

Министерство науки и высшего образования Российской Федерации
**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИТМО»**

Факультет Систем Управления и Робототехники

Направление подготовки:

15.04.06 Мехатроника и робототехника

КУРСОВАЯ РАБОТА

по дисциплине: Методы машинного обучения в робототехнике

по теме: *Метод сопоставления изображений, получаемых с квадрокоптера
и спутника, на основе нейронной сети*

Выполнили студенты

Веснин М.А.

Миргазов Э.Р.

Топольницкий А.А.

Преподаватель

Колюбин С.А.

Подпись преподавателя

Дата

Защита		
--------	--	--

Санкт-Петербург

2023 г.

Аннотация

Курсовая работа по дисциплине «Методы машинного обучения в робототехнике» по теме «Метод сопоставления изображений, получаемых с квадрокоптера и спутника, на основе нейронной сети».

Цель работы – реализация алгоритма сопоставления изображений, полученных с квадрокоптера, и изображений, снятых со спутника, на основе нейросетевого подхода.

Рассмотрены подходы к задаче локализации квадрокоптера. Рассмотрена структура свёрточных нейронных сетей и сетей трансформеров, методы внимания и самовнимания, применяемые в нейронных сетях, а также Swin Transformer архитектура. Реализован модифицированный вариант данной архитектуры на языке программирования Python с использованием фреймворка PyTorch. Проведено обучение данной архитектуры и архитектуры ResNet50 для датасета University-1652, и сделано сравнение результатов для данных моделей.

Работа изложена на 54 страницах печатного текста. Включает в себя 35 рисунков, 3 таблицы, 17 источников литературы и 1 приложение.

Оглавление

Аннотация	2
Введение	4
Глава 1. Сравнительный анализ подходов к локализации.....	6
Глава 1.1. Относительная визуальная локализация.....	6
Глава 1.2. Абсолютная визуальная локализация	6
Глава 1.2.1. Сопоставление шаблонов	8
Глава 1.2.2. Сопоставление характерных точек	9
Глава 1.2.3. Сопоставление визуальной одометрии.....	10
Глава 1.2.4. Сопоставление глубоким обучением	11
Глава 2. Описание Swin Transformer и предлагаемого на его основе решения задачи сопоставления изображений.	15
Глава 2.1. Внимание и самовнимание.....	15
Глава 2.2. Архитектура Swin Transformer	23
Глава 2.3. Модификация рассматриваемой архитектуры	26
Глава 3. Описание набора данных и параметры обучения нейронной сети...	30
Глава 4. Полученные результаты.	31
Заключение	37
Список литературы	39
Приложение	41

Введение

В современном мире квадрокоптеры находят всё большее применение, поскольку могут выполнять целый ряд задач: профилактика и ликвидация ЧС, обеспечение обороны и национальной безопасности объектов промышленности, сельского хозяйства и продовольствия, природных ресурсов, а также служб мониторинга, длительного авиационного патрулирования земной и водной поверхностей. Помимо этого, данный тип роботов можно применять для задач поиска пропавших людей или в задачах противодействия преступникам [1].

Для выполнения любой реальной задачи необходимо решить задачу локализации, то есть определить положение нашего мобильного робота в пространстве. Наиболее популярными подходами к локализации являются использование инерциальных навигационных систем и глобальной навигационной спутниковой системы, однако данные методы имеют ряд недостатков. Все инерциальные системы подвержены накоплению ошибок со временем, а поскольку используемые датчики, то есть акселерометры и гироскопы, не являются идеальными, то это приводит к неизбежному увеличению погрешности в определении местоположения. Кроме того, данные системы чувствительны к внешним воздействиям, что также приводит к искажениям измерений [2].

Конечно же можно отметить, что инерциальные системы редко применяют в чистом виде, а все больше используют совместно с глобальной навигационной спутниковой системой, однако и последняя система также не лишена недостатков:

1. Прием вне прямой видимости. Сигналы спутниковых систем требуют прямой видимости между несколькими спутниками и приемником для достижения точного позиционирования, однако в городских условиях, густых лесах или районах с высокими зданиями сигналы могут быть заблокированы.

2. Многолучевое распространение. Оно происходит, когда сигналы спутниковых систем достигают приемника по нескольким путям, включая прямые пути и пути, отраженные от таких поверхностей, как здания, деревья или водоемы. Эти отраженные сигналы могут мешать прямым сигналам, вызывая искажение сигнала и внося ошибки в расчеты местоположения.
3. Спуфинг. Спуфинг относится к преднамеренному манипулированию или имитации сигналов спутниковых систем для обмана приемников. В последнее время имели место атаки, нацеленные на слабые места БПЛА, использующие глобальную навигационную спутниковую систему, такие как прерывание сигнала для краха БПЛА или отправка ложных сигналов для захвата БПЛА [3].

Вышесказанное мотивировало к созданию нового подхода к самолокализации, который мог бы дополнить существующие методы. Таким образом был предложен метод локализации, основанный на компьютерном зрении. Его суть заключается в извлечении полезной информации с бортовых камер и сопоставлении с предварительно сохраненными изображениями, содержащими географическую привязку.

Целью курсовой работы является реализация алгоритма сопоставления изображений, полученных с квадрокоптера, и изображений, снятых со спутника, на основе нейросетевого подхода. Задачи, которые необходимо выполнить для достижения цели, следующие:

1. Провести сравнительный анализ подходов к локализации;
2. Провести описание предложенного решения на основе нейросетевого подхода;
3. Изучить набор данных, позволяющий реализовать обучение нейронной сети;
4. Произвести выбор или создать архитектуру нейронной сети, провести её обучение;
5. Оценить работу нейронной сети и сделать выводы.

Глава 1. Сравнительный анализ подходов к локализации

Глава 1.1. Относительная визуальная локализация

В целом методы оценки положения разделяют на 2 основных подхода: относительная визуальная локализация и абсолютная визуальная локализация. Для справки рассмотрим оба подхода [4, 5]. Сперва необходимо отметить, что данный подход не способен обеспечить глобальную оценку положения и включает в себя такие 2 метода как визуальная одометрия (VO) и одновременная локализация и построение карты (SLAM). VO — это метод, который сравнивает текущий кадр, наблюдаемый мобильным роботом, с предыдущим кадром для анализа различия в собственном движении. Новая оценка положения получается посредством добавления оценочной разницы к предыдущей оценке.

В отличие от VO метод SLAM помимо оценки положения строит карту неизвестного пространства по мере его исследования. Основным принципом SLAM является совместная оценка положения ориентиров и положения мобильного робота. Ключевой характеристикой SLAM является способность распознавать ранее посещенные места и корректировать оценку позы и карту в соответствии с этим наблюдением или, другими словами, замыкать цикл, чего лишен метод VO. Несмотря на данную способность его применение не всегда возможно, например, на больших территориях или в случаях, когда траектория не само пересекается.

У данных методов существует значительный недостаток аналогичный инерциальной навигационной системе: они накапливают ошибку с течением времени, поэтому их применение на больших территориях без замыкания цикла нецелесообразно.

Глава 1.2. Абсолютная визуальная локализация

В отличие от предыдущего подхода абсолютная визуальная локализация невосприимчива к дрейфу с течением времени, однако данный подход использует ранее собранные данные, обладающие точной географической привязкой, чего не требовалось в относительной визуальной локализации.

Эталонные данные представляют собой ортотрансформированные изображения, то есть изображения, в которых производится коррекция геометрии для устранения искажений, вызванных перспективой и наклоном съемочного устройства. В настоящее время подобные данные можно получить из картографических систем, например, Google Maps, Bing Maps, Mapbox, NASA Worldview и другие. Еще одним способом получить данные является использование изображений, собранных во время предыдущих полетов и привязанных к местности с помощью бортовой глобальной навигационной спутниковой системы. Очевидно, что в данном случае нам необходима точная навигационная система, однако она нам понадобится лишь единожды.

Основная цель абсолютной визуальной локализации – сопоставить текущий кадр с визуальной памятью, построенной из вышеупомянутых эталонных данных для глобальной локализации. Из этого и вытекает невосприимчивость к дрейфу, поскольку каждая оценка является независимой.

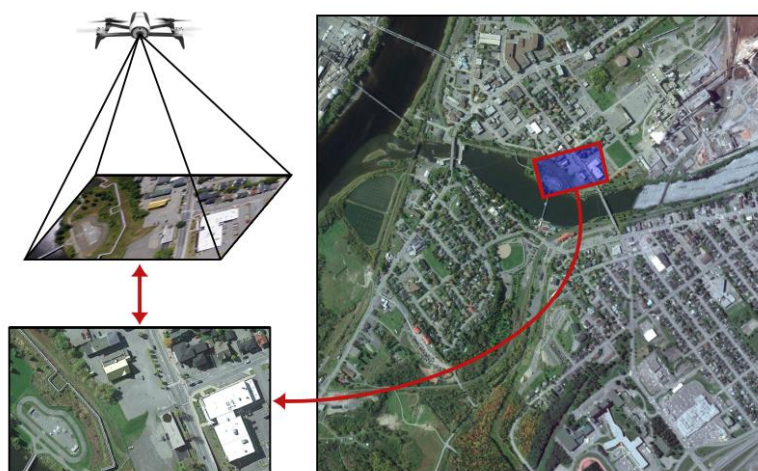


Рисунок 1. Концепт абсолютной визуальной локализации.

Однако данный подход также не лишен недостатков, которые в основном завязаны на эталонные данные. Самое очевидное это большой объем данных, для которых требуется соответствующая вычислительная мощность. Кроме этого, можно отметить, что изображения могут быть получены при совершенно разных внешних условиях, в том числе при разном уровне

освещенности, высоте, перспективе, времени года и так далее, что может повлиять на точность оценки или работоспособность методов если не учитывать эти особенности.

Наиболее широко распространены 4 метода сопоставления изображений: сопоставление шаблонов (template matching), сопоставление характерных точек (feature points matching), сопоставление визуальной одометрии (visual odometry matching) и сопоставление глубоким обучением (deep learning matching). Кратко рассмотрим каждый из них.

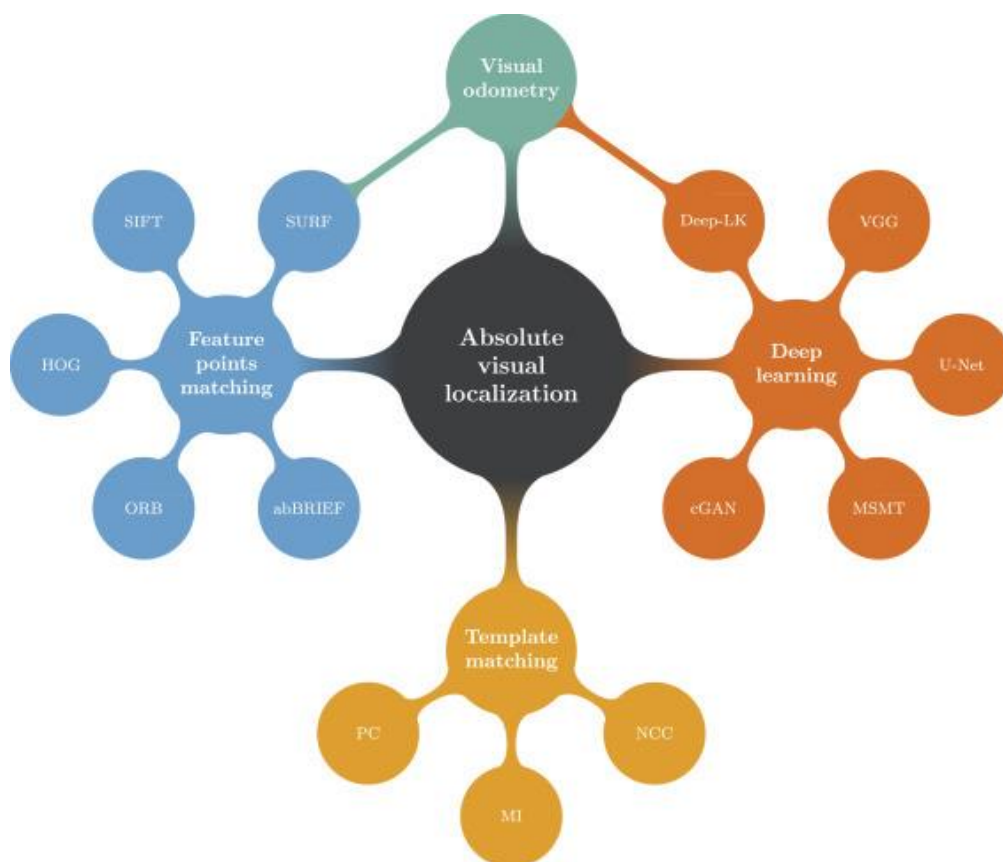


Рисунок 2. Методы абсолютной визуализации

Глава 1.2.1. Сопоставление шаблонов

В общем случае сопоставление с шаблоном является методом обработки цифровых изображений для поиска частей изображения, которые соответствуют шаблону. Применительно к нашей задаче текущий кадр с БПЛА является шаблоном, который ищется на карте.

Данный метод использует скользящее окно (sliding window), при котором окно фиксированного размера перемещается по изображению для

поиска совпадения с шаблоном. Далее используется оператор сравнения фрагментов изображения, такой как сумма квадратов разностей, для сравнения двух фрагментов изображения и получения меры сходства. Основным недостатком сопоставления шаблонов является высокая вычислительная ресурсоемкость меры подобия.

Глава 1.2.2. Сопоставление характерных точек

Сопоставление характерных точек используется как более эффективная альтернатива сопоставлению шаблонов по ряду причин:

1. Инвариантность к масштабу и поворотам: характерные точки выбираются таким образом, чтобы быть стабильными и неизменными при преобразованиях масштаба, поворотах и искажениях, что делает их более надежными для распознавания объектов.
2. Робастность к изменениям внешней среды: характерные точки выбираются на основе их отличительных особенностей, что делает их более устойчивыми к изменениям в освещении, тени и другим артефактам.
3. Сопоставление характерных точек позволяет снизить вычислительную сложность задачи поиска и сопоставления объектов. Вместо сравнения всего изображения или больших шаблонов, используется только небольшой набор характерных точек, что существенно сокращает количество операций.

Во время обнаружения характерных точек алгоритм ищет особые места на изображении, для чего могут использоваться оператор Харриса, который основан на вычислении угловых точек, которые являются местами существенных изменений в интенсивности изображения; метод SIFT (масштабно-инвариантная трансформация признаков), который использует гауссовскую пирамиду для поиска характерных точек или другие методы.

Каждая найденная точка описывается с помощью некоторого дескриптора, который содержит информацию о локальной окрестности точки, что позволяет создать компактное представление каждой характерной точки, которое сохраняет ее отличительные особенности. Соответственно

следующим шагом необходимо извлечь дескрипторы, для чего также существуют широко используемые методы, рассмотрим несколько. Метод SIFT, помимо обнаружения характерных точек, также извлекает их дескрипторы, которые основаны на распределении градиентов интенсивности вокруг точек. Метод SURF способен извлекать как характерные точки, так и дескрипторы. Дескрипторы в данном случае извлекаются на основе распределения градиентов интенсивности с помощью вейвлетов Хаара, благодаря чему метод SURF обладает более высокой скоростью работы по сравнению с методом SIFT.

Далее путем сравнения и сопоставления дескрипторов выполняется сопоставление характерных точек на разных изображениях, для чего применяются такие метрики как Евклидово расстояние.

В настоящее время большинство работ использует комбинацию сопоставления характерных точек и статистической фильтрации, например RANSAC, который используется для оценки параметров математических моделей на основе набора данных, содержащего выбросы или ошибки, или другие.

Глава 1.2.3. Сопоставление визуальной одометрии

VO был описан ранее однако он может быть расширен до абсолютной визуальной локализации с использованием данных о предыдущих полетах. Данные соответственно должны обладать географической привязкой, как и во всех других методах абсолютной визуализации, а далее собираются в базу данных для повторного использования.

Необходимо отметить, что данный подход отличается от рассмотренных ранее поскольку не опирается на сравнение кадра и карты, вместо этого сравнивается изображение с БПЛА и графом поз. Граф поз представляет собой структуру данных из положений мобильного робота в пространстве с течением времени. В связи с этим возникают и трудности, которые не были присущи предыдущим подходам, например, оптимизация графа поз. Однако данный метод может быть крайне полезен при резкой утрате связи с

глобальной навигационной системой, тогда мобильный робот может вернуться в исходную точку.

Глава 1.2.4. Сопоставление глубоким обучением

Сперва скажем пару слов о глубоком обучении в области компьютерного зрения, поскольку это направление является сравнительно новым. Данное направление получило широкое распространение после победы сверточной нейронной сети AlexNet в соревновании ImageNet Large Scale Visual Recognition Challenge (ILSVRC) в 2012 году с большим отрывом [6]. Данная победа доказала превосходство нейронных сетей по сравнению с более классическими методами.

Однако данная победа стала результатом нескольких факторов, приведем несколько: увеличение вычислительной мощности и, что еще более важно, большому набору обучающих данных (ImageNet содержит около 14 миллионов изображений). На настоящий момент не существует настолько больших датасетов в области локализации БПЛА, что несколько тормозит развитие в данной области и поэтому данный подход к решению проблемы локализации БПЛА находится только в начале своего пути.

Помимо указанной проблемы на глубокое обучение накладываются еще некоторые ограничения, например, обучение происходит довольно медленно даже на современных вычислительных мощностях полноразмерных компьютеров, не говоря уже о вычислительных способностях беспилотника, что делает невозможным обучение в реальном времени из-за чего нельзя запомнить уже посещенные области. Кроме того, по причине ограниченности вычислительных ресурсов невозможно обучить нейросеть запомнить полноразмерную эталонную карту. Однако перечисленные недостатки не пересиливают возможную перспективу, которые открывают методы сопоставления глубоким обучением.

На рисунке ниже показано, что алгоритм SIFT, который является широко используемым традиционным сопоставлением точечных объектов, не может сопоставить спутниковые изображения (справа) и изображения с БПЛА

(слева). Область, отмеченная синим цветом, — это та же область, что и на изображении слева, но алгоритм SIFT не может найти синюю область [7].

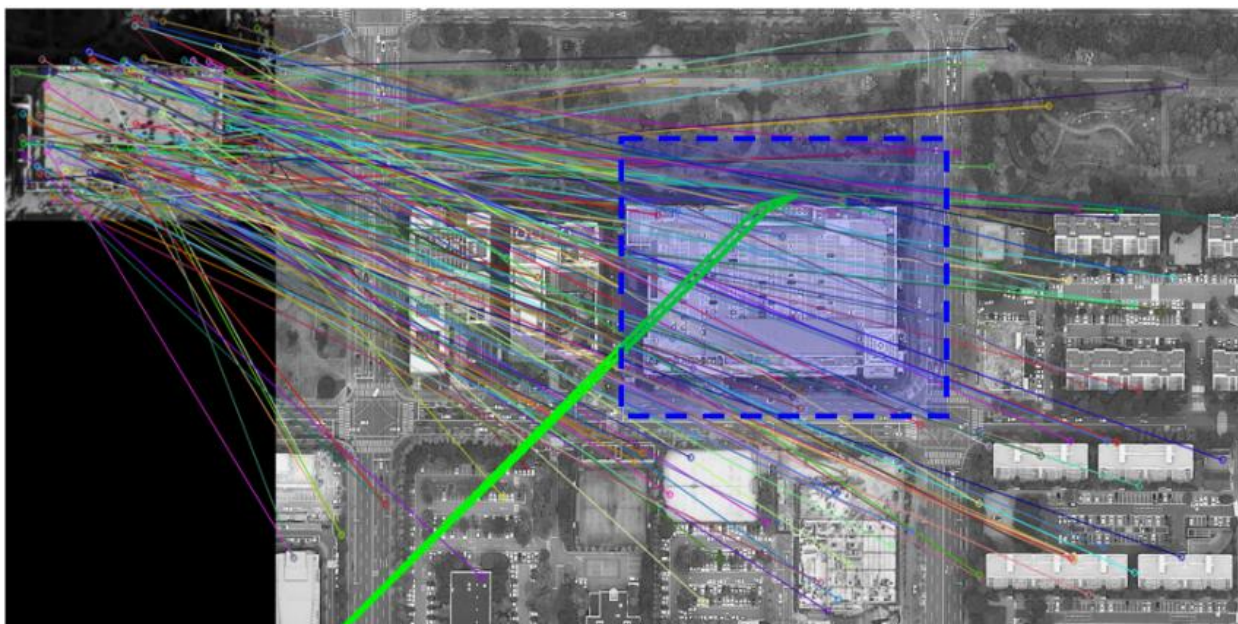


Рисунок 3. Сопоставление вида с БПЛА и спутникового изображения классическим методом компьютерного зрения.

Одним из сильных инструментов для работы с изображениями являются свёрточные нейронные сети, поэтому довольно часто именно этот тип нейросети используется для сопоставления объектов, потому рассмотрим его чуть более подробно. Пример свёрточной архитектуры приведён ниже.

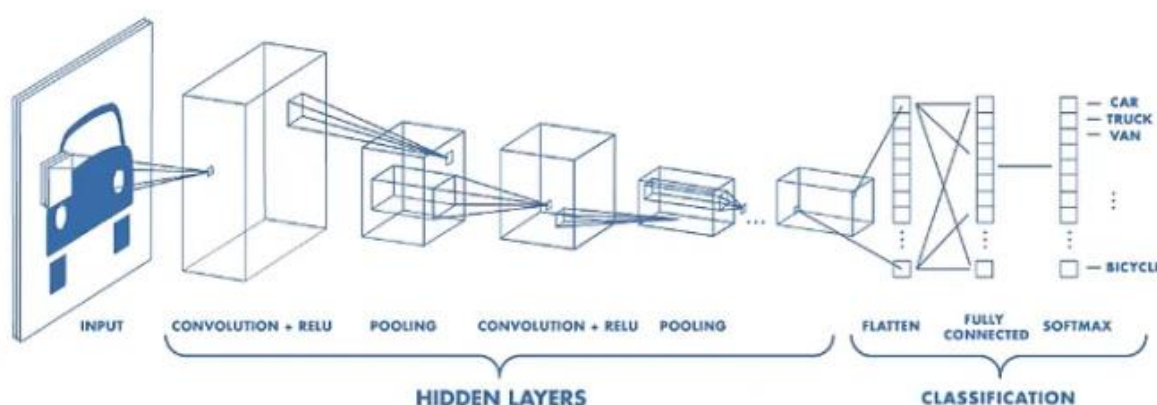


Рисунок 4. Пример свёрточной архитектуры

Основным элементом свёрточной нейронной сети является слой свёртки, принцип работы которых можно описать следующим образом. На

вход в слой подаётся тензор признаков изображения, обычно тензор имеет размер $3 \times H \times W$, где 3 – это количество цветовых каналов изображения, например, RGB, H и W – это размеры изображения. В свёрточном слое применяются фильтры, которые при умножении на элементы тензора дают результат. Разберём свёртку на примере одного пикселя. По каждому пикселю проходит фильтр, или ядро, свёртки. На картинке выбрали пиксель со значением 2. Сначала нужно перемножить по очереди выбранный пиксель и соседние с ним со значениями в матрице свёртки, а потом всё сложить. После этого на исходной картинке заменить первое значение пикселя на то, что получилось, — 13, а значения соседних пикселей не менять. Эти же действия нужно проделать с каждым пикселем на картинке.

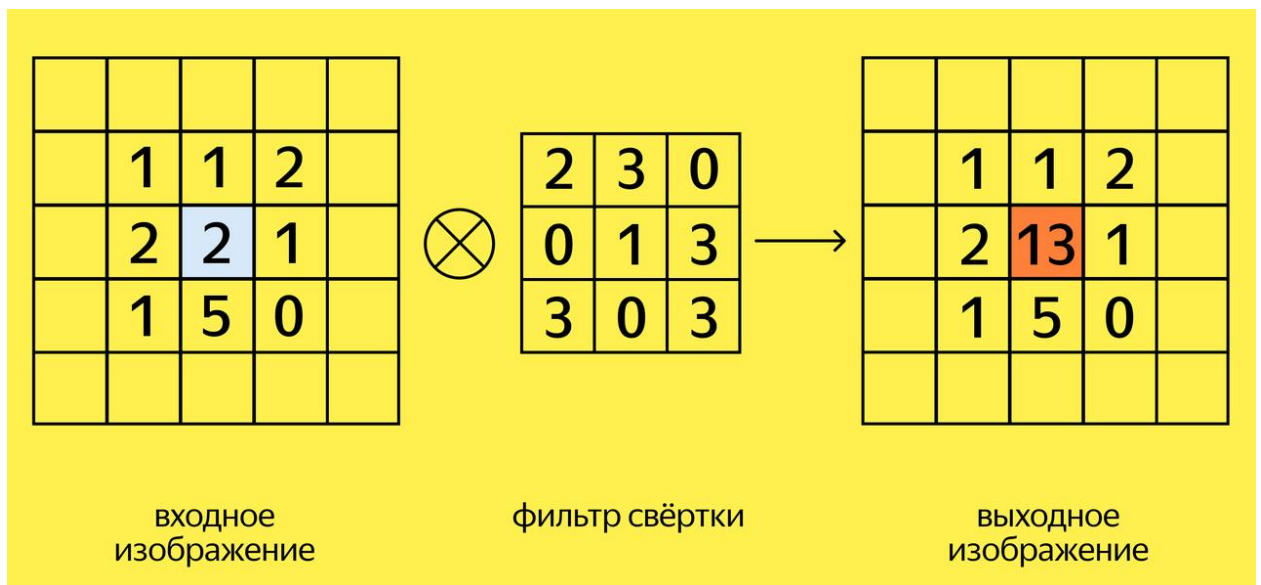


Рисунок 5. Применение операции свёртки на одном пикселе

Общий алгоритм работы свёртки выглядит следующим образом:

1. Фильтр проходит по каждому пикселю изображения, перемножает и суммирует числа своей матрицы и матрицы картинки. На выходе получается новая матрица. Числа полученных матриц суммируются в одну матрицу.
2. К каждому значению матрицы добавляется одинаковое число — значение, на которое переместился фильтр, или шаг свёртки. Шаг равен 1 —

фильтр перемещался на один пиксель. Шаг равен 2 — фильтр шагнул на 2 пикселя. Финальная матрица — это один канал выходной карты признаков.

3. Все каналы, или матрицы, которые получили после обработки изображения фильтрами, объединяются в один тензор. В итоге получается изображение другого размера и с другим числом каналов [8].

Вернёмся к нашей исходной задаче. Пример успешного использования свёрточной нейронной сети для задачи локализации квадрокоптера приведён в статье [9]. Исследователи использовали датасет University – 1652, состоящий из изображений, снятых с дрона, и изображений со спутников. Данный датасет более подробно будет описан дальше в соответствующей главе. В статье описывается использование свёрточной нейронной сети, состоящей из 13 свёрточных слоёв, одного глобального пуллинг-слоя, полносвязного слоя и классификатора. Пуллинг-слой — это слой, который вычисляет значения итогового тензора признака на основе соседей — например, максимальный элемент из 4 или средний элемент из 4 [10]. Пример пуллинг-слоя приведён ниже.

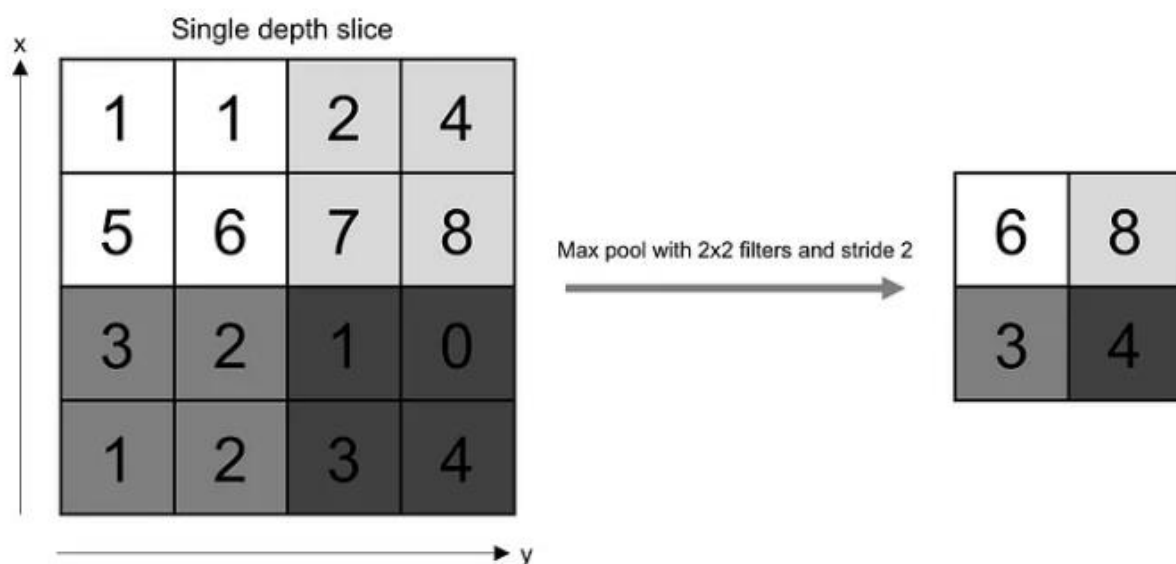


Рисунок 6. Применение операции максимального пуллинга (объединения)

В результате применения такого подхода авторы статьи смогли добиться значения в 87,34% по выбранной им метрике, что говорит о достаточно

высоком качестве работы нейронной сети для задачи сопоставления полученных с дрона изображений со спутниковыми снимками.

Поскольку прогресс не стоит на месте и на данный момент можно утверждать, что наблюдается так называемое «лето» в сфере искусственного интеллекта, то уже существуют более сложные и более эффективные нейронные сети для задачи сопоставления изображений. Одной из таких архитектур является рассматриваемая в данной курсовой работе архитектура на базе Swin Transformer, которая подробно будет описана в следующей главе.

Глава 2. Описание Swin Transformer и предлагаемого на его основе решения задачи сопоставления изображений.

Глава 2.1. Внимание и самовнимание

Применяемая в курсовой работе архитектура нейронной сети базируется на Swin Transformer архитектуре [11]. Перед подробным рассмотрением данной архитектуры необходимо рассмотреть понятия «внимания» и «самовнимания» в нейросетях.

Внимание – это техника, которая используется в рекуррентных и свёрточных нейронных сетях для поиска взаимосвязей между различными частями входных и выходных данных. Изначально этот термин был введён в статье 2017 года под названием «Attention Is All You Need» [12]. Однако, для простоты объяснения воспользуемся другим источником на русском языке, чтобы лучше донести смысл данного понятия. Пример взят из [13]. Рассмотрим на примере базовой архитектуры Seq2seq. Данная сеть состоит из двух рекуррентных сетей – энкодера и декодера. Про устройство рекуррентных нейронных сетей можно почитать в одном из источников [14, 15]. Энкодер – принимает предложение на языке A и сжимает его в вектор скрытого состояния, декодер – выдаёт слово на языке B, принимает последнее скрытое состояние энкодера и предыдущее предсказанное слово. Введём понятия x_i – слова в предложении на языке A и h_i – скрытое состояние энкодера. На рисунке ниже зелёным окрашены блоки энкодера – они получают

на вход x_i и передают скрытое состояние h_i на следующую итерацию. d_i – скрытое состояние декодера, y_i – слова в предложении на языке В.

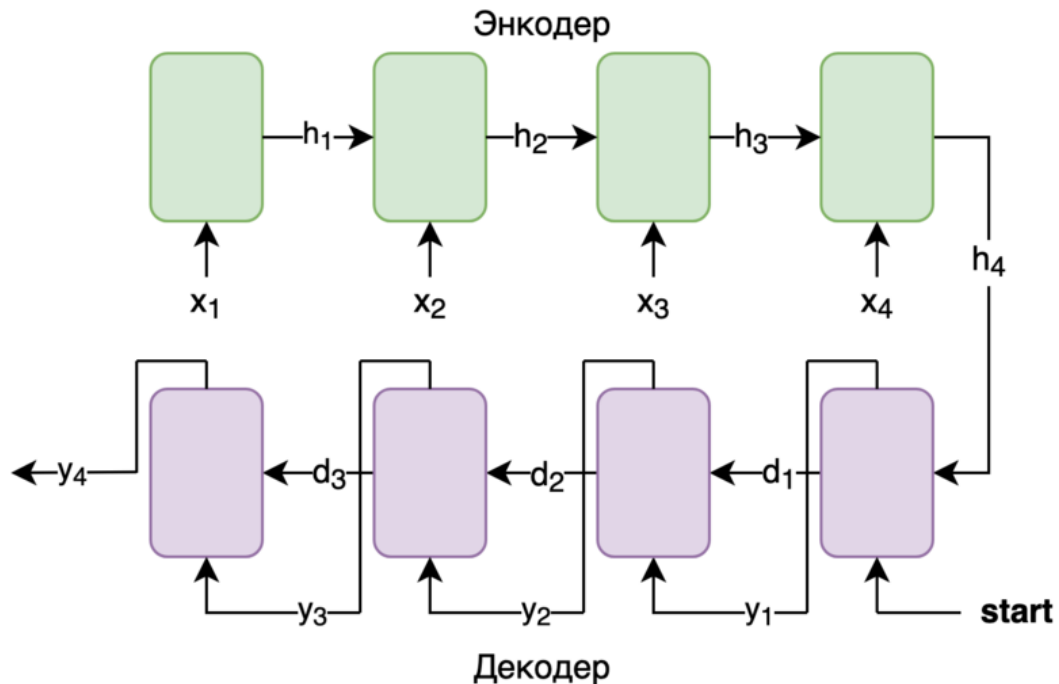


Рисунок 7. Пример работы базовой Seq2seq сети

Фиолетовые блоки декодера получают на вход y_{i-1} или специальный токен в случае первой итерации и возвращают y_i . Передают d_i на следующую итерацию. Перевод считается завершенным при y_i равным специальному токenu. Теперь можно рассмотреть применение внимания для данной архитектуры.

В обычной свёрточной нейронной сети результатом является только последнее скрытое состояние h_m , где m – длина последовательности входных данных, а в случае использования внимания позволяет получить информацию из любого скрытого состояния h_t . Слой механизма внимания часто представляет собой однослойную полносвязную нейронную сеть, на вход которой подаются состояния $h_t, t = 1 \dots m$ и вектор d , в котором содержится зависящий от задачи контекст. Для нашего примера d есть d_{i-1} предыдущей итерации декодера. Выходом данного слоя будет некий вектор s – вектор

оценок, на основании которых на скрытое h_i будет обращено внимание. Далее используется функция softmax:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{k=1}^K e^{z_k}}, \quad (1)$$

Далее считается вектор $c = \sum_{i=1}^m \sigma(z)_i h_i$, который содержит информацию обо всех скрытых слоях.

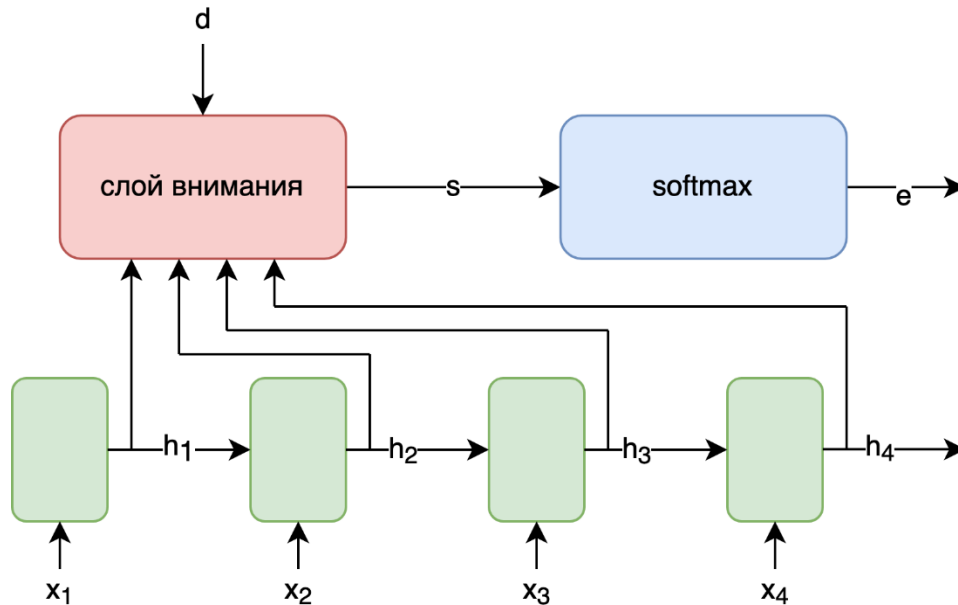


Рисунок 8. Обобщённый механизм внимания в рекуррентной нейронной сети

Наконец, можно рассмотреть общую схему. Механизм внимания добавляется в рекуррентную нейронную между энкодером и декодером. На схеме ниже жёлтым помечен агрегатор скрытых состояний энкодера, которые собирает их все в себе и возвращает всю последовательность скрытых состояний. c_i – вектор контекста на итерации i . Блок внимания принимает состояние и вектор d_{i-1} и возвращает вектор контекста c_i . На блок декодера теперь подаётся не только y_{i-1} , но y_{i-1} и c_i . Таким образом, декодер фокусируется на определённых скрытых состояниях, например, на какие слова или конкретные признаки.

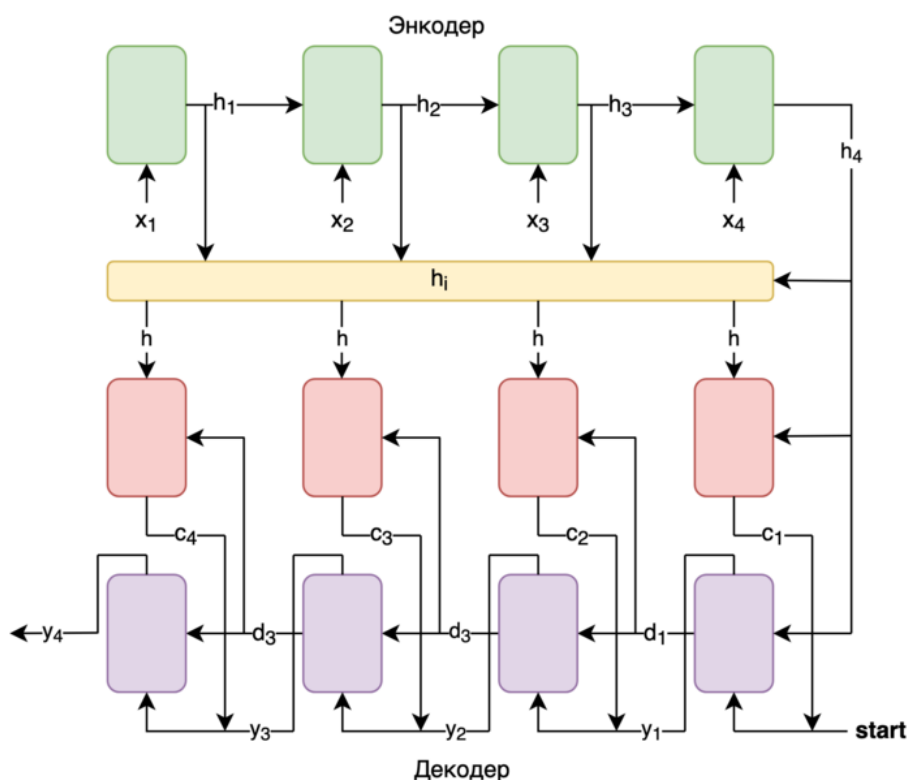


Рисунок 9. Пример работы Seq2seq с механизмом внимания

Теперь, рассмотрев, что такое внимание, можно перейти к понятию самовнимания. Данный механизм применяется в структурах под названием трансформеры, в основе которых лежат последовательные блоки энкодеров и декодеров со схожей архитектурой [12, 16]. Общее устройство приведено ниже.

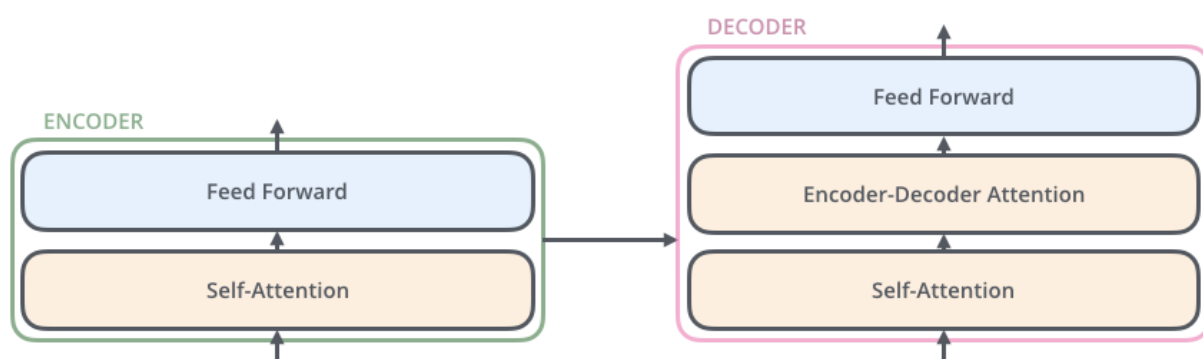


Рисунок 10. Блоки энкодера и декодера в трансформере в общем случае

Входные данные энкодера сначала проходят через слой внутреннего внимания — слой, который помогает кодировщику смотреть на другие слова

во входном предложении, когда он кодирует конкретное слово. Выходные данные уровня внутреннего внимания передаются в нейронную сеть с прямой связью. К каждой позиции независимо применяется одна и та же сеть прямой связи. У декодера есть оба этих слоя, но между ними находится слой внимания, который помогает декодеру сосредоточиться на соответствующих частях входного предложения.

Теперь рассмотрим блок самовнимания (Self-Attention) на примере, взятом из указанного выше источника. Предположим, что есть предложение: *“The animal didn’t cross the street because it was too tired”*. В нём не так просто понять, к чему относится местоимение “it” – к улице или к животному. Мы как люди понимаем, что речь идёт о животном, но не нейронная сеть. По мере того как модель обрабатывает каждое слово (каждую позицию во входной последовательности), собственное внимание позволяет ей искать в других позициях во входной последовательности подсказки, которые могут помочь улучшить кодирование этого слова. Самовнимание — это метод, который Трансформер использует, чтобы встроить «понимание» других релевантных слов в то, которое мы сейчас обрабатываем.

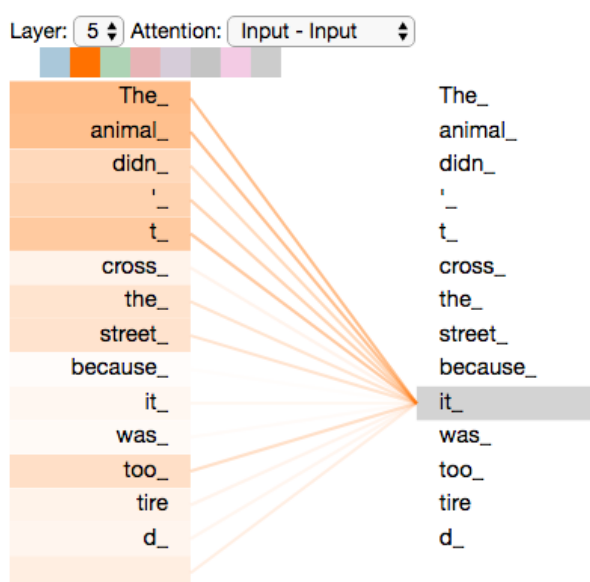


Рисунок 11. При кодировании местоимения “it” больше всего энкодер фокусируется на ‘the’, ‘animal’

Постараемся максимально кратко, но в то же время полно раскрыть работу самовнимания. **Первым шагом** в вычислении самовнимания является создание трех векторов из каждого из входных векторов кодировщика (в данном случае встраивание каждого слова). Итак, для каждого слова мы создаем вектор запроса, вектор ключа и вектор значения. Эти векторы создаются путем умножения вложения на три матрицы, которые мы обучали в процессе обучения. Новые векторы меньше по размеру, чем вектор вложения. Их размерность равна 64, в то время как векторы ввода/вывода встраивания и кодирования имеют размерность 512. Это не является правилом, а обусловлено выбором архитектуры примера, чтобы сделать вычисление многоголового внимания (в основном) постоянным.

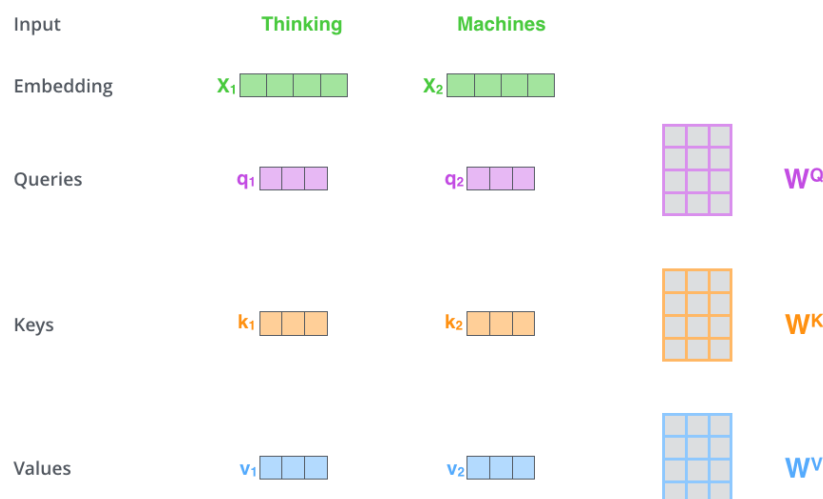


Рисунок 12. Общий вид расчёта в самовнимании

Умножение x_1 на весовую матрицу W^Q дает q_1 , вектор «запроса», связанный с этим словом. В итоге мы создаем «запрос», «ключ» и «значение» проекции каждого слова во входном предложении.

Второй шаг в подсчете внимания к себе — подсчет баллов. Скажем, мы вычисляем само-внимание для первого слова в этом примере «Думаю». Нам нужно сопоставить каждое слово входного предложения с этим словом. Оценка определяет, сколько внимания нужно уделять другим частям входного предложения, когда мы кодируем слово в определенной позиции. Оценка

рассчитывается путем скалярного произведения вектора запроса на ключевой вектор соответствующего слова, которое мы оцениваем. Итак, если мы обрабатываем самовнимание для слова в позиции № 1, первая оценка будет скалярным произведением q_1 и k_1 . Вторая оценка будет скалярным произведением q_1 и k_2 .

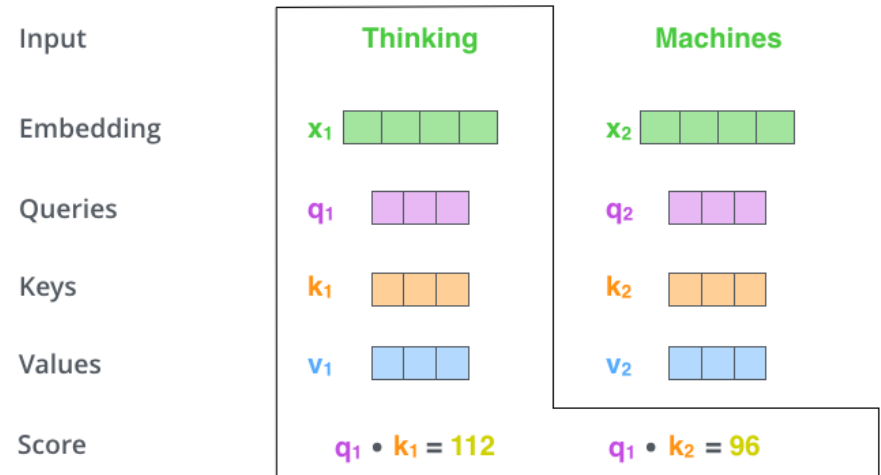


Рисунок 13. Второй шаг работы механизма самовнимания

Третий и четвертый шаги — разделить баллы на 8 (квадратный корень из размерности ключевых векторов, использованных в статье — 64. Это приводит к более стабильным градиентам. Здесь могут быть и другие возможные значения, но это общепринятая практика), затем передаётся результат через операцию softmax. Softmax нормализует оценки, чтобы все они были положительными и в сумме давали 1.

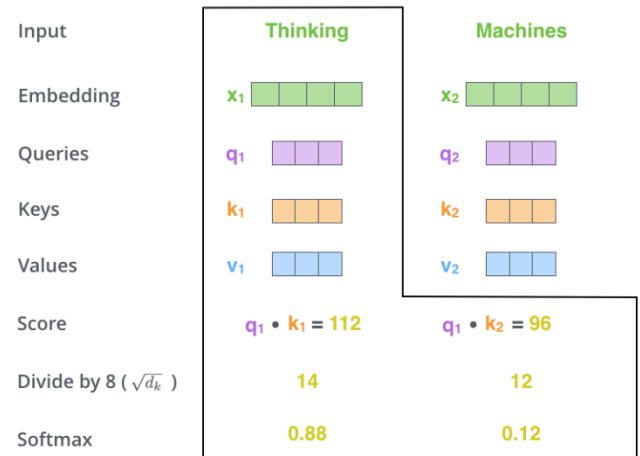


Рисунок 14. Третий и четвёртый шаги механизма работы самовнимания

Пятый шаг — умножить каждый вектор значений на оценку softmax (при подготовке к их суммированию). Смысл этого здесь состоит в том, чтобы сохранить нетронутыми значения слов, на которых мы хотим сосредоточиться, и заглушить нерелевантные слова (например, умножив их на крошечные числа, такие как 0,001). **Шестой** шаг заключается в суммировании взвешенных векторов значений. Это производит вывод слоя внутреннего внимания в этой позиции (для первого слова).

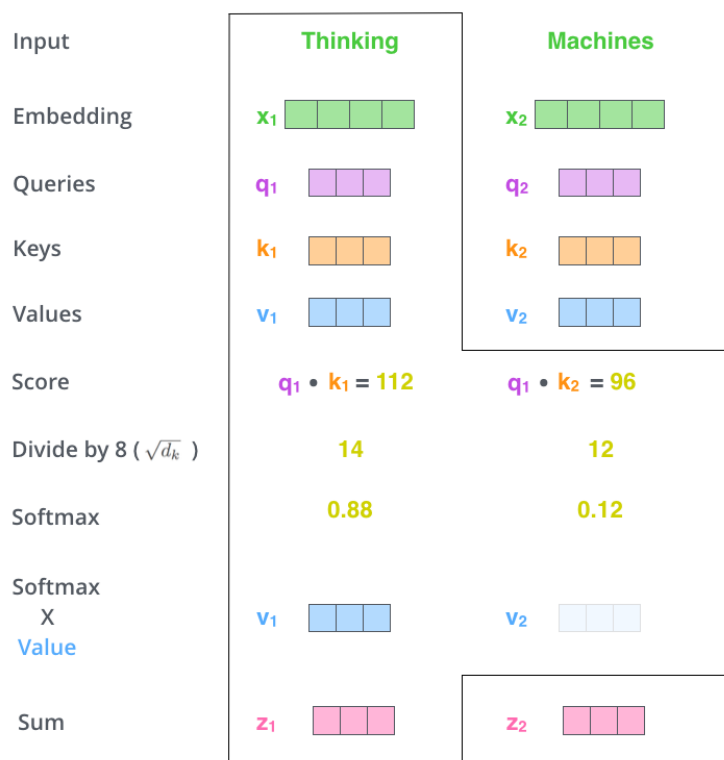


Рисунок 15. Пятый и шестой шаги в работе механизма самовнимания

На этом расчет собственного внимания заканчивается. Результирующий вектор — это тот, который мы можем отправить в нейронную сеть с прямой связью. Однако в реальной реализации этот расчет выполняется в матричной форме для более быстрой обработки. Однако, чтобы не перегружать отчет по курсовой работе матричный случай не будет здесь расписан. Также для улучшения работы данного алгоритма используется метод “multi-head” внимание, суть которого можно кратко описать так. Это дает слою внимания несколько «подпространств представления». При многоголовом внимании у нас есть не один, а несколько наборов весовых матриц Запрос/Ключ/Значение

(преобразователь использует восемь головок внимания, поэтому мы получаем восемь наборов для каждого кодировщика/декодера). Каждый из этих наборов инициализируется случайным образом. Затем, после обучения, каждый набор используется для проецирования входных вложений (или векторов из нижних кодеров/декодеров) в другое подпространство представления. Здесь тоже не будем совсем подробно описывать, чтобы не перегружать отчёт, учитывая, что это не является фокусом работы, приведём только общую схему.

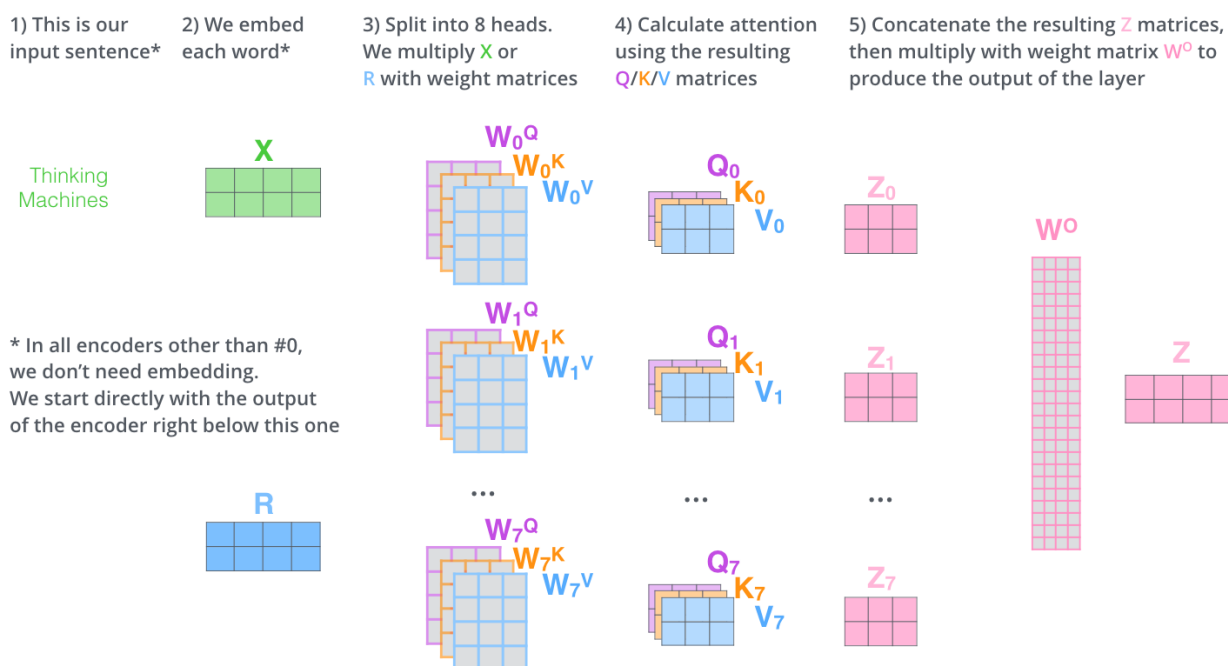


Рисунок 16. Общий вид "multi-head" внимания, подробнее про который можно почитать в [16].

Теперь, рассмотрев механизм внимания и самовнимания, можно, наконец, перейти к используемой в рамках курсовой работы архитектуре Swin Transformer.

Глава 2.2. Архитектура Swin Transformer

Данная архитектура была представлена впервые в статье [11]. Данная архитектура была создана, чтобы заменить упоминаемый выше блок "multi-head" внимания в трансформерах (MSA – блок) с помощью специального блока, использующего сдвинутое окно (shifted window). Сейчас опишем

принцип его работы. Сдвинутые окна соединяют окна предыдущего слоя, обеспечивая связи между ними.

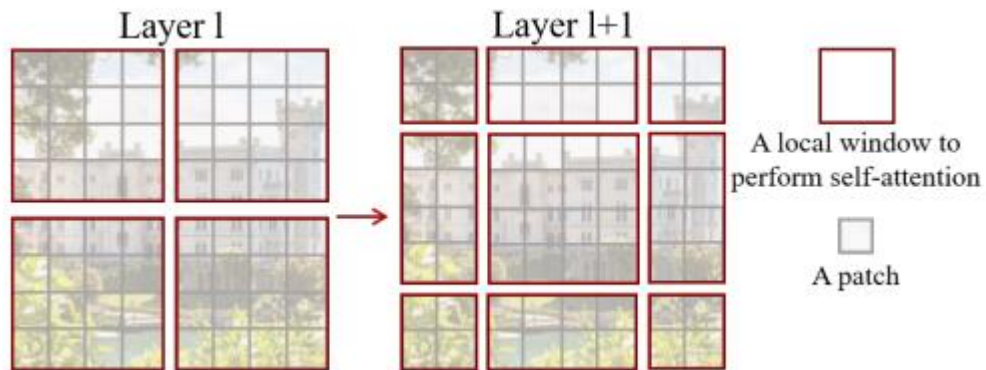


Рисунок 17. Иллюстрация работы сдвинутого окна в механизме самовнимания

В слое l (слева) используется обычная схема разбиения окон, и в каждом окне учитывается собственное внимание. В следующем слое $l + 1$ (справа) разбиение окон сдвигается, что приводит к появлению новых окон. Вычисление собственного внимания в новых окнах пересекает границы предыдущих окон в слое l , обеспечение связи между ними. Первый модуль использует обычную стратегию разделения окон, которая начинается с верхнего левого пикселя, а карта признаков 8×8 равномерно делится на окна 2×2 размером 4×4 ($M = 4$). Затем следующий модуль принимает конфигурацию окон, которая отличается от конфигурации предыдущего уровня, перемещая окна на $\left(\left\lceil \frac{M}{2} \right\rceil, \left\lceil \frac{M}{2} \right\rceil\right)$ пикселя от регулярно разделенных окон. Подход к разделению смещенного окна вводит связи между соседними неперекрывающимися окнами на предыдущем уровне и оказался эффективным.

Введём сокращения, которые будут полезны при дальнейшем рассмотрении Swin-архитектуры. MLP – Multilayer Perceptron – многослойный перцептрон, LN – LayerNorm – слой нормализации, W-MSA – слой multi-head самовнимания с обычным окном и SW-MSA – слой multi-head самовнимания со сдвинутым окном. Итоговая структура Swin приведена ниже.

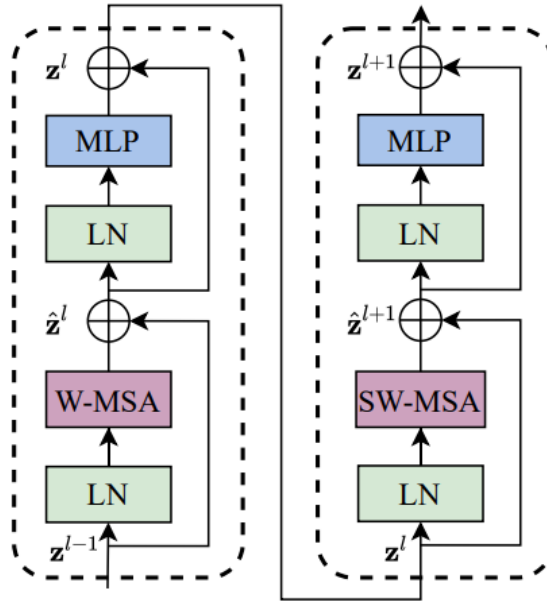


Рисунок 18. Два Swin Transformer блока

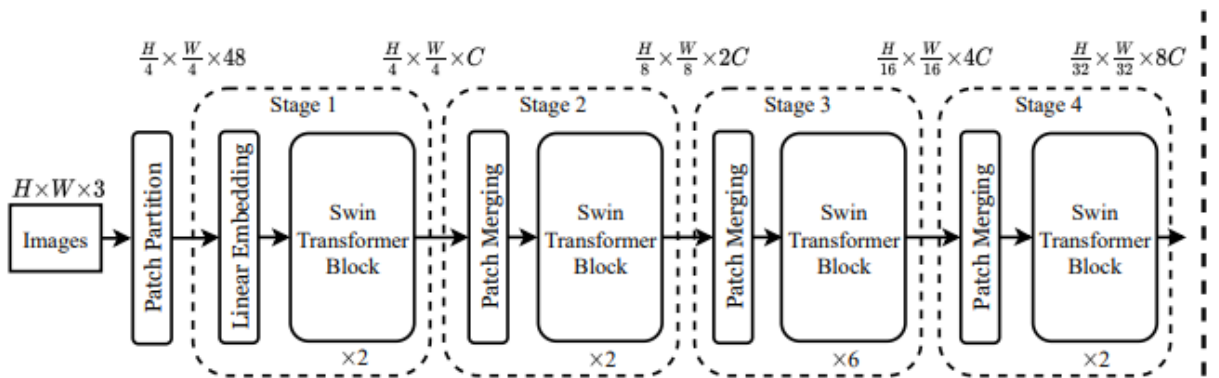


Рисунок 19. Общий вид архитектуры

Опишем архитектуру, поскольку она используется в работе. Модуль разделения патчей (patch splitting module) в нейронных сетях относится к методу разделения входного изображения на множество патчей (небольших фрагментов) и их последующей обработке независимо друг от друга. Это позволяет нейронной сети работать с более мелкими деталями изображения и улучшить ее способность к обнаружению и анализу локальных особенностей. Каждый патч обрабатывается как «токен», и его функция задается как конкатенация необработанных значений RGB пикселей. В нашей реализации мы используем размер патча 4×4 , и, таким образом, размер объекта каждого патча составляет $4 \times 4 \times 3 = 48$. К этому объекту с необработанными

значениями применяется слой линейного внедрения, чтобы проецировать его на произвольное измерение (обозначается как C). Несколько блоков Transformer с модифицированным вычислением внутреннего внимания (блоки Swin Transformer) применяются к этим токенам исправления. Блоки Transformer поддерживают количество токенов $\left(\frac{H}{4} \times \frac{W}{4}\right)$, и вместе с линейным встраиванием называются «этап 1».

Количество токенов уменьшается путем слияния слоев патчей по мере того, как сеть становится глубже. Первый слой слияния патчей объединяет объекты каждой группы 2×2 соседних патчей и применяет линейный слой к конкатенированным объектам размерности $4C$. Это уменьшает количество токенов на кратное $2 \times 2 = 4$ (уменьшение разрешения в 2 раза), а выходное измерение устанавливается равным $2C$. Блоки Swin Transformer применяются впоследствии для преобразования признаков с сохранением разрешения на уровне $H \times W$. Этот первый блок слияния исправлений и преобразования функций обозначается как «Этап 2». Процедура повторяется дважды на этапах «Этап 3» и «Этап 4» с выходными разрешениями в $\left(\frac{H}{16} \times \frac{W}{16}\right)$ и $\left(\frac{H}{32} \times \frac{W}{32}\right)$ соответственно. Теперь можно перейти к применяемой в работе архитектуре.

Глава 2.3. Модификация рассматриваемой архитектуры

Датасет, который подробно будет описан в следующей главе, состоит из изображений университетов, снятых с дрона, и изображений тех же университетов, полученных со снимков. На вход в нейронную сеть подаются сразу изображения с дрона и со спутника, что видно из картинки архитектуры, приведённой ниже.

Применяемая архитектура отличается от Swin Transformer 1 блоком под названием Semantic Guidance Module (SGM) [17]. Данный модуль позволяет учитывать контекстную информацию. Если принять карту признаков, поступающую на вход как M_j^i и M размером 64×768 в целом, то модуль SGM

суммирует M_j^i все карты по одному каналу. Операцию можно записать математически как

$$M_i = \sum_{j=0}^{768} M_i^j, \quad i \in [0,63] \quad (2)$$

После этой операции размер M_i составляет 64 x 1.

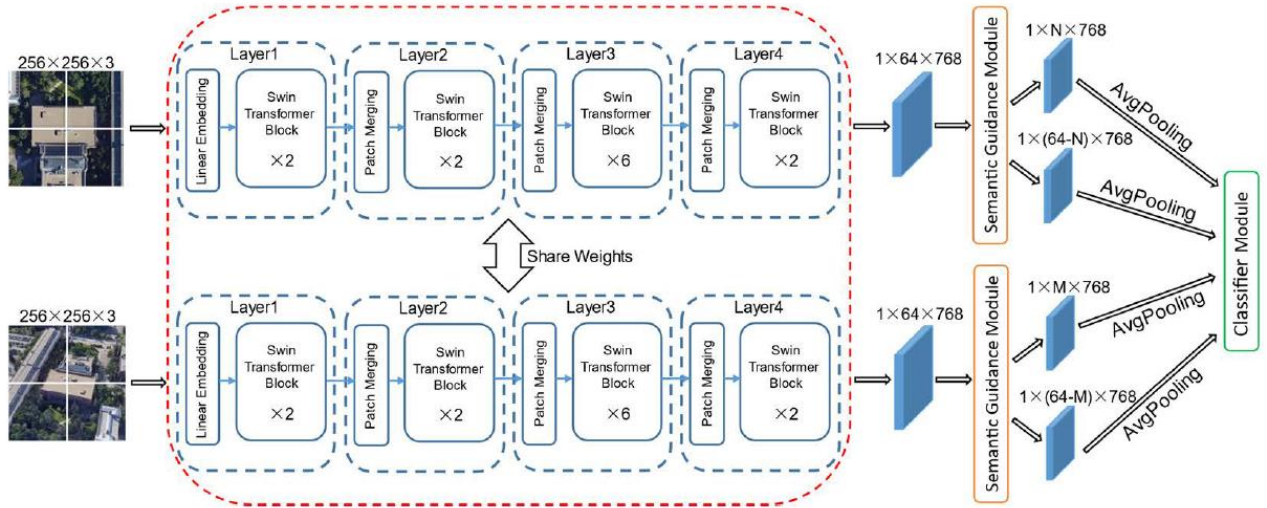


Рисунок 20. Используемая в работе архитектура модифицированного Swin Transformer

Далее проводится операция нормализации:

$$M_i = \frac{M_i - \text{Minimum}(M_i)}{\text{Maximum}(M_i) - \text{Minimum}(M_i)} \quad (3)$$

После данный слой считает градиент между соседями и делит общую карту признаков на две части. На рисунке ниже на правой части можно увидеть, что все карты поделены на две части – передний план (здание и архитектура) и задний план (окружающая территория). Такое разделение позволяет учитывать контекстную информацию. Формульно это выглядит:

$$i_{\text{position}} = \text{argmax} \left(\frac{M_{i+1} - M_i}{M_i} \right) \quad (4)$$

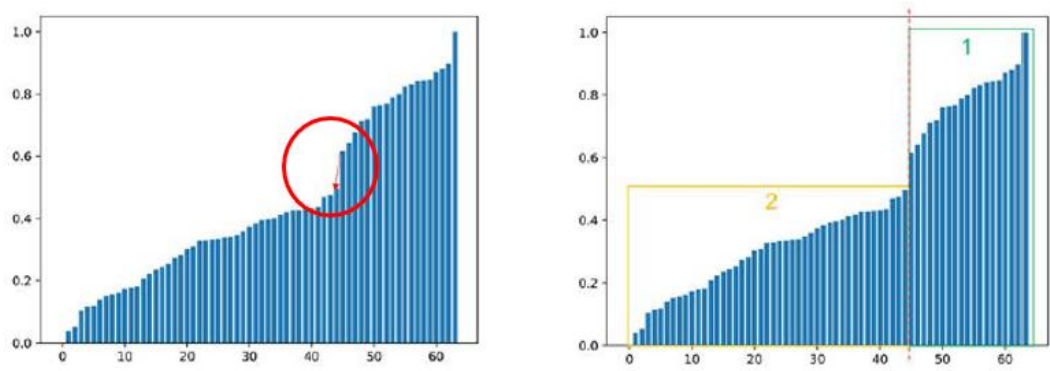


Рисунок 21. Расчёт градиентов и разделение карты признаков на две части

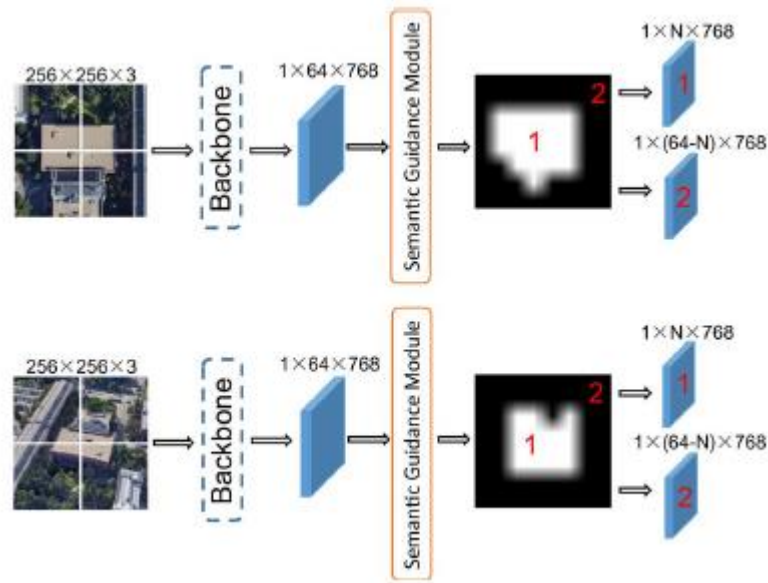


Рисунок 22. Работа блока SGM

Теперь для каждого набора карт признаков, будь то передний план или задний план и будь то фото с дрона или фото со спутника, применяется операция усреднённого пуллинга. И в итоге размер каждой карты 1×768 . Формульно это выглядит так:

$$Y_i = Avgpool(X_i) \quad i \in [0,1], \quad (5)$$

где X_i – это разделённая карта признаков, Y_i – итоговый вектор для этой карты.

Далее эти векторы отправляются на классификатор, который состоит из полносвязного слоя, слоя с батч-нормализацией, слоя дроп-аут и выходного классифицирующего слоя. Для каждого из четырёх векторов применяется

softmax, формула которого была приведена в разделе про внимание. В качестве оптимизатора используется кросс-энтропия, формула которой:

$$LOSS_{CE}(p, y) = \begin{cases} -\log(p) & y = 1 \\ -\log(1 - p) & y = 0 \end{cases}, \quad (6)$$

где p – предсказанный результат, y – абсолютная истина.

В человеческом понимании можно данную задачу сформулировать так: векторы признаков объектов, имеющих одно и то же географическое положение имеют меньшее расстояние, чем разные объекты. Итоговая функция потерь может быть записана:

$$Loss = \sum_{i=0}^n (L_{Drone}^i - L_{Satellite}^i), \quad (7)$$

n – количество разделённых карт признаков.

На этапе тестирования модуль классификатора убирается и получаем векторное представление изображений, на основе которого считается Евклидово расстояние по формуле для конкатенированных векторов:

$$D = \|V_{Drone} - V_{Satellite}\|_2 \quad (8)$$

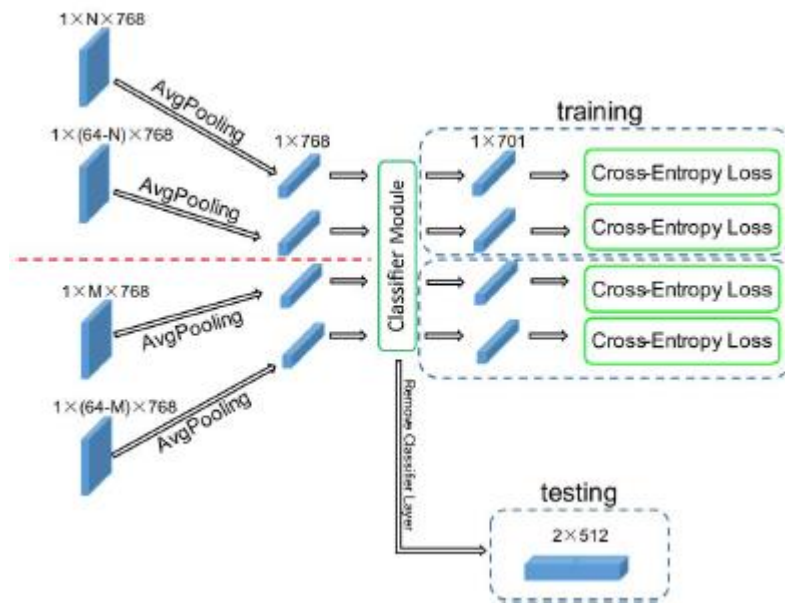


Рисунок 23. Окончание фазы обучения и фаза тестирования

Глава 3. Описание набора данных и параметры обучения нейронной сети.

Для проведения экспериментов использовался датасет University-1652, который состоит из 1652 географических объектов, располагающихся на территории 72 университетов со всего мира. Сам датасет состоит из трёх типов данных – виды с дрона, виды со спутника и виды с улицы (в нашей работе не рассматривался). У каждого объекта имеется только 1 спутниковый снимок и чуть более 50 фотографий, полученных с разных ракурсов с помощью квадрокоптера. Целью нейросети и работы в целом является соотнесение фотографии со спутника с фотографиями с дрона. Причём рассматривается задача «Дрон» → «Спутник», то есть для тестирования подаётся одно изображение с дрона и необходимо найти соответствующий спутниковый снимок. Общая статистика по датасету приведена ниже. Все картинки имеют размер 512 x 512 и являются цветными изображениями, то есть 3 канала.

Таблица 1. Описание датасета

Обучающий датасет		
Тип вида	Количество зданий	Число изображений
С дрона	701	37 854
Со спутника	701	701
Тренировочный датасет		
С дрона	951	51 355
Со спутника	951	951

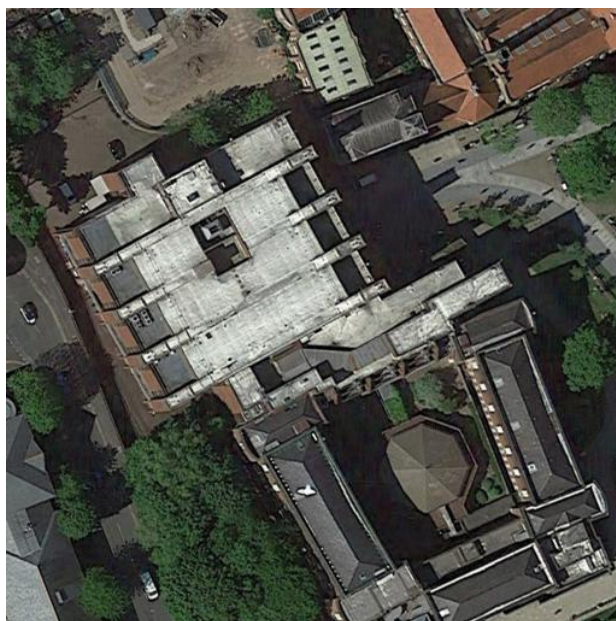


Рисунок 24. Пример спутникового снимка из обучающей выборки

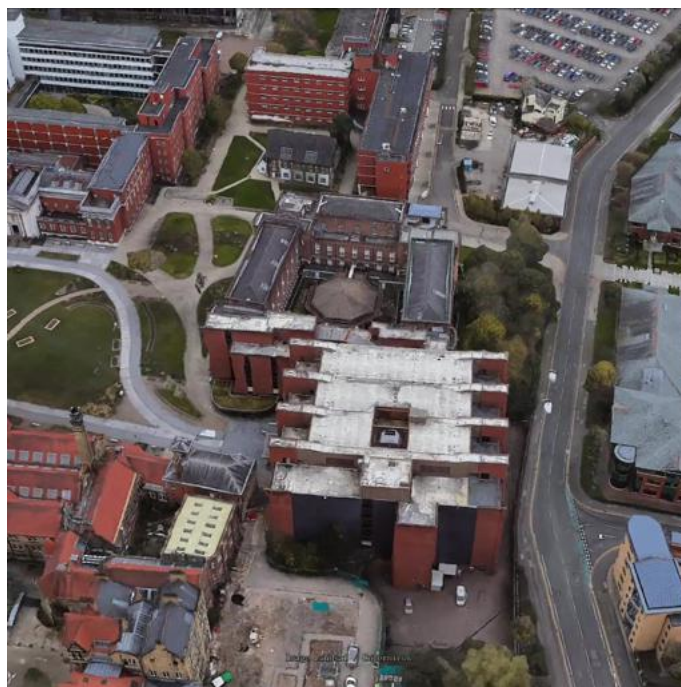


Рисунок 25. Пример фотографии того же здания, полученного с помощью дрона

Глава 4. Полученные результаты.

Для сравнения работы описанного выше решения использовались две архитектуры: Swin Transformer + SGM и ResNet50. Почему использовались 2 нейросети? Основная трудность, связанная с обучением таких сетей, состоит

в том, что необходимо очень много времени на обучение. В первый раз была обучена Swin Transformer + SGM с оптимизатором Adam и размером батча в 8, скорость обучения была 0.002, момент = 0. После 15 часов обучения и 15 эпох было обнаружено, что нейросеть не обучилась – функция потерь упала с 26.870 до 26.700 и при проверке на тестовых данных был сделан вывод, что нейросеть не была обучена. Далее было проведено повторное обучение для двух нейросетей и с другим оптимизатором для Swin-архитектуры. К сожалению, в силу ограниченности вычислительных ресурсов и времени каждая нейронная сеть обучалась только 10 эпох. Обе модели (ResNet50 и базовая Swin архитектура). загружались с предобученными весами «Imagenet1K». Модуль Swin был взят из PyTorch, модуль классификатора, SGM для данной архитектуры были реализованы нами.

Таблица 2. Параметры обучения двух моделей

Параметры	Swin Transformer + SGM	ResNet50
Количество эпох обучения	10	10
Оптимизатор	SGD, момент = 0.9	SGD, момент = 0.9
Скорость обучения	0.01	0.001
Размер батча	32	10
Точность вычислений	FP16	FP16
Функция потерь	Кросс-энтропия	Кросс-энтропия

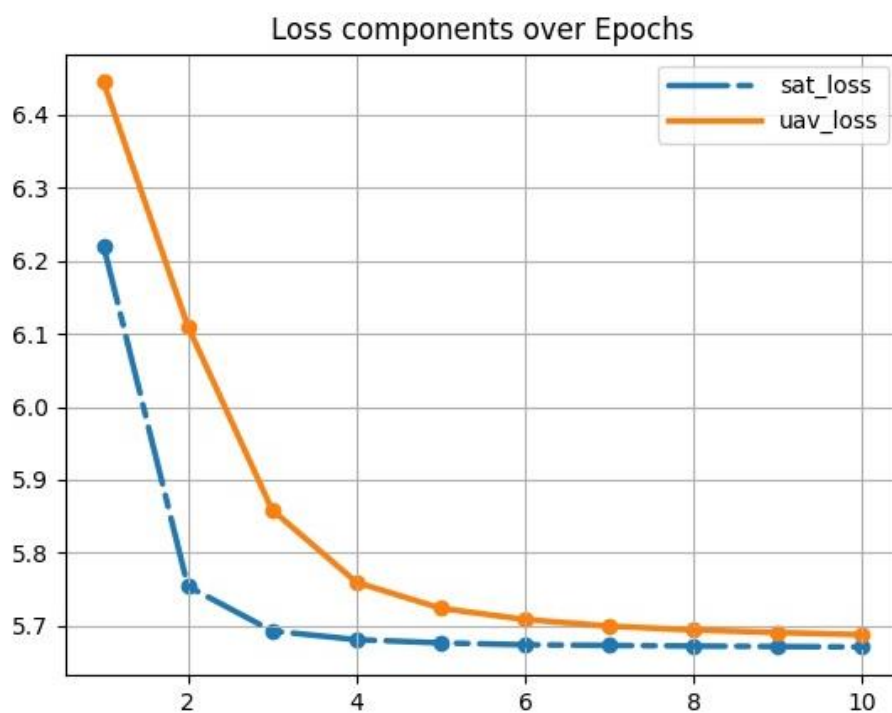


Рисунок 26. Обучение ResNet50 и компоненты функции потерь

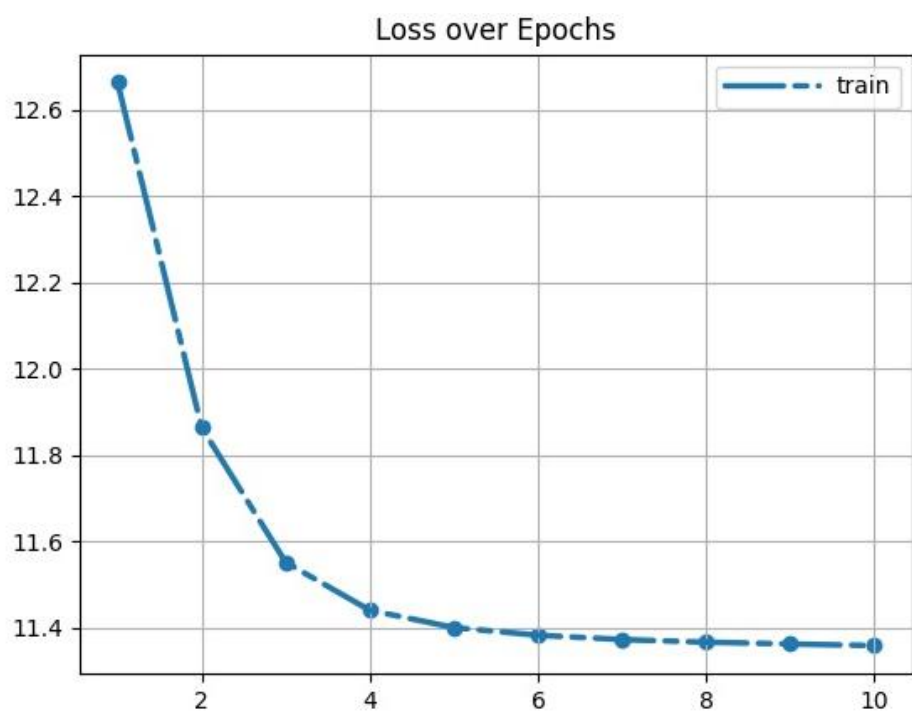


Рисунок 27. Обучение ResNet50 и общая функция потерь

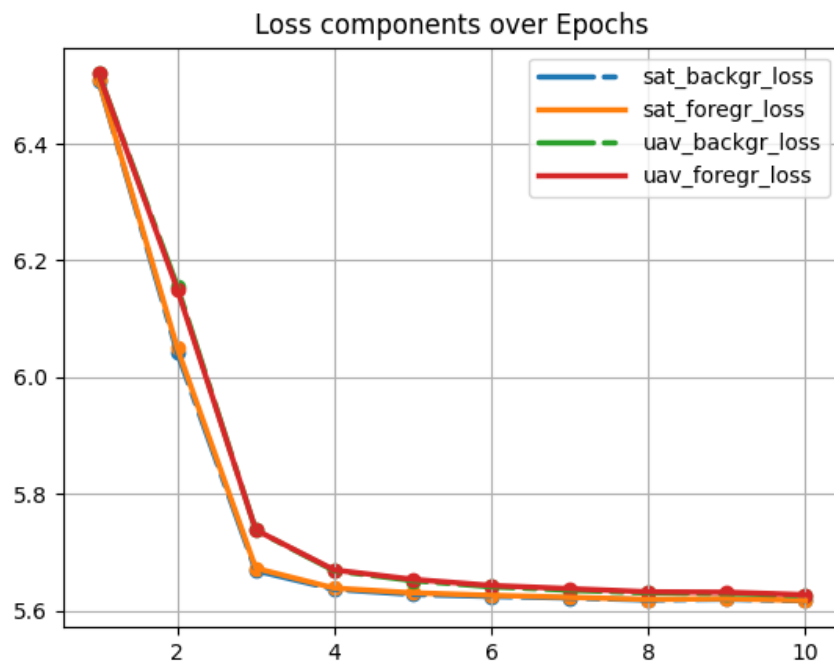


Рисунок 28. Обучение Swin Transformer и компоненты функции потерь

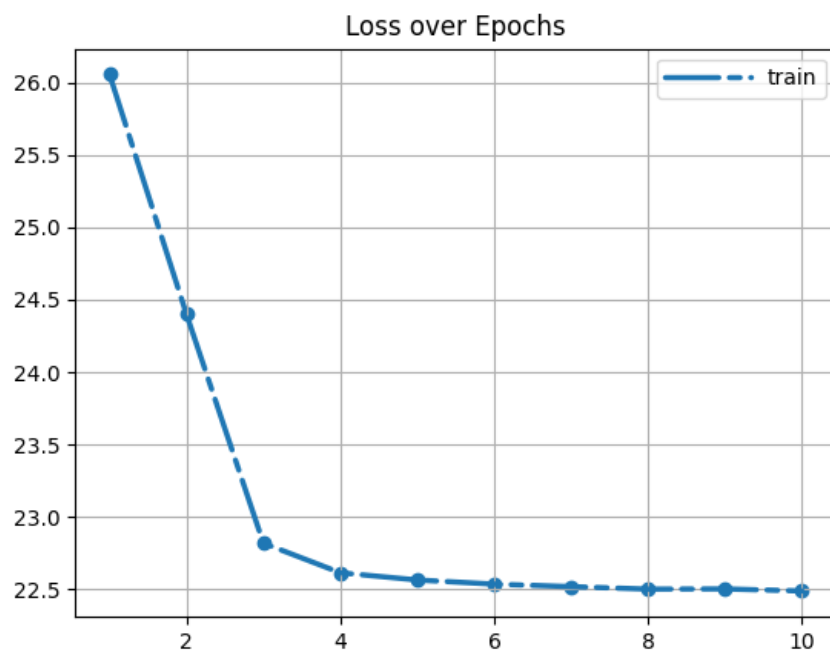


Рисунок 29. Обучение Swin Transformer и общая функция потерь

По приведённым выше графикам можно сделать вывод, что обе нейросети обучаются, поскольку происходит уменьшение функции потерь

Для верификации предложенных архитектур нейронных сетей использовалась метрика Recall@k. Математически эта метрика может быть записана следующим образом [9]:

$$Recall@K = f(x) = \begin{cases} 1, & \text{if } order_{true-matched} < K + 1 \\ 0, & \text{otherwise} \end{cases} \quad (9)$$

Поясним словами: если корректно определённое изображение входит в первые K результатов, то есть идёт до $K + 1$, то значение метрики равно 1. В противном случае оно равно 0. В ходе работы K вычислялась для 1, 5 и 10 фотографий. Для тестирования работы нейросетей использовались не все файлы для вычисления метрики, а какая-то их часть. Такое решение было обосновано двумя факторами: во-первых, данная нейросеть должна обеспечивать квадрокоптеру возможность ориентироваться на местности вдоль заданного маршрута, а если маршрут известен, то известны и сооружения, находящиеся на этом пути, потому нецелесообразно использовать весь тренировочный датасет для оценки; во-вторых, из-за ограничений на вычислительные мощности и временные рамки.

Поэтому метрика считалась для 20, 35, 50 и 100 объектов (местоположений) для ResNet50 и для 20 и 35 объектов для Swin Transformer с модулем SGM.

Таблица 3. Итоговая таблица расчёта метрики Recall@k в %. Сравнение результатов

		20	35	50	100
ResNet50	Recall@1	87,3	69,3	58,2	49,7
	Recall@5	98,1	91,3	87,1	76,7
	Recall@10	99,9	96,9	94,4	85,3
Swin Transformer	Recall@1	70,4	59,1	-	-
	Recall@5	94,2	92,3	-	-
	Recall@10	99,3	97,7	-	-

Видно, что по итоговой метрике для 20 объектов результат для ResNet50 оказался лучше, чем для Swin Transformer. Полагаем, что это можно объяснить тем, что Swin Transformer медленнее обучался в силу ограниченности времени и вычислительных мощностей, а также в силу того, что могли быть выбраны не самые лучшие параметры обучения. Однако, для 35 объектов по R@5 и R@10 у Swin Transformer метрика оказалась выше, что говорит о том, что данная модель работает как минимум не хуже ResNet50 для данной задачи.

Заключение

В ходе выполнения курсовой работы был реализован алгоритм сопоставления изображений, полученных с квадрокоптера и спутника из датасета University-1652, на основе нейросетевого подхода, а именно с помощью двух нейронных сетей – Swin Transformer с модификацией SGM и ResNet50. Для того, чтобы выполнить эту цель, были решены следующие задачи.

Были рассмотрены основные подходы к решению задачи локализации, рассмотрены положительные и отрицательные стороны каждого из подходов, в итоге был выбран нейросетевой подход. Была описана общая структура свёрточной нейронной сети. Также были рассмотрены механизм внимания и, как его продолжение, механизм самовнимания, который активно применяется в нейросетях типа трансформер. Приведено общее описание архитектуры трансформеров. Была описана архитектура Swin Transformer и её модификация с добавлением Semantic Guidance Module, позволяющая лучше учитывать контекст данных.

Был описан датасет University-1652 и на его основе проведено обучение двух нейросетей. Для этого базовая часть Swin Transformer и ResNet50 были загружены с весами на основе ImageNet-1K, однако Swin Transformer был дополнен модулем Semantic Guidance Module, классификатором, которые были реализованы в ходе выполнения работы. Также реализована загрузка датасета в Data Loader для последующей загрузки в нейросети.

Было проведено три обучения – Swin Transformer с оптимизатором Adam, Swin Transformer с оптимизатором SGD, ResNet50 с SGD. Первое обучение велось 15 часов и за 15 эпох результат обучения был практически нулевым из-за некорректно выбранного оптимизатора и маленького шага обучения. Два других процесса обучения суммарно заняли около 25 часов, каждая сеть обучалась в силу ограниченности вычислительных ресурсов по 10 эпох. В итоге по метрике Recall@5 были получены следующие результаты для обеих нейросетей для 20, 35 и 50 объектов соответственно: ResNet50 – 98,1;

91,3; 87,1; Swin Transformer – 94,2 для 20 и 92,3 для 35 объектов; Данный результат можно объяснить тем, что Swin Transformer мог недообучаться, так как было проведено всего 10 эпох в силу ограничений на вычислительные мощности. При этом обе нейросети показали достаточно высокий результат по выбранной метрике. И для метрики R@5 и R@10 для 35 объектов Swin архитектура показала результат лучше ResNet50. Можно предположить, что данная модель лучше работает для большого количества объектов, чем ResNet50, однако для этого необходимо провести дополнительное исследование.

Список литературы

1. Варламова Л.П. Применение беспилотных летательных аппаратов в обеспечении технологической безопасности // Journal of Technical and Natural Sciences 5(14), 2019. – 54 – 90.;
2. Трефилов П.М. Сравнительный анализ улучшения точностных характеристик инерциальных навигационных систем // Сборник трудов XIII Всероссийского совещания по проблемам управления ВСПУ-2019. – с. 470-474.;
3. Stephanie A.C. Schuckers. Spoofing and Anti-Spoofing Measures. Information Security Technical Report, vol. 7, No. 4, 2002, p. 56-62.;
4. Andy Couturier, Moulay A. Akhloufi, A review on absolute visual localization for UAV, Robotics and Autonomous Systems, Volume 135, 2021, 103666, ISSN 0921-8890, <https://doi.org/10.1016/j.robot.2020.103666>.
5. M. H. Mughal, M. J. Khokhar and M. Shahzad, "Assisting UAV Localization Via Deep Contextual Image Matching," in IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, vol. 14, pp. 2445-2457, 2021, doi: 10.1109/JSTARS.2021.3054832.
6. Krizhevsky, Alex & Sutskever, Ilya & Hinton, Geoffrey. (2012). ImageNet Classification with Deep Convolutional Neural Networks. Neural Information Processing Systems. 25. 10.1145/3065386.
7. Choi, Junho and Hyun Myung. "BRM Localization: UAV Localization in GNSS-Denied Environments Based on Matching of Numerical Map and UAV Images." 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (2020): 4537-4544, doi: 10.48550/arXiv.2008.01347
8. Водолазский Д., Патрушева А., Блог Яндекс Практикума, Как свёрточные нейросети имитируют работу мозга [Электронный ресурс] .- <https://practicum.yandex.ru/blog/svertochnye-neyronnye-seti/> (дата обращения 24.06.2023);

9. Ding, Lirong, Ji Zhou, Lingxuan Meng and Zhiyong Long. "A Practical Cross-View Image Matching Method between UAV and Satellite for UAV-Based Geo-Localization." Remote. Sens. 13 (2020): 47, doi: 10.3390/rs13010047
10. M. Mishra, Medium, Convolutional Neural Networks explained, 26.09.2020 [Электронный ресурс] .- <https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939> (дата обращения 24.06.2023)
11. Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted Windows," in Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV), Oct. 2021, pp. 11_17.
12. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł. & Polosukhin, I. (2017). Attention is all you need. Advances in Neural Information Processing Systems (p./pp. 5998--6008)
13. Вики-конспекты ИТМО, Механизм внимания [Электронный ресурс] .- https://neerc.ifmo.ru/wiki/index.php?title=Механизм_внимания (дата обращения 16.06.2023)
14. Вики-конспекты ИТМО, Рекуррентные нейронные сети [Электронный ресурс] .- https://neerc.ifmo.ru/wiki/index.php?title=Рекуррентные_нейронные_сети (дата обращения 16.06.2023)
15. IBM, What are recurrent neural networks [Электронный ресурс] .- <https://www.ibm.com/topics/recurrent-neural-networks> (дата обращения 16.06.2023)
16. Alammar, J (2018). The Illustrated Transformer [Blog post]. Retrieved from <https://jalammar.github.io/illustrated-transformer/> (дата обращения 17.06.2023)
17. A Semantic Guidance and Transformer-Based Matching Method for UAVs and Satellite Images for UAV Geo-Localization JIEDONG ZHUANG, <https://doi.org/10.48550/arXiv.2201.09206>

Приложение

Ниже приведён программный код на Python.

Файл Classifier.py

```
from torch import nn

class Classifier(nn.Module):
    def __init__(self, num_classes, is_train, input_size=768, hidden_size=512):
        super(Classifier, self).__init__()

        self.fc1 = nn.Linear(input_size, hidden_size)
        self.batch_norm = nn.BatchNorm1d(hidden_size)
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(p=0.5)
        self.classifier = nn.Linear(hidden_size, num_classes)
        self.softmax = nn.Softmax(dim=0)
        self.is_train = is_train

    def forward(self, x):
        x = self.fc1(x)
        if self.is_train:
            x = x.view(x.size(0), -1)
            try:
                x = self.batch_norm(x)
            except ValueError:
                pass
            x = self.relu(x)
            x = self.dropout(x)
            x = self.classifier(x)
            x = self.softmax(x)
        return x
```

Файл DualBranchModel.py

```
import torch
from torch import nn
from Classifier import Classifier

class DualBranchModel(nn.Module):
    def __init__(self, backbone, device='cpu', is_train=True, num_classes=701):
        super(DualBranchModel, self).__init__()
        self.device = device
        self.backbone_ = backbone
        self.classif = Classifier(num_classes=num_classes, is_train=is_train)

    def forward(self, x1, x2):
        def SGM(f_tens: torch.Tensor, device):
            """
            Разделение исходного тензора на два - foreground and background

            Input:
            tens: torch.Tensor

            Output:
            Background
            Foreground
            """

            backgr = torch.zeros(len(f_tens), 768, device=device)
            foregr = torch.zeros(len(f_tens), 768, device=device)

            for i in range(len(f_tens)):
                tens = f_tens[i]
                M_s = torch.sum(tens, dim=1)
                M_border = [min(M_s), max(M_s)]
                M_norm = [(M_s[i] - M_border[0]) / (M_border[1] - M_border[0]) for i in range(len(M_s))]
```

```

        sorted_indices = torch.argsort(torch.tensor(M_norm))
        sorted_tens = tens[sorted_indices, :]

        M_norm = sorted(M_norm)

        diff = []
        for j in range(len(M_norm) - 1):
            if M_norm[j] == 0:
                diff.append(0)
            else:
                diff_i = (M_norm[j + 1] - M_norm[j])
                diff.append(diff_i)

        backgr_ = sorted_tens[torch.argmax(torch.tensor(diff)):]
        backgr_ = torch.mean(backgr_, dim=0)
        backgr[i] = backgr_

        foregr_ = sorted_tens[:torch.argmax(torch.tensor(diff))]
        foregr_ = torch.mean(foregr_, dim=0)
        foregr[i] = foregr_

    return backgr, foregr

x1 = self.backbone_(x1)
x2 = self.backbone_(x2)

x1 = x1.reshape(x1.size(0), x1.size(1) * x1.size(2), x1.size(3))
x2 = x2.reshape(x2.size(0), x2.size(1) * x2.size(2), x2.size(3))

x1_backgr, x1_foregr = SGM(x1, self.device)
x2_backgr, x2_foregr = SGM(x2, self.device)

x1_backgr = self.classif(x1_backgr)
x1_foregr = self.classif(x1_foregr)
x2_backgr = self.classif(x2_backgr)
x2_foregr = self.classif(x2_foregr)

return x1_backgr, x1_foregr, x2_backgr, x2_foregr

```

Файл UAV_project_SWIN_train.py

```

import os
import torch
from torch import nn
import torch.utils.data as data
from PIL import Image
import torchvision.transforms as transforms
from torchvision.models import swin_t
import torch.optim as optim
from torch.cuda import amp
import itertools
import numpy as np
from DualBranchModel import DualBranchModel
import matplotlib.pyplot as plt

epochs = 10
dir = r'E:\Datasets\University-Release'
batch_size = 6

# dataset
def create_vectors(length):
    vectors = []
    for i in range(length):
        vector = np.zeros(length)
        vector[i] = 1
        vectors.append(vector)
    return vectors

```

```

class BuildingDataset(data.Dataset):
    def __init__(self, root_dir, transform=None):
        self.root_dir = root_dir
        self.transform = transform
        self.buildings = sorted(os.listdir(os.path.join(self.root_dir, 'train', 'drone')))

    def __len__(self):
        return len(self.buildings)

    def __getitem__(self, index):
        building_name = self.buildings[index]
        drone_dir = os.path.join(self.root_dir, 'train', 'drone', building_name)
        satellite_dir = os.path.join(self.root_dir, 'train', 'satellite', building_name)

        satellite_path = os.path.join(satellite_dir, os.listdir(satellite_dir)[0])
        satellite_img = Image.open(satellite_path).convert('RGB')

        def drone_imgs():
            for drone_img_name in os.listdir(drone_dir):
                drone_img_path = os.path.join(drone_dir, drone_img_name)
                drone_img = Image.open(drone_img_path).convert('RGB')
                yield drone_img

        if self.transform:
            satellite_img = self.transform(satellite_img)
            dataset = [(satellite_img, self.transform(drone_img)) for drone_img in
drone_imgs()]

        return dataset

transform = transforms.Compose([
    transforms.Resize((256, 256)),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

dataset = BuildingDataset(root_dir=dir, transform=transform)

result_list = list(itertools.chain.from_iterable(dataset))
vectors = create_vectors(701)
result_vectors = [torch.tensor(x) for x in vectors for _ in range(54)]

result_list = [x + (y,) for x, y in zip(result_list, result_vectors)]

dataloader = data.DataLoader(result_list, batch_size=batch_size, shuffle=True)

# NN
device = 'cuda' if torch.cuda.is_available() else 'cpu'
# device = 'cpu'
print(device, '\n')

swin_model = swin_t(weights='IMAGENET1K_V1')
backbone = nn.Sequential(*list(swin_model.children())[:-5])
model = DualBranchModel(backbone, device=device).to(device)

# X = torch.randn(8, 3, 256, 256)
# Y = torch.randn(8, 3, 256, 256)
# target = torch.zeros(8, 701)
# target[0, 42] = 1
# dataset = data.TensorDataset(X, Y, target)
# dataloader = data.DataLoader(dataset, batch_size=4, shuffle=True)

lossFunc = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

```

```

scaler = amp.GradScaler()

train_losses = []
sat_backgr_losses = []
sat_foregr_losses = []
uav_backgr_losses = []
uav_foregr_losses = []
for epoch in range(epochs):
    trainLoss = 0
    sat_backgr_l = 0
    sat_foregr_l = 0
    uav_backgr_l = 0
    uav_foregr_l = 0
    for i, (sat_img, uav_img, target) in enumerate(dataloader):
        sat_img = sat_img.to(device)
        uav_img = uav_img.to(device)
        target = target.to(device)
        sat_backgr_out, sat_foregr_out, uav_backgr_out, uav_foregr_out = model(sat_img,
uav_img)

        optimizer.zero_grad()
        with amp.autocast():
            sat_backgr_loss = lossFunc(sat_backgr_out, target)
            sat_foregr_loss = lossFunc(sat_foregr_out, target)
            uav_backgr_loss = lossFunc(uav_backgr_out, target)
            uav_foregr_loss = lossFunc(uav_foregr_out, target)
            loss = sat_backgr_loss + sat_foregr_loss + uav_backgr_loss + uav_foregr_loss

        scaler.scale(loss).backward()
        scaler.step(optimizer)
        scaler.update()

        batch_num = i + 1
        if (batch_num) == 1 or batch_num % 100 == 0:
            print("Epoch {}/{}", Batch {}/{}".format(epoch + 1, epochs, batch_num,
len(dataloader)))

            trainLoss += loss.item()
            sat_backgr_l += sat_backgr_loss.item()
            sat_foregr_l += sat_foregr_loss.item()
            uav_backgr_l += uav_backgr_loss.item()
            uav_foregr_l += uav_foregr_loss.item()

    trainLoss = trainLoss / len(dataloader)
    train_losses.append(trainLoss)

    sat_backgr_l /= len(dataloader)
    sat_backgr_losses.append(sat_backgr_l)

    sat_foregr_l /= len(dataloader)
    sat_foregr_losses.append(sat_foregr_l)

    uav_backgr_l /= len(dataloader)
    uav_backgr_losses.append(uav_backgr_l)

    uav_foregr_l /= len(dataloader)
    uav_foregr_losses.append(uav_foregr_l)

    if min(train_losses) == train_losses[-1]:
        torch.save(model.state_dict(), 'model.pt')
        print('weight saved')
    print('epoch: [{}]/{}; train loss: {:.3f}'.format(epoch + 1, epochs, trainLoss))

def plot_help1(epochs, y_train_m, title):
    plt.rc('lines', linewidth=2.5)

```

```

fig, ax = plt.subplots()
ax.set_title(title)
# Using set_dashes() and set_capstyle() to modify dashing of an existing line.
line1, = ax.plot(np.linspace(1, epochs, num=epochs), y_train_m, label='train')
line1.set_dashes([10, 2, 2, 2]) # 10pt line, 2pt break, 2pt line, 2pt break.
line1.set_dash_capstyle('round')
ax.scatter(np.linspace(1, epochs, num=epochs), y_train_m)

# Using plot(..., dashes=..., gapcolor=...) to set the dashing and
# alternating color when creating a line.

ax.legend(handlelength=4)
plt.grid()
plt.savefig(f'{title}.png')
return fig, ax

def plot_help2(epochs, l1, l2, l3, l4, title):
    plt.rc('lines', linewidth=2.5)
    fig, ax = plt.subplots()
    ax.set_title(title)
    # Using set_dashes() and set_capstyle() to modify dashing of an existing line.
    line1, = ax.plot(np.linspace(1, epochs, num=epochs), l1, label='sat_backgr_loss')
    line1.set_dashes([10, 2, 2, 2]) # 10pt line, 2pt break, 2pt line, 2pt break.
    line1.set_dash_capstyle('round')
    ax.scatter(np.linspace(1, epochs, num=epochs), l1)

    # Using plot(..., dashes=...) to set the dashing when creating a line.
    line2, = ax.plot(np.linspace(1, epochs, num=epochs), l2, label='sat_foregr_loss')
    ax.scatter(np.linspace(1, epochs, num=epochs), l2)

    line3, = ax.plot(np.linspace(1, epochs, num=epochs), l3, label='uav_backgr_loss')
    line3.set_dashes([10, 2, 2, 2]) # 10pt line, 2pt break, 2pt line, 2pt break.
    line3.set_dash_capstyle('round')
    ax.scatter(np.linspace(1, epochs, num=epochs), l3)

    line4, = ax.plot(np.linspace(1, epochs, num=epochs), l4, label='uav_foregr_loss')
    ax.scatter(np.linspace(1, epochs, num=epochs), l4)

    # Using plot(..., dashes=..., gapcolor=...) to set the dashing and
    # alternating color when creating a line.

    ax.legend(handlelength=4)
    plt.grid()
    plt.savefig(f'{title}.png')
    return fig, ax

title = 'Loss over Epochs'
fig, ax = plot_help1(epochs, train_losses, title)

title = 'Loss components over Epochs'
fig1, ax1 = plot_help2(epochs, sat_backgr_losses, sat_foregr_losses, uav_backgr_losses,
uav_foregr_losses, title)

```

Файл UAV_project_SWIN_test.py

```

import os
import torch
from torch import nn
from torchvision.models import swin_t
from DualBranchModel import DualBranchModel
import torch.utils.data as data
from torch.utils.data import DataLoader
import torchvision.transforms as transforms
from PIL import Image
import itertools

```

```

device = 'cuda' if torch.cuda.is_available() else 'cpu'
# device = 'cpu'
print(device, '\n')

swin_model = swin_t()
backbone = nn.Sequential(*list(swin_model.children())[:-5])
model = DualBranchModel(backbone, device=device, is_train=False).to(device)
model.load_state_dict(torch.load('model.pt'))
model.eval()

transform = transforms.Compose([
    transforms.Resize((256, 256)),
    transforms.ToTensor(),
    # transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

batch_size = 1

# img1 = Image.open(r'E:\Datasets\University-Release\test\gallery_drone\0001\image-06.jpeg')
# img1 = transform(img1).to(device)
# img2 = Image.open(r'E:\Datasets\University-Release\test\gallery_satellite\0004\0004.jpg')
# img2 = transform(img2).to(device)
# img3 = Image.open(r'E:\Datasets\University-Release\test\gallery_satellite\0008\0008.jpg')
# img3 = transform(img3).to(device)
# img4 = Image.open(r'E:\Datasets\University-Release\test\gallery_satellite\0001\0001.jpg')
# img4 = transform(img4).to(device)
# l = [(img1, img2, 0), (img1, img3, 0),
#      (img1, img4, 1)]
# dataloader = DataLoader(l, batch_size=batch_size)

# input_tensor1 = torch.randn(3, 256, 256).to(device)
# input_tensor2 = torch.randn(3, 256, 256).to(device)
# input_tensor3 = torch.randn(3, 256, 256).to(device)
# input_tensor4 = torch.randn(3, 256, 256).to(device)
# l = [(input_tensor1, input_tensor2, 0), (input_tensor3, input_tensor4, 1),
#      (input_tensor1, input_tensor4, 0), (input_tensor3, input_tensor2, 0)]
# dataloader = DataLoader(l, batch_size=batch_size)

class BuildingDataset(data.Dataset):
    def __init__(self, root_dir, transform=None, mode='both'):
        self.root_dir = root_dir
        self.transform = transform
        self.buildings = sorted(os.listdir(os.path.join(self.root_dir, 'test', 'gallery_drone')))
        self.mode = mode

    def __len__(self):
        return len(self.buildings)

    def __getitem__(self, index):
        building_name = self.buildings[index]
        drone_dir = os.path.join(self.root_dir, 'test', 'gallery_drone', building_name)
        satellite_dir = os.path.join(self.root_dir, 'test', 'gallery_satellite', building_name)

        satellite_path = os.path.join(satellite_dir, os.listdir(satellite_dir)[0])
        satellite_img = Image.open(satellite_path).convert('RGB')

        if self.mode == 'drone':

            def drone_imgs():
                for drone_img_name in os.listdir(drone_dir):
                    drone_img_path = os.path.join(drone_dir, drone_img_name)
                    drone_img = Image.open(drone_img_path).convert('RGB')
                    yield drone_img

            if self.transform:
                dataset = [self.transform(drone_img) for drone_img in drone_imgs()]

            return dataset

        if self.mode == 'satellite':
            if self.transform:
                satellite_img = self.transform(satellite_img)

            return satellite_img

```

```

else:
    def drone_imgs():
        for drone_img_name in os.listdir(drone_dir):
            drone_img_path = os.path.join(drone_dir, drone_img_name)
            drone_img = Image.open(drone_img_path).convert('RGB')
            yield drone_img

    if self.transform:
        satellite_img = self.transform(satellite_img)
        dataset = [(satellite_img, self.transform(drone_img)) for drone_img in drone_imgs()]

    return dataset

transform = transforms.Compose([
    transforms.Resize((256, 256)),
    transforms.ToTensor(),
    # transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

dir = r'E:\Datasets\University-Release'
dataset_satellite = BuildingDataset(root_dir=dir, transform=transform, mode='satellite')
dataset_drone = BuildingDataset(root_dir=dir, transform=transform, mode='drone')

dataset_satellite_loader = [dataset_satellite[i] for i in range(len(dataset_satellite))]

len_dataset = 20
result_dataset = []
# создаем список для теста
for position in range(len_dataset):
    vector = [0] * len_dataset
    vector[position] = 1
    for j in range(54):
        dataset_drone_loader = [dataset_drone[position][j]] * len(dataset_drone)
        result = [(x, y, z) for x, y, z in zip(dataset_satellite_loader[:len_dataset],
        dataset_drone_loader, vector)]
        result_dataset.append(result)

print('Старт')
R1 = 0
R5 = 0
R10 = 0
count = 0
for j in range(len(result_dataset)):
    dataloader = DataLoader(result_dataset[j], batch_size=batch_size)

    distances = []
    for i, (sat_img, uav_img, target) in enumerate(dataloader):
        sat_img = sat_img.to(device)
        uav_img = uav_img.to(device)
        target = target.to(device)
        sat_backgr_out, sat_foregr_out, uav_backgr_out, uav_foregr_out = model(sat_img, uav_img)

        sat = torch.cat((sat_backgr_out, sat_foregr_out), dim=0)
        uav = torch.cat((uav_backgr_out, uav_foregr_out), dim=0)

        distance = torch.linalg.vector_norm(uav - sat, ord=2)
        distances.append((distance.item(), i))
        if target == 1:
            true_index = i

    sorted_distances = sorted(distances, key=lambda x: x[0])
    for distance, idx in itertools.islice(sorted_distances, 5):
        print(f"Distance: {distance}, i: {idx}")
        print()

    count += 1
    if sorted_distances[0][1] == true_index:
        R1 += 1
    for d in range(5):
        if sorted_distances[d][1] == true_index:
            R5 += 1
            break
    for k in range(10):
        if sorted_distances[k][1] == true_index:

```

```

        R10 += 1
        break

print(R1 / count)
print(R5 / count)
print(R10 / count)

```

Файл UAV_project_resnet_train.py

```

import os
import torch
from torch import nn
import torch.utils.data as data
import torchvision.transforms as transforms
import torch.optim as optim
from torch.cuda import amp
from PIL import Image
import itertools
import numpy as np
from torchvision.models import resnet50
from Classifier import Classifier
import matplotlib.pyplot as plt

epochs = 10
dir = r'E:\Datasets\University-Release'
batch_size = 10

# dataset
def create_vectors(length):
    vectors = []
    for i in range(length):
        vector = np.zeros(length)
        vector[i] = 1
        vectors.append(vector)
    return vectors

class BuildingDataset(data.Dataset):
    def __init__(self, root_dir, transform=None):
        self.root_dir = root_dir
        self.transform = transform
        self.buildings = sorted(os.listdir(os.path.join(self.root_dir, 'train', 'drone')))

    def __len__(self):
        return len(self.buildings)

    def __getitem__(self, index):
        building_name = self.buildings[index]
        drone_dir = os.path.join(self.root_dir, 'train', 'drone', building_name)
        satellite_dir = os.path.join(self.root_dir, 'train', 'satellite', building_name)

        satellite_path = os.path.join(satellite_dir, os.listdir(satellite_dir)[0])
        satellite_img = Image.open(satellite_path).convert('RGB')

        def drone_imgs():
            for drone_img_name in os.listdir(drone_dir):
                drone_img_path = os.path.join(drone_dir, drone_img_name)
                drone_img = Image.open(drone_img_path).convert('RGB')
                yield drone_img

        if self.transform:
            satellite_img = self.transform(satellite_img)
            dataset = [(satellite_img, self.transform(drone_img)) for drone_img in drone_imgs()]

        return dataset

transform = transforms.Compose([
    transforms.Resize((256, 256)),

```



```

        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
    ])

dataset = BuildingDataset(root_dir=dir, transform=transform)

result_list = list(itertools.chain.from_iterable(dataset))
vectors = create_vectors(701)
result_vectors = [torch.tensor(x) for x in vectors for _ in range(54)]

result_list = [x + (y,) for x, y in zip(result_list, result_vectors)]

dataloader = data.DataLoader(result_list, batch_size=batch_size, shuffle=True)

class DualBranchModel(nn.Module):
    def __init__(self, backbone, device='cpu', is_train=True, input_size=1000, num_classes=701):
        super(DualBranchModel, self).__init__()
        self.device = device
        self.backbone_ = backbone
        self.classif = Classifier(num_classes=num_classes, input_size=input_size, is_train=is_train)

    def forward(self, x1, x2):
        x1 = self.backbone_(x1)
        x2 = self.backbone_(x2)

        x1 = self.classif(x1)
        x2 = self.classif(x2)
        return x1, x2

device = 'cuda' if torch.cuda.is_available() else 'cpu'
# device = 'cpu'
print(device, '\n')

backbone = resnet50(weights='IMAGENET1K_V2')
model = DualBranchModel(backbone, device=device).to(device)

# X = torch.randn(8, 3, 256, 256)
# Y = torch.randn(8, 3, 256, 256)
# target = torch.zeros(8, 701)
# target[0, 42] = 1
# dataset = data.TensorDataset(X, Y, target)
# dataloader = data.DataLoader(dataset, batch_size=4, shuffle=True)

lossFunc = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
scaler = amp.GradScaler()

train_losses = []
sat_losses = []
uav_losses = []
for epoch in range(epochs):
    trainLoss = 0
    sat_l = 0
    uav_l = 0
    for i, (sat_img, uav_img, target) in enumerate(dataloader):
        sat_img = sat_img.to(device)
        uav_img = uav_img.to(device)
        target = target.to(device)
        sat, uav = model(sat_img, uav_img)

        optimizer.zero_grad()
        with amp.autocast():
            sat_loss = lossFunc(sat, target)
            uav_loss = lossFunc(uav, target)
            loss = sat_loss + uav_loss

        scaler.scale(loss).backward()
        scaler.step(optimizer)
        scaler.update()

        batch_num = i + 1
        if (batch_num) == 1 or batch_num % 100 == 0:
            print("Epoch {}/ {}, Batch {}/ {}".format(epoch + 1, epochs, batch_num, len(dataloader)))

```

```

        trainLoss += loss.item()
        sat_l += sat_loss.item()
        uav_l += uav_loss.item()

    trainLoss = trainLoss / len(dataloader)
    train_losses.append(trainLoss)

    sat_l /= len(dataloader)
    sat_losses.append(sat_l)

    uav_l /= len(dataloader)
    uav_losses.append(uav_l)

    if min(train_losses) == train_losses[-1]:
        torch.save(model.state_dict(), 'model_res.pt')
        print('weight saved')
    print('epoch: [{}/{}]; train loss: {:.3f}'.format(epoch + 1, epochs, trainLoss))

def plot_help1(epochs, y_train_m, title):
    plt.rc('lines', linewidth=2.5)
    fig, ax = plt.subplots()
    ax.set_title(title)
    # Using set_dashes() and set_capstyle() to modify dashing of an existing line.
    line1, = ax.plot(np.linspace(1, epochs, num=epochs), y_train_m, label='train')
    line1.set_dashes([10, 2, 2, 2]) # 10pt line, 2pt break, 2pt line, 2pt break.
    line1.set_dash_capstyle('round')
    ax.scatter(np.linspace(1, epochs, num=epochs), y_train_m)

    # Using plot(..., dashes=..., gapcolor=...) to set the dashing and
    # alternating color when creating a line.

    ax.legend(handlelength=4)
    plt.grid()
    plt.savefig(f'{title}_res.png')
    return fig, ax

def plot_help2(epochs, l1, l2, title):
    plt.rc('lines', linewidth=2.5)
    fig, ax = plt.subplots()
    ax.set_title(title)
    # Using set_dashes() and set_capstyle() to modify dashing of an existing line.
    line1, = ax.plot(np.linspace(1, epochs, num=epochs), l1, label='sat_loss')
    line1.set_dashes([10, 2, 2, 2]) # 10pt line, 2pt break, 2pt line, 2pt break.
    line1.set_dash_capstyle('round')
    ax.scatter(np.linspace(1, epochs, num=epochs), l1)

    # Using plot(..., dashes=...) to set the dashing when creating a line.
    line2, = ax.plot(np.linspace(1, epochs, num=epochs), l2, label='uav_loss')
    ax.scatter(np.linspace(1, epochs, num=epochs), l2)

    # Using plot(..., dashes=..., gapcolor=...) to set the dashing and
    # alternating color when creating a line.

    ax.legend(handlelength=4)
    plt.grid()
    plt.savefig(f'{title}_res.png')
    return fig, ax

title = 'Loss over Epochs'
fig, ax = plot_help1(epochs, train_losses, title)

title = 'Loss components over Epochs'
fig1, ax1 = plot_help2(epochs, sat_losses, uav_losses, title)

```

Файл UAV_project_resnet_test.py

```

import os
import torch
from torch import nn
import torch.utils.data as data

```

```

from torch.utils.data import DataLoader
import torchvision.transforms as transforms
from PIL import Image
import itertools
from Classifier import Classifier
from torchvision.models import resnet50

class DualBranchModel(nn.Module):
    def __init__(self, backbone, is_train=True, input_size=1000, num_classes=701):
        super(DualBranchModel, self).__init__()
        self.device = device
        self.backbone_ = backbone
        self.classif = Classifier(num_classes=num_classes, input_size=input_size, is_train=is_train)

    def forward(self, x1, x2):
        x1 = self.backbone_(x1)
        x2 = self.backbone_(x2)

        x1 = self.classif(x1)
        x2 = self.classif(x2)
        return x1, x2

device = 'cuda' if torch.cuda.is_available() else 'cpu'
# device = 'cpu'
print(device, '\n')

backbone = resnet50().to(device)
model = DualBranchModel(backbone, is_train=False).to(device)
model.load_state_dict(torch.load('model_res.pt'))
model.eval()

# transform = transforms.Compose([
#     transforms.Resize((256, 256)),
#     transforms.ToTensor(),
#     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
# ])

batch_size = 1

# img1 = Image.open(r'E:\Datasets\University-Release\test\gallery_drone\0001\image-06.jpeg')
# img1 = transform(img1).to(device)
# img2 = Image.open(r'E:\Datasets\University-Release\test\gallery_satellite\0004\0004.jpg')
# img2 = transform(img2).to(device)
# img3 = Image.open(r'E:\Datasets\University-Release\test\gallery_satellite\0005\0005.jpg')
# img3 = transform(img3).to(device)
# img4 = Image.open(r'E:\Datasets\University-Release\test\gallery_satellite\0001\0001.jpg')
# img4 = transform(img4).to(device)
# l = [(img1, img2, 0), (img1, img3, 0),
#       (img1, img4, 1)]
# dataloader = DataLoader(l, batch_size=batch_size)

# input_tensor1 = torch.randn(3, 256, 256).to(device)
# input_tensor2 = torch.randn(3, 256, 256).to(device)
# input_tensor3 = torch.randn(3, 256, 256).to(device)
# input_tensor4 = torch.randn(3, 256, 256).to(device)
# l = [(input_tensor1, input_tensor2, 0), (input_tensor3, input_tensor4, 1),
#       (input_tensor1, input_tensor4, 0), (input_tensor3, input_tensor2, 0)]
# dataloader = DataLoader(l, batch_size=batch_size)

class BuildingDataset(data.Dataset):
    def __init__(self, root_dir, transform=None, mode='both'):
        self.root_dir = root_dir
        self.transform = transform
        self.buildings = sorted(os.listdir(os.path.join(self.root_dir, 'test', 'gallery_drone')))
        self.mode = mode

    def __len__(self):
        return len(self.buildings)

    def __getitem__(self, index):
        building_name = self.buildings[index]
        drone_dir = os.path.join(self.root_dir, 'test', 'gallery_drone', building_name)

```

```

        satellite_dir = os.path.join(self.root_dir, 'test', 'gallery_satellite', building_name)

        satellite_path = os.path.join(satellite_dir, os.listdir(satellite_dir)[0])
        satellite_img = Image.open(satellite_path).convert('RGB')

        if self.mode == 'drone':

            def drone_imgs():
                for drone_img_name in os.listdir(drone_dir):
                    drone_img_path = os.path.join(drone_dir, drone_img_name)
                    drone_img = Image.open(drone_img_path).convert('RGB')
                    yield drone_img

            if self.transform:
                dataset = [self.transform(drone_img) for drone_img in drone_imgs()]

            return dataset

        if self.mode == 'satellite':
            if self.transform:
                satellite_img = self.transform(satellite_img)

            return satellite_img

        else:
            def drone_imgs():
                for drone_img_name in os.listdir(drone_dir):
                    drone_img_path = os.path.join(drone_dir, drone_img_name)
                    drone_img = Image.open(drone_img_path).convert('RGB')
                    yield drone_img

            if self.transform:
                satellite_img = self.transform(satellite_img)
                dataset = [(satellite_img, self.transform(drone_img)) for drone_img in drone_imgs()]

            return dataset

transform = transforms.Compose([
    transforms.Resize((256, 256)),
    transforms.ToTensor(),
    # transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

dir = r'E:\Datasets\University-Release'
dataset_satellite = BuildingDataset(root_dir=dir, transform=transform, mode='satellite')
dataset_drone = BuildingDataset(root_dir=dir, transform=transform, mode='drone')

dataset_satellite_loader = [dataset_satellite[i] for i in range(len(dataset_satellite))]

len_dataset = 200
result_dataset = []
# создаем список для теста
for position in range(len_dataset):
    vector = [0] * len_dataset
    vector[position] = 1
    for j in range(54):
        dataset_drone_loader = [dataset_drone[position][j]] * len(dataset_drone)
        result = [(x, y, z) for x, y, z in zip(dataset_satellite_loader[:len_dataset],
        dataset_drone_loader, vector)]
        result_dataset.append(result)

print('Старт')
R1 = 0
R5 = 0
R10 = 0
count = 0
for j in range(len(result_dataset)):
    dataloader = DataLoader(result_dataset[j], batch_size=batch_size)

    distances = []
    for i, (sat_img, uav_img, target) in enumerate(dataloader):
        sat_img = sat_img.to(device)
        uav_img = uav_img.to(device)
        target = target.to(device)
        sat, uav = model(sat_img, uav_img)

```

```

distance = torch.linalg.vector_norm(uav - sat, ord=2)
distances.append((distance.item(), i))
if target == 1:
    true_index = i

sorted_distances = sorted(distances, key=lambda x: x[0])
for distance, idx in itertools.islice(sorted_distances, 5):
    print(f"Distance: {distance}, i: {idx}")
print()

count += 1
if sorted_distances[0][1] == true_index:
    R1 += 1
    print('R1 +')
for d in range(5):
    if sorted_distances[d][1] == true_index:
        R5 += 1
        print('R5 +')
        break
for k in range(10):
    if sorted_distances[k][1] == true_index:
        R10 += 1
        print('R10 +')
        break

print('R1:', R1 / count)
print('R5:', R5 / count)
print('R10:', R10 / count)

```

```

0.8731481481481481
0.9814814814814815
0.9990740740740741

```

Рисунок 30. Значение метрики для 20 объектов для ResNet50

```

R1:0.6936507936507936
R5:0.9132275132275133
R10:0.9698412698412698

```

Рисунок 31. Значение метрики для 35 объектов для ResNet50

```

R1: 0.5814814814814815
R5: 0.8714814814814815
R10: 0.9440740740740741

```

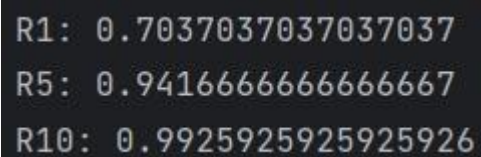
Рисунок 32. Значение метрики для 50 объектов для ResNet50

```

R1:0.49666666666666665
R1:0.7670370370370371
R1:0.8529629629629629

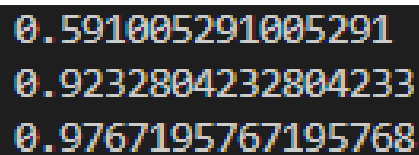
```

Рисунок 33. Значение метрики для 100 объектов для ResNet50. При выводе допущена текстовая опечатка, выводятся R1, R5, R10.



R1: 0.7037037037037037
R5: 0.9416666666666667
R10: 0.9925925925925926

Рисунок 34. Значение метрики для 20 объектов для Swin Transformer



0.591005291005291
0.9232804232804233
0.9767195767195768

Рисунок 35. Значение метрики для 35 объектов для Swin Transformer