Введение

Для того чтобы представить пользователю результаты работы нашей нейронной сети по определению траектории движения тайфуна, мы используем GUI, в котором для конечного результата отведено специальное пространство (Picture Box). И в качестве результата работы конечный пользователь видит у себя область карты, на которой изображена траектория движения тайфуна. В этой главе мы расскажем, как и какие инструменты мы применили для создания этих изображений.

Теория

API

API (Application Programming Interface) — это программный интерфейс, который позволяет разработчикам программного обеспечения взаимодействовать с другими приложениями или сервисами. API предоставляет набор функций, методов и протоколов для взаимодействия между двумя приложениями или сервисами, позволяя им обмениваться данными и выполнять определённые операции.

API может использоваться в различных областях. Таких, как веб-разработка, мобильные приложения игры, облачные вычисления и так далее, например, API может использоваться для доступа к данным из баз данных, для получения вычислительных мощностей удалённой вычислительной платформы (Google Collab), для получения результата работы других сервисов и многое другое.

API (интерфейс прикладного программирования) упрощает процесс программирования при создании приложений, абстрагируя базовую (внутреннюю) реализацию и предоставляя только объекты, действия или их результаты, необходимые разработчику. Если графический интерфейс для почтового клиента может предоставить пользователю кнопку, которая выполнит все шаги для выборки и выделения новых писем, то API для ввода/вывода файлов может дать разработчику функцию, которая копирует файл из одного места в другое, не требуя от разработчика понимания операций файловой системы, происходящих за кулисами.

HTTP

HTTP — широко распространённый протокол передачи данных, изначально предназначенный для передачи гипертекстовых документов (то есть документов, которые могут содержать ссылки, позволяющие организовать переход к другим документам).

Аббревиатура HTTP расшифровывается как HyperText Transfer Protocol, «протокол передачи гипертекста». В соответствии со спецификацией OSI, HTTP является протоколом прикладного (верхнего, 7-го) уровня.

Протокол HTTP предполагает использование клиент-серверной структуры передачи данных. Клиентское приложение формирует запрос и отправляет его на сервер, после чего серверное ПО обрабатывает данный запрос, формирует ответ и передаёт его обратно клиенту. После этого клиентское приложение может продолжить отправлять другие запросы, которые будут обработаны аналогичным образом.

Задача, которая традиционно решается с помощью протокола HTTP — обмен данными между пользовательским приложением, осуществляющим доступ к веб-ресурсам (обычно это веб-браузер) и веб-сервером. На данный момент именно благодаря протоколу HTTP обеспечивается работа Всемирной паутины.

Также HTTP часто используется как протокол передачи информации для других протоколов прикладного уровня, таких как SOAP, XML-RPC и WebDAV. В таком случае говорят, что протокол HTTP используется как «транспорт».

API многих программных продуктов также подразумевает использование HTTP для передачи данных — сами данные при этом могут иметь любой формат, например, XML, HTML, JSON или даже байтовое представление изображения.

Как правило, передача данных по протоколу HTTP осуществляется через TCP/IP-соединения. Серверное программное обеспечение при этом обычно использует TCP-порт 80 (и, если порт не указан явно, то обычно клиентское программное обеспечение по умолчанию использует именно 80-й порт для открываемых HTTP-соединений), хотя может использовать и любой другой.

**Цель программы**

По полученным из нейронной сети координатам получить изображение области на карте (используются Яндекс карты), а также нарисовать на нём траекторию движения тайфуна. Все действия выполняются с помощью обращения к Static API Яндекс карт через HTTP запрос. Static API используется именно для получения статичного изображения карты по заданным в запросе параметрам.

Внутри кода

Код для формирования изображения карты был написан на ЯП Python, с применением модулей requests (для отправки и обработки HTTP запросов), os (для работы с операционной системой), numpy (для обработки больших массивов числовых данных). Также для удобства разработки и непосредственного предпросмотра полученных изображений, использовался Jupyter Notebook.

Сама программа поделена на 2 файла. Один из них представляет из себя реализацию класса MapParams, который выступает, как структура для хранения параметров, которые мы будем указывать в запросе к Static API. Второй же файл хранит в себе всю логику программы (считывание и парсинг данных, построение запроса, его обработка, создание картинки в байтовом представлении, сохранение её на ПК и тд).

YandexMap

В полях width и height хранятся параметры выходного изображения в пикселях (ширина и высота соответственно), при чём они по умолчанию установлены в максимально возможные значения, поле dots используется для хранения координат точек, по которым мы будем строить траекторию. Поле zoom отвечает за уровень приближения карты, а type – за тип слоя карты (схема, спутник или гибрид).

*class* MapParams(*object*):  
 *def \_\_init\_\_*(*self*, data: numpy.array, width: *int* = 650, height: *int* = 450):  
 *# Точки движения тайфуна  
 self*.dots = data  
 *self*.zoom: *int* = 3 *# Масштаб карты на старте. Изменяется от 1 до 17  
 self*.type: *str* = "map" *# Другие значения "sat", "sat,skl"*

Многие функции в этом классе отвечают за то, чтобы преобразовать значения полей в необходимую для запроса форму, так, например, размер изображения указывается без пробелов через запятую, а перечисление нескольких нарисованных линий через знак тильда (~).

Такими функциями являются size\_key, line\_key, bbox\_key.

Единственная функция, которая занимается вычислениями – find\_borders, которая находит граничные значения точек, чтобы все точки траектории обязательно поместились на изображении (по умолчанию Static API тоже справляется с этой задачей).

*# Размеры выходного изображения  
def* size\_key(*self*):  
 *return str*(*self*.width) + "," + *str*(*self*.height)  
  
*def* bbox\_key(*self*):  
 borders = *self*.find\_borders()  
 result = f'{borders[0][1]:.1f},{borders[0][0]:.1f}~{borders[1][1]:.1f},{borders[1][0]:.1f}'  
 *return* result  
  
*def* find\_borders(*self*):  
 max\_coord = *list*(numpy.amax(*self*.dots, axis=0))  
 min\_coord = *list*(numpy.amin(*self*.dots, axis=0))  
 *return* min\_coord, max\_coord  
  
*def* line\_key(*self*):  
 result = ''  
 *for* i *in self*.dots:  
 result += f'{i[1]:.2f},{i[0]:.2f},'  
 *return* result[:-1]

**YandexRequest.py**

В этом файле происходит основная работа по обращению к API Яндекс карт, а также обработка входных данных (CSV), определение их типа, сохранение изображений и их вывод.

Начать стоит с обработки входных данных и определения типа файла. На вход может податься 2 типа:

1. Файл только с исходными данными (до обработки нейронной сетью)
2. Файл, который включает и исходные, и итоговые значения (после обработки)

За определение типа отвечает функция find\_type\_file:

*def* find\_type\_file(data: *list*):  
 cnt\_strings = 0  
 *for* i *in* data:  
 *if* i != '':  
 cnt\_strings += 1  
 *else*:  
 *break  
 return* cnt\_strings

Функция, отвечающая за создание запроса, его отправку, сохранение изображения, а также обработку всех возможных исключений, которые могут быть вызваны при неудачном HTTP запросе:

* Размер (sz) – размер выходного изображения (Высота x Ширина)
* Тип (l) – тип слоя карты
* Линия (pl) – параметр для отрисовки линий на карте. Параметр «c» задаёт цвет, «w» - толщина линии, «bw» - толщина обводки.

*def* load\_map(mp, name: *str*, mp\_res=*None*):  
 *#ll - центр (автомат)  
 #z(zoom) - масштабирование (автомат)  
 #size - разрешение изображения  
 #bbox - границы отображения* second\_line = ""  
 *if* mp\_res *is not None*:  
 second\_line = f"~c:0xFFA500FF,w:6,bw:3,{mp\_res.line\_key()}"  
  
 map\_request = "http://static-maps.yandex.ru/1.x/?&size={sz}&l={type}&pl=w:6,bw:3,{coords}{second\_coords}". \  
 format(type=mp.type,  
 sz=mp.size\_key(),  
 coords=mp.line\_key(),  
 bbox=mp.bbox\_key(),  
 second\_coords=second\_line)  
  
 response = requests.get(map\_request)  
 *if not* response:  
 *print*(f"Ошибка с файлом {name}")  
 *print*("Ошибка выполнения запроса:")  
 *print*(map\_request)  
 *print*("Http статус:", response.status\_code, "(", response.reason, ")")  
 *#sys.exit(1)  
  
 # Запись полученного изображения в файл.* map\_file = f"results/{name}.png"  
 *try*:  
 *with open*(map\_file, "wb") *as* file:  
 file.write(response.content)  
 *except IOError as* ex:  
 *print*("Ошибка записи временного файла:", ex)  
 sys.exit(2)  
 *return* map\_file

За обработку файлов, фильтрацию данных и распределение в массивы numpy отвечает функция main:

*def* main():  
 *# if len(args) != 2:  
 # sys.exit(2)  
 #  
 # filename = args[1]* isExist = os.path.exists("results")  
 *if not* isExist:  
 os.makedirs("results")  
  
 *with open*(f"data.txt") *as* file:  
 *# Убираем пустые строки* dataset = file.read().split('\n')  
 cnt\_strings = find\_type\_file(dataset)  
 dataset = *list*(*filter*(*None*, dataset))  
  
 *if* cnt\_strings == 2:  
 *# Проходимся по каждому тайфуну  
 for* name, coords *in zip*(dataset[::2], dataset[1::2]):  
 data = numpy.array(*list*(coords.strip().split(',')), dtype=*float*).reshape(-1, 2)  
 *print*(data)  
 mp = MapParams(data)  
 load\_map(mp, name)  
  
 *if* cnt\_strings == 3:  
 *for* name, orig, result *in zip*(dataset[::3], dataset[1::3], dataset[2::3]):  
 orig\_data = numpy.array(*list*(orig.strip().split(',')), dtype=*float*).reshape(-1, 2)  
 result\_data = numpy.array(*list*(result.strip().split(',')), dtype=*float*).reshape(-1, 2)  
  
 orig\_mp = MapParams(orig\_data)  
 res\_mp = MapParams(result\_data)  
 load\_map(orig\_mp, name, res\_mp)

Приложения

Приложение 1

**YandexMap.py**

*import* numpy  
  
  
*class* MapParams(*object*):  
 *def \_\_init\_\_*(*self*, data: numpy.array, width: *int* = 650, height: *int* = 450):  
 *# Точки движения тайфуна  
 self*.dots = data  
  
 *self*.zoom: *int* = 3 *# Масштаб карты на старте. Изменяется от 1 до 17  
 self*.type: *str* = "map" *# Другие значения "sat", "sat,skl"  
  
 # Разрешение изображения (максимум 650x450)  
 if* 0 < width <= 650:  
 *self*.width: *int* = width  
 *else*:  
 *raise ValueError*("Максимальная ширина: 650")  
 *if* 0 < height <= 450:  
 *self*.height: *int* = height  
 *else*:  
 *raise ValueError*("Максимальная высота: 450")  
  
 *# Размеры выходного изображения  
 def* size\_key(*self*):  
 *return str*(*self*.width) + "," + *str*(*self*.height)  
  
 *def* bbox\_key(*self*):  
 borders = *self*.find\_borders()  
 result = f'{borders[0][1]:.1f},{borders[0][0]:.1f}~{borders[1][1]:.1f},{borders[1][0]:.1f}'  
 *return* result  
  
 *def* find\_borders(*self*):  
 max\_coord = *list*(numpy.amax(*self*.dots, axis=0))  
 min\_coord = *list*(numpy.amin(*self*.dots, axis=0))  
 *return* min\_coord, max\_coord  
  
 *def* line\_key(*self*):  
 result = ''  
 *for* i *in self*.dots:  
 result += f'{i[1]:.2f},{i[0]:.2f},'  
 *return* result[:-1]

Приложение 2

**YandexRequest.py**

*import* numpy  
*import* requests, sys, os  
*from* IPython.display *import* Image, display  
*from* YandexMap *import* \*  
  
  
*# Создание карты с соответствующими параметрами.  
def* load\_map(mp, name: *str*, mp\_res=*None*):  
 *#ll - центр (автомат)  
 #z(zoom) - масштабирование (автомат)  
 #size - разрешение изображения  
 #bbox - границы отображения* second\_line = ""  
 *if* mp\_res *is not None*:  
 second\_line = f"~c:0xFFA500FF,w:6,bw:3,{mp\_res.line\_key()}"  
  
 map\_request = "http://static-maps.yandex.ru/1.x/?&size={sz}&l={type}&pl=w:6,bw:3,{coords}{second\_coords}". \  
 format(type=mp.type,  
 sz=mp.size\_key(),  
 coords=mp.line\_key(),  
 bbox=mp.bbox\_key(),  
 second\_coords=second\_line)  
  
 response = requests.get(map\_request)  
 *if not* response:  
 *print*(f"Ошибка с файлом {name}")  
 *print*("Ошибка выполнения запроса:")  
 *print*(map\_request)  
 *print*("Http статус:", response.status\_code, "(", response.reason, ")")  
 *#sys.exit(1)  
  
 # Запись полученного изображения в файл.* map\_file = f"results/{name}.png"  
 *try*:  
 *with open*(map\_file, "wb") *as* file:  
 file.write(response.content)  
 *except IOError as* ex:  
 *print*("Ошибка записи временного файла:", ex)  
 sys.exit(2)  
 *return* map\_file  
  
  
*def* find\_type\_file(data: *list*):  
 cnt\_strings = 0  
 *for* i *in* data:  
 *if* i != '':  
 cnt\_strings += 1  
 *else*:  
 *break  
 return* cnt\_strings  
  
  
*def* main():  
 *# if len(args) != 2:  
 # sys.exit(2)  
 #  
 # filename = args[1]* isExist = os.path.exists("results")  
 *if not* isExist:  
 os.makedirs("results")  
  
 *with open*(f"data.txt") *as* file:  
 *# Убираем пустые строки* dataset = file.read().split('\n')  
 cnt\_strings = find\_type\_file(dataset)  
 dataset = *list*(*filter*(*None*, dataset))  
  
 *if* cnt\_strings == 2:  
 *# Проходимся по каждому тайфуну  
 for* name, coords *in zip*(dataset[::2], dataset[1::2]):  
 data = numpy.array(*list*(coords.strip().split(',')), dtype=*float*).reshape(-1, 2)  
 *print*(data)  
 mp = MapParams(data)  
 load\_map(mp, name)  
  
 *if* cnt\_strings == 3:  
 *for* name, orig, result *in zip*(dataset[::3], dataset[1::3], dataset[2::3]):  
 orig\_data = numpy.array(*list*(orig.strip().split(',')), dtype=*float*).reshape(-1, 2)  
 result\_data = numpy.array(*list*(result.strip().split(',')), dtype=*float*).reshape(-1, 2)  
  
 orig\_mp = MapParams(orig\_data)  
 res\_mp = MapParams(result\_data)  
 load\_map(orig\_mp, name, res\_mp)  
  
  
*if* \_\_name\_\_ == "\_\_main\_\_":  
 main()