель:

* создать нейронную сеть для сопоставления ключевых точек изображений.

Для достижения данной цели необходимо выполнить ряд задач:

* ознакомиться с задачей сопоставления ключевых точек
* разработать архитектуру нейронной сети для сопоставления ключевых точек
* провести обучение и в случае необходимости доработку нейронной сети
* сравнить полученные результаты с известными методами
* оформить текст диссертации

В моей работе задача сопоставления ключевых точек рассматривается в контексте выделения структуры из движения (Structure from motion).

Это технология построения трехмерной структуры объекта по набору изображений (2D фотографий). Она является объектом исследования в таких областях как компьютерное зрение и зрительное восприятие (изучение зрительной системы головного мозга) [3].

## Актуальность

Данная технология построения трехмерных структур используется во многих сферах деятельности: медицине, спелеологии, геологии. Существует множество инструментов для решения данной задачи. Наиболее популярными из них являются те, которые строят трехмерные структуры по видео или фото-потоку.

Специалистам требуется стоить трехмерные модели пещер и различных предметов местности. Существующие на сегодняшний день технологии имеют ряд недостатков. Так же показывают плохой результат работы в “специфических” условиях, к которым можно отнести резкий перепад света на последовательных кадрах.

Более того, при построении топологических карт местности данная технология по сравнению с аэросъемкой позволяет добиться большей точности при минимальном количестве затрат.

Также построение трехмерной структуры используется для оценки, мониторинга, представления и реставрации памятников культурного наследия[3].

## Структура работы

Работа состоит из введения, оглавления, трёх глав, заключения и списка использованных источников.

В первой главе даётся подробное описание задачи сопоставления точек и её решения, также описываются основные термины, используемые в рассматриваемой области. Вместе с этим приводится оценка состояния темы исследования.

Во второй главе описывается решение задачи исследования посредством использования нейронных сетей. Даются основные определения, описываются используемые слои и структуры при построении и использовании нейронных сетей при решении задачи. Также в данной главе представлены основные результаты исследования.

Третья глава посвящена разработке обучающей выборки для моделей обучаемых искусственных нейронных сетей. Описывается формат наименования областей точек для формирования выборки и основные преобразования, участвующие в генерации.

# Задача сопоставления ключевых точек

Задача моей научной работы рассматривается в контексте задачи «Structure from motion» - технологии получения трехмерных моделей предметов и карт местности по данным видео и фотосъёмки. В свою очередь её можно разделить на ряд подзадач:

1. Работа детектора: поиск точек интереса (угловых);
2. Работа дескриптора: описание характеристических особенностей найденных точек
3. Поиск соответствия между дескрипторами точек для пар изображений;
4. Фильтрация ложных соответствий;
5. Определение смещения камеры между кадрами по векторам смещения пикселов;
6. Нахождение трёхмерной структуры.

Моя работа сфокусирована на 2-ой и 3-ей подзадачах, которые и выполняют сопоставление ключевых точек.

## Основные определения и общее решение задачи

Прежде чем делать дальнейшее подробное рассмотрение данных пунктов дадим определение основным терминам и понятиям, которые будут использоваться в дальнейшем:

Ключевая точка — это такая точка изображения, которую можно отличить от любой другой точки изображения. На изображении можно выделить 2 типа особых точек: углы и ребра. Самым распространенным типом таких точек являются углы. Они, в отличие от ребер, не зависят от масштаба и расположения изображения[4]. Следовательно, их можно однозначно сопоставить для пар изображений. Углы можно определить, используя различные детекторы.

Детектор — это метод получения особых точек изображения. На вход детектору подается черно-белое изображение. На выходе получается матрица, значения элементов которой определяют степень правдоподобности нахождения угла в исследуемых пикселях изображения. После выполняется отсечение участков пикселей со степенью правдоподобности, меньшей некоторого порога. Оставшиеся после отсечения точки являются особыми.

Для нахождения соответствия между точками для пар изображений к каждой угловой точке должно прилагаться её описание (дескриптор). Если дескрипторы точек на разных изображениях близки, то можно считать, что один и тот же физический объект.

Дескриптор — это небольшая окрестность точки. В идеале, дескриптор должен быть независим от масштаба и ориентации изображения. Тогда поиск соответствий сводится к обходу всех дескрипторов одного изображения и поиску наиболее близкого дескриптора на другом изображении.

Теперь опишем общий алгоритм решения задачи сопоставления точек:

Пусть нам дано: два изображения, причём они являются «похожими», т.е. получены смещением, поворотом, проекцией точки, добавлением шума, изменением положения камеры или уровня освещенности.

1. С помощью известного алгоритма компьютерного зрения находим ключевые точки двух изображений
2. Далее, используя свой оригинальный алгоритм/программный инструмент получаем дескрипторы точек изображений
3. Используя заранее определенный критерий и стандартный алгоритм (или собственный) сопоставляем дескрипторы точек изображений. Здесь можно использовать как простой перебор точек, так и метод поиска k-ближайших соседей.
4. Так как каждый дескриптор соответствует своей точке изображения, то мы получаем пары сопоставленных точек двух изображений

## Оценка состояния темы исследования

В своей работе я буду искать новый способ получения дескриптора ключевой точки изображения. При этом, полученный в результате исследования программный продукт будет проанализирован и подвержен сравнению с уже известными аналогами для решения данной задачи, ведь сейчас существует множество алгоритмов для сопоставления особых точек на изображениях. Самыми известными и широко используемыми являются SURF (Speeded Up Robust Features) и SIFT (Scale-invariant feature transform) [1]. Также надо упомянуть быструю и оптимизированную альтернативу предыдущим алгоритмам - Oriented FAST and rotated BRIEF (ORB). Он объединяет детектор локальных точек FAST и дескриптор BRIEF[6]. Некоторые из представленных выше алгоритмов являются запатентованными, и показывают нерезультативность работы в сложных условиях. Недавние исследования по нейронным сетям предлагают использовать алгоритм LIFT, который превосходит результаты своих аналогов SIFT и SURF[2]. Но при этом всегда есть возможность улучшить скорость обучения и структуру сети для достижения более точных показателей работы.

# Разработка архитектуры нейронной сети

Использование нейронных сетей для оптимизации и замены эвристических традиционных алгоритмов компьютерного зрения подтвердило свою уместность, как об этом говорилось в предыдущей главе. Поэтому, для достижения поставленной цели исследования было решено использовать искусственные нейронные сети.

## Определение и основные термины

Искусственная нейронная сеть — математическая модель, а также её программное или аппаратное воплощение, построенная по принципу организации и функционирования биологических нейронных сетей — сетей нервных клеток живого организма. Это понятие возникло при изучении процессов, протекающих в мозге, и при попытке смоделировать эти процессы. Первой такой попыткой были нейронные сети У. Маккалока и У. Питтса.

Единицей данной сети является нейрон, модель которого схематически представлена на рис.1.



Рисунок 1 Модель нейрона

Совокупность данных нейронов образует сеть из параметров весов, которые регулируются в течение обучения нейронной сети: на вход ей даются данные из выборки, которые также содержат соответствующий идеальный отклик, который должна дать сеть на эти данные. После обработки входных данных, результат сети сравнивают с желаемым откликом. На этом этапе вычисляется функция потерь (loss function), которая высчитывает ошибку. После выполняется процесс обратного распространения ошибки - распространение сигналов ошибки от выходов сети к её входам, в направлении, обратном прямому распространению сигналов в обычном режиме работы.

## Общая структура сети для решения задачи

Обращаясь к задаче исследования, нам нужно получить нейронную сеть, которая будет сравнивать два дескриптора (описания окрестности точки изображения) и вычислять, описывает ли данный дескриптор одну и ту же точку, или же разные. Предполагаемая схема поиска соответствий представлена ниже на рис.2.



Рисунок 2 Схема нахождения соответствий

Как видим, для этой задаче лучше всего подходит модель сиамской нейронной сети – которая была впервые использована для нахождения соответствия между подписями группой ученых из AT&T Laboratories в США в 1994 году [5].

Положили для начала, что модель-прототип будет выглядеть следующим образом (рис.3):



Рисунок 3 Модель-прототип

Как видно из рис.3, на вход подаются два окрестности точек изображения и параметр сигма, который указывает, принадлежат эти области одной и той же точке изображения (1) или разным (0). В основе сиамской нейронной сети лежит обычная нейронная сеть. При этом две половинки сиамской сети объединяются критерием L2-нормы, которую можно записать как:

$$\left‖x\right‖\_{2}=\sqrt{\sum\_{i}^{}x\_{i}^{2}}$$

Данная норма известна как евклидова норма, которая обычно используется для вычисления расстояния между векторами. Вычисляется как разность векторов (евклидово расстояние):

$$\left‖x\_{1}-x\_{2}\right‖\_{2}=\sqrt{\sum\_{i}^{}(x\_{1\_{i}}-x\_{2\_{i}})^{2}}$$

Для получения же дескриптора, мы программными средствами будем обрезать половинку сиамской сети, которая возвращает описательный вектор области.

## Первоначальные вариации структур базовой нейронной сети

Получив простое начальное представление модели, нам предстоит настроить её параметры и работу. Первым из таких параметров выступает половинка сиамской сети – базовая нейронная сеть.

Для начала были рассмотрены две модели для сравнения, взятые из разных источников (представлены на рис. 4).



Рисунок 4 Первоначальные структуры нейронной сети

На рис.4. первая структура а – это сверточная нейронная сеть, изначально решала близкую задачу – определяла похожие изображения (места отдыха). Была представлена программистом из Сан-Франциско Суджит Палом. Состоит из 8 слоёв:

Convolution2D – слой свёртки, выполняет преобразования схожие с оконным преобразованием фильтра: проходит по всей матрице изображения и перемножает пиксели вокруг точки на веса, затем получает суммарный отклик всех перемноженных величин. При этом веса данного фильтра меняются в процессе обучения сети.

MaxPooling2D - слой субдискретизации, он выполняет операцию максимума на тензоре (многомерной матрице), в данном случае – матрице изображения, получая максимальное значение из всех значений взятых из области, определяемой размером окна этого слоя. Например, MaxPooling2D (2x2, stride =2) имеет окно размером 2x2, соответственно берёт максимум 4-х значений рассматриваемой области 2x2.

Параметр stride - это шаг окна субдискретизации и свёртки, который позволяет обрабатывать не все пиксели изображения, а пропускать их часть, что не даёт сети создавать слишком много параметров для обучения и оптимизирует время её работы.

Вторая структура сети «b» является простой, состоит из Dense-полносвязных слоёв. Наглядно их можно представить, как переплетение выходов всех нейронов предыдущего слоя с входами нейронами последующего (рис.5).



Рисунок 5 Полносвязный слой Layer 2

Также здесь используется слои Dropout – они защищают сеть с такой переплетенной структурой от переобучения, отбрасывая часть используемых нейронов сети.

Сгенерировав обучающую выборку размером 20х20 вырезанных областей точек изображения и обучив несколько моделей сетей «а» и «b», были получены следующие результаты:

Таблица 1

|  |  |  |
| --- | --- | --- |
| **Используемая модель сети:****Название (размер обучающей выборки)** | **Тип структуры сети** | **Точность сопоставления на 500 точках первого и второго изображений** |
| choppedmySiameseArtificial\_model(239054).h5 | а | 31.800% |
| choppedmySiameseArtificial\_model\_V1(900000).h5 | а | 48.800% |
| choppedmySiameseArtificial\_model\_V2(200000).h5 | а | 34.200% |
| choppedmySiameseArtificial\_model\_V3(100000).h5 | а | 32.600% |
| choppedMySiameseNet1\_model\_V0(100000).h5 | b | 22.600% |
| choppedMySiameseNet1\_model\_V1(239054).h5 | b | 7.000% |

Как видим из таблицы 1, лучше всего справилась с задачей модель сети со структурой «а» - сверточная сеть. Но при этом точность была заметно низкой, поэтому было решено улучшить данную модель сети, изменив её параметры.

## Доработка структуры нейронной сети

Изучив статью по обучению «сверточного характеристического дескриптора ключевых точек» (Discriminative Learning of Deep Convolutional Feature Point Descriptors), разработал новую структуру сверточной сети. Данный выбор был продиктован и результатами эксперимента со структурами сетей, и ключевыми особенностями сверточных нейронных сетей[8]:

* 1. Зависимости (шаблоны), которым они обучаются инвариантны относительно переноса (translation invariant): если какая-то характерная особенность изображения переместиться в другой его угол, то сверточная сеть без труда определит её, а вот плотная (dense) сеть будет воспринимать такое изменение как новую зависимость (шаблон);
	2. Сверточные сети также могут обучиться «пространственной многоуровневости зависимостей» (spatial hierarchies of patterns): т.е. на первом слое, они могут распознавать углы, ребра, пятна, следующий слой: комбинации из рёбер и углов – многоугольники, далее – узоры, и т.д.

Новая структура сети имеет следующий вид (рис.6):



Рисунок 6 Новая структура сети

Оставляя неизменным основную последовательность слоёв, было решено поэкспериментировать с параметрами слоёв. В результате обучили порядка 7 новых моделей на основе сетей Net2-4. Итоговая сравнительная характеристика представлена на таблице 2 и соответствующими выводами по ней:

1. Наибольшая точность у моделей сети Net3;
2. Приём с переходом от 2D сверток к 1D сверткам уменьшил количество нейронов и значительно понизил точность моделей;
3. Модели, обученные на 35К и 70К различаются по точности только на 4-5%, следовательно можно предположить, что увеличение обучающей выборки в 2 раза даёт незначительный прирост точности
4. В случае с сетями Net2 и Net4 при увеличении выборки точность уменьшилась. Предполагаю, что это связано с хаотичным созданием «плохих» пар. Если данная гипотеза верная, то можно прибегнуть к «осмысленному» генерированию плохих пар для обучения.
5. При увеличении числа точек для сопоставления (например, для Net3\_model\_V1 – 1000 – 44%) точность уменьшается.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Название модели сети** | **Тип входных данных** | **Тип свёрток** | **Число нейронов** | **Размер выборки для обучения** | **Время обучения** | **Точность сопоставления\* (500 т)**  | **Точность сопоставления\* (1000 т)**  | **Точность сопоставления\* (1500 т)**  |
| choppedMySiameseNet2\_model\_V0.h5 | (50, 50, 3) цвет | 2D | 349120 | 35К | 4 ч | 25.800% | 24.100% | 22.533% |
| choppedMySiameseNet2\_model\_V1.h5 | (50, 50, 3) цвет | 2D | 349120 | 70K | 10 ч 30 м | 4.200% | 3.100% | 2.667% |
| choppedMySiameseNet3\_model\_V0.h5 | (50, 50, 1) ч/б  | 2D | 345984 | 35K | 3 ч 40 м | 45.200% | 39.200% | 36.533% |
| choppedMySiameseNet3\_model\_V1.h5 | (50, 50, 1) ч/б  | 2D | 345984 | 70K | 7 ч 40 м | 51.000% | 43.100% | 40.667% |
| choppedMySiameseNet4\_model\_V0.h5 | (50, 50) ч/б | 1D | 97568 | 35K | 10 м | 13.400% | 11.900% | 10.867% |
| choppedMySiameseNet4\_model\_V1h5 | (50, 50) ч/б | 1D | 97568 | 70K | 30 м | 10.000% | 8.000% | 8.733% |
| choppedMySiameseNet4\_model\_V2.h5 | (50, 50) ч/б | 1D | 97568 | 280K | 1 ч 36 м | 4.600% | 3.600% | 4.733% |
| **\*Точность сопоставления**  считается как ср.арифметическое из 10 полученных результатов  |  |
| Для вычисления точности используется **L2-норма и данные из набора вырезанных областей** |  |

Таблица 2.

# Доработка архитектуры нейронной сети

## Подход hard-mining

Было решено подробнее рассмотреть процесс генерации обучающих пар выборки в статье «Discriminative Learning of Deep Convolutional Feature Point Descriptors»[7]. Оказалось, авторы, основываясь на наблюдении, что после определенного момента времени обучения большинство пар правильно классифицируются сетью, и прекращение их использования улучшает натренированные весы модели, предлагают стратегию агрессивной добычи «жестких» хороших и плохих пар, которая называется “hard-mining”.

Из статьи можно вынести общий абстрактный алгоритм генерации выборки методом «hard-mining»:

1. Прежде сеть обучается до определенного уровня точности;
2. Далее, для каждой последующей эпохи генерируется набор плохих пар;
3. Плохие пары прогоняются через сеть и считается числовое значение функции потери;
4. Далее оставляют только самые «трудноотличимые» плохие пары, т.е. такие, которые дают меньший числовой результат предсказания сети и высокое значение потери

(в моём случае, «трудноотличимыми» плохими парами будут те, на которых сеть выдаёт результат от 0 до 0.5, соответственно, для хороших «трудноотличимых» пар моя сеть будет выдавать результат, больший 0.5);

1. Далее сеть обучают на отобранных «трудноотличимых» плохих парах.

Также, авторы добавляют, что такая процедура может быть использована и при генерации хороших пар.

## Первая реализация метода hard-mining

Далее реализуем метод поиска «трудноотличимых» пар для расширения выборки с помощью python. Для начала оценим, сколько потенциальных пар можно получить из набора вырезанных областей 16 изображений.

 Генерация обучающей выборки для одного изображения:

1. Генерация хороших пар:

На 1 точку приходится 79 областей (оригинал + преобразования). т.к. для нас не имеет значение порядок выбора областей, то это превращается в классическую задачу комбинаторики про сочетания:

«сколькими способами можно выбрать m из n различных предметов?»

$$С\_{n}^{m}=\frac{n!}{m!\left(n-m\right)!}$$

Т.е. сколькими способами можно выбрать 2 области из 79:

$$C\_{79}^{2}=\frac{79!}{2!\*\left(79-2\right)!}=\frac{77!\*78\*79}{2\*77!}=\left\{сокращаем\right\}=$$

$$=\frac{78\*79}{2}=3081$$

На 400 точек получаем: 400 \* 3081 = 1232400 (1.2М) потенциальных пар можно получить из набора №4.

1. Генерация плохих пар:

Здесь мы будем выбирать 2 области из 400\*79=31600 (всех имеющихся областей для картинки). Однако, мы не рассматриваем области одних и тех же точек:

Выбираем первую область: 400\*79

Выбираем вторую область: 399\*79

Мы рассматриваем задачу: взять 2 области из 400 точек:

$$C\_{400}^{2}=\frac{400!}{2!\*\left(400-2\right)!}=\frac{398!\*399\*400}{2\*398!}=\left\{сокращаем\right\}=$$

$$=\frac{399\*400}{2}=399\*200=79800$$

$$79800$$

 Итого 79800 сочетаний точек возможно. Теперь нужно добавить к этим

 данным выбор областей для этих точек:

Домножить на 79\*79, так как дальнейший выбор не зависит дальнейший от сочетаний (79 областей одной точки и 79 областей другой отличаются друг от друга).

Итого: 79800\*79\*79 = 498031800 (498М) потенциальных плохих пар.

Так как, кол-во вариантов для поиска «трудноотличимых» пар велико, то было решено использовать потоки для распараллеливания вычислений и увеличения скорости работы. Также, весь комплекс программ был перенесен с ноутбука (на котором изначально выполнялись вычисления) на стационарный компьютер с большей мощностью процессора для сокращения времени работы программ.

**Алгоритм для “hard mining”:**

*Предварительные параметры:*

*N – кол-во точек,*

*len\_pairs - сколько хор. и плох. пар мы хотим получить*

*index\_pict - номер картинки*

*dir – директория, где лежат области*

*handleArea(name) – функция загружает область name и конвертирует её в формат входных*

 *данных модели нейронной сети*

*GoodL – список хороших пар типа: <val, (img1, img2, 1)>*

*BadL – список плохих пар: <val, (img1, img2, 0)>*

*Алгоритм:*

1. *Загружаем модель нейронной сети*

*#Возможные хорошие пары:*

1. ***Для каждой точки*** *I из N точек* ***делаем****:*
	1. ***Для каждой области*** *ia1 = 0* ***с шагом*** *1* ***до*** *78(включая)* ***делаем****:*
		1. ***Для каждой области*** *ia2 = ia1+1* ***с шагом*** *1* ***до*** *78(включая)* ***делаем****:*
			1. *Загружаем img1 = handleArea(<i><index\_pict><ia1>.jpg)*
			2. *Загружаем img2 = handleArea(<i><index\_pict><ia2>.jpg)*
			3. *Получаем значение val, запуская модель сети на параметрах:*

 *(img1, img2, 1)*

* + - 1. ***Если*** *val < 0.5* ***то****:*

*Переходим к следующей итерации цикла 2.1.1.*

* + - 1. *Идём по списку GoodL (с конца): j = len(GoodL)-1; j >= 0; j--*

 *По очереди сравниваем значение val со значениями*

 *других пар из GoodL:*

***Если*** *val > GoodL[j].val* ***то****:*

***Если*** *мы рассматриваем первый элемент (j == 0)* ***то****:*

*вставляем <(img1, img2, 1), val> в начало*

*выход из цикла по GoodL*

***Иначе:***

*переходим на новую итерацию*

***Если*** *val <= GoodL[j].val* ***то****:*

***Если*** *мы рассматриваем последний элемент (j == len(GoodL)-1* ***и*** *длина списка ещё не len\_pairs* ***то****:*

*вставляем <(img1, img2, 1), val> в конец*

*выход из цикла по GoodL*

***Иначе:***

 *Вставляем <(img1, img2, 1), val> на позицию j*

*выход из цикла по GoodL*

* + - 1. ***Если*** *длина списка стала больше len\_pairs* ***то****:*

*удаляем последний элемент списка*

 *#Возможные плохие пары:*

1. ***Для каждой точки*** *I= 0* ***с шагом*** *1* ***до*** *N-1(включая)* ***делаем****:*
	1. ***Для каждой точки*** *J=I+1* ***с шагом*** *1* ***до*** *N-1(включая)* ***делаем:***
		1. ***Для каждой области*** *ia1 точки I* ***делаем****:*
			1. ***Для каждой области*** *ia2 точки J* ***делаем****:*
				1. *Загружаем img1 = handleArea(<i><index\_pict><ia1>.jpg)*
				2. *Загружаем img2 = handleArea(<i><index\_pict><ia2>.jpg)*
				3. *Получаем значение val, запуская модель сети на параметрах:*

 *(img1, img2, 0)*

* + - * 1. ***Если*** *val > 0.5* ***то****:*

*Переходим к следующей итерации цикла 3.1.1.1.*

* + - * 1. *Идём по списку BadL (с конца): j = len(BadL)-1; j >= 0; j--*

 *По очереди сравниваем значение val со значениями*

*других пар из BadL:*

***Если*** *val < BadL[j].val* ***то****:*

***Если*** *мы рассматриваем первый элемент (j == 0)* ***то****:*

*вставляем <(img1, img2, 0), val> в начало*

*выход из цикла по BadL*

***Иначе:***

*переходим на новую итерацию*

***Если*** *val >= BadL[j].val* ***то****:*

***Если*** *мы рассматриваем последний элемент (j == len(BadL)-1* ***и*** *длина списка ещё не len\_pairs* ***то****:*

*вставляем <(img1, img2, 1), val> в конец*

*выход из цикла по BadL*

***Иначе:***

 *Вставляем <(img1, img2, 1), val> на позицию j*

*выход из цикла по BadL*

* + - * 1. ***Если*** *длина списка стала больше len\_pairs* ***то****:*

*удаляем последний элемент списка*

Программная версия алгоритма на языке python описана в Приложении Б: Исходные коды программ.

Для использования потоков воспользуемся простым разделением алгоритма:

1. Сначала создаются и вычисляются потоки для хороших пар:

 Действия 2.1.1.1. – 2.1.1.6. разделяются по точкам:

 Пусть всего потоков - P, а точек N, тогда каждому потоку

 по P/N точек.

1. Затем создаются и вычисляются потоки для плохих пар:

Действия 3.1.1.1.1. – 3.1.1.1.6. разделяются по преобразованиям:

 Пусть всего потоков - P, а преобразований - A, тогда каждому потоку

 по P/A преобразований со всевозможными точками.

Также отметим, что программная реализация отличается от алгоритма тем, что потоки считают каждый свой список точек (хороших/плохих), а затем, по окончании работы происходит слияние списка отдельного потока с общим списком. Так как и списки в потоках, и общий список – отсортированы, то лучше всего с задачей слияния справляется алгоритм, основанный на кучах - heap queue algorithm. Слияние в таком случае выполняется за линейное время O(n).

Такой изменение было использовано, чтобы общий большой список (плохих/хороших) пар реже использовался и не создавалась большая очередь из потоков, ожидающих своего доступа на изменение.

## Результаты первого применения hard-mining

Было сразу же решено перейти на вариант поиска «трудноотличимых» пар на 100 точках (вместо 400), так как время работы на всевозможных хороших парах областей из 100 точек (всего 308100 пар) с извлечением 25К пар с максимальным значением сети заняло 1 час 15 минут, 900 Мб ОП.

Для плохих пар, т.к. даже для 100 точек элементов слишком много: 15М, по сравнению c предыдущим – 0.3М (приблизительно в 50 раз больше), запустили на 20 точках (585К пар) – получили время работы - 2 часа 14 минут.

Сгенерированная выборка из «трудноотличимых» пар сохраняется в текстовых файлах (рис.7).



Рисунок 7. Выборка из "трудноотличимых" пар

Таким образом получили: 25К - хороших пар, 25K - плохих. Далее их слили в общую выборку и продолжили обучать лучшую сохраненную модель – Net3 (51%). Однако, точность во время обучения (было 98%) начинает расти с 40% и останавливается на 90%. Но данная не относится к сопоставлению точек, на котором дообученная выдала 1-4%.

После 3-х прогонов дообучения сети на полученных данных с помощью hard-mining хороших и плохих пар 1,2,3 картинок, наблюдалась следующая динамика: 66%, 16%, 20%, 6% соответственно. Данный результат указал на наличие проблем в реализации сети и применении метода поиска «трудноотличимых» пар.

## Новые изменения сети

Сначала была решена первая проблема – это подготовка сети к дообучению её сохраненных моделей. Для этого весь процесс обучения, сопоставления точек и распечатки графиков был автоматизирован при помощи скриптов batch windows.

Далее решалась проблема изменения параметров при продолжении обучения сохраненных моделей сети. Было решено заменить оптимизатор rms на sgd. Несмотря на то, что rms также может продолжить обучение сохраненных моделей, sgd показал себя с лучшей стороны и увеличил точность сети на 20% без использования метода hard-mining. Так получилась новая сеть Net6p, которая отличается от Net3 оптимизатором, явным указанием скорости обучения = 0.01 и моментом = 0.9, которые были подобраны исходя из нескольких экспериментов с обучением сети. Следующая таблица 3 показывает точность сети Net6p при сопоставлении точек на 4-х картинках с подробным описанием характеристик моделей.

Надо отметить, что сеть обучалась с помощью набора №4 выборки на первой картинке и модель chopped\_net6p\_V12.h5, обучавшаяся в течение 12 эпох показала лучший результат в 77%. Данная сеть была также обучена на различных точках из всех 16 картинок, однако наибольшим результатом было 69% точности после 16 эпох.

Вновь заметим, что ключевую роль играет подбор пар для обучающей выборки, который все еще остается произвольным, поэтому вернемся к проблеме продолжения обучения. Удостоверившись в корректности реализации автоматизированного дообучения сохраненных моделей сети Net6p c оптимизатором sgd было выяснено, что спад точности наблюдается еще при обучении и только тогда, когда мы при загрузке уже обученной модели меняем выборку (на выборку из hard-mining, например).

Таблица 3. Характеристики обучения моделей сети Net6p



## Вторая реализация метода hard-mining

Первое применение hard-mining не сработало из-за ряда проблем:

1. Выборка, подаваемая на вход сети могла быть плохой, т.к. плохие пары генерируются каждый раз произвольно;
2. Перебор пар для поиска «трудноотличимых» берёт области, которые не пригодятся для различения точек: то есть, могут быть области с преобразованием масштаба исходной области и преобразованием поворота исходной. Таким образом сеть обучается комбинации двух преобразований, когда нам достаточно, чтобы сеть знала, что исходная область и её преобразование относятся к одной точке;
3. И соответственно, обучение сохраненной модели сети на новой выборке, полученной методом hard-mining ухудшает её точность.

Для разрешения первой проблемы была написана программа, которая «фиксирует» выборку и сохраняет тройки в текстовый файл. Таким образом, можно отличать различные выборки и выбирать ту, на которой сеть обучается лучше.

Вторая же проблема была решена изменением алгоритма hard-mining при переборе точек. Теперь рассматриваются все возможные комбинации плохих и хороших пар с тем условием, что одна из областей пары представляет непреобразованную исходную область (где преобразование = 0, как например в ‘1003000.jpg’). Такой же подход используется при генерации исходных плохих пар для перового обучения сети.

Также, для устранения третьей проблемы, было «расширять» исходную выборку, а не заменять её полностью. И уже на расширенной выборке обучать сеть по новой. Для наглядности приведем алгоритм обучения сети с помощью hard-mining.

ОБЩИЙ АЛГОРИТМ ОБУЧЕНИЯ:

1. Указываем, сколько всего будет итераций I\_end.
2. Начальная нулевая итерация I = 0, картинка K=1:

Обучаем сеть на созданном заранее, фиксированном наборе данных setHM0.txt в течение 12-ти эпох

1. Сохраняем полученную модель сети
2. Итерация I = I + 1:

Используем метод hard-mining и области следующей картинки:

K = K + 1

1. Используем hardMiningGood для модели сети итерации I-1, получаем список хороших пар (время ожидания будет сокращено, так как используя комбинацию пар с начальной областью, для каждой точки будем перебирать 78 вариантов пар (78 преобразований), итого всего вариантов = 78 \* 400 точек = 31200
2. С помощью hardMiningGood получаем 5000 тысяч хороших «трудноотличимых» пар с наибольшим значением для модели сети итерации I-1 и сохраняем результат в файле goodHardTriples.txt.
3. Используем hardMiningBad для модели сети итерации I-1, получаем список плохих пар (здесь будем использовать 100 точек, в силу использования комбинаций пар с начальной областью, нам не нужно рассматривать все множество произвольных комбинаций 79\*79, поэтому тут кол-во пар для перебора тоже будет уменьшено:

$$C\_{100}^{2}=\frac{100!}{2!\*\left(100-2\right)!}=\frac{98!\*99\*100}{2\*98!}=\left\{сокращаем\right\}=$$

$$=\frac{99\*100}{2}=99\*50=4950$$

И домножая на кол-во преобразований, получаем:

4950 \* 8 = 386100 всевозможных пар для перебора.

1. С помощью hardMiningBad получаем 5000 тысяч плохих «трудноотличимых» пар с наибольшим значением для модели сети итерации I-1 и сохраняем результат в текстовый файле badHardTriples.txt
2. С помощью программы enlargeSet.py объединяем set0.txt, badHardTriples.txt и goodHardTriples.txt, получая расширенную выборку setHM1.txt
3. Запускаем дообучение сохраненной модели итерации I-1 на выборке setHM1.txt в течение 12-ти эпох.
4. Сохраняем полученную модель сети.
5. Повторяем пункты 3-10 пока I < I\_end.

## Результаты второго применения метода hard-mining

Согласно алгоритму, описанному в предыдущем параграфе реализовал 2 итерации. Результат представлен в таблице 4.

Таблица 4. Результат второго применения hard-mining (2 итерации)

|  |  |
| --- | --- |
|   | Картинка № |
| Итерация  | 1 | 2 | 3 | 4 |
| 0 (lr=0.01) | 75.500% | 64.000% | 66.000% | 64.000% |
| 1 (lr=0.001) | 58.250% | 25.750% | 57.250% | 46.250% |
| 1 (lr=0.01) | 70.750% | 38.750% | 73.000% | 58.500% |

Как видно из таблицы, лучшие результаты показало начальное обучение сети. Также надо отметить, что полученная фиксированная выборка с новым способом генерирования плохих пар показывает хорошую точность. Однако, расширение выборки получением «трудноотличимых» пар по-прежнему ухудшает показатели точности модели. При этом, если мы уменьшаем скорость обучения (learning rate), то точность также падает.

# Генерация обучающей выборки

## Общий алгоритм получения набора вырезанных областей

Задача обученной нейронной сети - проверять на равенство дескрипторы (фрагменты изображения с ключевыми точками), принимаемые на вход. Для подготовки обучающей выборки удобно иметь уже готовый набор вырезанных областей, из которого уже посредством программы будет составляться обучающая выборка.

Для начала, мы сформируем порядок наименования областей для их дальнейшего хранения и использования (Рис.7).



Рисунок 8 Формат наименования вырезанных областей

Пары изображений на вход сети будут формироваться так:

(<1001000.jpg>, <1001001.jpg>, 1) – два изображения одной области

(<2001000.jpg>, <1001001.jpg>, 0) – два изображения разных областей.

Далее опишем алгоритм работы генератора областей:

1. Цикл по отобранным N изображениям для генерации выборки

{Ik: k = 0…N}:

1. На текущим изображении Ik находим с помощью алгоритма SIFT/SURF/ORB ключевые точки {P}
2. Вырезаем области точек определенного размера
3. Цикл по точкам j = 0…кол-во\_точек(P):
	1. В соответствии с форматом, после каждого преобразования сохраняется новая область, относящаяся к точке j изображению k и преобразованию m:

<j><k><m>.jpg

* 1. Неизмененная область
	2. Наложение шума (соль и перец)
	3. Масштабирование (фильтр Гаусса х2)
	4. Изменение яркости (х4)
	5. Поворот с шагом 5 градусов (от 0 до 360) (х71)

В итоге на одну точку изображения получаем 78 преобразований. Таким образом для одной точки можем спродуцировать 79\*78 возможных комбинаций хороших пар областей. Набор данных преобразований был выбран, чтобы сделать конечный продукт исследования эффективным в специфических природных условиях (в особенности в условиях пещер, низкой освещенности, и т.д.).

Наглядно применение фильтров к вырезанной области представлено на рис.7: 1 – исходная область, 2 – наложение шума, 3 – масштабирование, 4 – изменение яркости и 5 – преобразование поворота.



Рисунок 9 Генерация вырезанных областей

# Заключение

В этом семестре была проведена работа по созданию и настройке оптимальной модели нейронной сети для решения задачи сопоставления ключевых точек. Было спроектировано и обучено 15 различных моделей нейронных сетей для достижения поставленной цели. Также написана программа для сопоставления ключевых точек изображений и проверки точности обученных моделей. При этом был проведён анализ полученных моделей для дальнейшего их улучшения и повышения точности сопоставления.

Основные результаты:

1. Разработана лучшая архитектура сети net6p на основе библиотеки keras для решения задачи сопоставления точек. Она представляет собой сиамскую нейронную сеть с половинками в виде сверточных сетей, критерием объединения половинок – L2-нормой и sgd-оптимизатором со скоростью обучения 0.01;
2. Получена обрезанная модель нейронной сети net6p, которая считает дескриптор области ключевой точки. Даная модель является лучшей среди остальных обученных и показывает среднюю точность в 70% при сопоставлении областей ключевых точек 4-х изображений.

 Данная модель обучалась на выборке из 77К элементов, созданной из набора областей №4 (см. таблица 4);

1. Реализованы 2 подхода для применения метода hard-mining для повышения точности модели нейронной сети, однако в результате точность сети ухудшилась после прохода первых итераций алгоритма.

# Список использованных источников

1. Конушин А. «Сопоставление изображений и локальные особенности» // Лекции по курсу «Введение в компьютерное зрение»,

2012, стр. 45–79.

1. Белоусов С. «LIFT: Learned Invariant Feature Transform» // Хабрахабр: интернет-сообщество индустрии высоких технологий [Электронный ресурс] <https://habrahabr.ru/post/323688/>

(дата обращения 27.12.2017)

1. Structure from motion // Wikipedia, The Free Encyclopedia [Электронный ресурс] <https://en.wikipedia.org/w/index.php?title=Structure_from_motion&oldid=816884383> (дата обращения 27.12.2017)
2. Обзор дескрипторов ключевых точек // НОУ «ИНТУИТ» [Электронный ресурс] [http://www.intuit.ru/studies/courses/10621/1105/lecture/17983](http://www.intuit.ru/studies/courses/10621/1105/lecture/17983?page=2)

(дата обращения 9.04.2018)

1. Bromley J., Guyon I., LeCun Y., Sickinger E., Shah R. «Signature Verification using a "Siamese" Time Delay Neural Network», 1994
2. ORB (Oriented FAST and Rotated BRIEF) // OpenCV-Python Tutorials [Электронный ресурс] <https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_orb/py_orb.html#orb>

(дата обращения 9.04.2018)

1. Simo-Serra E., Trulls E., Ferraz L., Kokkinos I., Fua P., Moreno-Noguer F. Discriminative Learning of Deep Convolutional Feature Point Descriptors // ICCV (2015)

<http://hi.cs.waseda.ac.jp/~esimo/publications/SimoSerraICCV2015.pdf>

1. Chollet F. Deep Learning with Python. NY, 2018.