Adversarial text recognition

Слайд 1 (Гриша)

Здравствуйте! Меня зовут Григорий Бартош, это Александр Федотов и Михаил Ютман. И наш проект: “Соревновательное распознавание текста”.

Слайд 2 (Гриша)

Итак. В интернете существует множество картинок с текстом. И хочется уметь качественно распознавать этот текст. Такая задача называется “Optical Character Recognition” или OCR.

Слайд 3 (Гриша)

Общая схема обучения OCR модели выглядит так. Берем размеченные картинки с текстом (то есть для каждой картинки известно, что за текст на ней написан и, скорее всего, где он находится) и подаем такие картинки на вход в OCR модель. На выходе модель выдает информацию о том, где находится текст и что это за текст. Этой информацией могут быть, к примеру, баундинг боксы для предложений, слов или букв с соответствующими лейблами.

Слайд 4 (Гриша)

Главная проблема с таким подходом - необходимость размеченных данных. Особенно, если модели требуются баундинг боксы. Потому что таких данных мало, а размечать их дорого. Однако, так как нас интересуют картинки с напечатанным текстом, мы можем сгенерировать синтетические данные в неограниченном количестве и сразу же автоматически их разметить. Но у такого подхода тоже есть проблема, ведь, если генерировать данные просто рандомно, мы получим смещенные и простые примеры. Обычно это решается наложением эвристических ограничений на распределение вероятности для генерируемых данных и аугментацией. Также можно вводить “соревновательную аугментацию”. То есть аугментировать картинки именно таким образом, чтобы максимально усложнить задачу OCR моделе.

Слайд 5 (Гриша)

Но это все же адаптация аугментации. Мы же хотим адаптировать само наложение текста на картинку. И сделать это через соревновательное обучение. Таким образом нам нужно научиться генерировать параметры для наложения текста. Сгенерировать само наложение текста. И, собственно, самое важное - научиться распознавать текст.

Слайд 6 (Гриша)

Соответственно общая архитектура будет выглядеть вот так. Получаем на вход картинку и текст, который будем накладывать, (также в виде картинки). Generator’ом генерируем параметры наложения. Staker’ом выполняем это самое наложение. И с помощью OCR модели распознаем текст. Аналогично GAN’ам здесь получается min-max игра. Generator стремится максимизировать ошибку, а OCR модель минимизировать. Stacker же не обучается, а просто выполняет механическую работу.

Слайд 7 (Миша)

Перейдем к реализации. И первое, с чем мы столкнулись - Stacker. Он должен применять параметры преобразования текста. Что это за параметры? Мы выбрали следующие параметры: поворот, растяжение, масштаб, сдвиг и цвет. Казалось бы - в чем проблема? А проблема в том, что если имея черно-белою картинку текста довольно просто сделать ее нужного цвета и положить на картинку, то вот с остальными параметрами все сложнее. Дело в том, что мы работаем с сетями. А значит через все операции должен проходить градиент, таким образом выход Stacker'а должен быть дифференцируем по входным параметрам. Но например операция сдвига, это не преобразование значений тензора, а преобразование координат его элементов, то есть это некое переупорядочивание его элементов. А переупорядочивание не дифференцируемо по параметрам этого переупорядочивания.

Собственно, изначально мы думали, что эта проблема как-то не очень сложно решится, но она никак просто не решается. Так что пришлось решать ее сложно. Мы взяли все недифференцируемые параметры. И сделали простую функцию, которая просто применяет эти параметры. И начали генерировать с помощью нее синтетические данные: исходная картинка, параметры преобразования, результат. А с помощью этих данных можно обучить нейросеть, которая будет делать то, что нужно, ведь все операции нейросети дифференцируемы по значению. Эту часть мы назвали Mover.

Слайд 8 (Миша)

В итоге мы пришли к вот такой модели. На вход принимается картинка и параметры. Картинка проходит через энкодер (у нас это ResNet18), параметры через трехслойный перцептрон. Они конкатенируются и все вместе это идет в декодер, который выполняет реконструкцию (у нас это инвертированный ResNet18). После чего стоит дискриминатор, который отличает настоящие картинки от сгенерированных (еще одна ResNet18). Получился такой параметризованный автоэнкодер-ган.

Слайд 9 (Миша)

Прогресс выглядел следующим образом. Изначально мы учили модель просто как автоэнкодер, то есть параметры были нулевыми, а дискриминатора не было. Ошибкой был MSE. Работало это просто прекрасно. Обучалось очень быстро и хорошо.

Тогда мы начали обучать сеть с параметрами. И все перестало работать совсем. Единственное, что получалось - размытые белые пятна там, где должен был быть текст.

Тогда мы начали параллельно учить модель и как автоэнкодер, и как параметризованный автоэнкодер. Стало чуть-чуть получше. Он начал восстанавливать какие-то контуры при малых трансформациях и у крупных символов.

Слайд 10 (Миша)

После этого было принято волевое решение отказаться от поворота, что, очевидно, привело к положительному результату. Начали вырисовываться слова по отдельности.

Но все еще реконструкция маленьких слов работала сильно хуже реконструкции крупных. Поэтому мы заменили MSE на сумму квадратов разности деленную на площадь всего текста. Это существенно помогло улучшить реконструкцию маленьких слов.

Однако текст все еще оставался размытым. Поэтому мы решили добавить соревновательности, введя дискриминатор, в надежде на увеличение резкости. И резкость действительно увеличилась, но появилась проблема с тем, что одна буква может превратиться в другую (к примеру l в i) и появились артефакты.

В общем довести до того состояния, до которого хотелось, мы, увы, не успели, но это совершенно точно можно сделать поиграв с размерами слоев и их числом и, может быть, какой-то аугментацией. В свое оправдание можем сказать, что мы все-таки работали с довольно большими картинками 300 на 300.

Слайд 11 (Саша)

Вернемся к основной архитектуре. Генератор параметров. Он принимает две картинки: саму картинку и текст. Пропускает их через две модели (здесь это опять же ResNet18) затем конкатенирует эмбеддинги и отправляет их в MLP.

Слайд 12 (Саша)

Распознавание текста. Мы распознавали символы текста по одному и использовали для это модель распознавание объектов SSD (Single Shot multibox Detector). Если вкратце, модель выглядит так. Вот вы знаете как выглядит архитектура VGG или ResNet. Там берутся тензоры, к которым применяются свертки и постепенно размерность тензоров уменьшается, а число каналов растет. Тензоры различных размеров - это различные уровни сети. В SSD берется последний тензор с каждого уровня и по нему предсказываются баундинг боксы различных размеров. С начальных уровней предсказывается много маленьких боксов, а с более глубоких уровней меньше боксов, но уже больших размеров. Всего сеть предсказывает 8732 бокса с лейблами. Это объяснение очень на пальцах, просто для общего понимания.

Слайд 13 (Саша)

Что у нас в итоге получилось. Мы полностью реализовали все пайплайны: и для обучения Stacker’a, и для обучения распознавания текста на случайных примерах, и с соревновательной генерацией параметров наложения. Мы исследовали процесс обучения Stacker’a. Обучили распознавание на случайных примерах (результат можно увидеть на картинках). И увы не обучили распознавание с соревновательной генерацией, так как не доучен Mover.

Вывод можно сделать такой, что мы в очередной раз убедились, что свертки гораздо хуже справляются с преобразованиями отличными от преобразования текстур.

На этом у нас все.