Learning to Infer Graphics Programs from Hand-Drawn Images

Mikhailov Nikita M.
Faculty of Computer Science
Highter School of Economics
Moscow
nmmikhaylov@edu.hse.ru

Abstract

Это промужуточный отчет проекта по воспроизведению статьи

Learning to Infer Graphics Programs from Hand-Drawn Images

В рамках требований к промежуточномой контрольной точке тут выполнен разбор используемого в работе DSL, реализован работающий модуль для рендера, а так же модуль генерации данных

Ссылка на GitHub проекта

1 Описание задачи

Часто мы нуждаемся в различных диаграммах. Для этого мы пользуемся различными инструментами для их создания. На каждую такую картинку мы тратим время, а хотелось бы просто от руки схематично нарисовать, чтобы потом программа сама все выравняла, соединила нужные точки (например, чтобы два круга, центры которых на одной прямой, были соединены ровно по этой прямой). Этим и вдохновились авторы статьи. Сама задача состоит в том, чтобы по нарисованным от руки картинкам выдавать

1. Программу, с помощью которой можно эту диаграмму создать
2. Изображение, построенное на основе программы. В идеале ровную картинку со входа

Мы работаем с диаграммами, а они часто состоят из простых объектов, например, прямоугольников, окружностей, отрезков. Отступим от такой классической схемы лишь в отрезках: добавим им возможность быть направленными – со стрелочкой, и быть пунктирной. Авторы статьи назвали эту задачу «Program Learning». Так же мы не хотим выдавать простой набор команд, например:

1. Прямоугольник с точками $(1, 1), (2, 2)$
2. Прямоугольник с точками $(2, 2), (3, 3)$
3. Прямоугольник с точками $(3, 3), (4, 4)$
4. Прямоугольник с точками $(4, 4), (5, 5)$

Очевидно, такие прямоугольники можно нарисовать циклом:

Цикл по $i \in [1, 4]$:

Прямоугольник с точками $(i, i), (i + 1, i + 1)$

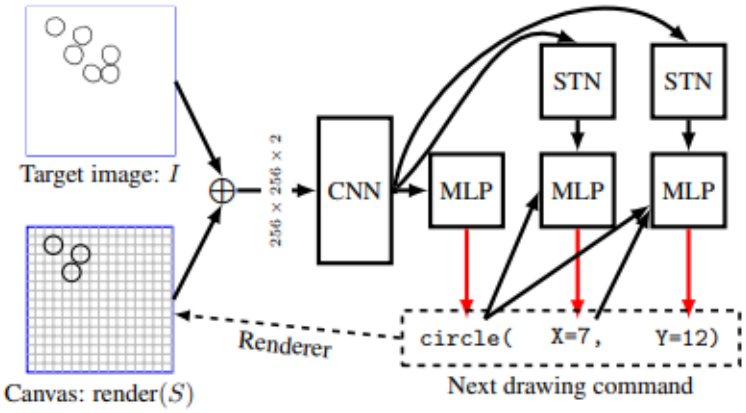
После того, как мы выяснили, какие объекты должны быть на сцене, мы должны выдать оптимальную с некоторой точки зрения программу для их рендеринга.

1nd Отчет проекта по курсу DeepLearning HSE AMI2019), Moscow, Russia.

2 Подход к решению задачи

Предсказывать все объекты целиком слишком сложно, поэтому авторы статьи предлагают действовать итерационно. Пусть у нас есть входное изображение $S(ource)$ и изображение $R(endered)$, полученное после рендера. Изначально R – пустое:

1. Подаем на вход сети двухканальное изображение $[S, R]$
2. Сеть сравнивает S и R и решает, какой объект рисовать дальше. То есть выдает объект O
3. Генерируем новое изображение R уже с объектом O.
4. Возвращаемся к п.1, пока $O \neq END$



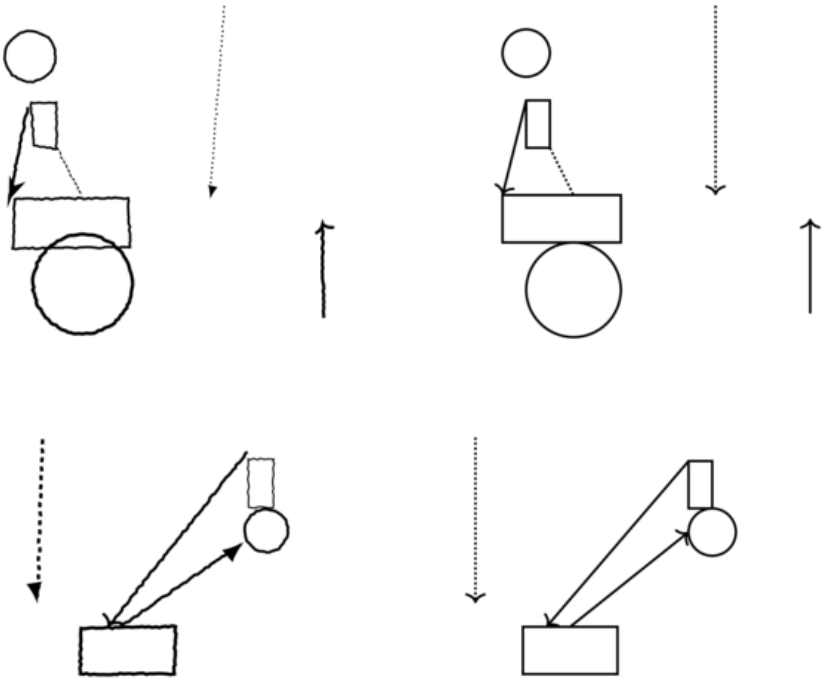
Саму сеть будем проектировать в следующей части проекта. Прежде всего нужно подготовить данные для обучения. Авторы статьи утверждают, что сеть хорошо учится даже на синтетических данных.

3 Генерация данных

Генерировать данные будем с помощью LaTeX, в котором есть TikZ. Диаграмма – это прежде всего набор различных фигур. Разобьем изображение на сетку размера 16×16. Введем описание каждого из объектов:

1. Точка. У нее есть координаты x, y. Генерировать ее координаты будем с некоторой вероятностью из уже имеющихся точек на сцене, либо случайно из сетки.
2. Окружность. У нее есть центр – точка, и радиус. Генерировать ее будем следующим образом: сначала пройдемся по всем существующим окружностям, далее с некоторой вероятностью переиспользуем один из радиусов и сгенерируем центр. Если полученная окружность выходит за рамки сетки, то повторяем заново, пока не получим валидную окружность.
3. Прямоугольник. У него есть четыре угла – четыре точки. Задается однозначно лишь по двум противоположным. Семплируем две точки. Одну из них назначаем верхним левым углом, вторую правым нижним (если нужно меняем координаты так, чтобы все было хорошо).
4. Линия. У нее есть две точки. Самое сложное в плане генерации. На диаграммах линии чаще всего используются, чтобы соединять объекты, поэтому точки для линий мы будем семплировать из специальных точек существующих объектов. Каждый объект имеет точки, которыми его можно присоединить. Генерируем все такие прямые, а далее с некоторой вероятностью выбираем случайную из них, а иначе случайно семплируем две точки линии.

Казалось бы этого достаточно, но мы не учли пересечения объектов. Будем добавлять на текущую сцену объект, только если он не пересекает никакой из уже существующих объектов.
Для генерации «чистых» данных этого достаточно, однако нам нужно имитировать нарисованные от руки изображения. Для этого каждому объекту добавим шум в координаты и будем использовать атрибут «pencildraw», который делает как раз то, что нам нужно. После этого «перегоним» все объекты в LaTeXпрограмму, скомпилируем *pdf*-файл, сконвертируем изображения оттуда в *png*-картинки. Вот что из этого получилось:



Слева – имитация рукописных изображений, справа – ровное изображение, полученное в результате генерации. Если посмотреть на оригинальную статью, то синтетические данные у них не лучше, поэтому считаем что мы пришли к успеху и реализовали модуль генерации данных, а так же их рендера. Все же это не диаграммы, но для данных, на которых мы будем обучаться должно подойти. Все такие самое главное – научиться детектировать объект, отнести его к нужному классу и предсказать его правильные координаты так, чтобы все было ровно.

4 DSL

Начнем с того, а что вообще такое DSL? Domain-Specific Language(DSL) – язык программирования специализированный для конкретной области применения. В нашем случае – для изображения диаграмм.

Мы разрешаем (`for`), условия (`if`), вертикальные/горизонтальные отражения (`reflect`), переменные (Var) и афинные преобразования ($\mathbb{Z}\times\text{Var}+\mathbb{Z}$).

Program→	Statement; $\cdots$; Statement
Statement→	`circle`(Expression,Expression)
Statement→	`rectangle`(Expression,Expression,Expression,Expression)
Statement→	`line`(Expression,Expression,Expression,Expression,Boolean,Boolean)
Statement→	`for`$(0 \leq Var < Expression)$ `{ if` $(Var > 0)$ `{` Program `}; ` Program `}`
Statement→	`reflect`$(Axis)$ `{` Program `}`
Expression→	$\mathbb{Z}\times$Var+$\mathbb{Z}$
Axis→	`X =` $\mathbb{Z}$ | `Y =` $\mathbb{Z}$
$\mathbb{Z}\to$	an integer

5 Дальнейшие планы.

К финальной части проекта я предоставлю обученную под эту задачу нейросеть и, возможно, можно будет самому что-нибудь нарисовать.