Инструкция по эксплуатации ML модели распознающей границы зданий

Термины, используемые в инструкции

Инференс - запрос к модели для получения результатов.

Дообучение модели - процедура обновления весов модели машинного обучения для повышения качества инференса.

Докер - программное обеспечение для автоматизации развёртывания и управления приложениями в средах с поддержкой контейнеризации.

GeoJSON - открытый формат, предназначенный для хранения географических структур данных и дополнительных свойств.

Тайл - единичный квадратный кроп части ортофотоплана используемый для обучения и инференса.

Shape-файл - векторный формат для хранения объектов, описываемых геометрией и сопутствующими атрибутами.

ML сервис - разрабатываемый программный продукт - сервис распознающей границы зданий по изображениям.

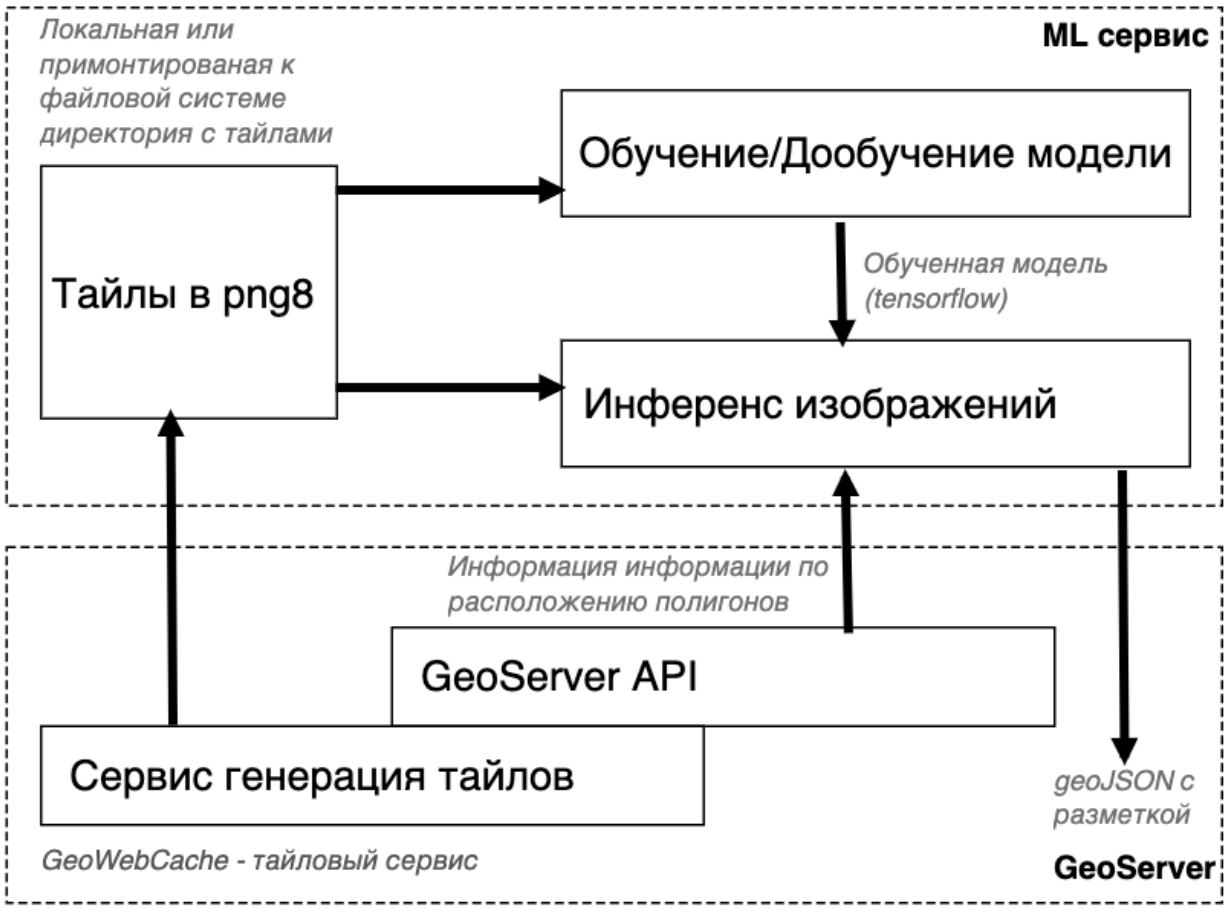
ML модель - файл keras модели (tensorflow background) с подобранными весами

Аугментация - расширение обучающего датасета за счет преобразования исходных изображений.

Инфраструктура сервиса

В основе используемого сервиса находится модель глубокого машинного обучения на сверточных нейронных сетях. Для обучения и инференса модели развернуто вспомогательное окружение в виде сервиса, который принимает на вход RGB изображение, а на выходе генерирует geoJSON файл с разметкой найденных крыш. Более подробная информация по входных данных содержиться в спецификации “Требования к формату входных данных для работы модели машинного обучения распознающей границы зданий”. Модель довольно успешно может использовать обобщение информации полученной на этапе обучения, но при появлении в инференсе новых изображений, похожих на которые не содержалось в обучающей выборке, может снижаться качество работы (разметки), поэтому иногда следует использовать дообучение.

Общая схема взаимодействия ML сервиса и вспомогательного сервиса для пространственной привязки полигонов:



Описание потоков данных между GeoServer и ML сервисом содержится в разделе “Требования к GeoServer для получения дополнительных гео-данных в процессе разметки”.

Требования к формату входных данных

Для работы модели машинного обучения требуется доступ к следующим компонентам:

1. Геосервер с тайлами
2. Папка с загруженными тайлами содержащими ортофотопланы

Требования к ортофотопланам

Характеристики ортофотоплана:

- Необходимое разрешение ортофотоплана: 5-10 см/пикс.
- Ортофотоплан должен иметь ровный тон, нормальную контрастность и хорошую проработку деталей как в освещенных, так и затененных местах;
- Не допускается использование снимков, имеющих сдвиг (рекомендуется использовать выдержку не менее 1:1000) или со сбитой фокусировкой;
- На ортофотоплане по возможности не должно быть мест, закрытых облачностью или туманом. Рекомендуется, чтобы все снимки были сделаны в теплое время года (отсутствие снега);
- Ортофотоплан должен иметь минимальное количество искажений геометрии на всей охватываемой площади (в том числе и на краях);
- Все здания отображенные на ортофотоплане должны иметь как можно более четко различимые края и должны быть отражены в ортогональной проекции. Рекомендуется избегать перспективный вид зданий;

Образцы входных данных:

Ортофотоплан (входные данные)	

Образец не подходящего тайла. Нарушение ортофото.	

Требования к структуре и формату передаваемых данных

Технические характеристики ортофотоплана и карты высот:

- Размер тайлов: 2048x2048 пикселей.
- Размер каждого файла должен быть не более 10 Мб.
- Тайлы для ортофотоплана и карты высот должны быть в графическом формате “.png8” и должны иметь уникальные названия, по которым можно с помощью ПО “GeoServer” однозначно привязать найденные полигоны с реальными географическим координатами.
- Все файлы должны находиться в одной папке.

Примеры выходных данных работы модели

Выходными данными сервиса являются geoJSON файл (схема данных описана в разделе “Процедура запуска инференса”).

После преобразования geoJSON в shapefile и импорта в ПО GeoServer, найденные в процедуре инференса маски должны совпадать с географическими объектами:

Пример наложения найденных границ зданий на географическую подложку (выходные данные)

Требования к серверу

Общие аппаратные требования, предъявляемые к конфигурации серверного оборудования, зависят от типа выполняемых задач: инференса или обучения.

Серверное оборудование / тип задач	Минимальные	Стандартные	Рекомендуемые
Запуск инференса			
Количество вычислительных потоков процессоров (шт.)	4	4	8
Тактовая частота процессора (Ггц)	2,4	2,4	3,5
Оперативная память (Гб)	16	16	16

Свободное дисковое пространство	128Гб	1Тб	1Тб (ssd)
Оперативная память GPU (Гб)	0	4	8
Необходимое количество ядер GPU	0	1	1
Используемое ядро GPU (NVIDIA)	-	GTX 1060	GTX 2080
Пропускная способность локальной сети Мбит/сек	1Мб/сек	10Мб/сек	100Мб/сек
Запуск обучения/дообучения моделей			
Количество вычислительных потоков процессоров (шт.)	8	24	24
Тактовая частота процессора (Ггц)	2,4	3,5	3,5
Оперативная память (Гб)	32	64	128
Свободное дисковое пространство	1Tb	4Tb	4Tb (SSD)
Необходимое количество ядер GPU	1	1	1
Оперативная память GPU (Гб)	8	11	32
Используемое ядро GPU (NVIDIA)	GTX 1070	GTX 2080 Ti	Tesla V100

Пропускная способность локальной сети Мбит/сек	-	100Мб/сек	1Гб/сек

Указанные в таблице параметры (размер дискового пространства, объем оперативной памяти) могут отличаться в случае специфических задач и зависят от количества тайлов и площади распознаваемых территорий.

Требования к операционной системе

Учитывая специфику настройки и работы ML стека для развертывания системы рекомендуется использовать операционную систему Ubuntu Server 18.04.x (64-bit). Допускается использование других версий Ubuntu, например, 16.04, а также (так как Ubuntu является представителем семейства OS Debian) серверов с Debian, но в таком случае порядок установки зависимостей будет отличаться (особенно в части работы с GPU и библиотекой tensorflow).

Установка модели

Настройка сервера

Доступно вызова инференса несколькими способами:

- Запуск распознавания из **Docker** контейнера (контейнер содержит в себе все необходимые зависимости, поэтому дополнительно устанавливать ничего не требуется).
- Скачивание проекта из git-репозитория и первичная настройка сервера.

Важно: в процессе установки библиотек происходит их скачивание из репозиториев linux, python и conda. Необходимо наличие доступа к указанным ресурсам.

Создание пользователя

```
> sudo useradd <username> -m -d /home/<username> -s /bin/bash
```

Установка пакета nvidia-cuda-toolkit

Выполним команду:

```
> sudo apt install nvidia-cuda-toolkit
```

Узнать, установлен ли пакет CUDA и его версию, можно, выполнив команду:

```
> nvcc --version
```

Загрузка и установка Anaconda

Ссылку на актуальную версию Anaconda можно найти на официальном сайте проекта по адресу: **https://www.anaconda.com/distribution/**. Необходимо выбирать дистрибутив, содержащий Python версии 3. При этом не имеет значения, какая версия Python уже установлена в системе, так как Anaconda создает изолированное виртуальное окружение, в котором и будут выполняться скрипты. Пример для установки Anaconda версии 2019.10:

1. При помощи команды wget загрузим установочный пакет.

```
> wget https://repo.anaconda.com/archive/Anaconda3-2019.10-Linux-x86_64.sh
```

2. Запустим установочный скрипт. Все параметры установщика можно оставить по умолчанию.

```
> bash ./Anaconda3-2019.10-Linux-x86_64.sh
```

Установка CUDA

Все действия по установке и настройке conda важно производить от имени одного и того же пользователя.

Предварительно необходимо убедиться, что виртуальная среда conda активна. Индикатором этого является отображение имени виртуальной среды в начале приглашения терминала ко вводу, например:
(base) user@workstantion:~$

В данном случае активна виртуальная среда с именем base.

По умолчанию при входе в систему conda активирует виртуальное окружение **base,** создаваемое при установке conda. Эту опцию можно отключить, добавив в конфигурационный файл **.condarc**, находящийся в домашнем каталоге пользователя, от имени которого была произведена установка conda, следующую строку:
auto_activate_base: false

Виртуальное окружение conda предназначено для изолирования среды python. Таким образом, если для разных проектов требуется разный набор библиотек или их разные версии, мы можем выделить под каждый свое виртуальное окружение, что позволит избежать конфликтов и в дальнейшем легко проводить манипуляции с набором зависимостей для каждого конкретного проекта, не затрагивая остальные. По умолчанию при установке conda создается окружение base. Можно устанавливать все зависимости прямо в него, однако хорошим тоном считается создание нового виртуального окружения для каждого проекта. Будем следовать этому принципу и создадим новое окружение:

Для выхода из текущей виртуальной среды выполним команду:
> conda deactivate

Создадим новое виртуальное окружение:
> conda create -n <имя_среды> python=x.x anaconda,
где вместо x.x указать версию интерпретатора Python. Для работы всех модулей и библиотек необходимо использование Python 3-ей версии (3.6 или 3.7)

Для активации виртуальной среды необходимо выполнить команду:
> conda activate <имя_среды>

Для удаления виртуального окружения используется команда:
> conda env remove -n имя_окружения
При этом предварительно необходимо выйти из окружения, которое удаляется.
1) Наиболее удобным способом установить инструменты поддержки CUDA является загрузка пакетов при помощи conda:

```
> conda install tensorflow-gpu==1.14
```

Подходящие пакеты CUDA и CudNN будут скачаны автоматически. Рекомендуется использовать этот способ. Для обеспечения совместимости с кодом проекта использовать tensorflow версии 1.14.

2) В качестве альтернативного варианта можно использовать установку через pip:

```
> sudo apt install nvidia-cuda-toolkit
> pip install tensorflow_gpu==1.14
```

3) Также возможна установка CUDA при помощи пакетов с официального сайта NVidia. Подробнее см. главу “Приложение”.

Необходимые зависимости для работы модели

Ниже представлен список Open Source решений, необходимых для запуска сервиса. Следует отметить, что некоторые библиотеки, которые необходимы для работы скрипта, устанавливаются как зависимости других библиотек. Перечислим сначала те, которые требуется дополнительно установить внутри виртуального окружения:

```
> conda install -c conda-forge opencv=4.2
> conda install pyproj
> conda install -c conda-forge Imutils
> conda install keras
```

Следующие же библиотеки уже были установлены нами ранее либо явно, либо как зависимости других пакетов; их повторная установка не требуется. В случае необходимости, эти пакеты можно установить командой **`conda install имя_пакета`** внутри виртуального окружения:

```
> tensorflow-gpu==1.14
> numpy
> IPython
> requests
> matplotlib
> scikit-image
```

Установка проекта с ML сервером

Проект с ML сервисом не требует установки, для начала работы достаточно распаковать[1] в выбранную директорию, на которую у текущего пользователя есть права на запуск исполняемых файлов, архив “TFS.ML.RoofDetect.zip”.

После распаковки zip архива “TFS.ML.RoofDetect.zip” , содержащего проект, в корне находятся 4 директории:

- **eval** директория с набором скриптов и библиотек для запуска процедуры инференса изображений. Подробнее в разделе “Запуск распознавание нового облета”.
- **train** директория с набором скриптов и библиотек для запуска процедуры обучения модели. Подробнее в разделе “Обучение модели”.
- **models** - директория с ML моделями для инференса
- **dataset** - директория для хранения разметки и изображений для обучения

Важно отметить, что при распаковке структура папок должна быть сохранена. Для разархивации перейдем в папку с архивом и выполним команду:

> unzip TFS.ML.RoofDetect.zip

[1] Более правильным вариантов установки является загрузка проекта из git репозитория командой
> git clone URL

Запуск распознавание нового облета

Распознавание происходит в несколько этапов:

- Скрипт просматривает указанную директорию на наличие графических файлов (тайлов);
- Модель в каждом слое находит полигоны (обводит крыши по контуру);
- Найденные координаты точек конвертируются в географические координаты. Для преобразования используются запросы к geoServer, который генерировал тайлы;
- Полигоны с соседних тайлов объединяются;
- Все полигоны объединяются в единый geoJSON объект.

Процедура запуска инференса

Запуск распознавания нового облета осуществляется скриптом **python main.py** (находится в директории {BASE}/eval) непосредственно из указанной директории с параметрами[2] (значения параметров указаны в качестве примера):

--convert_to_GPS
*(необязательный, по умолчанию **Pseudo-Mercator (EPSG:3857)**, если 1, то данные конвертируются в **WGS 84 (EPSG:4326)**)*

--export_file=./export/monino_3857_with_guid.json
(необязательный, директория и название файла, куда сохранять сгенерированный geoJSON)

--path_to_images=/home/user/datasets/Blob/MONINO_Monino-Aerial/BIG-EPSG_3857_18/0155_0176/
(обязательный, путь от корня до папки с тайлами)

--use_GPU=-1
(необязательный, для использования GPU для инференса можно передать id gpu, по умолчанию используется вычисления на CPU (-1))

--image_limit=3000
(необязательный, лимит по количеству тайлов для распознавания тайлов)

--layer_name='MONINO:Monino-Aerial'
(обязательный, название слоя подложки в geoserver)

[2] Стандартной практикой для проектов на python является добавление директории с запускаемым скриптом в список PYTHONPATH

`--geoserver_url='http://80.252.146.214:8810'`
(обязательный, доступный урл запущенного геосервера)

`--min_area=15`
(необязательный, минимально допустимая площадь полигона в м^2 *)*

`--max_area=1500`
(обязательный, максимальная площадь допустимая полигона в м^2*)*

`--min_confidence=0.9`
(обязательный, минимально допустимое значение confidence модели)

```
1/30 (00158867_00180305.png8) - 0
2/30 (00158874_00180286.png8) - 0
3/30 (00158843_00180320.png8) - 0
4/30 (00158881_00180294.png8) - 0
5/30 (00158863_00180324.png8) - 0
6/30 (00158867_00180288.png8) - 0
7/30 (00158877_00180294.png8) - 0
8/30 (00158848_00180312.png8) - 0
9/30 (00158862_00180291.png8) - 0
10/30 (00158880_00180308.png8) - 3
11/30 (00158876_00180296.png8) - 1
12/30 (00158880_00180310.png8) - 1
13/30 (00158862_00180293.png8) - 0
14/30 (00158856_00180299.png8) - 0
15/30 (00158863_00180314.png8) - 0
16/30 (00158865_00180307.png8) - 0
17/30 (00158870_00180290.png8) - 0
18/30 (00158869_00180322.png8) - 0
19/30 (00158874_00180318.png8) - 10
20/30 (00158858_00180303.png8) - 4
21/30 (00158880_00180301.png8) - 0
22/30 (00158860_00180313.png8) - 8
23/30 (00158863_00180285.png8) - 0
24/30 (00158843_00180286.png8) - 0
25/30 (00158871_00180292.png8) - 1
26/30 (00158873_00180318.png8) - 10
27/30 (00158875_00180304.png8) - 0
28/30 (00158870_00180318.png8) - 3
29/30 (00158865_00180304.png8) - 0
30/30 (00158865_00180286.png8) - 0
Проанализировано и добавлено 41 полигонов/а
1/1980
2/1980
3/1980
4/1980
5/1980
6/1980
7/1980
8/1980
9/1980
10/1980
11/1980
12/1980
13/1980
14/1980
15/1980
16/1980
17/1980
18/1980
19/1980
```

Интерфейс процесса поиска полигонов и генерирования geoJSON

Учитывая тот факт, что процесс инференса может занимать продолжительное время (например, 30000 тайлов размера 2048x2048 на GPU NVIDIA 1070 занимает порядка 5 часов) удобно запускать процессы инференса и обучения в фоновом режиме (без необходимости держать открытой консоль с ssh подключением). Для этого можно использовать встроенную POSIX команду nohup. При этом важно передавать вывод

сервера в отдельный файл лога для возможности отслеживания состояния процесса. Формат вызова: **nohup** запуск скрипта с параметрами **>** путь к файлу с логами **&.**

Пример:

```
> nohup python main.py --export_file ./test.json --
layer_name="MONINO:Monino_HightMap_1.0"  --convert_to_GPS=0 --
path_to_images="/home/user/Загрузки/MONINO heightmap/BIG-
EPSG_3857_18/0155_0176" --geoserver_url="http://192.168.7.84:8080" --
use_GPU=0 > /home/user/logs/eval_log.txt &
```

Каждый полигон в итоговом geoJSON содержит в себе дополнительную структурированную информацию.
Схема данных:

Поле	**Тип**	**Размер**	**Пример**	**Описание**
guid	character	64	b09f605b9d0cf0e09af31d7fd7b5a3a492f89b110955270a613c4135ecc1ca50	*Внутренний id полигона. Содержит в себе строку с информацией по названию тайла, даты и времени инференса (поиска крыш моделью), координаты первой точки полигона.*
area	real	6	71.22	*Площадь дома в m2*
confidence	real	6	0.9898	*Уверенность модели в прогнозе*
start_tile	character	22	00158861_00180305.png8	*Название файла с тайлом, где был найден полигон*
join_directions	character	5	rul__	*Схема движения для склейки полигона от start_tile до finish_tile.* ***rrul**_ - вправо, вверх, влево.*

finish_tile	character	22	00158861_00180308.png8	*Название файла с тайлом, где был найден последний полигон при склейке*
long_polygon	boolean	1	False	*True - если полигон был длинее 5 тайлов или возникли проблемы при склейке. Для привлечения внимания оператора.*

Дополнительные настройки распознавания

Дополнительные настройки могут быть полезными при снижении качества распознавания на новых облетах. Настройки находятся в файле ***eval/polygone_collector.py***

Граница по которой считать точки полигона граничными, в %
PADDING = 0.03

Минимальная значимая длина линии на границе у разрезанного тайла
MIN_LINE_LENGTH = 0.001

Максимальный радиус до ближайшей точки в граничном полигоне
MAX_RADIUS_LENGTH = 0.05

Размер полигона
TILE_SIZE = 2048

Коэффициент пересчета px в м
TILE_SCALE = 0.03732

Обучение модели

В дистрибутиве уже находится обученная ML модель, которая способна находить крыши объектов капитального строительства. Одной из особенностей использования машинного обучения является тот факт, что модели имеют более высокое качество инференса в том случае, если в обучающей выборке находились похожие изображения. Поэтому важной составляющей проекта является раздел по обучению/дообучению модели на новых объектах, которых не присутствовало при первоначальном обучении. В первую очередь это тайлы с другой цветовой гаммой, временем года, материалами покрытия и размерами крыш.

Необходимое и достаточное количество тайлов/зданий сложно определить заранее (до запуска обучения, подраздел “Запуск процесса обучения), но следует исходить из базовых предположений:

- Чем больше изображений, тем лучше;
- Обычно количество изображений в обучающей выборке должно находиться в пределах от 200 до 100000;
- Чем больше объекты отличаются друг от друга, тем больше нужен объем обучения;
- Изображения, которые находятся в отложенной части для оценки качества модели, в обучении не используются;
- Чем сложнее используется аугментация изображений, тем меньше минимальный порог количества изображений.

Подготовка к обучению происходит в несколько этапов:

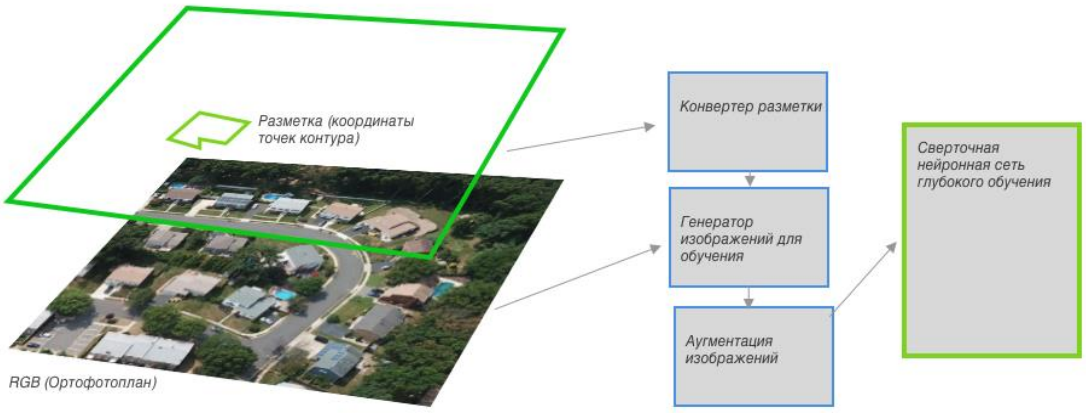
1. Выбор тайлов для обучения с наличием объектов капитального строительства;
2. Разметка тайлов масками, т.е. ломаными линиями по периметру крыши;
3. Преобразование разметки в формат обучения.

Необходимые зависимости для запуска процесса обучения

Для запуска обучения новой модели необходимы все зависимости, используемые для распознавания (см. главу “Установка модели”), а также пакет **scikit-learn**:

```
> conda install scikit-learn
```

Разметка изображений для запуска обучения



Для разметки новых объектов можно использовать любой инструмент с возможностью создания аннотаций полигонами. Общий набор правил для специалиста, выполняющего работы по разметке приведена в **Приложении 2**.

Установка и использование инструмента для разметки (VIA)

В данном проекте рекомендуется использовать opensource решение - VGG Image Annotator (VIA). Для стабильной работы необходима версия 2.0.8, для других мажорных версий и минорных версий возможно потребуется разработка специального конвертера формата разметки.

Пример запуска инструмента разметки для системы MacOS (версии 10.5+):

1. Скачать архив с дистрибутивом via-2.0.8.zip (ссылка), распаковать в выбранную директорию на жестком диске;
2. Внутри директории найти файл **via.html**, в контекстном меню выбрать пункт “Открыть в программе” - “Safari”. Должна открыться страница в браузере с VGG Image Annotator.

Далее в VIA нужно загрузить те тайлы, которые будут использованы для обучения/дообучения модели[3]. Требования, предъявляемые к тайлам для обучения аналогичны требованиям для инференса (см. главу “Требования к формату входных данных”). В обучении могут использоваться только те тайлы, для которых была выполнена разметка. При разметке крайне важно не пропускать крыши, а также не выделять объекты, крышами не являющиеся. Все изображения, которые не содержат крыш, следует пропустить при разметке. Более подробная информация содержится в **Приложении 2**. Допускается использовать изображения в формате jpg или png. Для

[3] В случаях снижения отзывчивости интерфейса программы рекомендуется загружать не более 100 изображений за один раз

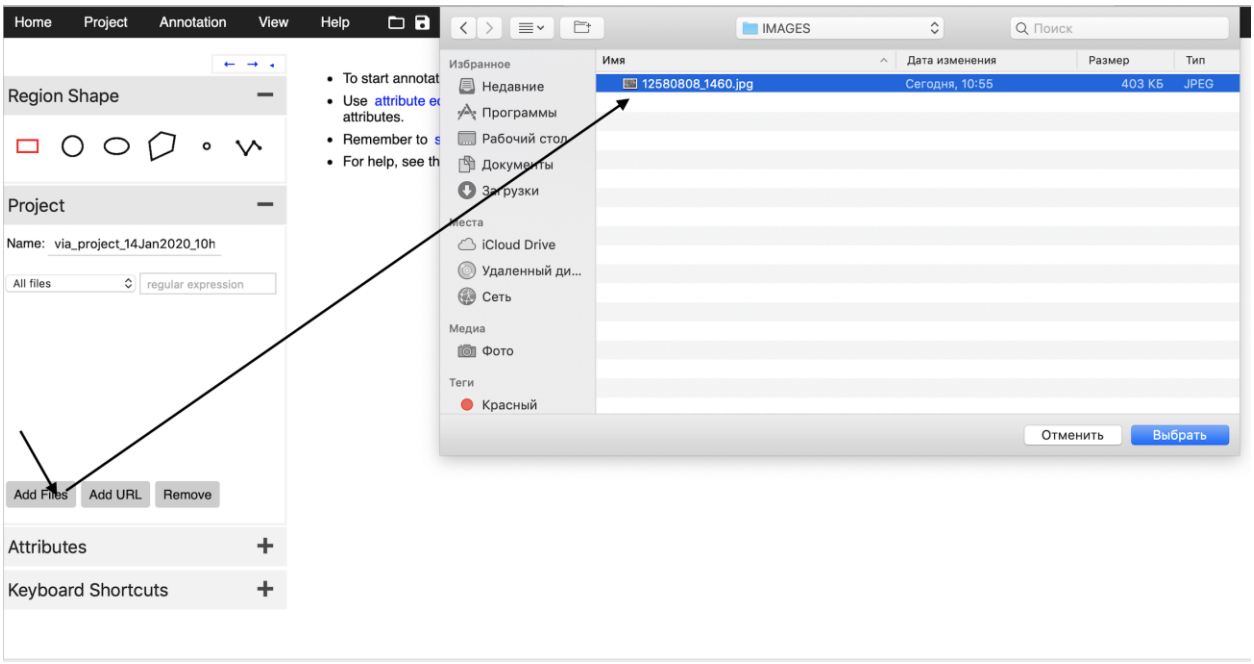
смены расширения группы файлов с png8 на png можно, предварительно перейдя в директорию с изображениями, выполнить команду:

```
> rename 's/.png8$/.png/' *.png8
```

Результатом разметки будет являться выходной json-файл, содержащий сведения о координатах полигонов.

Пример последовательности разметки изображения:

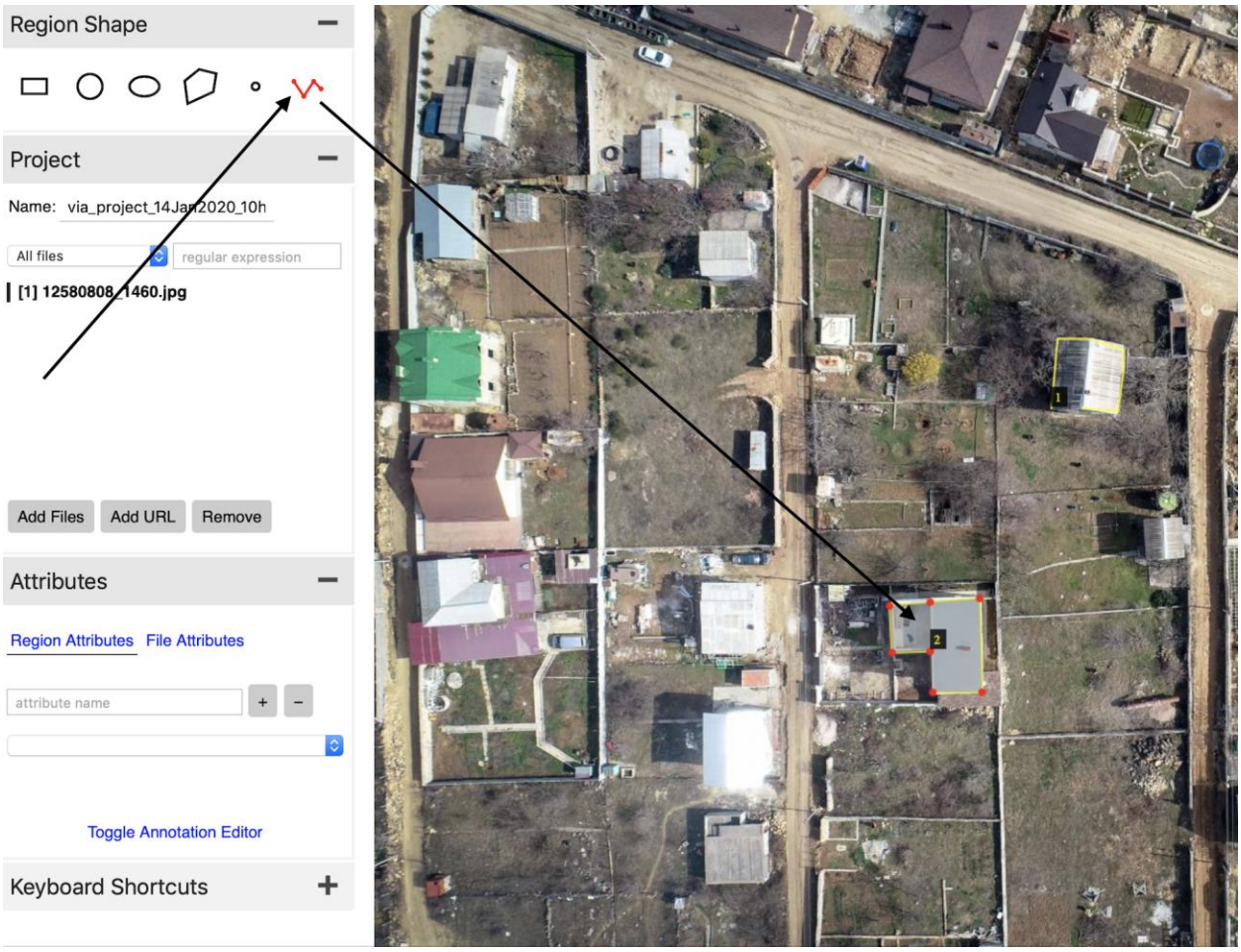
1. Скопировать тайлы на ту рабочую станцию, с который будет производиться разметка оператором;
2. Загрузить все отобранные изображений для обучения в проект в VIA[4].



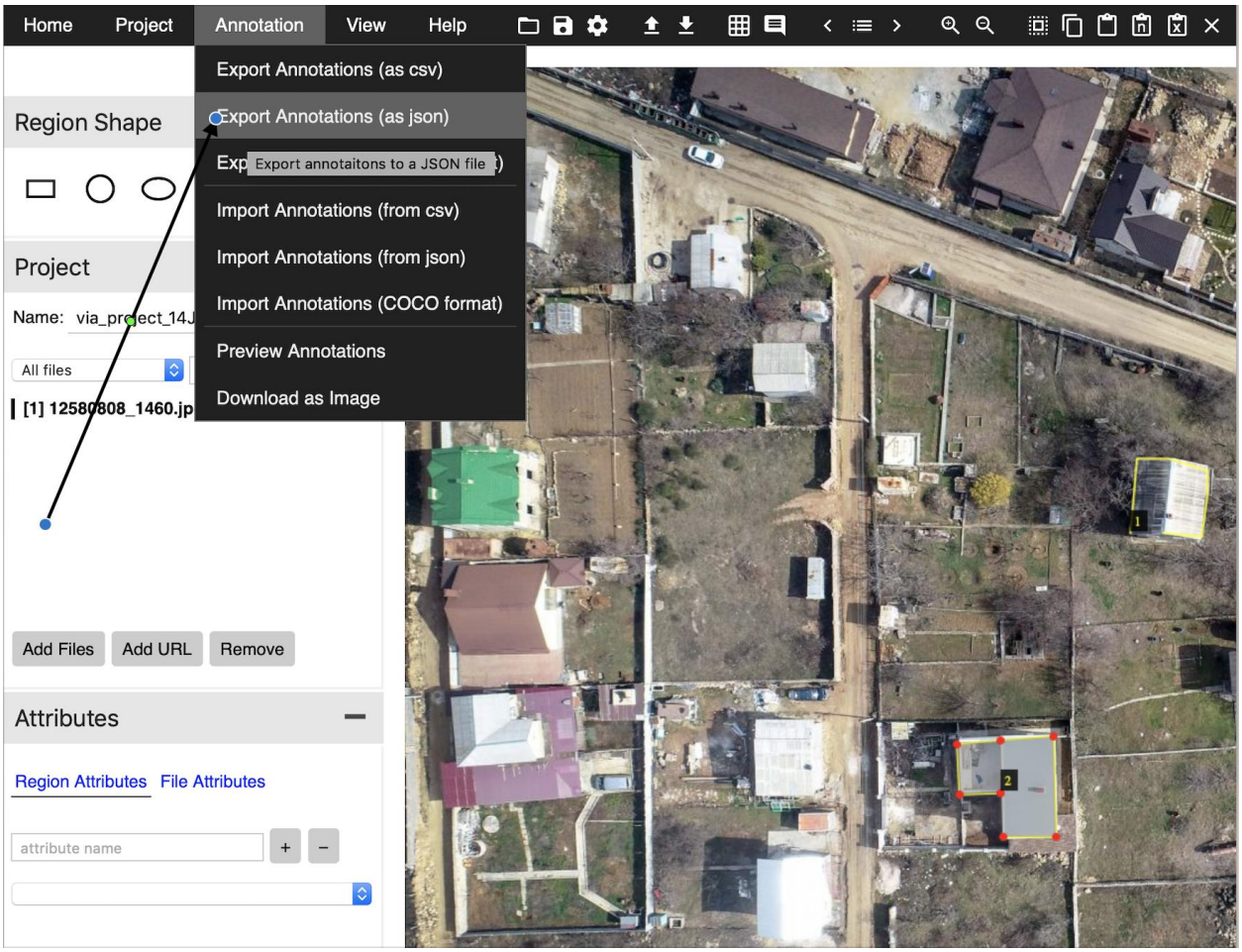
3. Выбрать разметку ломаной линией (Polylyne) и разметить все крыши на всех загруженных изображениях (тайлах)[5].

[4] В файле с разметкой будет указаны конкретные названия файлов изображений, поэтому переименование файлов и расширений запрещено.

[5] Ломаная линия должна представлять собой замкнутый контур (для автоматического замыкания контура нужно нажать клавишу Enter).



4. После окончания разметки выполнить экспорт в формат “Annotations”. При этом выгружается файл json с разметкой, который после преобразования можно использовать для обучения модели.



Добавление разметки новой области облета

Файл разметки полученные через VIA необходимо добавить в список на обучение. Для этого он должен быть размещен в директории **{BASE}/train/marcup** и быть добавлен в файле **{BASE}/train/train_markup.json.**

Файл **train_markup.json** содержит ссылки на все файлы разметки и соответствующие директории, в которых находятся фото, которые будут использованы для обучения/дообучения[6]. Этот файл требуется для того, чтобы модель понимала соответствие между разметкой и изображениями. Необходимо заполнить этот файл в соответствии со структурой приведенной в примере ниже.

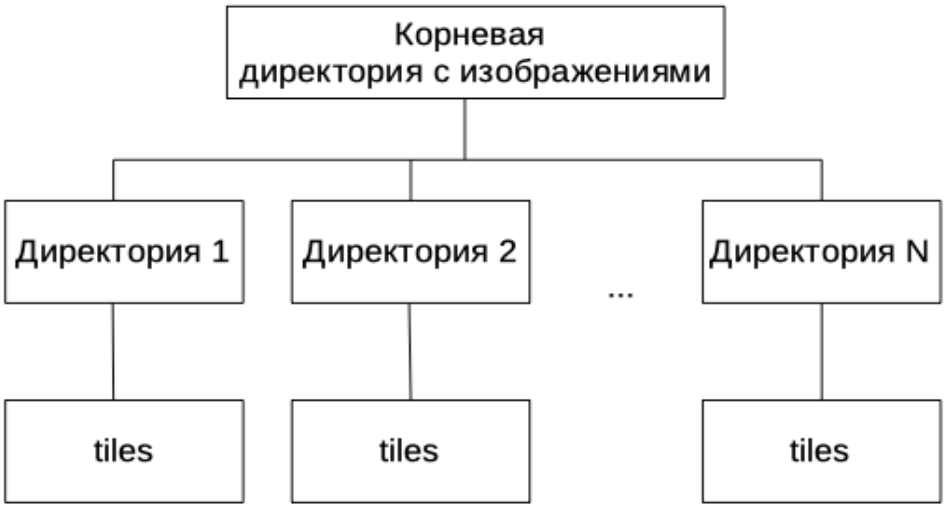
Описание параметров файла **train_markup.json**:

title — имя директории

[6] Важно! Для тех разметок, которые получены иными способами чем VIA и имеющими другой формат структуры данных, их добавление должно осуществляться непосредственно в файле **roof_dataset_RGB.py** в соответствии с Приложением 3. “Преобразование данных по разметке в формат для обучения модели”

tiles — имя тайловой директории (не нуждается в редактировании)
extension — расширение изображений
markup — имя файла разметки для данных, находящегося в папке **{BASE}/train/marcup**

Пример: в качестве базовой директории для изображений используется **/home/user/images.** Указанная директория будет являться корневой для всех изображений, а иерархическая структура будет иметь следующий вид:



То есть, например, если необходимо обучать модель на двух городах: **city1** и **city2**, то тайлы должны находиться в директориях **/home/user/images/city1/tiles** и **/home/user/images/city2/tiles** соответственно. При этом перечень городов указывается в файле **{BASE}/train/train_markup.json**. В рассматриваемом случае он будет иметь следующий вид:

```
{
"cities": [
        {
                "title": "city1",
                "tiles": "tiles",
                "extension": "png8",
                "markup": "city1.json"
        },
        {
                "title": "city2",
                "tiles": "tiles",
                "extension": "png8",
                "markup": "city2.json"
        }
        ]
}
```

Запуск процесса обучения

Процесс обучения проходит в несколько этапов:

1. В самом начале процесса отделяется от основной выборки отложенная часть, на которой впоследствии будет определяться качество работы модели. Такое разделение данных необходимо для того, чтобы получить достоверные сведения о качестве модели, так как возможна ситуация, называемая переобучением, когда модель слишком сильно настраивается под обучающую выборку, показывая высокую точность при обучении, но низкую на новых для себя данных. Размер тестовой части данных, которую необходимо оставить для пост-валидации качества моделей, рекомендуется использовать в объеме 15% от всех данных для обучения. Значение можно установить в свойстве класса **RoofDataset** - **TRAIN_LIMIT**, в данный параметр передается объем обучающей выборки - 85%, что соответствует необходимым 15% тестовых данных;
2. В течение одной эпохи все изображения обрабатываются моделью, при этом изменяются веса нейронной сети;
3. В конце эпохи проводится измерение качества инференса на отложенной части;
4. Подобранные веса в течение одной эпохи сохраняются в новый файл.

Перед запуском обучения важно убедиться, что файл **train_markup.json** заполнен в соответствии с форматом, представленным выше: тайлы не переименовывались после выполнения разметки, в поле “*title”* указано корректное имя директории с файлами, в поле “markup” указано имя json-файла с разметкой, соответствующей этим тайлам, а расширение, указанное в поле “extension” соответствует действительному расширению тайлов.

Запуск распознавания нового облета осуществляется скриптом **python roof_dataset_RGB.py** (находится в директории **{BASE}/train**) непосредственно из указанной директории с параметрами:

```
> python roof_dataset_RGB.py train
```

Параметры запуска (значения параметров указаны в качестве примеров):

--dataset=None
(обязательный, путь к директории с изображениями)

--weights=/home/user/TFS.ML.RoofDetect/models/mask_rcnn_roof-summer-30-_0027.h5
(обязательный, путь к файлу с весами предобученной модели. Для первичного обучения необходимо использовать значение параметра --weights=coco или --weights=imagenet. Для дообучения модели в значение параметера должен быть передан путь к той модели, которую требуется дообучить. Текущая модель, которая

используется для инференса, расположена по пути: **{BASE}/models/common_model.h5 (**о том, как заменить модель, см. главу “Выбор модели для инференса” ниже**))**

`--logs=./logs`
(обязательный, папка для сохранения моделей. После каждой эпохи модель сохраняется на диске в указанной директории (поддиректория для каждого отдельного запуска создается автоматически))

Визуальный контроль процесса обучения

Процесс обновления весов в зависимости от используемой hardware части может занимать десятки часов, поэтому очень важно иметь возможность следить за процессом обучения. Для визуального контроля процесса обучения можно использовать дашбоард tensorboard (входит в пакет Tensorflow). Общая рекомендация для интерпретация результатов - ошибка по каждому критерию качества обучения должна быть минимальной.

Команда для запуска:

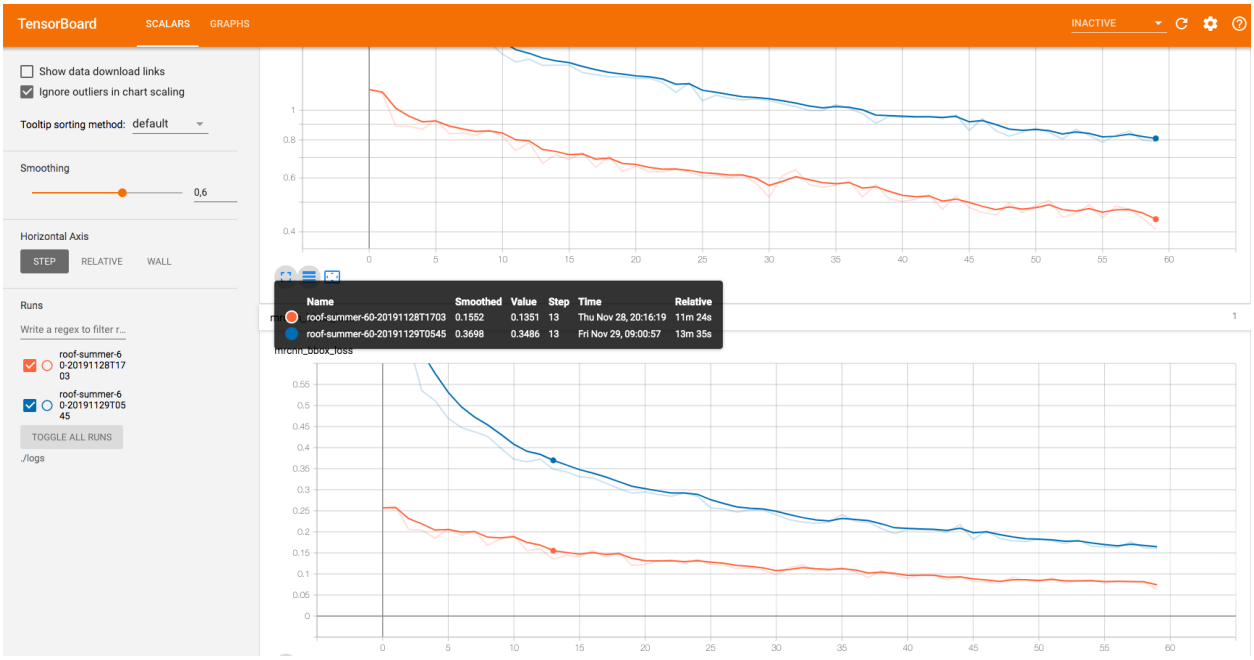
```
> tensorboard
```

Параметры запуска дашборда (значения параметров указаны в качестве примера):

`--logdir ./logs`
(обязательный, папка с сохраненными моделями. Параметр “--logs” который использовался при запуске обучения)

`--host 0.0.0.0`
(необязательный, 0.0.0.0 - добавляет возможность подключения к удаленному серверу)

`--port 8809`
(необязательный, порт для доступа из браузера. По умолчанию - 6006)



Интерфейс процесса обучения модели

Для того, чтобы открыть графический интерфейс Tensorboard необходимо запустить браузер и перейти по адресу: 127.0.0.1:port, где вместо port указать номер порта, переданный в параметре `--port` при запуске Tensorboard (по умолчанию - 6006).

Выбор модели для инференса

Модель, которая будет использоваться для инференса, расположена по пути: **{BASE}/models/common_model.h5**. Для того, чтобы использовать новую модель, необходимо переименовать ее в **common_model.h5** и переместить в директорию[7] **{BASE}/models.**

[7] Важно предусмотреть создание резервной копии старой модели на случай, если новая окажется менее точной.

Установка и настройка GeoServer

Так как внутри тайлов не содержится информации по пространственному расположению полигонов, то для позиционирования используются запросы к API ПО GeoServer - OpenSource сервер для отображения и управления геоданными. Так же необходимость использования обусловлена требованиями модели работать с отдельными тайлами, а не с целым geoTiff изображением.

Возможен вариант использование удаленного геосервера[8], но рекомендуемым способом работы является использование установленного локально (на том же самом физическом сервере) по отношению к ML сервису GeoServer по нескольким причинам:

1. Время инференса и обучения очень сильно зависит от скорости доступа к файлам тайлов. Чтение с локального диска всегда выше, чем по локальной/интернет сети;
2. В момент инференса ML сервис запрашивает гео-информацию о тайлах, поэтому внутренние (localhost) запросы будут всегда выполняться быстрее;

Требования к GeoServer для получения дополнительных гео-данных в процессе разметки

Для стабильной работы необходима версия 2.16.x, для других мажорных версий и минорных версий необходимо проверить доступность следующих endpoint (информационных ресурсов):

- Сервис предоставляющий информацию по отдельному полигону.
    - Пример вызова: HTTP GET */geoserver/gwc/service/tms/1.0.0/{LAYER_NAME}@BIG-EPSG%3A3857@png8/{ZOOM}/{X}/{Y}.png8*
- Сервис предоставляющий информацию по масштабам, названиям слоев и т.д.:
    - Пример вызова: HTTP GET */geoserver/gwc/service/wms?SERVICE=WMS&VERSION=1.1.1&REQUEST=getcapabilities&TILED=true.png8*

[8] При этом необходимо выполнить несколько дополнительных действий (монтирование удаленной директории с тайлами в используемую файловую систему, использование внешнего адреса GeoServer)

Требования к GeoServer для тайлов для обучения и инференса

Установка GeoServer на сервере

Для установки ПО GeoServer необходимо выполнить следующие шаги:

1. Установить Java версии 8. Скачать дистрибутив можно по ссылке.
2. Скачать GeoServer версии 2.16.x. Последнюю версию ПО можно скачать на официальном сайте - ссылка, удобнее использовать “Platform Independent Binary”.
3. Распаковать в выбранную директорию на сервере, например, /usr/share/geoserver.
4. Добавить в переменную окружения путь к директории

> echo "export GEOSERVER_HOME=/usr/share/geoserver" >> ~/.profile

5. Создать группу пользователей геосервера:

```
> sudo groupadd geoserver
```

6. Добавить пользователя, который будет работать с моделью, в группу геосервера:

```
> sudo usermod -a -G geoserver <user_name>
```

7. Установить группу-владельца директории геосервера:

```
> sudo chown -R :geoserver /usr/share/geoserver/
```

8. Добавить права доступа для группы:

```
>  sudo chmod -R g+rwx /usr/share/geoserver/
```

9. Запустить GeoServer

```
> cd geoserver/bin && sh startup.sh
```

10. Проверить работу сервера в браузере по адресу: http://server_ip:8080/geoserver.

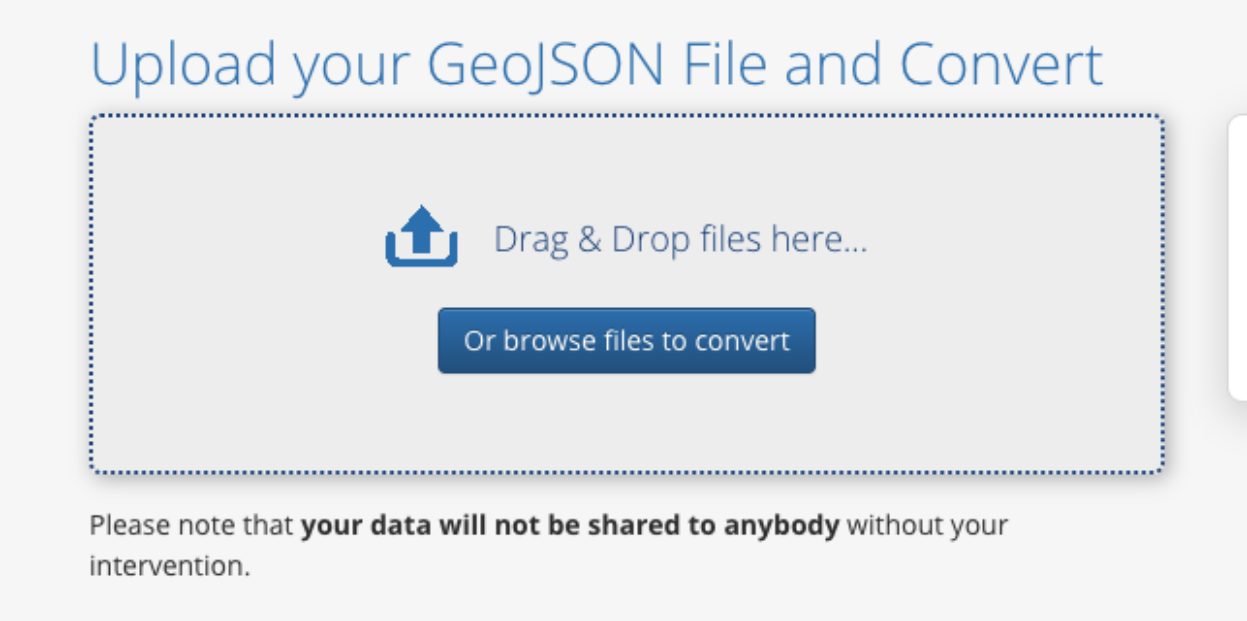
Добавление нового слоя в GeoServer

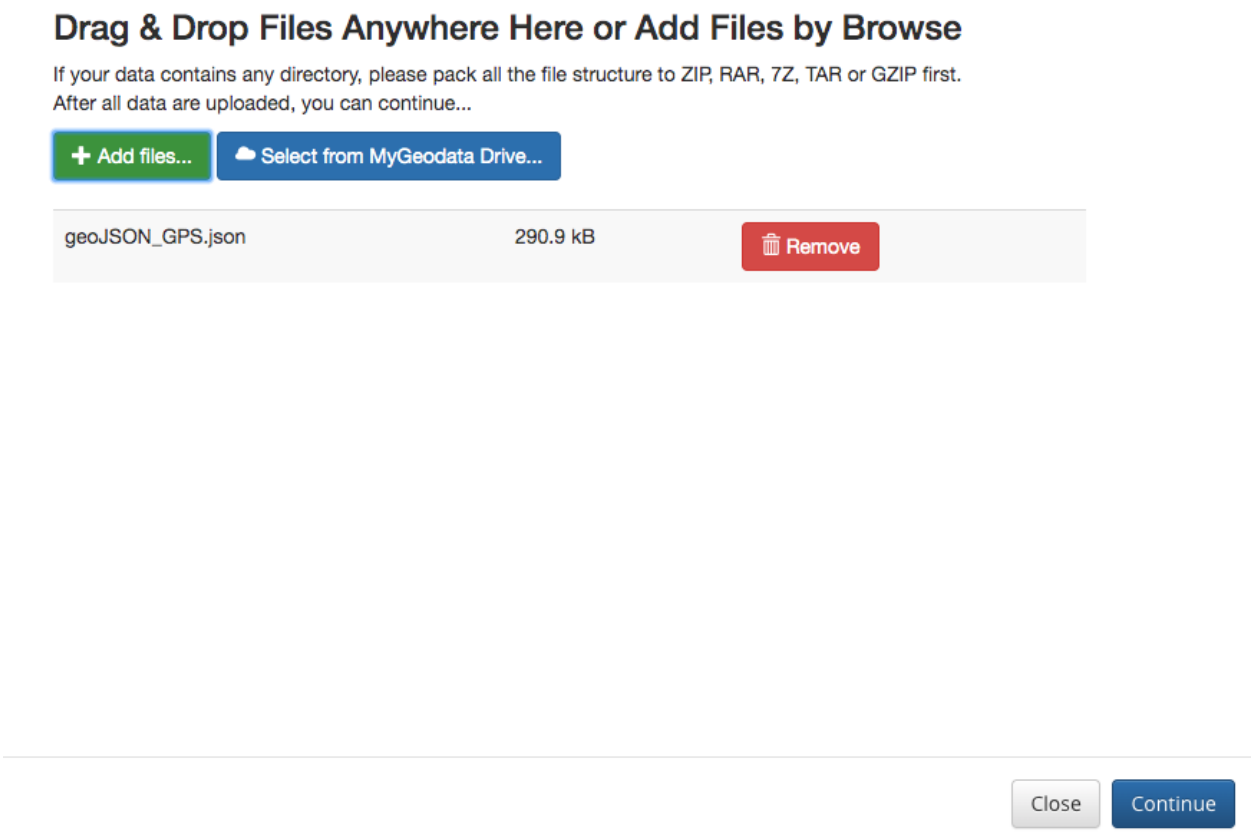
Экспорт geoJSON в shapefile

Для конвертации geoJSON в shapefile (формат geoserver) можно воспользоваться сервисом или любым другим.

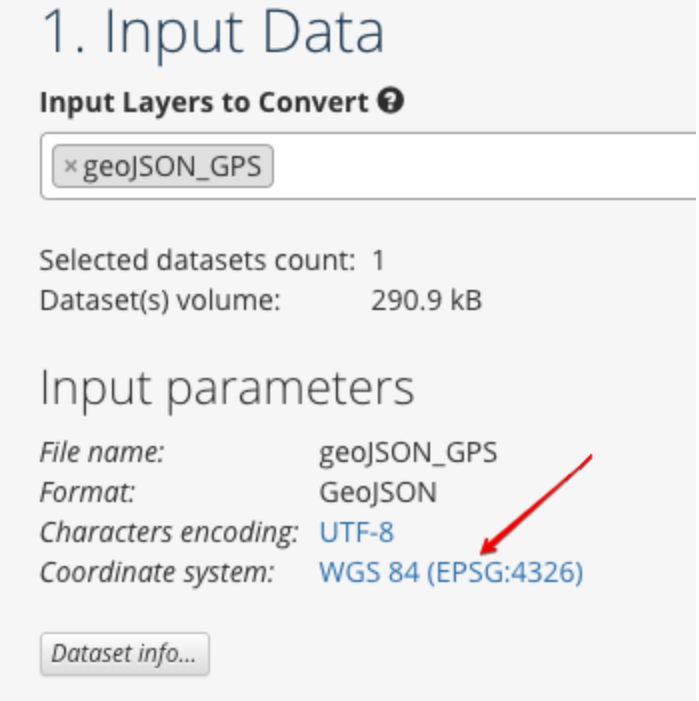
Последовательность этапов выполнения:

- Импортировать geoJSON в сервис;

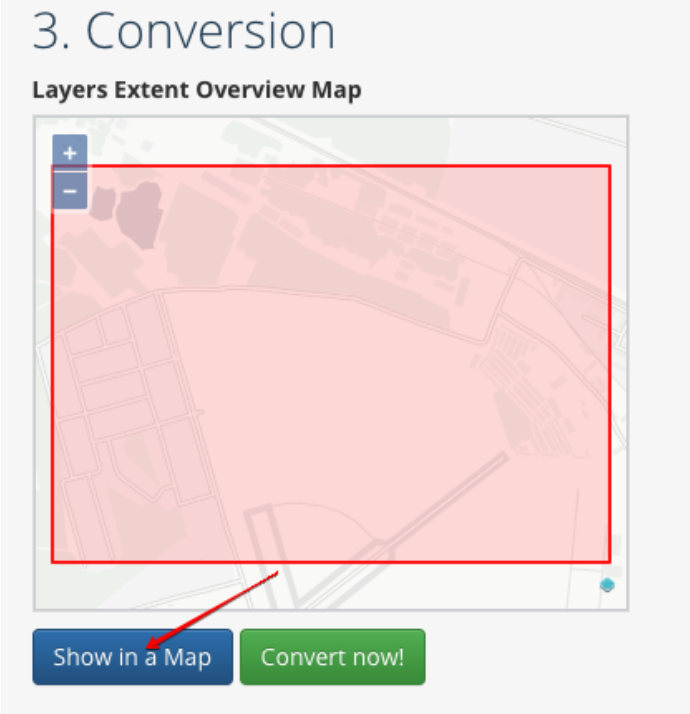


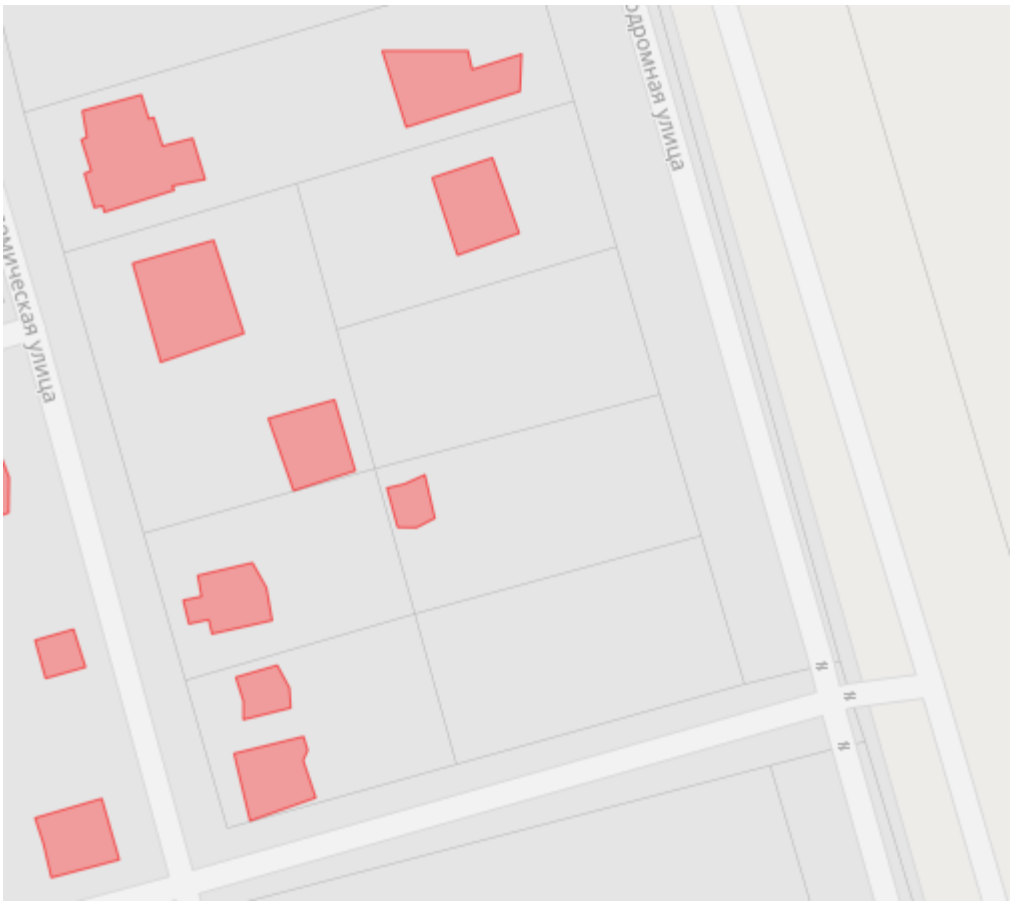


- Нажать **Continue**;
- Изменить “**Coordinate system**” с WGS 84 на **WGS 84 / Pseudo-Mercator (EPSG:3857);**



- Убедиться, что разметка попадает в объекты на карте;

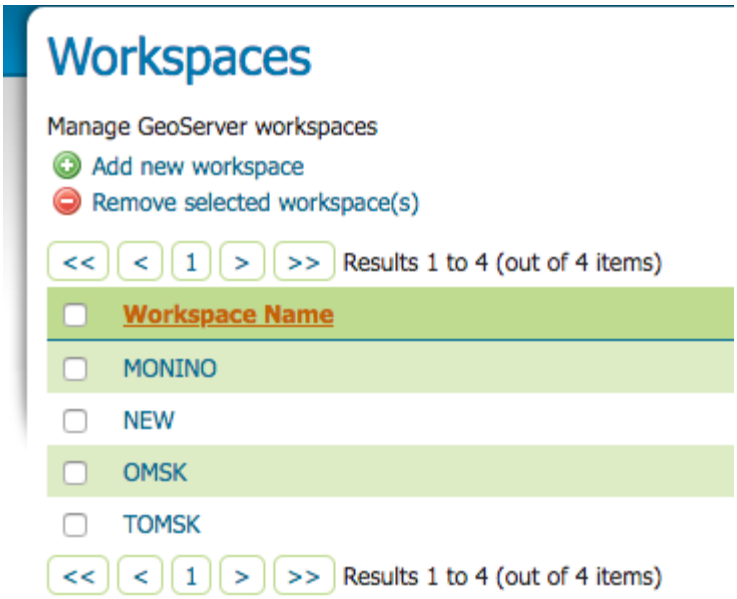




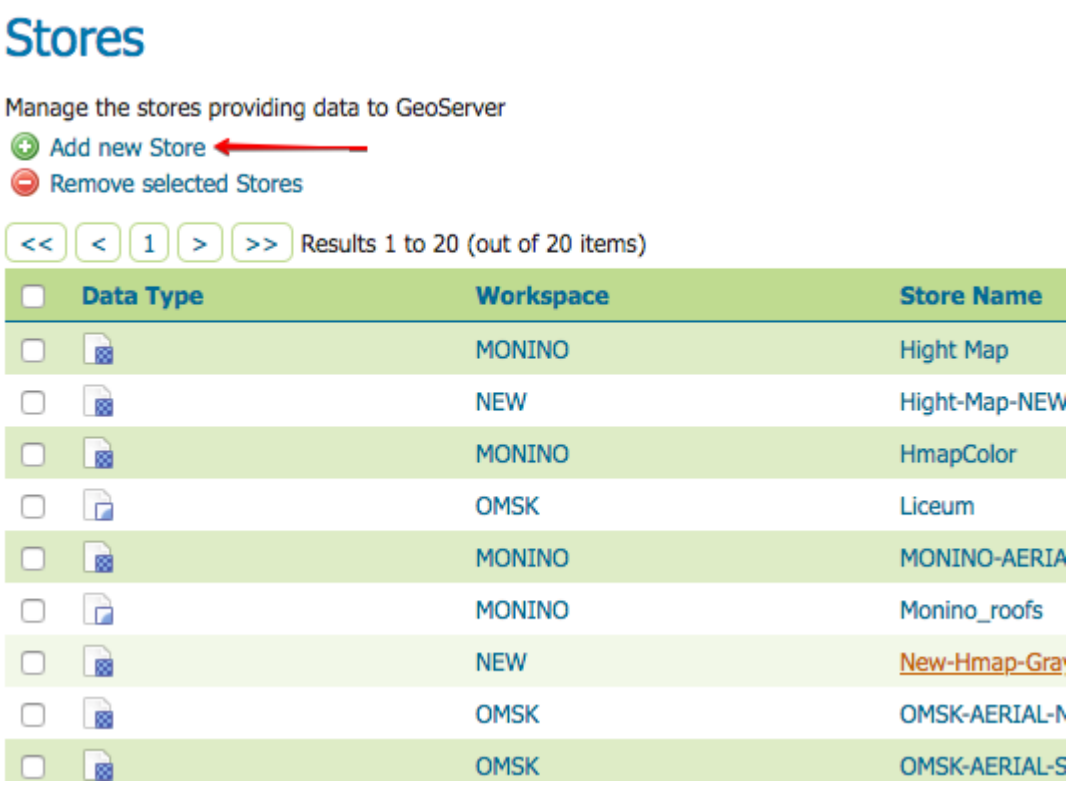
- Запустить конвертацию (Convert now!) и скачать архив с shapefile;

Добавление нового слоя в GeoServer

- Разархивировать shapefile и положить файлы в любое место на сервере где установлен Geoserver;
- Войти в свой аккаунт в Geoserver;
- Проверить, что уже создан workspace под выбранный проект;



- Добавить свой слой (полная инструкция доступна в официальной документации ссылка);
- Создать новое хранилище из своего shapefile и указать любое имя;



New data source

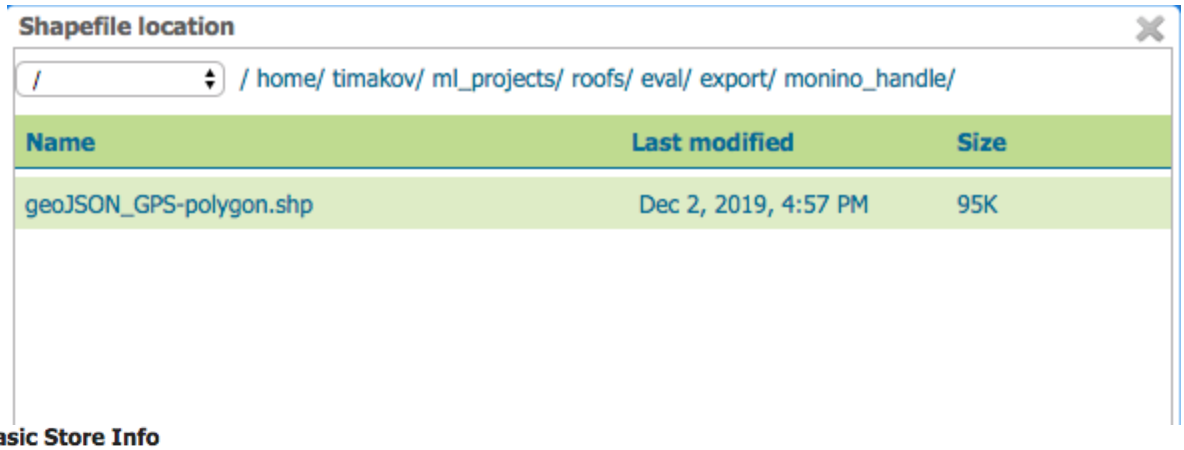
Choose the type of data source you wish to configure

Vector Data Sources

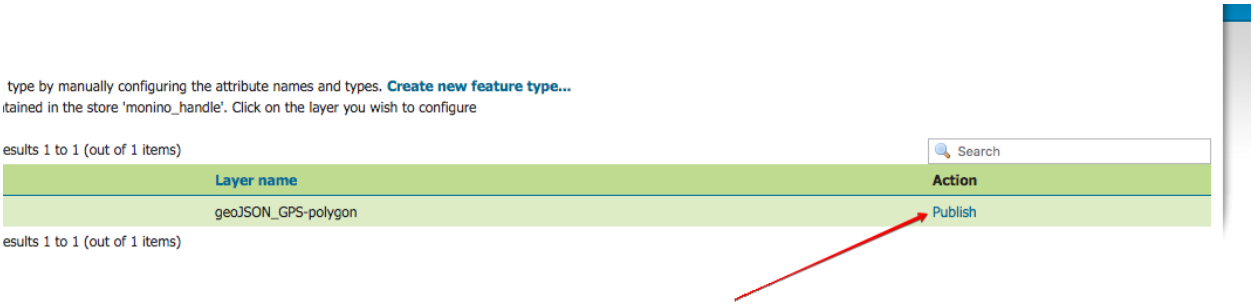
- CSV - Comma delimited text file
- Directory of spatial files (shapefiles) - Takes a directory of shapefiles and exposes it as a data store
- GeoPackage - GeoPackage
- PostGIS - PostGIS Database
- PostGIS (JNDI) - PostGIS Database (JNDI)
- Properties - Allows access to Java Property files containing Feature information
- Shapefile - ESRI(tm) Shapefiles (*.shp)
- Web Feature Server (NG) - Provides access to the Features published a Web Feature Service, and the abil

Raster Data Sources

- ArcGrid - ARC/INFO ASCII GRID Coverage Format
- GeoPackage (mosaic) - GeoPackage mosaic plugin
- GeoTIFF - Tagged Image File Format with Geographic information
- ImageMosaic - Image mosaicking plugin
- ImageMosaicJDBC - Image mosaicking/pyramidal jdbc plugin
- ImagePyramid - Image pyramidal plugin
- WorldImage - A raster file accompanied by a spatial data file



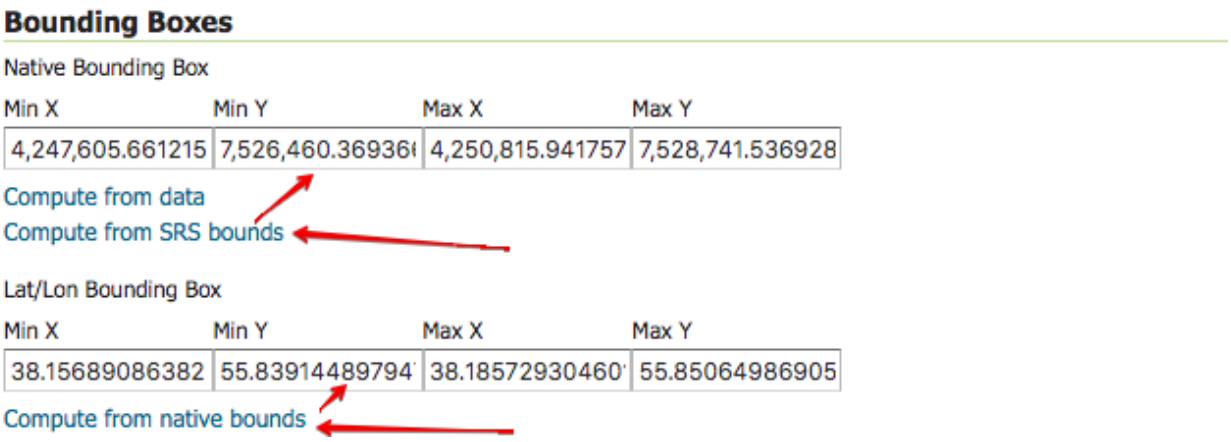
- Опубликовать новый слой кнопкой **Publish**



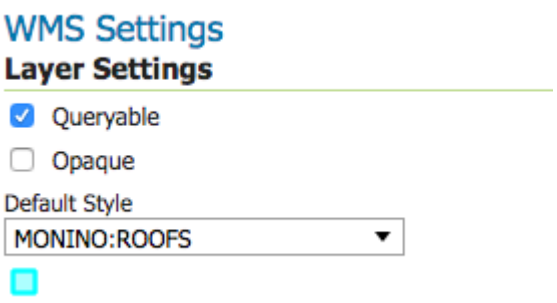
- Указать настройки слоя. Имя[9]:

[9] В именах слоев GeoServer нельзя использовать знак "-"

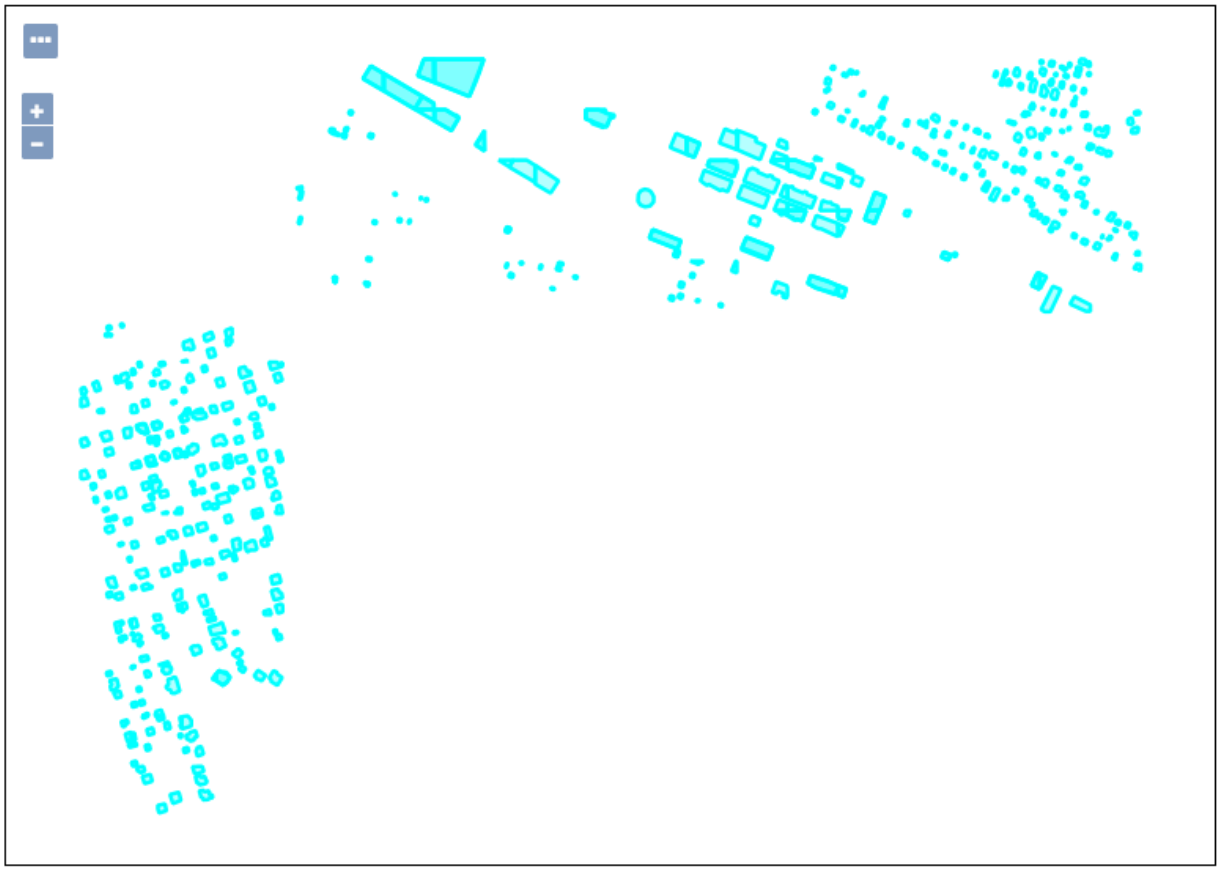
- Параметры границ(можно автоматически указанными кнопками):



- На странице Publishing указать тип(стиль) отображения тайлов (MONINO:ROOFS):



- Проверить новый слой в разделе **Layer Preview**. Должны отображаться полигоны и информация по ним (если кликнуть по любому из них)



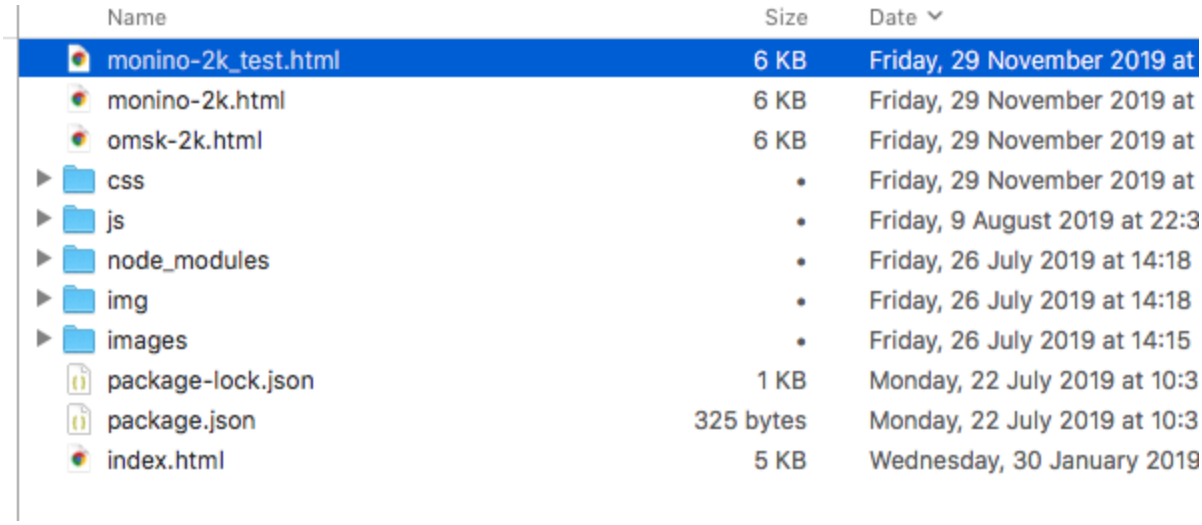
Scale = 1 : 17K 4250881.90258, 7526586.24442

monino_handle

fid	guid	area	confidence	tile_name	nex
monino_handle.309	0c31a7dd75f8f7849f0a1d94b515f6df0c4b204b4f4dea8557cccd0a1a3c8c5c	0.0	1.0	00158870_00180317.png	

Добавление слоя на сайт

- Открыть директорию с файлами сайтов (можно в ftp браузере);
- Открыть существующий сайт (html файлы) (или сделать дубликат в той же папке) в любом html редакторе;



- Добавить JavaScript код:

```
roofs = L.tileLayer('http://80.252.146.214:8810/geoserver/gwc/service/tms/1.0.0/MONINO:monino_3857_fix-polygon@BIG-EPSG%3A3857@png8/{z}/{x}/{y}.png8', {
        crs: L.CRS.EPSG3857,
        tms: true,
        maxZoom: 19,
        minZoom: 15,
        subdomain: 'http://80.252.146.214:8810/geoserver/gwc/service/tms/1.0.0/MONINO:monino_3857_fix-polygon@BIG-EPSG%3A3857@png8',
        maxNativeZoom: 19,
        attribution: ""});
```

- Заменить выделенное название слоя (MONINO:monino_3857_fix-polygon) на нужный (например, **MONINO:monino_handle**). Посмотреть название слоев можно в интерфейсе GeoServer:

- Добавить новый слой в список. Отображаемое название слоя можно выбирать произвольно:

```
var overlays = [
    {
        groupName : "ОртоФото",
        expanded : true,
        layers    : {
            "Совмещенный облет" : lyr,
            "Карта высот" : hmap2,
            "Полигоны": roofs,
            "Разметка handle": roofs_handle,
        }
    },
];
```

- Убедиться, что новый слой отображается в списк справа:

Работа с Docker

В контейнере Docker содержится образ базовой операционной системы, код приложения, библиотеки, от которого оно зависит. Всё это скомпоновано в виде единой сущности, на основе которой можно создать контейнер.

Файл Dockerfile содержит набор инструкций, следуя которым Docker будет собирать образ контейнера. Этот файл содержит описание базового образа, который будет представлять собой исходный слой образа.

Для хранения образа Docker, содержащего требуемые зависимости, дополнительно требуется 8 Гб дискового пространства.

Подготовка к использованию

Установим docker:

```
> sudo apt install docker
> sudo apt install docker.io
```

Для использования видеокарты в среде docker необходимо установить nvidia-docker.

Добавим репозиторий NVidia, последовательно выполнив команды:

```
> curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey | sudo apt-key add -

> distribution=$(. /etc/os-release;echo $ID$VERSION_ID)

> curl -s -L https://nvidia.github.io/nvidia-docker/$distribution/nvidia-docker.list | sudo tee /etc/apt/sources.list.d/nvidia-docker.list

> sudo apt-get update
```

Установим nvidia-docker и перезапустим фоновый процесс:

```
> sudo apt-get install nvidia-docker2

> sudo pkill -SIGHUP dockerd
```

Развертывание готового образа Docker

Для переноса образа между компьютерами, его можно упаковать в архив tar. Для того, чтобы развернуть образ необходимо выполнить команду:

```
> sudo docker load -i <ПУТЬ_К_TAR_ФАЙЛУ>.tar
```

После этого новый образ появится в списке образов, который можно отобразить командой

```
> sudo docker images
```

На основе образа создаются контейнеры, внутри которых происходит выполнение скрипта. Создадим и запустим контейнер на основе образа, например:

```
> docker run -e NVIDIA_VISIBLE_DEVICES=all -e
NVIDIA_DRIVER_CAPABILITIES=all -it -v
ПУТЬ_К_ПАПКЕ_С_ФОТОГРАФИЯМИ:/usr/src/app/data/images
-v ПУТЬ_К_ПАПКЕ_С_JSON:/usr/src/app/data/geojson/
--runtime=nvidia
--rm ИМЯ_ОБРАЗА
--image_limit=5000
--export_file ПУТЬ_К_ФАЙЛУ_JSON
--layer_name ИМЯ_СЛОЯ
--use_GPU 0
--convert_to_GPS 1
--geoserver_url=”http://80.252.146.214:8810”
```

Пояснения к команде:
1. Ключ **-it** означает запуск контейнера в интерактивном режиме. Для запуска контейнера в фоновом режиме используется ключ **-d**.
2. Ключ **-v** предназначен для подключения к файловой системе контейнера директории из основной ОС. Для того, чтобы не копировать каждый раз изображения в контейнер и иметь возможность выгружать файл из контейнера подключаем 2 директории: для тайлов и выходного json файла соответственно.
3. Если параметры скрипта, описанные в данной инструкции выше, не переданы, то используются значения по умолчанию. **ВАЖНО:** параметры скрипта нужно указывать после имени образа. Описание параметров скрипта представлено в главе “Распознавание нового облета”
4. Ключ **--runtime=nvidia** необходим для возможности использования видеокарты в среде docker.
5. Ключ **--rm** предназначен для автоматического удаления контейнера после завершения работы скрипта.
6. Переменная окружения **NVIDIA_VISIBLE_DEVICES=all** определяет доступность видеокарт в среде docker, переменная **NVIDIA_DRIVER_CAPABILITIES** определяет возможности, доступные видеокарте.

Создание образа на основе Dockerfile

Dockerfile предназначен для описания команд, выполняющихся при создании образа. В нем описываются зависимости, необходимые для работы приложения, при

необходимости изменяется структура внутренней файловой системы контейнера, определяются переменные окружения, точка входа в контейнер.

1) Создание образа:
> sudo docker build -t <image_name> <dockfile_directory>

image_name - имя, которое будет присвоено образу
dockfile_directory - директория, в которой расположен dockerfile

Справочник по основным командам Docker

Создание билда:

> sudo docker build -t <container_name> <dockfile_directory>

Интерактивный запуск контейнера:

> sudo docker run -it <container_name>

Смонтировать /tmp с главной машины как /root внутри image:

> docker run -itv /tmp:/root <image_name>

Информация о контейнере:

> docker inspect <container-id>

Сохранить изменения, внесенные в контейнер:

> docker commit <container-id>

Сохранить контейнер в tar-файл:

> docker save -o <path for generated tar file>.tar <image name>

Загрузить контейнер из tar-файла:

> docker load -i <path to image tar file>

Запуск того же контейнера во втором терминале:

> docker exec -it <container-id> bash

Инструкции для Dockerfile:

Установка пакета pip:

RUN pip install <packet-name>

Запуск bash при старте контейнера:

```
ENTRYPOINT ["/bin/bash"]
```

Запуск Python скрипта при старте контейнера:

```
ENTRYPOINT ["python", "/path/to/script.py"]
```

Установка рабочей директории:

```
WORKDIR /usr/src/app
```

Копирование файла в контейнер:

```
COPY <source_file> <destination_file>
```

Приложение 1. Установка CUDA с официального сайта NVidia

Скачиваем CUDA, подходящую для видеокарты с официального сайта Nvidia:

https://developer.nvidia.com/cuda-10.1-download-archive-base, например 10.1 для 2080ti: **https://developer.nvidia.com/compute/cuda/10.1/Prod/local_installers/cuda-repo-ubuntu1804-10-1-local-10.1.105-418.39_1.0-1_amd64.deb**

Устанавливаем скачанный пакет:

```
> sudo dpkg -i cuda-repo-ubuntu1804-10-1-local-10.1.105-418.39_1.0-1_amd64.deb
> sudo apt-key add /var/cuda-repo-<version>/7fa2af80.pub
> sudo apt-get update
> sudo apt-get install cuda
```

Добавляем CUDA в PATH:

```
> export PATH=/usr/local/cuda-10.1/bin${PATH:+:${PATH}}
> export LD_LIBRARY_PATH=/usr/local/cuda-10.1/lib64\${LD_LIBRARY_PATH:+:${LD_LIBRARY_PATH}}
```

Скачиваем CudNN:

https://developer.nvidia.com/compute/machine-learning/cudnn/secure/7.6.5.32/Production/10.1_20191031/cudnn-10.1-linux-x64-v7.6.5.32.tgz

Разархивируем:

```
> tar -xzvf cudnn-10.1-linux-x64-v7.6.5.32.tgz
```

Копируем файлы в директорию с CUDA:

```
> sudo cp cuda/include/cudnn.h /usr/local/cuda/include
> sudo cp cuda/lib64/libcudnn* /usr/local/cuda/lib64
```

Меняем права, чтобы файлы библиотеки были доступны всем пользователям:

```
> sudo chmod a+r /usr/local/cuda/include/cudnn.h /usr/local/cuda/lib64/libcudnn*
```

Теперь добавляем пути в bashrc:

```
>export
LD_LIBRARY_PATH="$LD_LIBRARY_PATH:/usr/local/cuda/lib64:/usr/local/cud
a/extras/CUPTI/lib64"
> export CUDA_HOME=/usr/local/cuda
> export PATH=/usr/local/cuda-10.1/bin${PATH:+:${PATH}}
>               export               LD_LIBRARY_PATH=/usr/local/cuda-
10.1/lib64\${LD_LIBRARY_PATH:+:${LD_LIBRARY_PATH}}

> source ~/.bashrc
> sudo ldconfig
> echo $CUDA_HOME
```

В случае успешной установки вывод должен стать: /usr/local/cuda

Приложение 2. Инструкция по разметке тайлов для новых облетов

Цель проведения разметки

Цель проведения разметки - создание обучающего датасета для создания модели машинного обучения, которая должна будет распознавать границы зданий и сооружений.

Постановка задачи

На каждом снимке необходимо выделить многоугольниками границы зданий и сооружений.

Пример разметки крыш

Разметка требуется для объектов следующих типов:

- Здания.

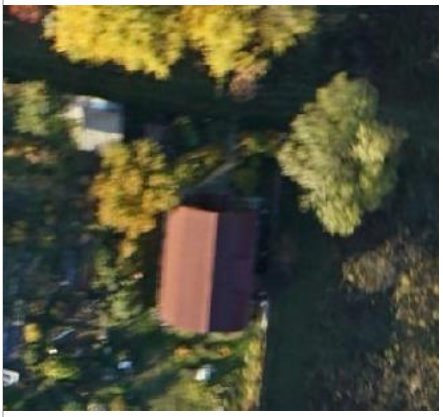
Разметка **не** требуется для объектов следующих типов:

- Здания менее 36м2 (меньше чем 5х5 м2). Для ориентира можно использовать легковой автомобиль в кадре, его длина примерно 4,5-5 метров;
- Небольшая будки охраны;
- Колодцы;
- Теплицы;

- Навесы для автомобилей;
- Веранды;
- Террасы;
- Бытовки;
- Теннисные корты;
- Недостроенные здания;
- Грядки;
- Объекты на границе слайда, которые видны менее чем на 30%.

Отбраковка снимков

В каких случая снимок необходимо пропускать в процессе разметки, чтобы в дальнейшем это не приводило к ухудшению качества работы обученной модели:

Описание	Пример тайла с нарушением
Снимок размыт, смазан, не в фокусе или дрожащий.	 

На снимке видны явные нарушения геометрии.	
Здания на изображении отображаются по под большим углом. Корректное изображение выглядит как строго вертикальное.	

Приложение 3. Преобразование данных по разметке в формат для обучения модели

Пример кода для преобразования разметки в нужный формат для обучения модели содержится в файле **{BASE}/train/roof_dataset_RGB.py** в методе **load_roof_from_json** класса **RoofDataset.** Для других инструментов разметки код будет отличаться.

Необходимость преобразования обусловлена унифицированным форматом данных для обучения и заключается в перекладывании полей данных по разметке из формата используемого инструмента аннотирования в формат добавления в генератор обучения:

1. **"Roof"** - класс разметки (в проекте используется только 1 - “roof”);
2. **image_id=image_name** - уникальный идентификатор тайла
3. **path=path** - путь к файлу с изображением;
4. **width=2048** - ширина тайла;
5. **height=2048** - высота тайла;
6. **polygons=polygons** - точки разметки крыши полигоном в формате [{'all_points_x': [], 'all_points_y': []}].