ИССЛЕДОВАНИЕ АРХИТЕКТУРЫ СЕТИ YOLO ДЛЯ ДЕТЕКТИРОВАНИЯ ОБЪЕКТОВ НА ИЗОБРАЖЕНИЯХ

А.П. Береснев, И.В.Зоев
Научный руководитель: Н.Г. Марков
Национальный исследовательский Томский политехнический университет
apb3@tpu.ru

Введение

На сегодняшний день в компьютерном зрении актуально не только задача распознавания объектов на изображениях, но и определение их местоположение, т.е. детектирование. В данной статье исследуются возможности архитектуры сверточной нейронной сети (СНС) YOLO с целью её дальнейшей реализации на программируемых логических интегральных схемах (ПЛИС). Обоснование выбора такой архитектуры СНС приведено статье «Перспективные нейронные сети для реализации на ПЛИС».

Архитектурные особенности сети

Было опубликовано две статьи [1, 2] по СНС YOLO. В первой приводится первая версия СНС, во второй её улученная версия. В данной работе рассматривается вторая версия этой СНС.

Детектирование объектов осуществляется с некоторой точностью. Пусть на изображении I расположено N объектов, которые ограничены прямоугольными областями Bgt_i *(ground truth)* где $i \in 1,..N$, Необходимо из I, найти такие Bp_i *(predicted)*, что

$$IOU(Bp_i, Bgt_i) = \frac{area(Bp_i \cap Bgt_i)}{area(Bp_i \cup Bgt_i)} > 0{,}5,$$

где $IOU(Bp_i, Bgt_i)$ – пересечение над объединением (*intersection over union – IOU*), $area(Bp_i \cap Bgt_i)$ – площадь пересечения областей Bp_i и Bgt_i, $area(Bp_i \cup Bgt_i)$ – площадь объединения областей Bp_i и Bgt_i. Если $IOU(Bp_i, Bgt_i) > 0{,}5$, то считается, что область найдена верно [3]. Помимо Bp_i из I необходимо определить классы к которым относятся объекты.

Система детектирования работает следующим образом. Изображение разбивается на *S* x *S* областей. Каждая область предсказывает *B* областей и степень уверенности в том, что этой области содержится объект. Каждая из областей предсказывает 5 переменных: *x, y, w, h, confidence. (x, y)* – координаты центра объекта относительно ячейки. W, h – ширина и высота объекта соответственно. *Confidence* показывает *IOU*. Каждая ячейка так же предсказывает *C* – вероятность принадлежности объекта к определенному классу. В результате, получается матрица, размерности *S* x *S* x *(B*5 + C)* элементов. Для Pascal VOC используется *S*=7, *B*=2. Количество классов *C*=20. Следовательно результирующая матрица представляется в виде 7 x 7 x 30. Что соответствует выходному слою сети. В качестве функции активации используется *leaky rectified linear unit (Leaky ReLU).* Для такой сети функция потерь *(loss function)* приведена в [1]. Одно из нововведений это батч-нормализация (batch normalization). Этот метод предложен в статье [4]. Метод решает проблему внутреннего ковариационного сдвига.

В *YOLOv2* удалены два полносвязных слоя на выходе, вместо них на используются сверточные слои. Вместо координат, сеть предсказывает смещения относительно якорных значений (anchor boxes). Для *YOLOv2* эти anchor boxes получаются с использованием метода *K-means* с использованием обучающей выборки. Авторы используют значение *K=5*, для лучшего результата детектирования. Выходных карт признаков недостаточно для получения выходной матрицы нужной размерности. Поэтому добавляются карты из предыдущих слоев (выход 13-го объединен с 20).

Вместе с архитектурой *YOLOv2* состоящей из 22 сверточных слоев была представлена и уменьшенная модель сети, в 9 слоев и с меньшим количеством фильтров свертки. Уменьшенная модель необходима для увеличения скорости работы, хотя приходится пожертвовать точностью работы сети. Авторы отмечают, что такая топология сети занимает меньший объем памяти, что критически важно на встраиваемых системах.

Обучение вариантов сети

Для обучения использовалась модификация библиотеки *Caffe* в которую были добавлены необходимые входной и слой, реализующий функцию потерь [5].

Для ускорения процесса обучения используется GPU GTX 1060. Размер батча для обучения составляет 8 изображений, а для тестирования 2. Разрушающий параметр для весовых коэффициентов *weight decay* (позволяет предотвращать переобучение) составляет значение равное 0,0005. На начальном этапе (первые 100 итераций) скорость обучения равна 0,0001. Далее до 25000 итерации она имеет значение равное 0,001. Далее с каждой 10000 последующей итерацией скорость уменьшается на десяток. Всего обучение занимает 45000 итераций. Для обучения были использованы архитектуры *YOLOv2* (с батч-нормализацией и без батч-нормализации), *Tiny-YOLO* с батч-нормализацией.

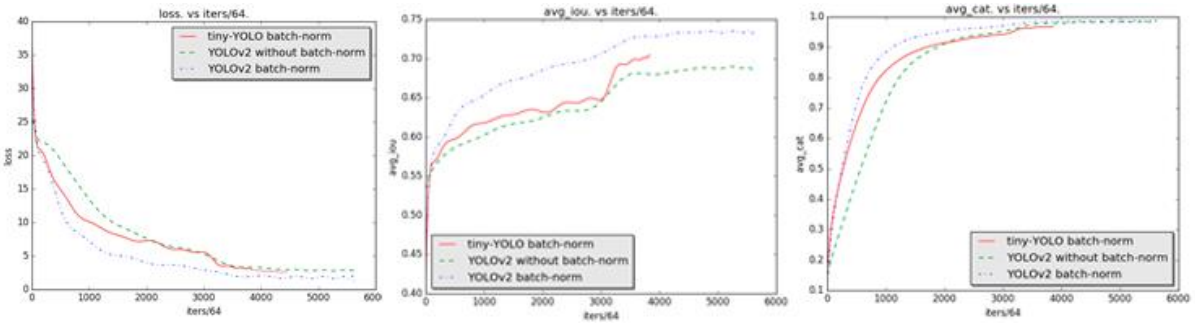

Рис. 1. Графики обучения вариантов архитектур *YOLO*. Слева функция ошибки (loss), по центру средняя точность выделения объектов регионами (avg_ iou), справа средняя точность отнесения объекта к классу (avg_ cat)

Анализ результатов

Из полученных графиков (рис. 1) обучения сетей можно сделать некоторые выводы. Вариант *YOLOv2* с использованием батч-нормализации имеет наилучшие результаты, относительно других вариантов. В среднем, два других варианта имеют сравнимую точность к окончанию процесса обучения. Несмотря на относительно небольшие размеры сети *tiny-YOLO*, она показывает сравнимые результаты по точности детектирования объектов. Можно сделать вывод, что исследователи пытались добиться минимальных размеров сети, при которой она будет распознавать объекты и такой сетью является *tiny-YOLO.*

Из-за того, что реализация операции батч-нормализации на ПЛИС может иметь некоторые сложности, в дальнейшем необходимо провести дополнительные исследования по использованию данной операции.

Рисунок 2 иллюстрирует пример работы сети *YOLOv2*. Как видим, имеются правильно распознанные объекты и имеется и ошибка распознавания. Хотя семантическая связь между классом *horse* и *cow* довольно сильна.

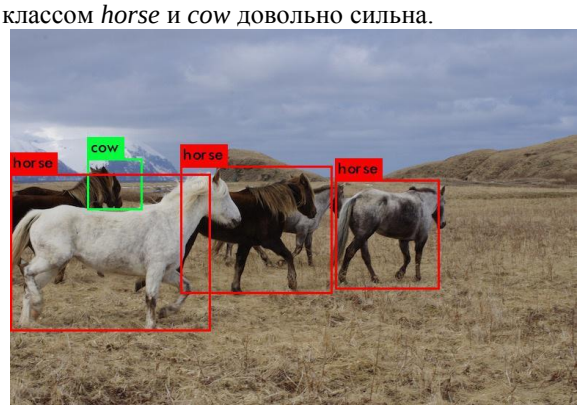

Рис. 2. Пример детектирования объектов

Заключение

В ходе работы были проанализированы архитектуры *YOLOv2* различный модификаций. Произведено обучение различных вариаций архитектуры как с использованием батч-нормализации, так и без неё с использованием фреймворка *Caffe*. Анализ графиков показал преимущество *YOLOv2* с использованием батч-нормализации перед другими вариантами сети.

Для реализации на ПЛИС, можно предложить варианты сети *YOLOv2* без использования батч-нормализации и *tiny-YOLO* с использованием батч-нормализации. Каждый из предложенных вариантов имеет свои преимущества и недостатки.

Данные варианты детектора имеют сравнимую точность распознавания. *Tiny-YOLO* требует использования операции батч-нормализации, что ведет к усложнению реализации итогового устройства. Хотя в этой архитектуре используется всего 9 слоёв, но накладные расходы на вычисления могут нивелировать это преимущество.

YOLOv2 без использования батч-нормализации не требует реализации этой операции, хотя использование 22 сверточных слоев могут увеличить время выполнения прямого прохода сети.

Список использованных источников

1. Joseph Redmon. You Only Look Once: Unified, Real-Time Object Detection. [Электронный ресурс] – URL: https://arxiv.org/pdf/1506.02640.pdf (дата обращения 18.09.2017)

2. Joseph Redmon. YOLO9000: Better, Faster, Stronger. [Электронный ресурс] – URL: https://arxiv.org/pdf/1612.08242.pdf (дата обращения 18.09.2017)

3. M. Everingham, L. V. Gool, C. K. I. Williams, J. W. A. Zisserman. The PASCAL Visual Object Classes (VOC) Challenge. [Электронный ресурс] – URL: https://pdfs.semanticscholar.org/0ee1/916a0cb2dc7d3add086b5f1092c3d4beb38a.pdf (дата обращения 25.10.2017)

4. Sergey Ioffe. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. [Электронный ресурс] – URL: https://arxiv.org/abs/1502.03167 (дата обращения 18.09.2017)

5. Caffe: a fast open framework for deep learning. [Электронный ресурс] – URL: https://github.com/meikuam/caffe (дата обращения 18.09.2017)