МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Институт интеллектуальной робототехники

Кафедра Интеллектуальных систем теплофизики ИИР
Направление подготовки: 15.03.06 Мехатроника и робототехника
Направленность (профиль): Мехатроника и робототехника

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Ло Цзюнь
(Фамилия, Имя, Отчество автора)

Тема работы Разработка алгоритмов навигации БПЛА на основе глубоких нейронных сетей

«К защите допущен(-а)»
и.о. зав. кафедрой ИСТИИР,
д.т.н, доцент

Назаров А.Д. /_____
(ФИО) / (подпись)

«……»……2024г.

Руководитель ВКР
программист

Забоцкий А.Е. /_____
(М.П.) (ФИО) / (подпись)

«……»……2024г.

Новосибирск, 2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Навигация беспилотных летательных аппаратов (БПЛА) с использованием компьютерного зрения - это метод навигации, при котором используется камера, встроенная в БПЛА, и одноплатные компьютеры, такие как OrangePI5[1], для работы в реальном времени с целью определения положения БПЛА на земле. Преимущество этого метода заключается в том, что он сохраняет автономность дрона, и ему не требуются внешние источники информации, такие как GPS. Навигация БПЛА с использованием компьютерного зрения является эффективным способом определения пространственного положения БПЛА, особенно когда ресурсы ограничены, что делает БПЛА более способными и гибкими при навигации в конкретных областях. По сравнению с GPS-навигацией, навигация с использованием компьютерного зрения может эффективно избегать помех и перехвата GPS.

Суть навигации с использованием компьютерного зрения заключается в получении различных параметров движения дрона через гироскоп, получении высоты дрона через датчик давления воздуха и применении нейронной сети через одноплатный компьютер. Это является основой для реализации навигации с использованием компьютерного зрения БПЛА.

Камера вычисляет смещение дрона относительно земли и с помощью нейронной сети соотносит положение дрона на спутниковом снимке, тем самым получая смещение дрона относительно земли. С использованием алгоритма SIFT на дроне получаются характерные точки непрерывных изображений, и можно рассчитать соответствие характерных точек относительного движения между соседними кадрами изображения, что в конечном итоге позволяет реализовать расчет пройденного дроном расстояния.

Барометр также являются важным оборудованием для навигации БПЛА. Считывая показания барометра, а также используя формулу для расчета высоты по давлению и температуре воздуха, можно также получить данные о высоте БПЛА. Однако в реальных условиях точность оценки высоты дрона просто с

помощью барометра не высока. Поэтому мы рассматриваем реализацию алгоритма расчета высоты дрона после реализации алгоритма навигации дрона.

1 ИСПОЛЬЗУЕМЫЕ АППАРАТНЫЕ И ПРОГРАММНЫЕ СРЕДСТВА

MPU-6050[2] — IMU, включающий в себя 3-х осевой гироскоп и 3-х осевой акселерометр. Диапазон измерений акселерометра: ± 2 ± 4 ± 8 ± 16g. Диапазон измерений гироскопа: ± 250 500 1000 2000 ° / s. Интерфейс — I2C (с максимальной рабочей частотой до 400 кГц). По каждой оси и параметру датчик выдаёт 16-битное знаковое значение.

Orange PI 5 — микрокомпьютер, на котором осуществляется разработка ПО. Для подключения датчиков используется интерфейс I2C. Поддерживаемые I2C шины: I2C1, I2C3, I2C5.

Mpu6050-raspberrypi — API модуль Python, позволяющий получать с MPU-6050 преобразованные в физические величины данные с нужным диапазоном.

MPU-9250 — это 9-осевой IMU, который состоит из двух кристаллов, интегрированных в один QFN-пакет. Один кристалл содержит 3-осевой гироскоп и 3-осевой акселерометр. Диапазон измерений акселерометра: ± 2 ± 4 ± 8 ± 16g. Диапазон измерений гироскопа: ± 250 ± 500 ± 1000 ± 2000 ° / s. Диапазон измерений магнитометра: ± 4800 μT. Интерфейс — I2C или SPI. По каждой оси и параметру датчик выдаёт 16-битное знаковое значение.

BMP 280 — это компактный и низкопотребляющий модуль для абсолютного измерения атмосферного давления в мобильных приложениях. Он обладает высокой точностью, линейностью, стабильностью и устойчивостью к электромагнитным помехам, а также предлагает различные варианты интерфейса и режимы фильтрации. BMP 280 основан на проверенной технологии пьезорезистивных датчиков давления от Bosch, которые характеризуются высокой точностью и долговечностью. Диапазон измерений давления: 300…1100 гПа. Диапазон измерений температуры: -40…+85°C. Интерфейс — I2C или SPI. По каждой оси и параметру датчик выдаёт 24-битное знаковое значение.

PixHawk 4.1 — это полетный контроллер для беспилотных летательных аппаратов, основанный на процессоре STM32F765. Он поддерживает разные типы летательных аппаратов и датчиков, а также работает под управлением прошивки ArduPilot. Вот некоторые важные параметры PixHawk 4.1: Процессор: STM32F765, 32-битный, 216 МГц, 2 МБ флеш-памяти, 512 КБ ОЗУИнтерфейс: UART, I2C, SPI, CAN, PWM, ADC Датчики: IMU, барометр, компас, GPS Размеры: 44 x 84 x 12 мм Вес: 15.8 г

BNO 08x[3] — это 9-осевой IMU, который обрабатывает данные с датчиков и выдаёт точные результаты ориентации, движения и жестов. Он основан на процессоре Arm Cortex M0+ с прошивкой SH-2 от CEVA Hillcrest Labs. Вот некоторые важные параметры BNO 08x: Датчики: 3-осевой акселерометр, 3-осевой гироскоп, 3-осевой магнитометр Интерфейс: I2C, UART, SPI Диапазон измерений акселерометра: ± 16g Диапазон измерений гироскопа: ± 2000 ° / s Диапазон измерений магнитометра: ± 2500 uT Размеры: 5.2 x 3.8 x 1.1 мм Вес: 0.83 г.

2 ЗАДАЧИ

2.1 Сбор и аннотирование набора данных

Для сбора и аннотирования наборов данных мы максимально используем популярные на данный момент методы.

Здесь мы используем дроны для захвата изображений на месте, чтобы изображения, вводимые в сеть для обучения, могли максимально соответствовать реальным потребностям и давать лучшие результаты в процессе применения алгоритма. Для сбора наборов данных спутниковых изображений мы рассматриваем возможность загрузки с сервера спутниковых изображений Google Map для получения необходимых нам спутниковых изображений.

Для аннотирования набора данных мы рассматриваем возможность использования популярного в настоящее время инструмента аннотирования компьютерного зрения CVAT (Computer Vision Annotation Tool) для аннотирования отредактированных пар спутниковых изображений и изображений, снятых с дрона.

Выбираем следующие нейронные сети: ResNet (CLIP),LoFTR (Coarse LoFTR), PIDNet , YOLOv5. Эти сети были выбраны потому, что они достаточно быстры, чтобы работать в режиме реального времени на одноплатных компьютерах, таких как OrangePI5, и, таким образом, отвечают потребностям проекта.

В настоящее время мы рассматриваем возможность использования этих моделей, но будут ли они использоваться, будет зависеть от их точности прогнозирования для пар изображений во время обучения.

2.2.1 ResNet (CLIP)

В области кросс-модального обучения без присмотра CLIP [4] является новаторской работой. Используемый метод - обучение без учителя.Преимущество этого метода обучения состоит в том, что он позволяет избежать ограничений классификационных меток при обучении с учителем и имеет характеристики zero-shot learning, что значительно улучшает практическую эффективность модели. В то же время CLIP также имеет

возможность zero-shot learning, что значительно улучшает возможности CLIP в обобщенных приложениях.

Zero-shot learning выстрелом относится к обучению на других наборах данных и их прямой передаче для классификации.Хотя аналогичные идеи были предложены еще в 2017 году, результаты не идеальны, поскольку размер данных слишком мал. OPENAI отдельно собрала большой набор данных, содержащий 400 миллионов образцов, для достижения хороших результатов.

Как показано на рисунке 1, архитектура этой сети основана на двух разных типах кодировщиков.

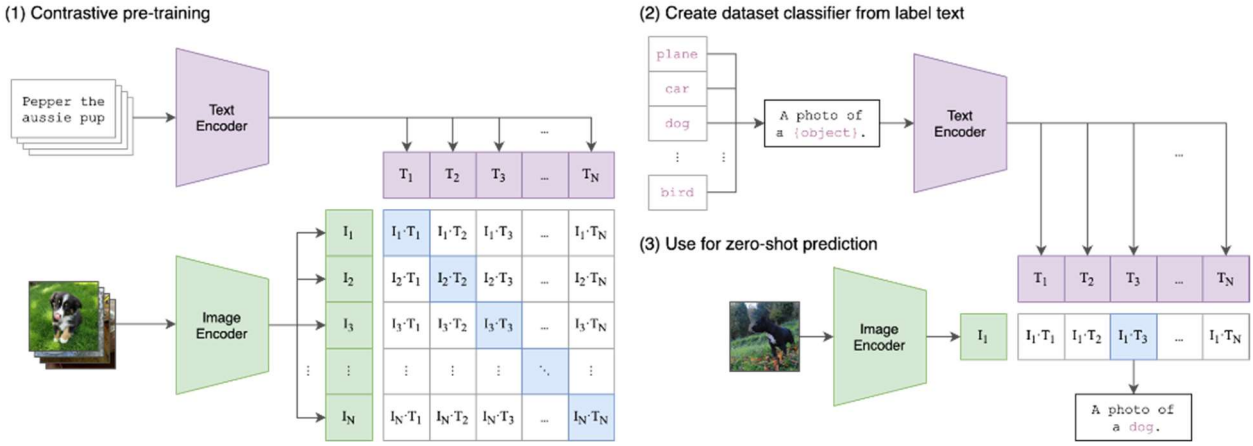

Рисунок 1 - Структура сети CLIP

Мы обнаружили, что CLIP соответствует производительности исходного ResNet50. на ImageNet «с нуля» без использования каких-либо оригинальных примеров с маркировкой 1,28M, что позволяет преодолеть несколько серьезных проблем в области компьютерного зрения.

Но нам явно не нужно, чтобы наша нейронная сеть имела семантическое понимание, поэтому мы заменяем один из кодировщиков, как показано на рисунке 2. Эта замена станет основой наших инноваций в этой сети.

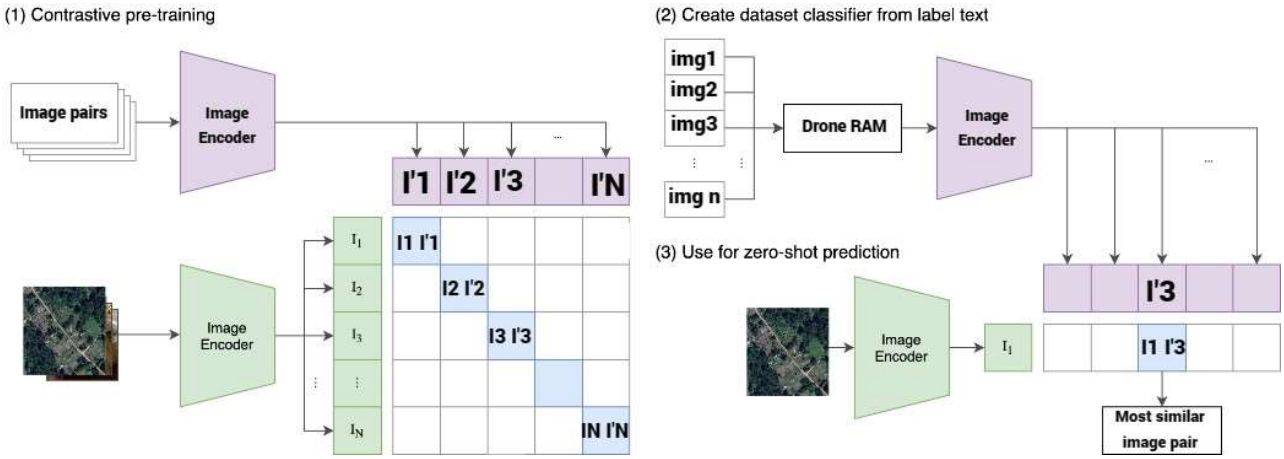

Рисунок 2 - Структура сиамской сети

Когда мы используем сеть CLIP, мы заменим кодировщик текста в сети CLIP кодировщиком изображений. Причина такой настройки в том, что после ввода изображения с камеры дрона в сеть CLIP нейросеть может найти наиболее похожее изображение в растровой карте, предварительно загруженной в память устройства.

Удалив текстовый кодер, мы вполне можем сохранить способность к обобщению модели CLIP (ее способность к обобщению намного превосходит широко используемый Resnet101) и лучше адаптироваться к нашему процессу разработки алгоритма навигации БПЛА.

Наша конечная цель — дать возможность нашей сети найти карту Heat_map, созданную с помощью более крупной спутниковой карты, которая наиболее похожа на фотографию, сделанную дроном, как показано на рисунке 3.

Рисунок 3 - Heat_map

Как показано на рисунке, это именно то, что мы ожидаем увидеть в сети CLIP. Сравнивая изображение, полученное на спутниковой карте, с изображением, хранящимся в памяти дрона, мы можем получить область изображения, которая с наибольшей вероятностью будет соответствовать целевой области на спутниковом изображении, что позволит дрону ориентироваться.

2.2.2 Coarse LoFTR

Этот проект разработал глубокую модель, которая способна сопоставлять локальные особенности на двух изображениях с высокой точностью и производительностью. Модель оптимизирована для использования на встроенных устройствах, таких как NVidia Jetson Nano 2 ГБ, и способна обрабатывать до 5 кадров в секунду. Основой алгоритма является реализация метода "LoFTR: сопоставление локальных признаков без детекторов с трансформаторами" [5]. Этот подход позволяет эффективно сопоставлять особенности изображений без использования детекторов, что делает модель компактной и высокоэффективной. Такая модель может быть полезна для различных приложений, включая системы навигации, виртуальную и дополненную реальность, а также робототехнику.

Его сетевая архитектура показана на рисунке 4.

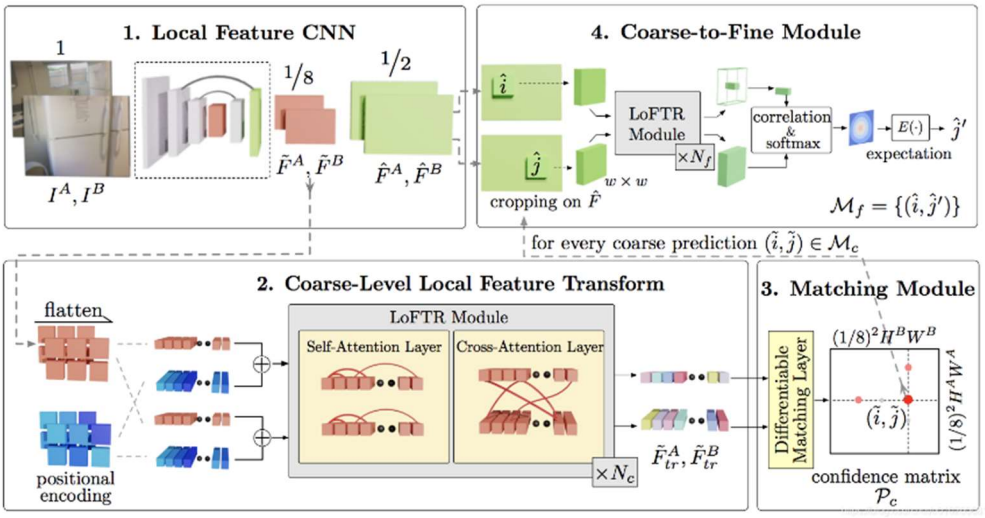

Рисунок 4 – Сетевая архитектура Coarse-Lofter

Но в модели уменьшено количество слоев ResNet и грубого преобразователя, поэтому потребление памяти значительно ниже, а производительность выше. Требуемый уровень точности был достигнут за счет применения методики дистилляции знаний и обучения на наборе данных BlendedMVS.

2.2.3 YOLOv5

YOLOv5[6] представляет собой алгоритм обнаружения объектов, который работает в один этап. Он был улучшен на основе YOLOv4, внедряя некоторые новые идеи, которые существенно увеличили его скорость и точность. Для улучшения обучения модели применяется метод Mosaic, который использует четыре изображения и комбинирует их с помощью случайного масштабирования, кадрирования и размещения. В YOLOv5 также используется метод улучшения данных CutMix, который, в отличие от Mosaic, объединяет только два изображения. Это позволяет еще более эффективно обучать модель. Конкретные изменения, достигнутые с помощью этих методов, проиллюстрированы на рисунке ниже.

Его сетевая архитектура показана на рисунке 5.

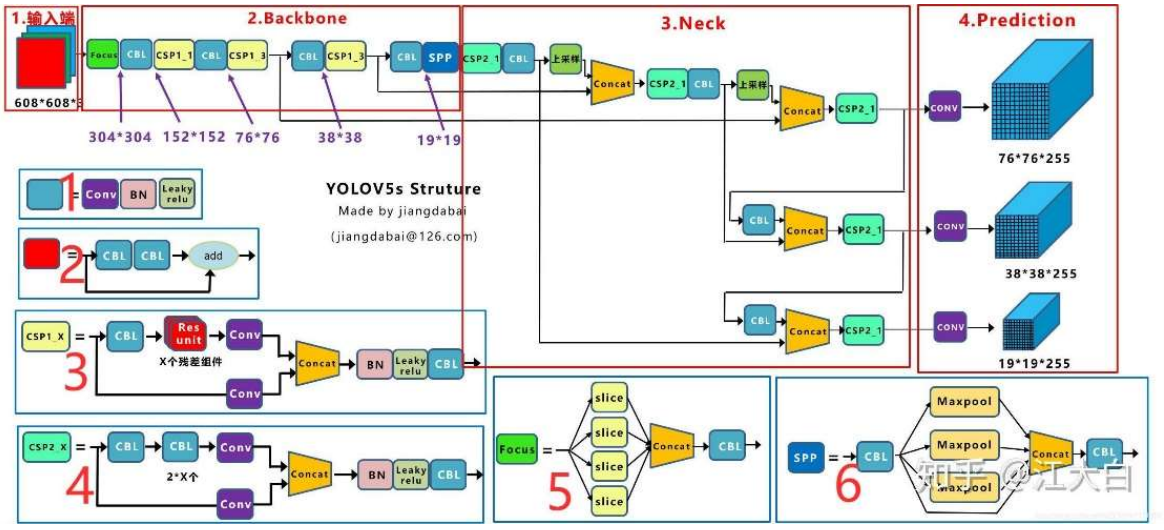

Рисунок 5 – Схема сетевой архитектуры YOLOv5

Этот метод улучшения позволяет объединить несколько изображений в одно, что не только обогащает набор данных, но и значительно повышает скорость обучения сети, а также может снизить требования к памяти модели.

2.2.4 PIDNet

PIDNet[7][8] представляет собой открытый проект сети для семантической сегментации изображений. Он выделяется своей высокой точностью, особенно в обработке изображений городской среды. Новая структура сети включает в себя дополнительную ветвь для обработки границ, что позволяет избежать проблем перерегулирования, характерных для предыдущих моделей. Благодаря этим особенностям, PIDNet может быть использован напрямую в реальном времени, что делает его привлекательным для таких приложений, как автономные транспортные средства и медицинская визуализация. В нашем проекте, который связан с дронами, PIDNet может использоваться для выполнения задач сегментации сцены, что помогает дронам лучше понимать окружающую среду и принимать более информированные решения.

Его сетевая архитектура показана на рисунке 6.

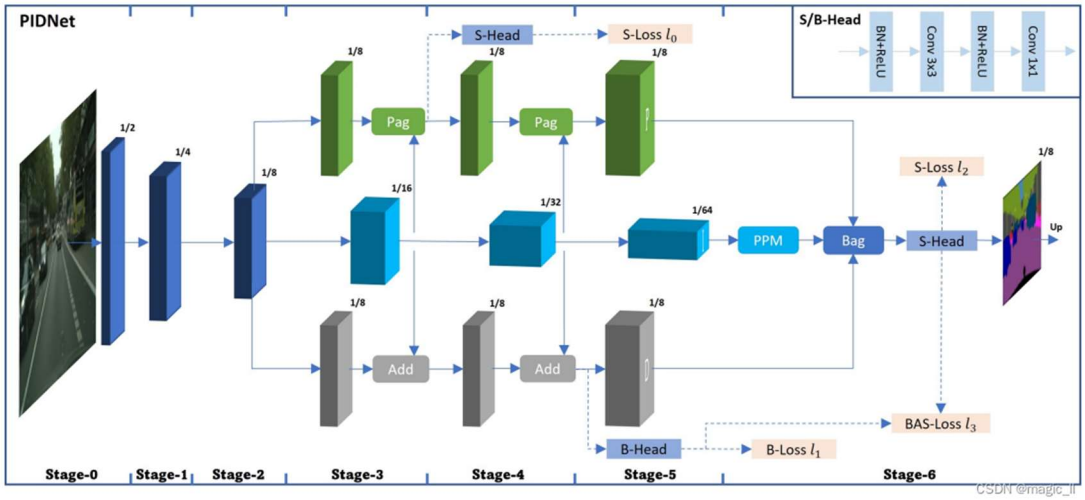

Рисунок 6 Схема архитектуры сети PIDNet

2.3 MAVLink

MAVLink[9] представляет собой легковесный протокол обмена сообщениями, который предназначен для общения с дронами и между их бортовыми компонентами. В нашем проекте мы предпочитаем использовать MAVLink для обмена сообщениями, так как он обеспечивает эффективную и надежную связь с минимальными накладными расходами. Использование такого

типа сообщений позволяет нам максимально свободно передавать информацию, что способствует общей производительности нашей сети. Кроме того, MAVLink предоставляет широкий набор сообщений и команд, что упрощает взаимодействие с дронами и обеспечивает гибкость в разработке программного обеспечения для автономных систем.

2.4 Определение высоты дрона

Барометр на дроне позволяет нам рассчитать приблизительную высоту дрона по показаниям барометра. Конкретный метод заключается в следующем:

Барометрический метод основан на использовании закономерного изменения атмосферного давления с высотой. Зависимость давления воздуха до 11000 м выражается формулой. Как показано в формуле (1).

$$P_{\text{н}} = P_0 \left(1 - \frac{t_{\text{гр}} H}{T_0}\right) \frac{1}{R t_{\text{гр}}} \quad (1)$$

Рецая это уравнение относительно высотыб формулу (2).

$$H = [1 - \left(\frac{P_{\text{н}}}{P_0}\right)^{t_{\text{гр}}{}^{R}}] \frac{T_0}{t_{\text{гр}}} \quad (2)$$

где R-газовая постоянная (29,27 м/град)

Из формуды видно что измерясмая высота является функцией четырех параметров давления на высоте полета Рн, давления и температуры на уровне начала отсчета высоты Ро и То и температурного градисита t гр.

Мы можем получить необходимые нам параметры с таких платформ, как Яндекс Погода, но на этот алгоритм сильно влияют погодные факторы в реальном приложении, поэтому этот алгоритм имеет низкий приоритет в последовательности разработки.

3 ПРОЦЕСС

3.1 Сбор и аннотирование набора данных

Пары изображений генерируются путем запуска дрона в поле и загрузки данных, которые лучше всего соответствуют изображению дрона, из Карт Google с помощью скрипта Python. И мы приводим практический пример на рисунке 7.

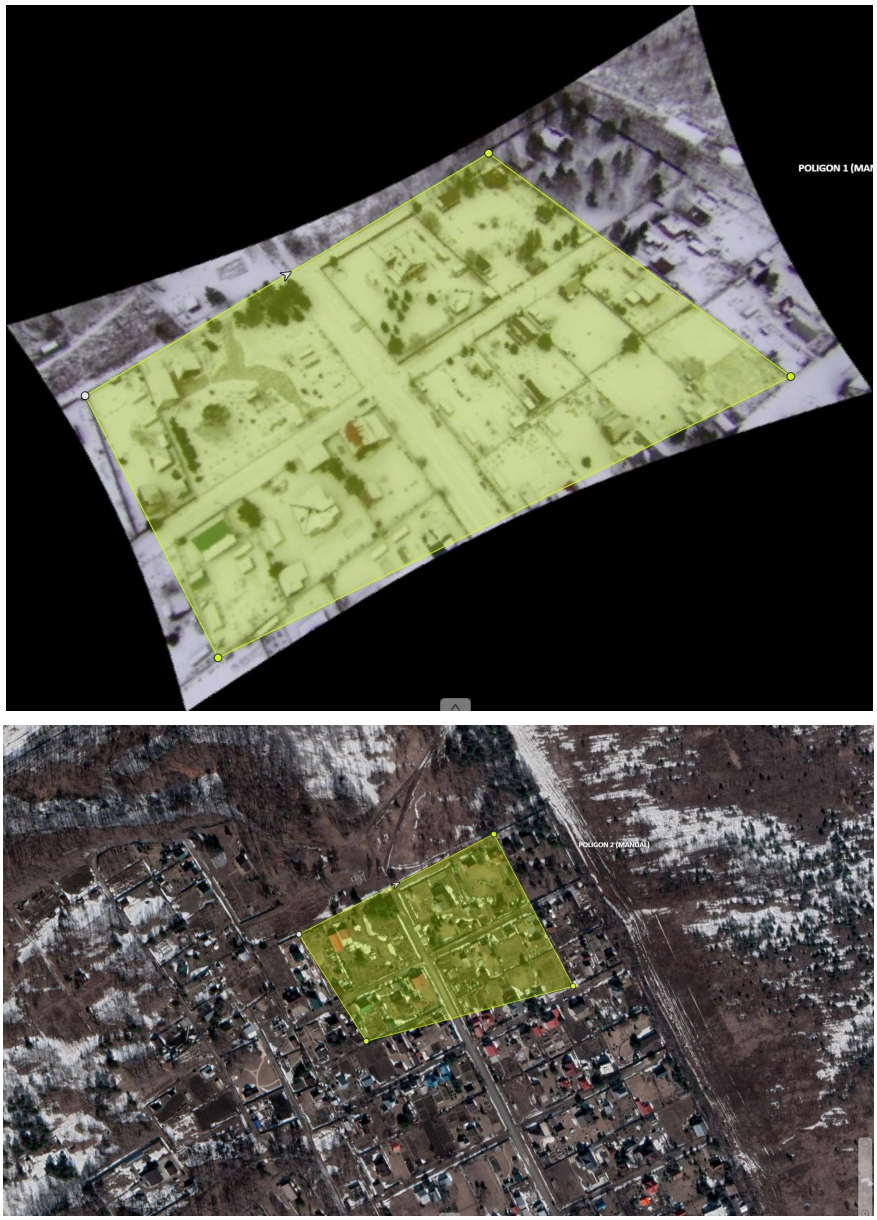

Рисунок 7 – Практические примеры аннотации набора данных

Задачи второго этапа заключаются, по сути, в дополнении и расширении набора данных на первом этапе, делая набор данных богаче и тем самым повышая надежность модели. Добавляя пары изображений с различными особенностями местности, модель имеет лучшие возможности обобщения.

Прежде чем маркировать пары изображений, изображения, снятые монокулярной камерой дрона, сначала корректируются. Монокулярная камера будет производить искажение соответствующей формы из-за линзы, содержащейся в объективе. Мы исправляем искажение с помощью следующего кода [a]. Конкретный эффект исправления искажений изображения показан на рисунке 8.

Рисунок 8 – Пример исправления искажений изображения дрона

CVAT (Computer Vision Annotation Tool) - это инструмент для аннотации данных в области компьютерного зрения, который также может использоваться для создания наборов данных для обучения моделей, таких как

для задачи матчинга точек на изображении. Вот подробное описание процесса аннотации и экспорта данных в формат COCO с использованием CVAT:

3.1.1 Создание задачи

В CVAT создается новая задача, где устанавливаются параметры, такие как тип аннотации «полигон», класс объекта (если требуется), а также другие настройки.

3.1.2 Добавление аннотаций

На выбранном кадре видео начинается процесс добавления аннотаций. Используется инструмент «полигон», с помощью которого добавляются точки на объектах, предназначенных для сопоставления. Как мы показываем на рисунке 9.

Рисунок 9 – Примеры ручных аннотаций в CVAT

3.1.3 Сохранение аннотаций

Готовые аннотации сохраняются в формате CVAT.

Затем мы реализуем наш скрипт Python, чтобы проверить, сможем ли мы сделать хорошую гомографию с парами изображений, которые мы сопоставили, и поместить хорошо совпадающие пары в набор данных, чтобы их можно было обучить в нейронной сети.

Таким образом, процесс заключается в аннотации точек на изображениях в CVAT, сохранении аннотаций в формате CVAT, а затем экспорте и преобразовании данных в формат COCO JSON для использования в обучении моделей компьютерного зрения.

3.1.4 Проверка достоверности данных

Для каждых аннотированных данных используйте библиотеку cv2 для создания перекрывающихся изображений, чтобы проверить достоверность четырех выбранных характерных точек и обеспечить правильное обучение модели.

Мы рассматриваем возможность перекрытия двух изображений, показанных на рисунке 10, чтобы проверить эффективность аннотации данных.

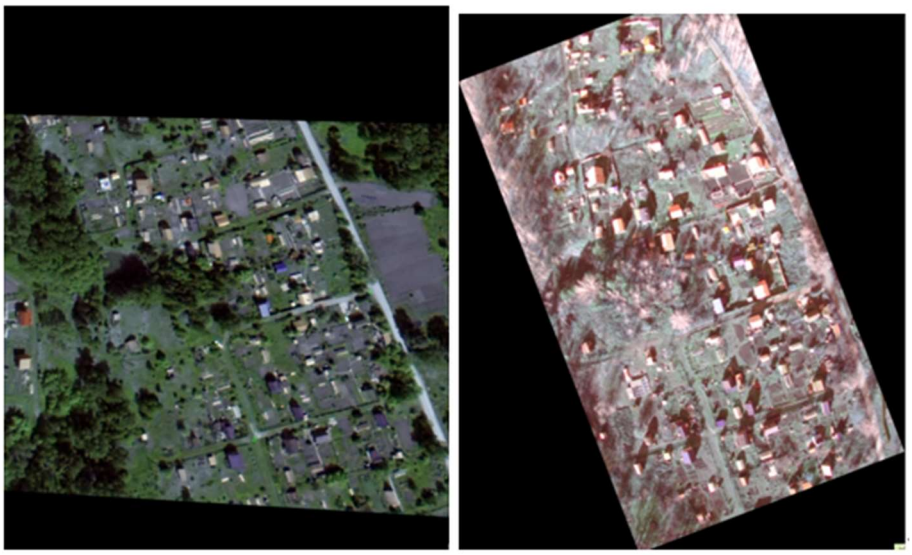

Рисунок 10 – Пример пары изображений

На рисунке 11 мы перекрываем две картинки:

Рисунок 11 – Пример перекрывающегося изображения

С помощью метода, показанного на рисунке выше, можно хорошо обеспечить качество ввода данных в модель, тем самым достигая требуемых функций.

Вышеуказанные функции реализованы с помощью кода в Приложении [1].

3.2 Обучение сети CLIP

Обучение сети CLIP в основном проводится в рамках официальной структуры OPENAI, но некоторые ее части были изменены в соответствии с реальной ситуацией. Ниже приведены ключевые части процесса обучения.

В сценарии обучения OPENAI мы видим [б], здесь определены основные параметры обучения модели. Тип модели — «resnet50». Это широко используемая остаточная нейронная сеть, имеющая широкий спектр приложений в области распознавания изображений.

Далее мы определяем класс AvgMeter[в], который используется для хранения среднего количества параметров производительности сети для оценки производительности сети.

Затем готовимся к обучению сети, предварительно подготавливая загрузчик данных.

Класс CLIPDataset[г]:Этот класс представляет собой класс набора данных PyTorch, используемый для загрузки изображений и соответствующих данных аннотаций.

Во время инициализации он сохраняет список имен файлов изображений, списки аннотаций и токенизатор и использует токенизатор для кодирования аннотаций.

__getitem__(self, idx): индексный метод набора данных, используемый для получения данных по указанному индексу. Он возвращает объект словаря, содержащий изображение и соответствующие аннотации.

В этом методе он сначала извлекает аннотацию по указанному индексу из закодированного списка аннотаций и преобразует ее в тензор PyTorch. Затем он считывает файл изображения, соответствующий индексу, выполняет предварительную обработку изображения (например, преобразование цветового пространства, изменение размера и нормализацию) и, наконец, преобразует данные изображения в тензор PyTorch с размером канала в качестве первого измерения.

Мы определяем функцию, которая нормализуется в соответствии с фактическим методом обработки данных[д]:

Если режим «train», возвращает серию операций преобразования данных, используемых для обучения, включая изменение размера и нормализацию изображений.

Если режим не «train», возвращает серию операций преобразования данных для тестирования, включая изменение размера и нормализацию изображения.

Среди кодеров, предоставленных OPENAI, есть кодеры текста и редакторы изображений, поскольку сеть CLIP изначально была сетью, которая генерирует совпадающие пары текст-изображение:

Image Encoder：В этом коде OPENAI[e] представляет определенный выше класс CFG, который содержит основные параметры обучения модели.

self.model = timm.create_model(model_name, pretrained, num_classes=0, global_pool="avg"): создала модель с помощью функции create_model в библиотеке timm (pytorch-image-models). Эта функция создает указанную модель на основе model_name, и вы можете выбрать, использовать ли предварительно обученные веса. num_classes=0 указывает, что последний уровень классификации удален, а global_pool="avg" указывает, что объединение глобальных средних значений используется в качестве окончательного метода извлечения признаков. Это делается для преобразования входного изображения произвольного размера в вектор признаков фиксированного размера.

for p в self.model.parameters(): p.requires_grad = trainable:мы перебираем параметры модели и устанавливаем, требуется ли градиент на основе обучаемого параметра. Если обучаемый имеет значение True, для установки параметров требуются градиенты, и можно выполнять обучение. Если обучаемость имеет значение False, заданные параметры не требуют градиентов, то есть остаются неизменными в процессе обучения. Обычно это используется для фиксации некоторых параметров при тонкой настройке модели.

Text Encoder：[ж]

self.model = DistilBertModel(config=DistilBertConfig()): создайте модель DistilBERT, которая не была предварительно обучена, и назначьте ее self.model. В этом случае для настройки модели используется DistilBertConfig по умолчанию.

DistilBERT — это небольшая, быстрая, дешевая и легкая модель трансформатора, обученная на основе дистиллированного BERT. Он имеет на 40 % меньше параметров, чем google-bert/bert-base-uncases, и работает на 60 % быстрее, сохраняя при этом более 95 % возможностей понимания языка BERT, измеренных на языках GLUE.

Очевидно, что в процессе разработки нашего алгоритма умение понимать язык не понадобится, поэтому мы рассматриваем возможность внесения в него определенных модификаций.

Для практических нужд нам необходимо внести изменения в энкодер, как показано на рисунке 2.

Другими словами, нам необходимо преобразовать исходный кодировщик текста и кодировщик изображений в кодировщик изображений дрона и кодировщик спутниковых изображений, чтобы, наконец, добиться генерации пар спутниковое изображение-изображение дрона и, таким образом, разработать алгоритм для маркировки дрона. местоположение на спутниковом снимке:

SatelliteEncoder：[з]

Класс SatelliteEncoder дополнительно реализует закрытый метод _get_features и атрибут Last_feature, которые используются для получения промежуточного представления признаков модели. Получив доступ к представлению промежуточных признаков модели, мы можем лучше изучить эффект этого Encodera.

DroneEnconder：[и]

Метод _get_features и атрибут Last_feature в DroneEncoder имеют те же имена и функции, что и соответствующие части в SatelliteEncoder. Оба позволяют получить представление признаков на определенных уровнях модели, тем самым используя их для сопоставления характерных точек модели, что в конечном итоге реализует общую функциональность алгоритма.

По форме два кодировщика примерно одинаковы, но есть различия в функциональности и использовании.

Мы определяем класс модели PyTorch ProjectionHead, который используется для проектирования входных функций при самостоятельном обучении. Функционал и структура точно такие же, как у оригинальных, но отличаются значениями параметров инициализации по умолчанию:[й]

Самое главное, определяем функцию для обучения нейронной сети:[к].

Метод инициализации модели принимает два параметра: температура представляет собой параметр температуры, который используется для настройки потерь при сравнении, а image_embedding представляет размер встраивания изображения. Эти параметры по умолчанию получаются из объекта глобальной конфигурации CFG.

В методе прямого распространения сначала кодируются пакеты спутниковых изображений для получения встроенного представления изображений, а затем рассчитывается потеря контраста между изображениями. Внутренний продукт между изображением дрона и спутниковым изображением рассчитывается с помощью матричного умножения, а затем делится на температурный параметр, чтобы получить начальные логиты потери контраста.

После этого рассчитывается целевая потеря контрастности. Матрицы подобия спутниковых изображений и изображений дронов усредняются и умножаются на температурный параметр, а затем с помощью функции softmax получается целевая потеря контрастности. Потери контрастности БПЛА и спутниковых изображений усредняются для получения окончательной потери контрастности. Наконец, среднее значение потери контрастности возвращается в качестве результата прямого прохода.

В основном мы следуем официальному процессу обучения пользовательскому набору данных для процесса обучения:[л]

Наконец, мы также используем официальную функцию main для обучения[м].

3.3 Практическая проверка эффекта сиамской сети

После завершения обучения сети CLIP мы попытались оценить эффект обработки сети CLIP на реальных спутниковых изображениях.

Представляем изображения и данные для проверки, класс DronePos определен в коде версии 3.4.

Мы преобразуем массив NumPy Satellite_features_map в тензор PyTorch и сохраняем его в одноименной переменной. Это для последующей обработки с помощью PyTorch[н].

Затем мы определяем функцию для преобразования искаженного исходного изображения, снятого дроном, в ортогональное изображение[о].

Основной алгоритм работы следующий:[п]

Получаем список файлов и сортируем их: Сначала код получает список файлов в указанном каталоге source_frames. Затем выполните сортировку по имени файла, которое обычно имеет числовую форму.

Читаем изображение и извлекаем метаданные: затем выбираем конкретный файл изображения для обработки.

Открываем изображение с помощью функции Image.open() библиотеки Pillow, а затем извлекаем информацию о положении и ориентации дрона из метаданных изображения.

Преобразуем изображение в массив NumPy: Преобразуем изображение дрона в массив NumPy, содержащий значения пикселей изображения.

Выполняем орфографическое преобразование изображения с помощью функции to_orto(). Это может включать изменение ориентации и/или масштабирования изображения.

Обрезаем изображение: посредством операции нарезки мы обрезаем интересующую область на изображении дрона, чтобы получить интересующие нас сегменты.

Читаем изображение карты и преобразуем его в формат RGB: читаем изображение карты с помощью функции cv2.imread() библиотеки OpenCV, а затем преобразуем его в формат RGB с помощью cv2.cvtColor().

Создаем копию изображения карты для последующей обрезки: Создайте копию изображения карты для последующей обрезки, чтобы избежать внесения изменений в исходное изображение карты.

Наконец, мы вводим всю последовательность изображений в сеть и получаем через сеть нужную нам тепловую карту и соответствующее изображение на спутниковой карте[р].

Но когда мы проверили, как этот алгоритм работает на реальных изображениях, результаты все равно были не идеальными. Как мы показываем на рисунке 12.

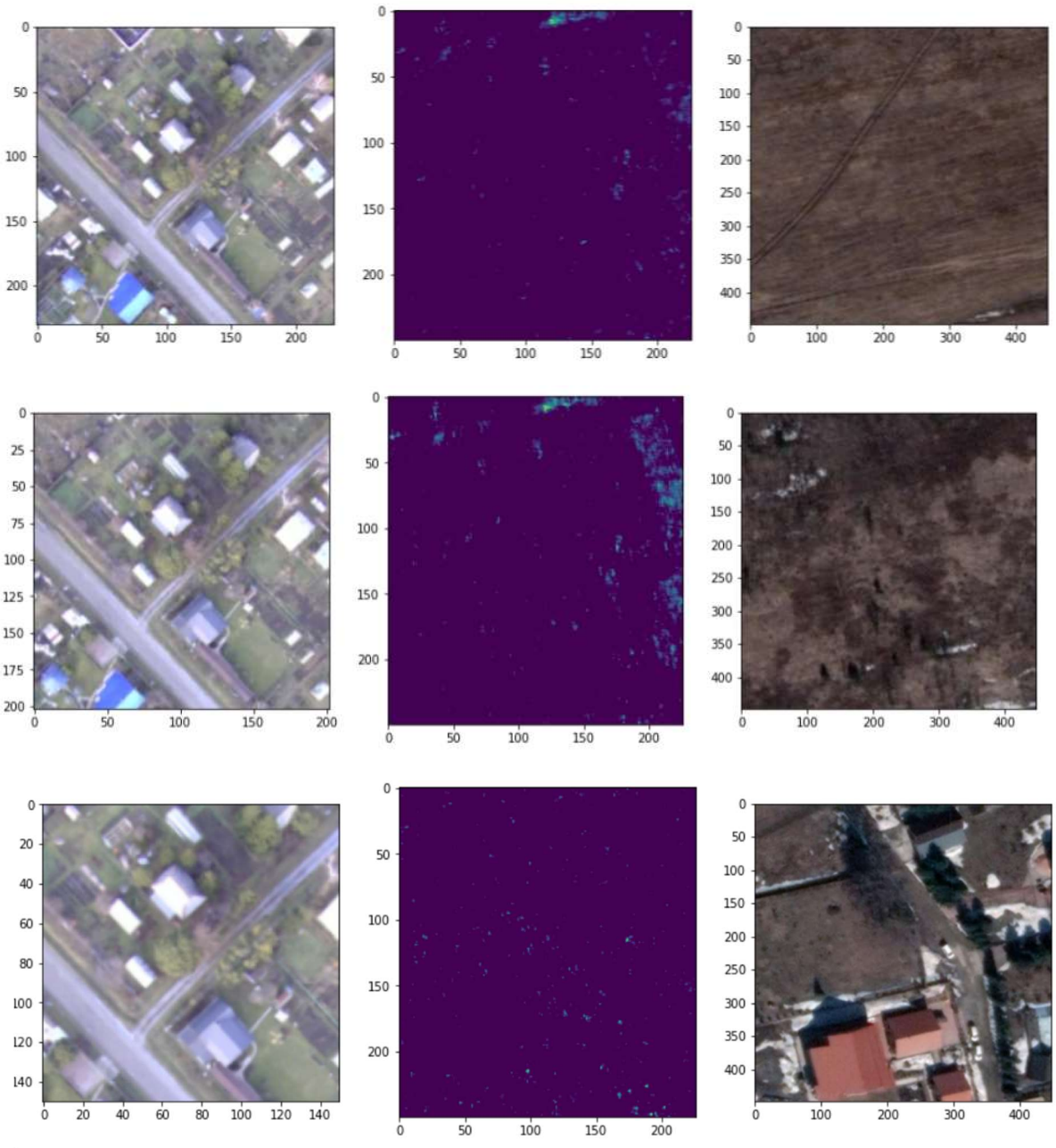

Рисунок 12 – Общие результаты, возвращаемые нейронными сетями

В большинстве случаев, когда мы пытаемся получить изображение, соответствующее области с наибольшим значением на тепловой карте, мы получаем результаты, аналогичные показанному на рисунке. Левая часть — это изображение, сделанное дроном, средняя — сгенерированная тепловая карта, а

правая — изображение области с самым высоким значением на соответствующей тепловой карте на спутниковом изображении.

Лишь на небольшом количестве изображений мы получаем результаты, показанные на рисунке 13. Хотя полученное изображение похоже на изображение, сделанное дроном, это явно не тот результат, который нам нужен.

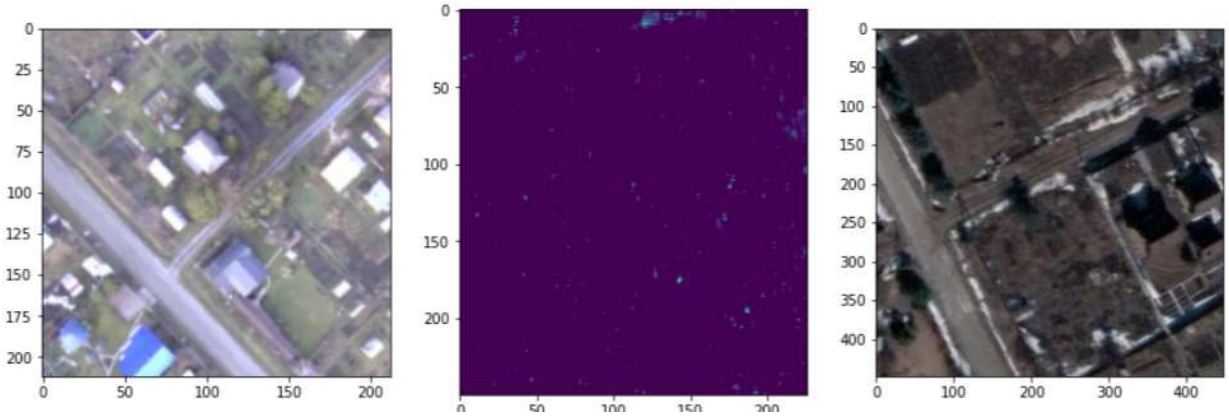

Рисунок 13 – Сеть возвращает некоторые результаты с большей достоверностью

3.4 Алгоритм обновления положения БПЛА в реальном времени

После завершения обучения сети CLIP мы попытались определить алгоритм, который позволит дрону обновлять свое местоположение в режиме реального времени с помощью тепловой карты, сгенерированной сетью.

Сначала мы определяем класс, который хранит информацию о местоположении дрона:

Класс DronPos[c] — важный класс, который мы используем для позиционирования дронов и построения маршрутов полета.

__init__ метод:

Метод инициализации, используемый для установки исходного положения и карты функций дрона.Принимает исходные координаты и карту объектов в качестве параметров и сохраняет их в свойствах класса.Вызовите метод update_area(), чтобы обновить карту объектов для текущей области.

метод update_area:

Обновите карту объектов области, где находится текущий дрон.Устанавливает всю карту объектов в качестве карты объектов текущего региона.

метод update_pos:

Местоположение дрона обновляется, и карта объектов его территории соответствующим образом обновляется.Принимает дополнительную координату в качестве аргумента, добавляя ее к текущим координатам положения для расчета новой позиции дрона.Обновите атрибут местоположения дрона и вызовите метод update_area(), чтобы обновить карту окружающих объектов.

метод get_area:

Возвращает характерную карту местности, где находится текущий дрон.Используется для получения карты объектов окружающей среды вокруг дрона для использования другими кодами.

Затем мы преобразуем изображение в тензорную форму, чтобы подготовить его к вводу в обученную сеть CLIP.После этого загружаем параметры обученной модели[т].

Мы определяем функцию get_features_map для извлечения карт объектов из входного изображения. Эта функция перемещает окна фиксированного размера по изображению и сшивает извлеченные объекты в каждом окне в карту объектов[у].

Мы в основном получаем карты признаков с помощью следующих четырех шагов:

Вычисление шага и числа срезов:

Во-первых, мы вычисляем шаг скольжения, исходя из параметра n, где n - это управляющий фактор, определяющий размер скользящего окна.Затем мы вычисляем, сколько срезов можно сделать в тензоре по строкам и столбцам, чтобы определить количество операций срезания на входном тензоре.

Определение необходимости среза:

Если количество строк или столбцов для среза меньше или равно 0, это означает, что размер входного тензора меньше размера скользящего окна, и мы не можем выполнить операцию срезания. В этом случае мы просто извлекаем все входные данные для извлечения признаков, затем нормализуем и возвращаем.

Перебирает срезы:

Мы проходимся по каждому срезу по строкам и столбцам, обходя все возможные позиции.На каждой итерации мы извлекаем срез данных из входного тензора и передаем его модели для извлечения признаков.

Затем мы транспонируем тензор признаков для последующей обработки.

Мы добавляем тензор признаков к карте признаков для построения полной карты признаков.

Возврат карты признаков:

После обхода всех срезов мы получаем полную карту признаков.

Мы нормализуем карту признаков и возвращаем нормализованную карту признаков.

Определена функция get_drone_features для извлечения функций из изображений дронов. Эта функция сначала обрезает и предварительно обрабатывает изображение, а затем использует предварительно обученную модель CLIP для извлечения функций[ф].

Затем мы определили функцию get_heat_map для создания тепловой карты местоположения дрона и окружающей территории. Эта функция выполняет операцию свертки с объектами БПЛА и региональными объектами для создания тепловой карты и выполнения постобработки.

Используем функцию get_pixel для получения координат самого большого пикселя на тепловой карте.

Наконец, мы определяем его логику в основной функции[х]:

Целью этой основной функции является управление работой дрона в заданном кадре и положении. Пошаговое объяснение следующее:

Получаем особенности изображений, снятых дронами: через функцию get_drone_features(frame) получаем особенности изображений в текущем кадре.

Получаем карту объектов окружающей области: используйте функцию dron_position.get_area(), чтобы получить карту объектов области вокруг дрона.

Получаем тепловую карту: используя полученные характеристики дрона и карту объектов окружающей местности, мы получаем тепловую карту с помощью функции get_heat_map(sat_features,drone_features).

Мы получаем координаты самого большого пикселя на тепловой карте: используйте функцию get_max_idxs(hm), чтобы получить координаты самого большого пикселя на тепловой карте и вернуть обновленную Heat_map и координаты самого большого пикселя.

Преобразуем координаты самого большого пикселя: с помощью функции get_pixel(y_x) преобразуем координаты самого большого пикселя в координаты пикселя и возвращаем смещение относительно текущей позиции.

Обновляем положение дрона: используя вычисленное смещение, используя функцию dron_position.update_pos(dy_dx).

Возврат результатов: возврат обновленной позиции дрона и тепловой карты в качестве результатов.

3.5 Алгоритм маркировки траектории полета БПЛА

Мы также разработали алгоритм, который может отображать путь дрона на спутниковых снимках. Его основная функция заключается в следующем.

Сначала преобразуем карту признаков спутникового изображения в необходимую форму и создаем экземпляр объекта дрона[ц].

Затем мы создадим функцию для преобразования искаженного изображения, снятого дроном, в ортогональное изображение[ч].

После этого мы разработали функцию, позволяющую найти точное местоположение дрона на графике и отметить на нем точки[ш].

Конкретные функции основной функции можно резюмировать следующим образом:

Читаем входной файл изображения карты source_map.

Наносим географические координаты на изображение карты и определяем пиксельные координаты маршрута.

Поворачиваем входное изображение дрона и регулируем ориентацию изображения дрона в соответствии с направлением компаса.

Обрезаем повернутое изображение дрона, чтобы сохранить интересующую область.

Делаем логические выводы на основе обрезанного изображения дрона, чтобы получить местоположение дрона на карте.

Наносим на карту текущее и прогнозируемое местоположение дрона.

Масштабируем изображение карты и отмечаем на нем прогнозируемое местоположение дрона.

Сохраняем аннотированное изображение карты в виде файла и возвращаем его.

В общем, функция этого алгоритма — нарисовать изображение траектории полета дрона на спутниковом изображении и вернуть его, чтобы пользователь мог его осмотреть.

ЗАКЛЮЧЕНИЕ

Цель этого проекта — найти подходящие решения для навигации дронов. Его главное преимущество заключается в том, что он не требует использования традиционных методов навигации, таких как GPS. Благодаря реализации зрелых алгоритмов нейронных сетей, таких как LoFTR ResNet YOLOv5 и PIDNet, будущее этого проекта очень светлое.

Этот алгоритм может в определенной степени избежать зависимости дронов от оборудования GPS, позволяя разработке алгоритмов дронов иметь больше новых путей и различных возможностей. Фактическая ценность этого проекта может иметь большое значение в будущем. другие поля были отражены.

Разработка базового алгоритма в основном завершена. После завершения всего проекта мы сможем выполнить: навигацию БПЛА, позиционирование БПЛА и прорисовку маршрута полета БПЛА без опоры на GPS и спутниковые снимки.

Основная проблема, с которой в настоящее время сталкивается этот проект, заключается в том, что обученная «сиамская сеть» кажется неудовлетворительной в реальном применении и, следовательно, не может быть использована в качестве основы для навигации БПЛА и связанных с ней алгоритмов, которые мы разработали. Мы рассматриваем возможность попробовать больше. Есть много способов. оптимизировать фактическую производительность этой сети, в том числе опробовать больше различных сетевых структур и добавить больше новых эффективных данных в набор данных.

Кроме того, мы попытались заменить официальную функцию потерь в процессе обучения CLIP на триплетные потери, поэтому более научное название разработанной нами сети — сиамская сеть.

Из-за неудовлетворительной фактической производительности нейронной сети мы временно не можем провести полевые испытания разработанного алгоритма на дронах, но надеемся завершить эту работу как можно скорее.

Мы также постараемся использовать другие сетевые структуры, упомянутые выше, такие как Course Lofter и т. д., для выполнения других необходимых функций, таких как использование дронов для распознавания объектов и других операций, чтобы сделать разрабатываемый нами алгоритм навигации дронов максимально эффективным. возможно, довольно всеобъемлющий.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1.Orange PI 5 User Manual [Электронный ресурс]. Режим доступа: https://drive.google.com/file/d/13LJk85ueOup2HqbEhcY2UhUw5lwESth_/view?pli=1(дата обращения: 14.12.2023).

2. Github. A Python module for accessing the MPU-6050 digital accelerometer and gyroscope on a Raspberry Pi. [Электронный ресурс]. Режим доступа: https://github.com/m-rtijn/mpu6050 (дата обращения: 14.12.2023).

3. Github. Helper library for the Hillcrest Laboratories BNO080 IMU [Электронный ресурс]. Режим доступа:

https://github.com/adafruit/Adafruit_CircuitPython_BNO08x. (дата обращения: 14.12.2023).

4. Learning Transferable Visual Models From Natural Language Supervision (Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, Ilya Sutskever) [Электронный ресурс] Режим доступа: https://arxiv.org/abs/2103.00020

5. LoFTR: Detector-Free Local Feature Matching with Transformers(Jiaming Sun, Zehong Shen, Yuang Wang, Hujun Bao, Xiaowei Zhou) [Электронный ресурс] Режим доступа: https://arxiv.org/abs/2104.00680

6. TPH-YOLOv5: Improved YOLOv5 Based on Transformer Prediction Head for Object Detection on Drone-captured Scenarios(Xingkui Zhu, Shuchang Lyu, Xu Wang, Qi Zhao) [Электронный ресурс] Режим доступа :https://arxiv.org/abs/2108.11539

7. PIDNet: A Real-time Semantic Segmentation Network Inspired by PID Controllers (Jiacong Xu, Zixiang Xiong, Shankar P. Bhattacharyya) [Электронный ресурс] Режим доступа : https://arxiv.org/abs/2206.02066.

8. Github. PIDNet: A Real Time Semantic Segmentation Network Inspired from PID Controller[Электронный ресурс] Режим доступа: https://github.com/XuJiacong/PIDNet(дата обращения: 14.12.2023).

9. Micro Air Vehicle Link (MAVLink) in a Nutshell: A Survey (Anis Koubaa, Azza Allouch, Maram Alajlan, Yasir Javed, Abdelfettah Belghith, and Mohamed Khalgu) [Электронный ресурс] Режим доступа: https://arxiv.org/pdf/1906.10641.pdf

ПРИЛОЖЕНИЕ А

Код программы

```
        dist_coefs = np.array([-0.35328536, 0.2090826, 0,  0,  -0.06274637])
        i = 0
        #for img_found in img_names_undistort:
        while i < len(img_names_undistort):
            img = cv2.imread(img_names_undistort[i])
            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

            h,  w = img.shape[:2]
                newcameramtx,  roi  =  cv2.getOptimalNewCameraMatrix(camera_matrix,
    dist_coefs, (w, h), 1, (w, h))
        def to_homo(pts):
            num_pts = pts.shape[0]
            homo = np.ones((num_pts,1), dtype=pts.dtype)
            return np.hstack((pts, homo))
        camPts = to_homo(pnts1)
        satPts = to_homo(pnts0)
        HoM, homask = cv2.findHomography(camPts, satPts, cv2.RANSAC, 5.0)
        warped = cv2.warpPerspective(camImg, HoM, satImg.shape[:2][::-1])
        plt.figure(figsize = (20, 15))
        plt.imshow(warped)
        plt.show()
```

[б]

```
    class CFG:
        debug = False
        image_path = "../input/flickr-image-dataset/flickr30k_images/flickr30k_images"
        captions_path = "."
        batch_size = 32
        num_workers = 4
        head_lr = 1e-3
        image_encoder_lr = 1e-4
        text_encoder_lr = 1e-5
        weight_decay = 1e-3
        patience = 1
        factor = 0.8
        epochs = 20
        device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

        model_name = 'resnet50'
        image_embedding = 2048
        text_encoder_model = "distilbert-base-uncased"
        text_embedding = 768
        text_tokenizer = "distilbert-base-uncased"
        max_length = 200

        pretrained = True # for both image encoder and text encoder
        trainable = True # for both image encoder and text encoder
```

```
    temperature = 1.0

    # image size
    size = 224

    # for projection head; used for both image and text encoders
    num_projection_layers = 1
    projection_dim = 256
    dropout = 0.1
```

[в]

```
class AvgMeter:
    def __init__(self, name="Metric"):
        self.name = name
        self.reset()

    def reset(self):
        self.avg, self.sum, self.count = [0] * 3

    def update(self, val, count=1):
        self.count += count
        self.sum += val * count
        self.avg = self.sum / self.count

    def __repr__(self):
        text = f"{self.name}: {self.avg:.4f}"
        return text

def get_lr(optimizer):
    for param_group in optimizer.param_groups:
        return param_group["lr"]
```

[г]

```
class CLIPDataset(torch.utils.data.Dataset):
    def __init__(self, image_filenames, captions, tokenizer, transforms):
        """
        image_filenames and cpations must have the same length; so, if there are
        multiple captions for each image, the image_filenames must have repetitive
        file names
        """

        self.image_filenames = image_filenames
        self.captions = list(captions)
        self.encoded_captions = tokenizer(
            list(captions), padding=True, truncation=True, max_length=CFG.max_length
        )
        self.transforms = transforms

    def __getitem__(self, idx):
        item = {
```

```
            key: torch.tensor(values[idx])
            for key, values in self.encoded_captions.items()
        }

        image = cv2.imread(f'{CFG.image_path}/{self.image_filenames[idx]}')
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        image = self.transforms(image=image)['image']
        item['image'] = torch.tensor(image).permute(2, 0, 1).float()
        item['caption'] = self.captions[idx]

        return item


    def __len__(self):
        return len(self.captions)
```

[д]

```
def get_transforms(mode="train"):
    if mode == "train":
        return A.Compose(
            [
                A.Resize(CFG.size, CFG.size, always_apply=True),
                A.Normalize(max_pixel_value=255.0, always_apply=True),
            ]
        )
    else:
        return A.Compose(
            [
                A.Resize(CFG.size, CFG.size, always_apply=True),
                A.Normalize(max_pixel_value=255.0, always_apply=True),
            ]
        )
```

[е]

```
class ImageEncoder(nn.Module):
    """
    Encode images to a fixed size vector
    """

    def __init__(
                        self, model_name=CFG.model_name, pretrained=CFG.pretrained,
trainable=CFG.trainable
    ):
        super().__init__()
        self.model = timm.create_model(
            model_name, pretrained, num_classes=0, global_pool="avg"
        )
        for p in self.model.parameters():
```

```
            p.requires_grad = trainable

        def forward(self, x):
            return self.model(x)
```

[ж]

```
    class TextEncoder(nn.Module):
        def __init__(self, model_name=CFG.text_encoder_model, pretrained=CFG.pretrained,
trainable=CFG.trainable):
            super().__init__()
            if pretrained:
                self.model = DistilBertModel.from_pretrained(model_name)
            else:
                self.model = DistilBertModel(config=DistilBertConfig())

            for p in self.model.parameters():
                p.requires_grad = trainable

            # we are using the CLS token hidden representation as the sentence's embedding
            self.target_token_idx = 0

        def forward(self, input_ids, attention_mask):
            output = self.model(input_ids=input_ids, attention_mask=attention_mask)
            last_hidden_state = output.last_hidden_state
            return last_hidden_state[:, self.target_token_idx, :]
```

[з]

```
    class SatelliteEncoder(nn.Module):
        """
        Encode images to a fixed size vector
        model_name = ResNet50, pretrained=True, trainable=True
        """

        def __init__(self, model_name='resnet50',
                     pretrained=True, trainable=True,
                     image_embedding=2048):
            super().__init__()
            self.resnet = timm.create_model(
                model_name, pretrained, num_classes=0, global_pool="avg"
            )
            for p in self.resnet.parameters():
                p.requires_grad = trainable
            self.satellite_projection = ProjectionHead(embedding_dim=image_embedding)
            self.resnet.layer4.register_forward_hook(self._get_features)

        def forward(self, x):
            x = self.resnet(x)
            return self.satellite_projection(x), self._avgpool_features

        @property
```

```
    def last_feature(self):
        """It works after 'forward' only"""
        return self._avgpool_features

    def _get_features(self, module, inputs, output):
        self._avgpool_features = output #output.data.cpu().numpy()
```

[и]

```
class DroneEncoder(nn.Module):
    """
    Encode images to a fixed size vector
    """

    def __init__(self, model_name='resnet50',
                 pretrained=True, trainable=True,
                 image_embedding=2048):
        super().__init__()
        self.resnet = timm.create_model(
            model_name, pretrained, num_classes=0, global_pool="avg"
        )
        for p in self.resnet.parameters():
            p.requires_grad = trainable
        self.drone_projection = ProjectionHead(embedding_dim=image_embedding)
        self.resnet.layer4.register_forward_hook(self._get_features)

    def forward(self, x):
        x = self.resnet(x)
        return self.drone_projection(x), self._avgpool_features

    @property
    def last_feature(self):
        """It works after 'forward' only"""
        return self._avgpool_features

    def _get_features(self, module, inputs, output):
        self._avgpool_features = output
```

[й]

```
class ProjectionHead(nn.Module):

    def __init__(self, embedding_dim, projection_dim=CFG.projection_dim,dropout=0.1):
        super().__init__()
        self.projection_dim = projection_dim
        self.projection = nn.Linear(embedding_dim, projection_dim)
        self.gelu = nn.GELU()
        self.fc = nn.Linear(projection_dim, projection_dim)
        self.dropout = nn.Dropout(dropout)
        self.layer_norm = nn.LayerNorm(projection_dim)

    def forward(self, x):
```

```
            projected = self.projection(x)
            x = self.gelu(projected)
            x = self.fc(x)
            x = self.dropout(x)
            x = x + projected
            x = self.layer_norm(x)
            return x
```

[к]:

```
class CLIPModel(nn.Module):
    def __init__(
        self,
        temperature=CFG.temperature,
        image_embedding=CFG.image_embedding,
    ):
        super().__init__()
        # Инициализация модели CLIP
        self.satellite_encoder = SatelliteEncoder()
    # Инициализация кодировщика спутниковых изображений
        self.drone_encoder = DroneEncoder()
          # Инициализация кодировщика изображений с беспилотного летательного аппарата
        self.temperature = temperature
          # Параметр температуры для регулировки потерь сравнения


    def forward(self, batch):
        # Процесс прямого распространения
        # Получение вложений изображений
        satellite_embeddings, _ = self.satellite_encoder(batch["satellite"])
          # Кодирование спутниковых изображений
        drone_embeddings, _ = self.drone_encoder(batch["drone"])
          # Кодирование изображений с беспилотного летательного аппарата

        # Вычисление потерь
        logits = (drone_embeddings @ satellite_embeddings.T) / self.temperature
          # Вычисление логитов предсказания для потерь сравнения
        satellite_similarity = satellite_embeddings @ satellite_embeddings.T
          # Вычисление матрицы сходства для спутниковых изображений
        drone_similarity = drone_embeddings @ drone_embeddings.T
          # Вычисление матрицы сходства для изображений с беспилотного летательного аппарата
        targets = F.softmax(
            (satellite_similarity + drone_similarity) / 2 * self.temperature, dim=-1
        )
          # Вычисление целевого распределения для сравнения, используя среднее значение сходства спутниковых и изображений с беспилотного летательного аппарата

        # Вычисление потерь сравнения
        drone_loss = cross_entropy(logits, targets, reduction='none')
```

```
            # Вычисление потерь сравнения для изображений с беспилотного летательного аппарата
            satellite_loss = cross_entropy(logits.T, targets.T, reduction='none')
            # Вычисление потерь сравнения для спутниковых изображений
            loss =  (satellite_loss + drone_loss) / 2.0
            # Усреднение потерь их изображений и спутников, форма (batch_size)
            return loss.mean()  # Возвращение среднего значения потерь
```

[л]

```
def train_epoch(model, train_loader, optimizer, lr_scheduler, step):
    loss_meter = AvgMeter()
    tqdm_object = tqdm(train_loader, total=len(train_loader))
    for batch in tqdm_object:
        batch = {k: v.to(CFG.device) for k, v in batch.items() if k != "caption"}
        loss = model(batch)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        if step == "batch":
            lr_scheduler.step()

        count = batch["image"].size(0)
        loss_meter.update(loss.item(), count)

        tqdm_object.set_postfix(train_loss=loss_meter.avg, lr=get_lr(optimizer))
    return loss_meter


def valid_epoch(model, valid_loader):
    loss_meter = AvgMeter()

    tqdm_object = tqdm(valid_loader, total=len(valid_loader))
    for batch in tqdm_object:
        batch = {k: v.to(CFG.device) for k, v in batch.items() if k != "caption"}
        loss = model(batch)

        count = batch["image"].size(0)
        loss_meter.update(loss.item(), count)

        tqdm_object.set_postfix(valid_loss=loss_meter.avg)
    return loss_meter
```

[м]

```
def main():
    train_df, valid_df = make_train_valid_dfs()
    train_loader = build_loaders(train_df, mode="train")
    valid_loader = build_loaders(valid_df, mode="valid")

    #Создаем экземпляр модели CLIP и перемещаем его на указанное устройство
    model = CLIPModel().to(CFG.device)
```

```
    # Определяем параметры модели и настройку оптимизатора
    params = [
        {"params": model.DroneEncoder.parameters(), "lr": CFG.DroneEncoder_lr},
        {"params": model.SatelliteEncoder.parameters(), "lr": CFG.SatelliteEncoder_lr},
        {"params": itertools.chain(
                                                model.DroneEncoder_projection.parameters(),
model.SatelliteEncoder_projection.parameters()
        ), "lr": CFG.head_lr, "weight_decay": CFG.weight_decay}
    ]

    optimizer = torch.optim.AdamW(params, weight_decay=0.)
    lr_scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(
        optimizer, mode="min", patience=CFG.patience, factor=CFG.factor
    )

    # определяем этапы обучения
    step = "epoch"

    #Инициализируем оптимальную ошибку проверки до положительной бесконечности
    best_loss = float('inf')

    for epoch in range(CFG.epochs):
        print(f"Epoch: {epoch + 1}")

        model.train()

        # Выполняем эпоху обучения и возвращаем ошибку обучения
        train_loss = train_epoch(model, train_loader, optimizer, lr_scheduler, step)

        # Устанавливаем модель в режим оценки
        model.eval()

        # Рассчитываем ошибку проверки без расчета градиента
        with torch.no_grad():
            valid_loss = valid_epoch(model, valid_loader)

        if valid_loss.avg < best_loss:
            best_loss = valid_loss.avg
            torch.save(model.state_dict(), "best.pt")
            print("Saved Best Model!")

        # корректируем скорость обучения на основе ошибки проверки
        lr_scheduler.step(valid_loss.avg)
```

[н]

```
source_frames: str = 'data_fall'
source_map: str = 'output19.tif'
source_coor = [83.195558, 54.798793, 83.214901, 54.786596,]
scale = 10
```

```
step = 10


with open('autumn_features.npy', 'rb') as f:
    satellite_features_map = np.load(f)
satellite_features_map = torch.from_numpy(satellite_features_map)


dron_position = DronePos((2500, 3100), satellite_features_map)
y, x = dron_position.y_x
```

[о]

```
def to_orto(pos):
    frame = pos['image']
#     pos['yaw'] = 0


    frame = undistortion(frame, crop=False)
    HoM = cam_to_iner_homo_matrix(pos)
    dst_size, THoM = get_dst_size_HoM(frame, HoM)
    tframe = cv2.warpPerspective(frame, THoM, dst_size)
    return tframe
```

[п]

```
# Получаем список файлов в директории исходного фрейма и сортируем их по имени файла
file_list = os.listdir(source_frames)
sorted_files = sorted(file_list, key=lambda x: int(x[:-4]) if x[:-4].isdigit() else float('inf'))
line_coor = []


# выбираем конкретный путь к файлу изображения для обработки
file_path = os.path.join(source_frames, '1698794738496807349.png')
im = Image.open(file_path)


# Разбираем метаданные файла изображения
metadata = ast.literal_eval(im.info['metadata'])
attitude = metadata['PixHawk_data']['ATTITUDE']
pitch = attitude['pitch']
roll = attitude['roll']
global_position_int = metadata['PixHawk_data']['GLOBAL_POSITION_INT']
relative_alt = global_position_int['relative_alt']/1000
lat = global_position_int['lat']/10**7
lon = global_position_int['lon']/10**7
compass = metadata['PixHawk_data']['VFR_HUD']['heading']


# Конвертируем изображение в массив NumPy
drone = np.array(im)
```

```
        pos = {
            'image': drone,
            'pitch': pitch,
            'roll': roll,
            'yaw': -compass,
        }

        # Выполняем орфографическое преобразование изображения
        drone = to_orto(pos)
        #Обрезаем изображение ортогональной проекции
        drone = drone[180:420, 80:320]

        img = cv2.imread(source_map)
        # Конвертируем изображение карты в формат RGB
        img = cv2.cvtColor(np.array(img), cv2.COLOR_BGR2RGB)
        # Создаём копию карты для последующей обрезки
        img_for_crops = img.copy()
```

[p]

```
        for scale in range(98):
            print(scale)
            # Рассчитываем смещение обрезки при текущем масштабе
            bias = int((crop.shape[0] / 2) * (scale / 100))
            # Обрезает ортогональное изображение при текущем масштабе.
            inp = drone[0 + bias: drone.shape[0] - bias, 0 + bias: drone.shape[1] - bias]
            #Вызов основной функции для обработки обрезанного изображения
            _, hm = main(inp, dron_position)

            # Получаем позицию пикселя результата предсказания
            size = 224
            _, y_x = get_max_idxs(hm)
            pred_y, pred_x = get_pixel(y_x)
            pred_x_map, pred_y_map = pred_x, pred_y
            # Выполнение граничной обработки результатов прогнозирования
            if pred_x_map < size:
                pred_x_map = size
            if pred_y_map < size:
                pred_y_map = size
            if pred_x_map > img.shape[1]:
                pred_x_map = img.shape[1] - size
            if pred_y_map > img.shape[0]:
                pred_y_map = img.shape[0] - size
             # Извлечение фрагментов изображения вокруг предсказанного местоположения с
карты
             crop_pred = img_for_crops[pred_y_map - size: pred_y_map + size, pred_x_map - size:
pred_x_map + size]

            # Отображение обработанного изображения и результатов прогнозирования
```

```
    fig, axs = plt.subplots(1, 3)
    axs[0].imshow(inp)
    axs[1].imshow(hm)
    axs[2].imshow(crop_pred)
    fig.set_size_inches(15, 10)
    plt.show()
```

[c]
```
class DronePos:
    # Инициализируем положение дрона и карту объектов
    def __init__(self, init_coord, features_map):
        self.y_x = init_coord  # Координаты пикселей
        self.sat_features_map = features_map  # Карта функций
        self.area = None
        self.update_area()

    def update_area(self):
        y, x = self.y_x
        self.area = self.sat_features_map

    def update_pos(self, y_x):
        dy, dx = y_x
        new_y_x = (self.y_x[0] + dy - self.a_shape[0], self.y_x[1] + dx - self.a_shape[1])
        # Обновляем положение дрона
        self.y_x = new_y_x
        self.update_area()

    def get_area(self):
        return self.area
```

[т]
```
#Предварительная обработка изображения
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )
])

def pre_proc(img):
    return transform(img)
```

```
device = 'cpu'
model = CLIPModel().to(device)

# Загружаем веса модели
model_path = f'best_autumn_margin_1.pt'
model.load_state_dict(torch.load(model_path, map_location=device))
```

```
model.eval()

# Получаем энкодеры для дронов и спутников
d_encoder = model.drone_encoder
s_encoder = model.satellite_encoder

# Удалить модель и оставить только кодировщик
del(model)
```

[у]

```
def get_features_map(model, input_tensor, n=1):
    step = (32 * n)
    #Рассчитываем размер скользящего шага
    n_col = (input_tensor.shape[3]) // step
    n_str = (input_tensor.shape[2]) // step
    if n_str <= 0 or n_col <= 0:
        # Получаем карту объектов и нормализуем ее
        features_map, _ = model(input_tensor)
        features_map = features_map.permute(1, 0).cpu().detach()
        return F.normalize(features_map, p=1, dim=0)
    for i in range(n_str):
        for j in range(n_col):
            # получаем функции окна обрезки и соединяем их
            crop = input_tensor[:, :, i * step:i * step + crop_size, j * step:j * step + crop_size]
            feature, _ = model(crop)
            feature = feature.permute(1, 0)
            feature = feature.cpu().detach()
            if j == 0:
                line = feature
                continue
            line = torch.cat((line, feature), 1)
            del feature

        if i == 0:
            features_map = line[:, None, :]
            continue
        features_map = torch.cat((features_map, line[:, None, :]), 1)
    features_map = features_map.cpu().detach()
    return F.normalize(features_map, p=1, dim=0)
```

[ф]

```
def get_drone_features(drone_img):
    img_shape = drone_img.shape[:2]
    # Вычисляем координаты центральной точки изображения
    up, left = img_shape[0] // 2, img_shape[1] // 2
    # обрезаем входное изображение по центральной точке
    input_crop = drone_img[up:up + crop_size, left:left + crop_size, :]
    # изменяем размер обрезанного изображения до размера, введенного моделью
    input_img = cv2.resize(input_crop, (input_image_size, input_image_size))
```

```
        # Предварительная обработка изображения с измененным размером и преобразование его в тензорный формат, требуемый моделью
        drone_tensor = pre_proc(input_img)
        # Извлечение функций с помощью кодировщика дронов
        _, frame_features = d_encoder(drone_tensor[None, :, :, :].to(device))
        # Стандартизируем извлеченные функции
        frame_features = frame_features.cpu().detach()
        return F.normalize(frame_features)
```

[x]

```
    def get_heat_map(sat_features, drone_features):
        # Устанавливаем порог тепловой карты
        tresh = 100
        # используем операцию свертки для расчета тепловой карты и преобразуем результат в массив numpy
        heat_map = F.conv2d(sat_features, drone_features).cpu().detach().squeeze().numpy()
        # Нулевое среднее значение тепловой карты
        heat_map -= np.mean(heat_map)
        #Unit нормализует тепловую карту
        heat_map /= np.std(heat_map)
        # Устанавливаем значения меньше нуля в тепловой карте на ноль
        heat_map[heat_map < 0] = 0
        # нормализуем тепловую карту до диапазона 0-255 и преобразуем ее в беззнаковый целочисленный тип.
        heat_map = heat_map / heat_map.max() * 255.
        heat_map = heat_map.astype(np.uint8)
        heat_map_c = heat_map.copy()
        #устанавливаем значение пикселя на тепловой карте, которое меньше порога, в качестве порога
        heat_map_c[heat_map_c < tresh] = tresh
        return heat_map_c


    def get_pixel(y_x):
        y, x = 32 * y_x[0], 32 * y_x[1]
        return y, x


    def main(frame, dron_position):
        # Получаем характеристики дрона
        drone_features = get_drone_features(frame)
        # Получаем карту объектов окрестностей
        sat_features = dron_position.get_area()
        # Получаем тепловую карту
        hm = get_heat_map(sat_features, drone_features)
        # Получаем координаты самого большого пикселя на тепловой карте
        hm, y_x = get_max_idxs(hm)
        # Преобразуем координаты самой большой точки пикселя на тепловой карте в координаты пикселей
        dy, dx = get_pixel(y_x)
        #Рассчитываем относительное смещение
```

```
            dy_dx = (dy, dx)
            #Последнее обновление положения дрона
            dron_position.update_pos(dy_dx)
            return dron_position, hm
```

[ц]

```
        # Читаем данные об объекте-спутнике из файла и сохраняем в переменную
        Satellite_features_map
        with open('autumn_features.npy', 'rb') as f:
            satellite_features_map = np.load(f)
        # Преобразование данных спутниковых объектов в тензоры PyTorch
        satellite_features_map = torch.from_numpy(satellite_features_map)
        # создаем объект экземплярadrone_position класса DronePos и передаем ему начальное значение
        координаты (2500, 3100) и карту объектов спутника.
        dron_position = DronePos((2500, 3100), satellite_features_map)
        #  получаем координаты y и x текущего положения дрона и сохраняем их в переменных y и x
        y, x = dron_position.y_x
```

[ч]

```
        def to_orto(pos):
            frame = pos['image']
            frame = undistortion(frame, crop=False)
            # Рассчитать однородную матрицу преобразования внутренних и внешних параметров на основе позы камеры
            HoM = cam_to_iner_homo_matrix(pos)
        #Выполняем проекционное преобразование изображения в соответствии с матрицей преобразования для получения ортогонального вида
            dst_size, THoM = get_dst_size_HoM(frame, HoM)
            tframe = cv2.warpPerspective(frame, THoM, dst_size)
            return tframe
```

[ш]

```
        def draw_line_on_map(source_map, source_coor, line_coor, drone, compass, dron_position):
            # Читаем изображение карты
            img = cv2.imread(source_map)
            img_for_crops = img.copy()
            # Получаем координаты верхнего левого и нижнего правого углов карты.
            left, top, right, bottom = map(float, source_coor)
            # Сопоставляем координаты маршрута с картой
            line_x = [(coor[0] - left) / (right - left) * img.shape[1] for coor in line_coor]
            line_y = [(coor[1] - top) / (bottom - top) * img.shape[0] for coor in line_coor]
            line_pts = [(int(x), int(y)) for x, y in zip(line_x, line_y)]
        # Пиксельные координаты на маршруте

            # получаем изображение дрона и поворачиваем его в зависимости от направления компаса
            crop = drone
            height, width = crop.shape[:2]
```

```
        rotation_matrix = cv2.getRotationMatrix2D((width / 2, height / 2), -compass, 1.0)
        crop = cv2.warpAffine(crop, rotation_matrix, (width, height), flags=cv2.INTER_LINEAR)

        # Обрезаем изображение дрона и сохраняем область интереса
        size = 56
        bias = 0
        center_x = crop.shape[1] // 2 + bias
        center_y = crop.shape[0] // 2 + bias
        left = center_x - size // 2
        right = center_x + size // 2
        top = center_y - size // 2
        bottom = center_y + size // 2
        crop = crop[top:bottom, left:right]

        # выполняем логический вывод по обрезанному изображению и получаем результат прогнозирования
        dron_position, hm = main(crop, dron_position)
        y, x = dron_position.y_x

        # Получаем пиксельные координаты результата прогноза
        _, y_x = get_max_idxs(hm)
        pred_y, pred_x = get_pixel(y_x)

        # Рисуем изображение местоположения дрона и прогнозируемое местоположение на карте
        img = cv2.resize(img, (img.shape[1] // scale, img.shape[0] // scale))
            img = cv2.putText(img, f'x: {pred_x}, y: {pred_y}', (50, 100), cv2.FONT_HERSHEY_SIMPLEX, 2, (255, 0, 0), 2, cv2.LINE_AA)
        cv2.imwrite(os.path.join(f'Map_with_route.png'), img)

        return img, dron_position
```

Выпускная квалификационная работа выполнена мной самостоятельно и с соблюдением правил профессиональной этики. Все использованные в работе материалы и заимствованные принципиальные положения (концепции) из опубликованной научной литературы и других источников имеют ссылки на них. Я несу ответственность за приведенные данные и сделанные выводы.

Я ознакомлен с программой государственной итоговой аттестации, согласно которой обнаружение плагиата, фальсификации данных и ложного цитирования является основанием для не допуска к защите выпускной квалификационной работы и выставления оценки «неудовлетворительно».

_____Ло Цзюнь_____ _____

ФИО студента *Подпись студента*

« ____ »_____20 __г.