Санкт-Петербургский национальный исследовательский университет
информационных технологий, механики и оптики
Факультет программной инженерии и компьютерной техники

Архитектура программных систем
Лабораторная №2

Выполнил:
Беляков Дмитрий
Группа:
P33122
Преподаватель:
Перл И. А.

Санкт-Петербург
2020

Factory method (GoF)

Сценарий:
Выделение фичей из входных данных с помощью разнообразных нейронных сетей. Для большинства задач recognition/embedding требуется выделение фич (определённых признаков) из входных данных, для этого можно использовать разнообразные эмпатические нейронные сети.
Способ организации: есть пользователь, который хочет натренировать свою часть нейронной сети, но перед этим ему нужно получить фичи из входных данных, обучение происходит с помощью класса Trainer, который отдаёт продукты в виде моделей нейронных сетей, дасатетов, даталоадеров и т. д. В зависимости от конфигурации (так как нейронные сети, их архитектуры и др. параметры могут отличаться), создаётся необходимая реализация класса Trainer, в которой присутствует своя логика под определённый класс сетей (для разнообразных сетей обучение может быть разное, например, различная loss function).
Возможные ограничения: для каждого типа сетей — мы регламентируем свой trainer, однако сеть с одними и теми же параметрами может обучаться по-разному, что ставит нас в рамки.

Сценарий:
Создание датасета для обучения. Часто при создании решения на основе нейронной сети, необходимо создать датасет для обучения, а не взять уже готовой. Способов создания может быть несколько. Например, у нас есть куча картинок, мы хотим их проанализировать, для определённой задачи нам может потребоваться их кластеризация, фильтрация, работа с шумами, добавление шумов и т. д. (пример из реального проекта), обычно такие способы создания строго определенны под каждую задачу и допускают вариацию только с гиперпараметрами. Тогда в таком случае мы можем создать фабричный метод, который бы делал наш датасет (продукт), а приложение, в зависимости от конфигураций будет выбирать необходимый DatasetCreator.
Возможные ограничения: большая архитектура для запуска всего несколько раз, возможно, проще создавать отдельные скрипты для запуска со своей конфигурацией под каждый тип датасета.

Builder (GoF)

Сценарий:
Вернёмся к первому примеру и предложим чуть более простой паттерн: будем использовать для создания такой схемы приложения паттерн Builder. В данном случае Trainer будет состоять из нескольких строительных блоков: feature model, additional model, loss, shedulare, dataset, dataloader etc и метод train. Часть таких модулей обязательная (например feature model и loss), однако, имеет и множество дополнений. Примерно такая же реализация есть в framework Pytorch Lightning, однако там разнообразие Trainer регламентируется на созданием разнообразных классов, а конфигурацией, что по сути является проблемой, но почему-то использовать так гораздо удобнее.
Возможные ограничения: некоторые части — обязательно должны быть (но это решается просто требованиями к созданию объекта Trainer). Необходимо писать большие конфигурации.

Сценарий:
Визуализатор 3-х мерных данных (например, LiDAR point clouds). Есть объект, который занимается визуализацией данных, у нас есть много параметров, таких как: координаты камеры, углы поворота, цветная палитра, размеры точек, система координат. Теперь мы можем не пользоваться огромным конструктором и пользоваться разнообразными методами.

Возможные ограничения: стоит выбор между параметрами по умолчанию или большим количеством сеттеров.

Creator (GRASP)

Сценираий:
Как мы понмим из первого примера, класс Trainer возвращает нам просто продукты, при этом возникала проблема, что есть объекты, которые должны быть созданы, без этой логики не будет работать логика системы, данную проблему может решить шаблон Creator, который регламентирует, какие объекты должны быть созданы, чтобы объекты правильно создавались. Так как наш сценарий предполагает логику создания, то такой паттерн отлично подойдёт для нашего сценария. (Кстати, по моему мнению, компоненты, которыми оперирует Trainer скорее всего удобнее создавать так же по шаблону Creator или хотя бы Information Expert, так как все эти компоненты знают, как взаимодействовать со своими данными, таким образом, они получаются малосвязанными и легко понимаемые для пользователя).
Возможные ограничения: немного усложняется логика, особенно для отладки, не очень понятно, как данные передаются от одного объекта к другому. Хоть иерархия усложняется, но, мне кажется, что это скорее плюс, явно видно за что отвечает каждый объект.

Сценарий:
Мы анализируем лица, передаваемые с камеры в реальном времени. Например, обработкой занимается класс Controller, однако при таком использовании, каждый раз, когда нам надо будет отдать очередное изображение на обработку, нам придётся преобразовать массив данных в класс изображения или массива (в зависимости от того, в каком виде нам приходят данных), таким образом, мы не сможем переиспользовать наш Controller, например, если вдруг поменяется формат (на самом деле, может много чего поменяться - цвет, разрешение, наличие шумов, фпс и т. д.). В таком случае лучше организовать логику в классе Recognizator, который и занимается распознованием. Его можно будет сконфигурировать под необходимые нужды, избавляя Controller от ненужных зависимостей.
Возможные ограничения: нам не следует полностью отдавать логику обработки Recognizator, так как, например, может увеличится fps и мы будем не успевать обрабатывать все изображения, это и не задача Recognizator, для этого контроллер нам и нужен. Хоть это и не совсем ограничение, но пока что других проблем с такой структурой я не вижу.

Вывод: в данной лабораторной работе я снова погрузился в мир шаблонов, повторил их, понял, что, если долго не работаешь с ООП (ну или очень редко), то быстро забываешь его основные парадигмы. Я пытался подбирать ситуации из собственной жизни, некоторые пришлось чутка изменить, которые подходили бы под какие-нибудь шаблоны, после чего я понял, что скорее всего, что проект чисто по шаблону я бы не стал делать, если только в части проекта или брать какие-то хорошие правила и концепты из различных шаблонов (главное, чтобы каши не было). В целом, ситуации подобрать получилось, однако, это была не такая простая задача.