Разработка алгоритмов генерации синтетических данных для обучения нейросетевых моделей детектирования объектов на изображении

В.И. Берзин, М.И. Судейкин
berzin@phystech.edu | MSudeykin@ermasoft.ru
Московский физико-технический институт (НИУ), Москва, Россия

Работа посвящена разработке алгоритмов генерации синтетических данных для обучения моделей детекторов объектов на изображении. В качестве целевых моделей рассматриваются современные SOTA-архитектуры на основе сверточных нейронных сетей, а также методы их обучения. Выявляются особенности, которыми должна обладать обучающая выборка на основе синтетических данных, для устойчивой работы модели на множестве натуральных данных. Описываются предлагаемые методы и принципы генерации таких данных. В качестве сопровождающего практического примера рассматривается задача детектирования товарных единиц на полках продуктовых супермаркетов, в контексте которой производилось тестирование имплементированных алгоритмов.

Ключевые слова: *алгоритмы, генерация синтетических данных, нейросетевые модели, детектирование, изображение.*

Development of synthetic data generation algorithms for training neural network models for detecting objects in an image

V.I. Berzin, M.I. Sudeykin
Moscow Institute of Physics and Technology (NRU), Moscow, Russia

The paper is devoted to the development of synthetic data generation algorithms for training models of object detectors in the image. Modern SOTA architectures based on convolutional neural networks, as well as methods for their training, are considered as target models. The features that a training set based on synthetic data must have for the stable operation of the model on a set of natural data are revealed. The proposed methods and principles for generating such data are described. As an accompanying practical example, the problem of detecting commodity items on the shelves of grocery supermarkets is considered, in the context of which the implemented algorithms were tested.

Key words: *algorithms, synthetic data generation, neural network models, detection, image.*

1. Введение

1.1. Задача детектирования объектов на изображении

Детектирование объектов (object detection) - одна из наиболее фундаментальных и сложных задач компьютерного зрения (CV) – позволяет определить, есть ли на изображении какие-либо экземпляры объектов из заданных категорий, и, если они присутствуют, то указать пространственное расположение каждого экземпляра объекта, например, через ограничивающую рамку (bounding box).

Будучи краеугольным камнем понимания изображений и компьютерного зрения, детектирование объектов формирует основу для решения сложных или высокоуровневых CV-задач, - таких как сегментация, отслеживание объектов, аннотирование изображений, обнаружение событий и распознавание активности.

Детектирование объектов имеет широкий спектр приложений, включая робототехнику, потребительскую электронику, безопасность, автономное вождение, человеко-компьютерные интерфейсы, поиск изображений на основе контента, интеллектуальное видеонаблюдение и дополненную реальность. В последнее время методы глубокого обучения стали мощными инструментами и позволили добиться значительных улучшений в задачах компьютерного зрения.

Задача детектирования объектов может быть разделена на два типа: обнаружение экземпляров конкретных объектов реального мира и обнаружения широких категорий. Исторически сложилось так, что большая часть усилий в области обнаружения объектов была сосредоточена на обнаружении одной категории (как правило, лиц, пешеходов, автомобилей) или нескольких категорий. Напротив, за последние несколько лет исследовательское сообщество начало продвигаться к более сложной цели - созданию систем *детектирования объектов общего назначения [1]*, где широта возможностей обнаружения объектов соперничает с широтой возможностей человека. Такой алгоритм в идеале должен достигать двух конкурирующих целей: высокого качества и высокой эффективности. Высококачественное обнаружение должно точно локализовать и распознать объекты на изображениях или видеокадрах. Высокая эффективность подразумевает выполнение всей задачи обнаружения в режиме реального времени с приемлемыми требованиями к памяти и хранилищу.

Проблемы в точности обнаружения проистекают из следующих причин:

1) широкий спектр внутриклассовых вариаций (мультимодальность распределений);
2) разнообразие окружающих условий при обнаружении;
3) огромное количество категорий объектов, проблемы таксономии.

С точки зрения внутренних факторов каждая категория может иметь множество различных экземпляров объекта, возможно изменение одного или нескольких цветов, текстуры, материала, формы и размера. Даже в таких более узко определенных классах, как, например, человек или лошадь, экземпляры объектов могут быть подвержены деформациям, находиться в различных позах и т.д.

Изменения условий визуализации вызваны воздействием окружающий среды на внешний вид объекта, - например, освещение (рассвет, день, сумерки, в помещении), физическое местоположение, погода, параметры камеры, фон, окклюзия и т.д. Все эти условия приводят к значительным изменениям внешнего вида объекта. Дополнительные проблемы могут быть добавлены артефактами оцифровки, шумовым искажением, сжатием и фильтрацией.

Большое количество категорий объектов требует большей мощности распознавания от детектора, чтобы различать близкие межклассовые вариации. На практике современные детекторы фокусируются в основном на структурированных категориях объектов. Для таких задач существуют такие открытые наборы данных, как PASCAL VOC (20 классов) [2], ILSVRC [3] (200 классов) и MS COCO [4] (91 класс). Очевидно, что количество рассматриваемых категорий объектов в существующих базовых датасетах так или иначе много меньше, чем множество, которое способен распознавать человек.

Наборы данных играют ключевую роль в современных задачах компьютерного зрения. Их используют не только для обучения моделей, но и в качестве общей основы для измерения и сравнения производительности конкурирующих моделей и алгоритмов. В частности, для качественного применения методов глубокого обучения критически важно большое количество аннотированных данных.

Для создания больших аннотированных наборов данных существует три этапа:

1) определение набора целевых категорий объектов;
2) сбор изображений для представления выбранных категорий;
3) аннотирование собранных изображений.

Из проблем задачи обнаружения объектов, описанных выше, вытекают и требования к данным, используемых для обучения моделей. В частности, при определении категорий объектов важна таксономия, иерархию нужно выстраивать исходя из распределения визуальных признаков тех или иных объектов. При сборе данных важно понимать, что каждый класс требует множество примеров изображений в различных условиях, причем тем больше, чем больше количество классов. Наконец, аннотирование изображений должно быть достаточно точным.

К сожалению, на сегодняшний день создание наборов данных для обучения моделей классификации, детектирования и сегментации изображений проходит как правило вручную, что негативно влияет на скорость решения вышеперечисленных задач. Трудоемкость при их создании, а также оносительно малое количество открытых датасетов тормозит развитие методов глубокого обучения для CV. Кроме того, человеческий фактор может негативно влиять на точность разметки данных, что часто приводит к некачественным результатам.

В настоящей работе для решения вышеперечисленных проблем предлагаются алгоритмы автоматической генерации аннотированных данных. В главе 2 рассмотрены современные SOTA[1]-модели классификации, детектирования и сегментации изображений, описана специфика данных для каждой из задач. В главе 4 приведен обзор существующих методов генерации данных, а в главах 5-6 описаны алгоритмы, предложенные автором данной работы для решения задачи детектирования товарных единиц на полках продуктовых супермаркетов.

1.2. Извлечение признаков из изображения и сверточные нейросети

Сверточная нейронная сеть (CNN[2] или ConvNet) представляет собой класс глубоких нейронных сетей, наиболее часто применяемых для анализа визуальных образов. Они также известны как инвариантные или пространственно-инвариантные искусственные нейронные сети (SIANN[3]), основанные на их архитектуре с общими весами и характеристиках трансляционной инвариантности.

Простая классификационная сверточная сеть состоит из следующих слоев: сверточные (convolutional) слои, субдискретизирующие (subsampling, подвыборка) слои и полносвязные слои, в соответствии с рисунком 1.

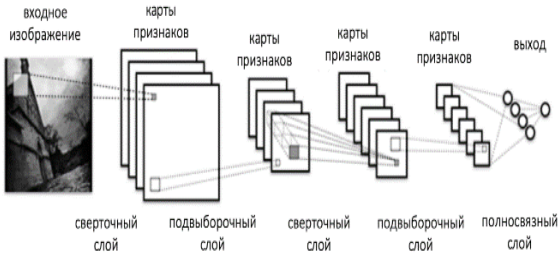


Рис.1. Пример архитектуры классифицирующей сверточной нейронной сети

Первые два типа слоев (convolutional, subsampling), чередуясь между собой, формируют входной вектор признаков для многослойного персептрона. Сверточный слой представляет из себя набор карт (другое название – карты признаков, в обиходе это обычные матрицы), у каждой карты есть синаптическое ядро (в разных источниках его называют поразному: сканирующее ядро или фильтр).

Размер у всех карт сверточного слоя – одинаковы и вычисляются по формуле:

[1] SOTA - state of the art

[2] CNN – convolutional neural network

[3] SIANN - space invariant artificial neural networks

$$(w, h) = (mW - kW + 1, mH - kH + 1),$$

где (w, h) – вычисляемый размер сверточной карты;

mW – ширина предыдущей карты;

mH – высота предыдущей карты;

kW – ширина ядра;

kH – высота ядра.

Ядро представляет из себя фильтр или окно, которое скользит по всей области предыдущей карты и находит определенные признаки объектов. Например, если сеть обучали на множестве лиц, то одно из ядер могло бы в процессе обучения выдавать наибольший сигнал в области глаза, рта, брови или носа, другое ядро могло бы выявлять другие признаки. Размер ядра обычно берут в пределах от 3x3 до 7x7. Если размер ядра маленький, то оно не сможет выделить какие-либо признаки, если слишком большое, то увеличивается количество связей между нейронами. Также размер ядра выбирается таким, чтобы размер карт сверточного слоя был четным, это позволяет не терять информацию при уменьшении размерности в подвыборочном слое.

Неформально эту операцию можно описать следующим образом — окном проходим с заданным шагом (обычно 1) все изображение, на каждом шаге поэлементно умножаем содержимое окна на ядро, результат суммируется и записывается в матрицу результата, как на рисунке 2:

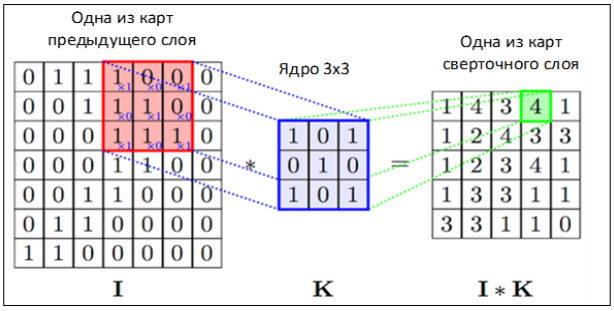


Рис.2. Пример работы свертки

В процессе сканирования ядром подвыборочного слоя (фильтром) карты предыдущего слоя, сканирующее ядро не пересекается в отличие от сверточного слоя. Обычно, каждая карта имеет ядро размером 2x2, что позволяет уменьшить предыдущие карты сверточного слоя в 2 раза. Вся карта признаков разделяется на ячейки 2x2 элемента, из которых выбираются максимальные по значению.

Одним из этапов разработки нейронной сети является выбор функции активации нейронов. Вид функции активации во многом определяет функциональные возможности нейронной сети и метод обучения этой сети. Классический алгоритм обратного распространения ошибки хорошо работает на двухслойных и трехслойных нейронных сетях, но при дальнейшем увеличении глубины начинает испытывать проблемы. Одна из причин — так называемое затухание градиентов. По мере распространения ошибки от выходного слоя к входному на каждом слое происходит домножение текущего результата на производную функции активации. Производная у традиционной сигмоидной функции активации меньше единицы на всей области определения, поэтому после нескольких слоев ошибка станет близкой к нулю.

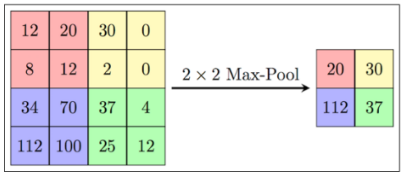


Рис.3. Результат работы подвыборочного слоя Max-Pool

Сегодня наиболее часто в качестве функции активации в сверточных нейронных сетях применяется функция ReLU, которая отчасти лишена подобных недостатков:

$$f(s) = \max(0, s)$$

$$f'(s) = \begin{cases} 1, & s > 0 \\ rand(0.01, 0.05), & s \le 0 \end{cases}$$

Наконец, полносвязные слои формируют выходной вектор, как правило, состоящий из нейронов, количество которых равно числу классов.

1.3. Region-proposal детекторы

Ранние подходы к задаче детектирования объектов состояли из двух шагов:

- деление изображения на несколько регионов;
- классификация каждого региона.

Алгоритм скользящего окна - один из способов достижения первого шага, когда прямоугольное окно проходит по всему изображению несколько раз с учетом различных соотношений сторон (размеров), углов, форм окна. Но, очевидно, что такой способ неэффективен и требует огромных вычислительных затрат.

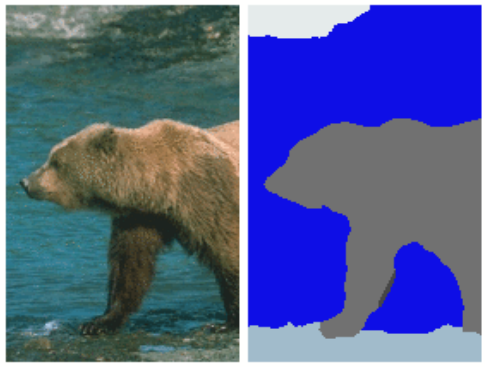


Рис.4. Пример текстурной сегментации изображения

В следующем наборе алгоритмов для локализации объектов использовался другой подход. Вместо того, чтобы перемещать окно по изображению, эти методы пытались сгруппировать похожие пиксели на изображении, чтобы сформировать области интереса. Эта группировка была сделана с использованием алгоритмов сегментации изображения. Такие регионы подаются в классификатор для идентификации класса.

Сегментация может выполняться различными способами. Например, с помощью алгоритма Фельзенсвальба [5], SLIC [6]. Однако такая сегментация может

выполняться и на уровне самой нейронной сети (рис. 5).

1.4. Однопроходные детекторы

Недостатки методов на основе предсказания отдельных регионов на изображении очевидны. Глобальное деление на два шага, каждый из которых требует больших вычислительных затрат. В частности, классификацию нужно проводить для каждого региона в отдельности.

Следующим шагом в развитии методов детектирования являются однопроходные детекторы, основная идея которых состоит в использовании внутренних представлений. Иными словами, сама сверточная сеть в процессе классификации уже выполняет большую часть работы детектора.

Например, если обратить внимание на результаты преобразований сигнала после слоев активации, то можно увидеть нечто похожее на тепловые карты (рис. 6).

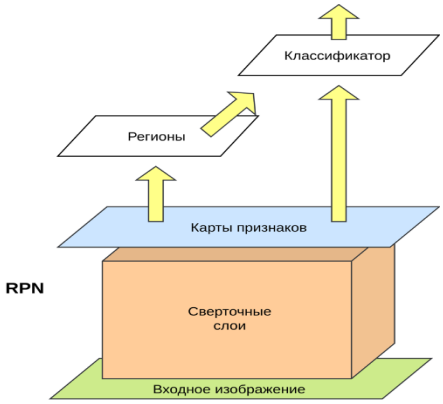


Рис.5. Архитектура Region Proposal Network (RPN)

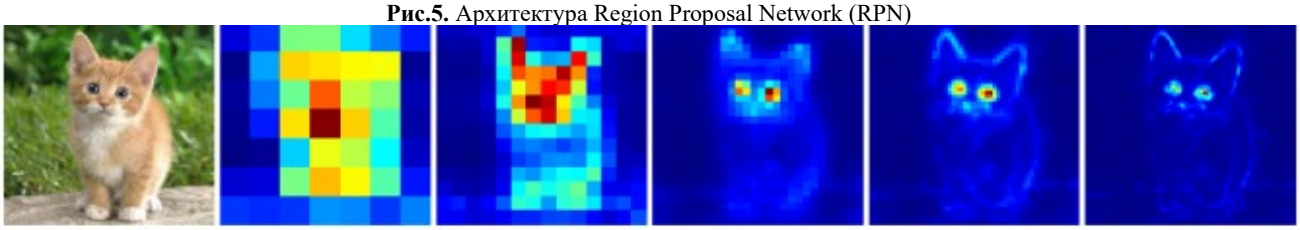


Рис.6. Слои активации нейронной сети во время распространения сигнала

Как говорилось в начале главы, существуют нелинейные функции активации, которые принимают на вход результаты сверток входных изображений с различными фильтрами. Выходы активационных слоев называют картами признаков (feature maps). Такая карта представляет собой матрицу, каждый элемент которой является композицией результата свертки входного изображения с фильтром и нелинейной функции активации. Такой элемент можно интерпретировать как меру его принадлежности к определенному классу, которому в свою очередь принадлежит конкретный фильтр.

Опираясь на такие внутренние представления, можно построить множество различных архитектур под различные задачи (рис. 7):

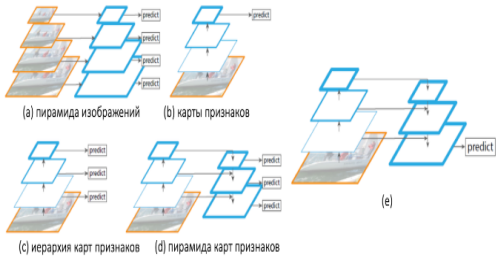


Рис.7. Основные типы сверточных архитектур

1. *Пирамида изображений (Image pyramid)* Такое представление активно использовалось до сверточных нейронных сетей в эпоху необучаемых методов компьютерного зрения. Здесь создается пирамида изображений с размерами кратным степени числа (обычно равного 2). Над каждым таким изображением независимо производятся преобразования, на основе каждого из которых дается ответ.
2. *Карты признаков одного изображения* Данное представление является базовым для сверточных нейронных сетей. На вход подается изображение, которое преобразуется в карту признаков. Далее операциями свертки,

либо с помощью подвыборочных слоев (pooling) понижается размер карты, после чего преобразования повторяются несколько раз.

3. *Иерархия карт признаков (Pyramidal feature hierarchy)*
   Карты признаков строятся от одного входного изображения, притом они являются зависимыми. Ответ дается путем агрегации нескольких выходов с каждого уровня пирамиды. Упускает возможность повторно использовать уже просчитанные карты признаков с более высоким разрешением, что негативно влияет на обнаружение небольших объектов. Типичным представителем, использующим такую архитектуру, являются детектор SSD.
4. *Пирамида карт признаков (Feature Pyramid Network)*
   Сочетает семантически сильные выходы с низким разрешением и семантически слабые выходы с высоким разрешением через прямой путь и боковые соединения, не влияя на использование вычислительных ресурсов. Данную архитектуру использует популярный дектектор YOLO (рис. 8).
5. *Модификация (d)*
   В отличие от (d) имеет один выход. Такой подход получил распространение в однопроходных сегментаторах, таких как TDM, SharpMask, RED-Net, U-Net.

Рассмотрим детектор YOLO. Данный детектор получает на вход изображение размера M x N, а на выходе предсказывают список прямоугольных рамок, которые описывают обнаруженный объект заданного класса C_k с достаточной достоверностью $P_k > T$, где T – настраиваемый порог.

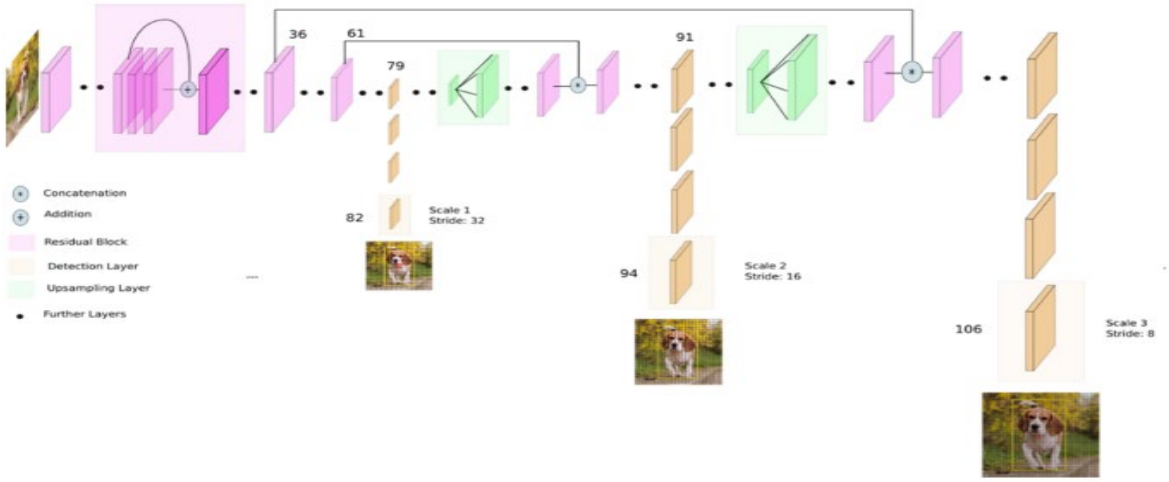


Рис.8. Архитектура детектора YOLO

Каждый выход представляет собой тензор размера $X_i \times X_i \times L$, где X_i – ширина выходной карты признаков для i-го выхода, L – длина вектора предсказаний для каждой ячейки. Каждая ячейка выходной карты предсказывает (рис. 9):

- в прямоугольников, обрамляющих объект *(x,y,w,h,o),* где
  x,y – координаты левого верхнего угла прямоугольника;
  w,h – ширина и высота прямоугольника;
  o – степень объективности предсказания, имеющая вероятностную интерпретацию.
- вероятности классов объекта $\{C_k,\ k \in [1, K] \cap \mathbb{N}\}$, K – количество классов.

Тогда имеем, что $L = 5B + K$.

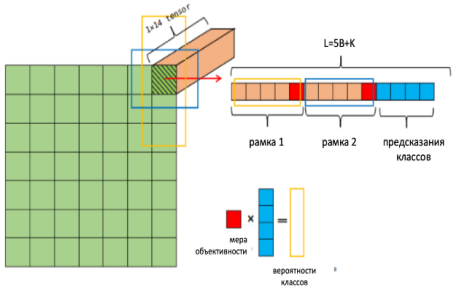


Рис.9. Структура выходного тензора YOLO

Конечная вероятность класса в каждой ячейке карты рассчитывается как $P_k = oC_k$. Таким образом, отфильтровав все ячейки, для которых $P_k > T$, получим список обрамляющих прямоугольников. Так как размер объекта может быть больше размера ячейки, то часто встречается ситуация, при которой один объект описывается несколькими прямоугольниками (рис. 10). В этом случае выбирают наилучший прямоугольник с помощью метода Non-Maximum Suppression (подавление не-максимумов).

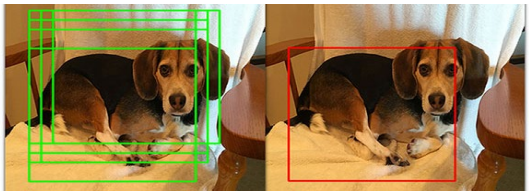


Рис.10. Пример работы алгоритма NMS

Метод NMS использует меру Жаккара. Пусть даны два множества *A* и *B*, мерой Жаккара *J(A,B)* называют отношений нормы пересечения этих множеств к норме их объединения:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

Алгоритм NMS

Входные данные: список обрамляющих прямоугольников, возвращаемых нейросетью L_{in} и пороговое значение перекрытия Q.

Вывод: список отфильтрованных рамок L_{out} (изначально пуст).

Ход алгоритма:

1. Выбирается рамка с наибольшей оценкой достоверности и перемещается из списка L_{in} в L_{out}.
2. Теперь для всех пар рамок L_{in} - L_{out} считается мера Жаккара. Если она оказывается меньше порога Q, то рамка из L_{in}, соответствующая данной паре, удаляется.
3. Теперь из L_{in} снова выбирается рамка с наибольшей оценкой достоверности, как в п.1. Этот процесс повторяется до тех пор, пока в L_{in} не станет пуст.

Таким образом, алгоритм NMS позволяет избавляться от большого количества пересечений, оставляя каждому обнаруженному объекту только одну рамку.

Процесс обучения

Обучение происходит путем минимизации следующего функционала (функции потерь):

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$

Первые два слагаемых отвечают за локализацию и производят регрессию координат обрамляющих прямоугольников.

Здесь:

x_i, y_i, ω_i, h_i – предсказываемые координаты;

$\hat{x}_i, \hat{y}_i, \hat{\omega}_i, \hat{h}_i$ – реальные координаты;

λ_{coord} – весовой параметр, обозначающий относительны вклад слагаемого локализации в функцию потерь;

$\mathbb{1}_{i,j}^{obj}$ – равен 1, если в j-й рамке предсказан верный объект, в остальных случаях 0;

S – размер карты признаков;

B – число детектируемых рамок одной ячейкой.

Третье и четвертое слагаемые отвечают за величину C_i, характеризующую объективность предсказания. Здесь $\hat{C}_i \equiv 1$, $\mathbb{1}_{i,j}^{noobj}$ комплементарен $\mathbb{1}_{i,j}^{obj}$, λ_{coord} – весовой параметр.

Последнее слагаемое производит классификацию, минимизируя сумму среднеквадратичных ошибок вероятности класса, $p_i(c)$.

1.5. Сети с вниманием и сегментаторы

В последнее время для задач детектирования объектов все чаще применяют модели сегментаторов. Задачи сегментации можно разделить на два типа: семантическая сегментация и инстанс-сегментация (рис. 11).

Первая предсказывает для изображения набор бинарных масок для каждого известного класса. Инстанс-сегментация обрабатывает несколько объектов одного класса как различные объекты и предсказывает набор бинарных масок для каждого объекта. На практике чаще всего важнее именно семантическая сегментация, и именно такой тип имеет больше всего общего с задачей детектирования объектов. Дальше под сегментацией будет пониматься именно семантическая сегментация.

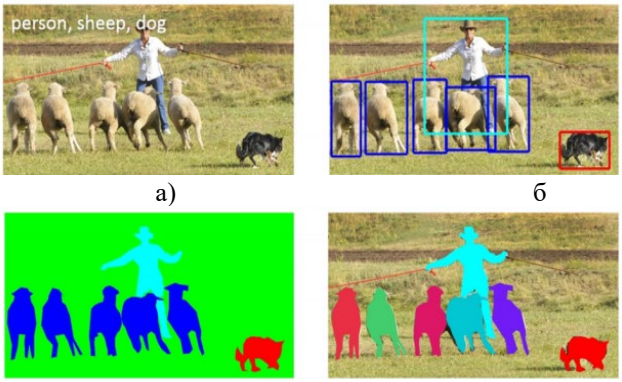


Рис.11. Классификация задач распознавания изображений: а) классификация, б) детектирование объектов, в) семантическая сегментация, г)инстанс- сегментация

Вместо набора прямоугольных рамок модель-сегментатор предсказывают бинарные маски для каждого класса в виде тензора размера *N* x *K* x *C*, где *N* x *K* – размер изображения, *C* – число классов.

В качестве функций потерь для задач сегментации используют следующие функционалы:

Сумма абсолютных разностей.

Является одним из самых простых функций:

$$L = \sum |m - \hat{m}|,$$

где $m, \hat{m}$ – настоящая и предсказанная маски соответственно. Очевидным недостатком такого функционала является слабая сходимость при небольших размерах объектов, поэтому его часто нормируют на сумму норм двух масок $|\hat{m}| + |m|$, но часто неплохо работает для грубых масок с небольшим разрешением.

Попиксельная кросс-энтропия

Предварительно к выходной маске $m_{n,k,c}$ применяется функция

$$softmax_{n,k,c} = \frac{e^{m_{n,k,c}}}{\sum_{\hat{c}=1}^{C} e^{m_{n,k,\hat{c}}}}$$

которая, возвращает тензор того же размера, сглаживая значения вдоль классов $c \in [0, C]$ для каждой пары координат $\{ n \in [0, N], k \in [0, K] \}$.

Далее функция перекрестной энтропии штрафует отклонение предсказанной маски от истинной внутри каждой пары координат (n,k):

$$L = -\sum_{(n,k)} \sum_{c} m_{n,k,c} \log (\hat{m}_{n,k,c})$$

Такой лосс зарекоменловал себя во многих современных сегментаторах, в частности U-Net, хорошо работает на точных масках больших размеров (равных входному изображению).

Мера Жаккара

Обычно используется как метрика качества:

$$J(m, \hat{m}) = \frac{m \cap bin(\hat{m})}{m \cup bin(\hat{m})},$$

где *bin* – пороговая функция бинаризации.

Но также хорошо работает и в качестве оптимизируемой функции или дополнительного слагаемого:

$$L = 1 - J(m, \hat{m}).$$

Таким образом, модели семантических сегментаторов являются очень мощным инструментом, который пригоден в том числе для эффективного решения задачи детектирования объектов. Такой подход не требует дополнительных затрат ресурсов на регрессию координат рамок, локализация объектов начинается уже на уровне базовой нейросети. При этом на выходе выдаются более точные маски.

2. Основная часть

2.1. Постановка задачи

Создание обучающего и тестового набора изображений достаточного объема для обучения модели сверточной нейронной сети является комплексной и нетривиальной задачей. Важно понимать, что состав тренировочной базы может влиять на качество получаемой системы распознавания больше, чем все остальные факторы. Несмотря на это, в большинстве научных работ этот важный этап полностью опущен.

Перед созданием базы данных образцов изображений и обучением нейронной сети необходимо уточнить техническую проблему. Понятно, что распознавание рукописного текста, эмоций человеческого лица или местонахождения фотографии - совершенно разные задачи. Распознавание изображений, полученных с камер высокого разрешения или расплывчатых изображений с веб-камеры без автофокуса, потребует совершенно других данных для обучения, тестирования и проверки. Вот почему свободно распространяемые базы данных изображений, рассмотренные выше, отлично подходят для академических исследований, но они почти всегда неприменимы в реальных задачах из-за их «общности».

В настоящей работе в качестве практического бэкграунда была выбрана задача детектирования товарных единиц на продуктовых полках супермаркета (рис. 12). Такие данные несут в себе ряд благоприятных особенностей для исследования, как например, большое разнообразие упаковок разных цветов и форм, а также плотная расстановка. Аннотирование подобных данных традиционными способами является очень трудоемкой задачей. Кроме того, товарные единицы часто меняют свой внешний вид, поэтому аннотирование должно происходить почти непрерывно.

Рис.12. Пример работы детектора объектов

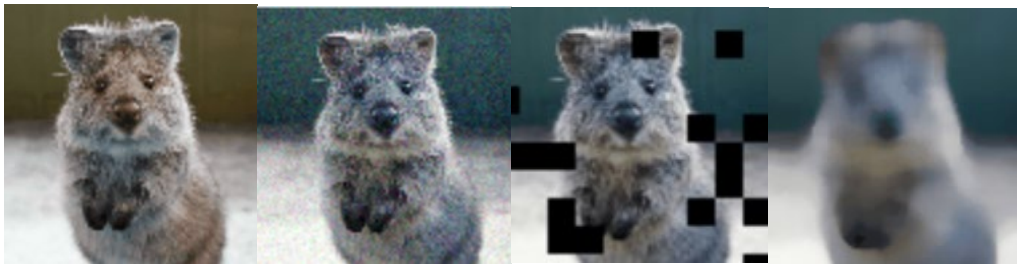
Здесь и далее под генератором будем понимать некую сложную функцию $\mathfrak{G}$, которая принимает на вход трехканальное изображение объекта $I_{x,y,ch}$, а также конечный набор параметров $\vec{p}$, описывающих объект (форма, размеры и т.д.), а на выходе бы возвращал множество пар $(\hat{I}_{x,y,ch}, m_{x,y,c})$ изображений и масок объектов в различных контекстах. В главе 6 будет описан и исследован такой генератор данных к задаче детектирования товарных единиц.

2.2. Существующие подходы к генерации данных для обучения

2.2.1. Аугментация изображений

Есть много способов увеличить количество данных для обучения сверточных нейронных сетей. Один из таких методов – аугментация изображений. Идея данного метода проста – ко входному изображению применяются различные преобразования, чтобы модель могла учиться на других примерах. Можно привести наиболее часто используемые преобразования при аугментации (рис. 13):

Исходное изобр. Поворот Отражение Кадрирование



Смешивание каналов. Зашумление. Дропаут Медианный фильтр

Рис.13. Примеры основных аугментирующих преобразований

- **Поворот** Самое распространенное преобразование. Поворот на небольшие углы часто бывает полезен при малом количестве примеров или при дисбалансе классов. Поворот на большие углы стоит использовать с осторожностью. Часто встретить перевернутый объект в тестовой выборке не представляется возможным, поэтому это действие может повредить качеству модели.

- **Отражение** В случае отражения следует обратить внимание на симметрию объекта. Существуют много неинвариантных относительно симметрии объектов. Например, текстовые символы при отражении могут потерять свое исходное значение, и выучивание таких ложных признаков негативно повлияет на работу модели.

- **Кадрирование** Используется почти всегда. Бывает особенно полезным при плотно упакованных сценах с большим количеством мелких объектов. Кадрирование позволяет извлечь большее количество признаков каждого объекта на изображении.

- **Перемешивание каналов** Здесь подразумеваются различные цветовые преобразования**.** Часто используются для баланса между геометрическими и цветовыми признаками. Например, если детектируемый класс имеет несколько мод в признаковом пространстве. Это могут быть автомобили разных цветов. В этом случае обесцвечивание изображений поможет обратить внимание сети на геометрические признаки при сравнении автомобиля с другими объектами, а не на цветовые.

- **Добавление шума** Добавление шумов различных распределений на фотографиях обучающей выборки делает работу сети инвариантной относительно шума и позволяет избежать включения воздействия шума в признаковое пространство.

- **Дропаут** Позволяет выучивать большее количество более мелких признаков, а также делает признаки более независимыми друг от друга. Часто помогает при переобучении.

- **Медианное сглаживание** Является очень важным преобразованием. Медианный фильтр является нелинейным, и его нельзя представить в виде свертки, а значит он плохо выучивается сетью. При этом на практике много программ для фотосъемки используют его для удаления импульсных помех.

Описанные выше преобразования оказываются полезными и помогают в борьбе с переобучением, обращая внимание сети на наиболее важные признаки. Стоит заметить, что этот процесс должен выполняться с осторожностью. Увеличение количества изображений может оказаться контрпродуктивным, если создаются примеры, которые очень отличаются от входных данных, на которых будет производиться тестирование модели.

Аугментация сегодня является одним из стандартных инструментов при обучении моделей, работающих с изображениями, заметно улучшая качество их работы. Но так или иначе данный метод не способен решить проблемы малых данных.

Несмотря на большое количество возможных преобразований, аугментация изображений не привносит много новой информации. Признаковое пространство вырастает не сильно. Многие из используемых преобразований являются довольно простыми и без проблем могут выучиться самой сетью.

2.2.2. Генеративно-состязательные сети

Основной целью генеративно-состязательных сетей GAN (Generative Adversarial Networks) является генерация данных из шума, в основном изображений.

Такая сеть состоит из двух частей: генератора и дискриминатора.

Генератор учится генерировать правдоподобные данные. Сгенерированные экземпляры становятся отрицательными примерами обучения для дискриминатора.

Дискриминатор учится отличать ложные данные генератора от реальных данных. Дискриминатор наказывает генератор за получение неправдоподобных результатов. Когда начинается обучение, генератор выдает явно фальшивые данные, и дискриминатор быстро учится говорить, что это фальшивая информация. В процессе обучения генератор становится ближе к получению выходных данных, которые могут обмануть дискриминатор.

Наконец, если тренировка генератора проходит хорошо, то качество работы дискриминатора становится хуже. Он начинает классифицировать поддельные данные как реальные. И генератор, и дискриминатор являются нейронными сетями. Выход генератора подключен непосредственно к входу дискриминатора. Благодаря обратному распространению ошибки классификация дискриминатора дает сигнал, который генератор использует для обновления своих весов (рис. 14):

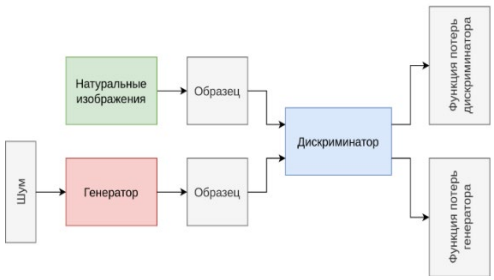


Рис.14. Архитектура GAN

Генератор и дискриминатор имеют разные процессы обучения. Обучение сетей проходит в чередующихся периодах:

1. Дискриминатор тренируется в течение одной или нескольких эпох.
2. Генератор тренируется в течение одной или нескольких эпох.

После шаги 1 и 2 повторяются.

В статье [7], в которой были впервые представлены GAN, генератор пытается минимизировать следующую функцию, а дискриминатор пытается максимизировать ее:

$$L(x) = E_x[\log(D(x))] + E_z[\log\left(1 - D(G(z))\right)]$$

Здесь:

• $D(x)$ - оценка дискриминатором вероятности того, что реальный экземпляр данных x является истинным;

• E_x - ожидаемое значение для всех реальных экземпляров данных;

• $G(z)$ – сгенерированный генератором экземпляр;

• $D(G(z))$ - оценка дискриминатора вероятности того, что поддельный (сгенерированный) экземпляр является реальным;

• E_z - ожидаемое значение для всех генерируемых поддельных экземпляров $G(z)$.

Формула представляет собой перекрестной энтропию между настоящим и сгенерированным распределениями. Генератор не может напрямую влиять на член $\log(D(x))$ в функции, поэтому для генератора минимизация потерь эквивалентна минимизации $\log\left(1 - D(G(z))\right)$.

2.2.3. Виртуальная фотосъемка

Наиболее самостоятельным методом генерации изображений является рендеринг.

Рендеринг - автоматический процесс генерации изображения из 2D (слои видео) или 3D модели с помощью компьютерной программы. Полученное изображение называется рендером.

На сцене могут быть определены сразу несколько моделей объектов. Каждая такая модель содержит информацию о геометрии (вершины, ребра, полигоны), текстурах, картах отражения (нормалей). Дополнительно сама сцена должна хранить информацию о параметрах камеры и освещении. Все эти данные затем передаются в программу рендеринга для обработки и вывода в файл графического изображения.

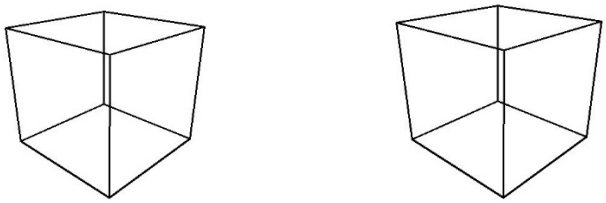


Рис.15. Построение описывающей рамки

Таким образом, для разметки изображений, полученных в результате рендеринга, можно использовать информацию о сцене и объектах, находящихся в ней. Очевидно, если программа-рендерер может выполнить фотореалистичный рендеринг для заданной сцены, то можно также дополнительно сделать рендеринг лишь одной геометрии каждого объекта, получив обрамляющие рамки или маски (рис. 15). Для этого достаточно выполнить проективное преобразование каждой отдельной вершины объекта на плоскость камеры:

$$\begin{bmatrix}\hat{x}\\ \hat{y}\\ \hat{z}\\ 1\end{bmatrix} = \begin{bmatrix}\frac{f\cdot pX}{2\cdot sX} & S & 0 & 0\\ 0 & \frac{f\cdot pY}{2\cdot sY} & 0 & 0\\ 0 & -1 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & \cos(rotX) & -\sin(rotX) & 0\\ 0 & \sin(rotX) & \cos(rotX) & 0\\ 0 & 0 & 0 & 1\end{bmatrix}\times$$

$$\begin{bmatrix}\cos(rotZ) & -\sin(rotZ) & 0 & 0\\ \sin(rotZ) & \cos(rotZ) & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & 0 & -locX\\ 0 & 1 & 0 & -locY\\ 0 & 0 & 1 & -locZ\\ 0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}X\\ Y\\ Z\\ 1\end{bmatrix}$$

Здесь:

$\hat{x}, \hat{y}, \hat{z}$ – координаты ортонормальной проекции вершины на плоскость камеры;

X, Y, Z – мировые координаты вершины;

locX, locY, locZ – мировые координаты камеры;

rotX, rotY, rotZ – углы вращения камеры в радианах;

pX, pY – разрешение выходного изображения (матрицы камеры) в пикселях;

sX, sY – размер сенсора, в мировых единицах измерения;

f – фокальное расстояние.

Далее необходимо преобразовать ортонормальные координаты к перспективным:

$$\begin{bmatrix}x\\ y\\ z\\ 1\end{bmatrix} = \begin{bmatrix}\frac{1}{z} & 0 & 0 & 0\\ 0 & \frac{1}{z} & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\hat{x}\\ \hat{y}\\ \hat{z}\\ 1\end{bmatrix}$$

Координату z точек можно отбросить. Получим набор точек $P_i = \{x_i, y_i\}$, для которого перебором находим прямые: $x = x_{max}$, $x = x_{min}$, $y = y_{max}$, $y = y_{min}$.

Очевидно, что по крайним диагональным точкам $A = x_{min} \cap y_{min}$ и $B = x_{max} \cap y_{max}$ можно построить обрамляющую рамку.

2.3. Стек алгоритмов генерации изображений из 3D-сцен

2.3.1. Используемы инструменты и технологии

Очевидно, наиболее перспективным методом к генерации изображений является рендеринг, так как такой подход позволяет физически корректно работать с исходными данными объектов реального мира, такими, как размер, форма и свойства материалов. Основная сложность заключается в создании самой сцены. Распространенными решениями сегодня являются 3D-сканирование реального окружения (например, Neuromation [8]). Однако, получение конечной трехмерной модели объекта путем сканирования – довольно сложный и ресурсоемкий метод, требующий постобработки.

Методы, предлагаемые в настоящей работе, требуют намного меньше исходных данных, так как генерируют 3D-модели объектов из примитивов и текстур, получение которых связано с несопоставимо меньшими затратами ресурсов (рис. 16).

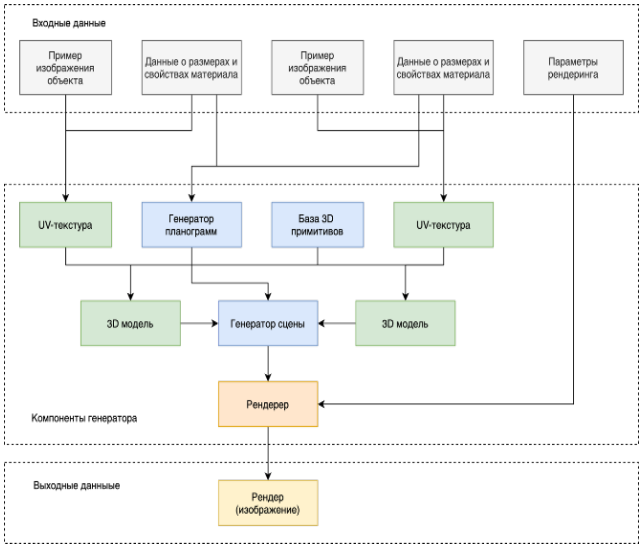


Рис.16. Предлагаемый стек алгоритмов генерации изображений

Предлагаемый подход должен хорошо работать при генерации сцен с продуктовыми полками в магазинах. Этому способствует специфика таких сцен, в частности:

- плотная упорядоченная расстановка объектов;
- примитивные геометрические формы (параллелепипед, цилиндр);
- большая повторяемость моделей.

При реализации выше обозначенного стека алгоритмов использовалось следующее ПО:

- Среда Python 3 с библиотеками:
  - Pandas – модуль обработки табличных данных, используется для операций с исходными данными товарных единиц.
  - OpenCV – графический фреймворк для обработки изображений, используется для генерации UV-текстур из UV-шаблонов для примитивов и изображений объекта.
  - TensorFlow 2 – фреймворк для машинного обучения.
  - Matplotlib – библиотека для визуализации.
- Blender 2.8 – программа для создания и изменения трехмерных моделей, материалов, а также фотореалистичного рендеринга.
- Unreal Engine 4 – трехмерный игровой движок, используется для рендеринга в режиме реального времени.

2.3.2. Подготовка исходных данных. Свойства объектов и паттерны

В качестве целевых классов для детектирования были отобраны упаковки картофельных чипсов различных брендов. С одной стороны на продуктовых полках данный вид продукта расположен отдельно, изолировано от других, а значит не нужно создавать дополнительный контекст из других out-of-domain (данные на которых не происходит обучение) товар-

ных единиц. С другой стороны, такой выбор не нарушает общности алгоритмов генерации. Упаковки разнообразны, имеют формы, которые наследуются от двух примитивов – подушка и цилиндр. Далее генерация, обучение и тестирования проводится для 10 меток классов, каждому и которых соответствует определенный бренд картофельных чипсов.

Рис.17. Примеры исходных изображений для генерации 3D-моделей

Набор исходных данных, которые принимаются на вход генератором представляет собой следующее:

- Пример изображения лицевой части объекта класса. Под лицевой частью понимается сторона, содержащая наибольшее количество отличительных признаков (рис. 17).
- Таблица с дополнительной информацией по каждому классу (табл. 1). Здесь поле **id** – артикул товара, которому соответствует пример изображения; **scale** – размер упаковки в сантиметрах; **template** – вид примитива, которым приближается модель упаковки; **class** – наименование бренда (целевая метка); **material** – светоотражающие свойства материала, которые будут учтены во время рендеринга (глянцевый или матовый).
- Заранее подготовленные модели 3D-примитивов. В случае упаковок для картофельных чипсов были использованы формы цилиндра и подушки (рис. 18).
- Параметры рендеринга являются опциональными, могут нести в себе информацию об интенсивности и цветовой температуре освещения, параметрах камеры (дисторсия, глубина резкости, апертурный угол) и т.д.

Табл.1. Исходные данные по генерируемым моделям

	id	scale	template	class	material
0	3229991H	195×85×255	pillow	lays	glossy
1	3209011H	305×220×50	pillow	lays	glossy
2	3113435H	225×60×260	pillow	lays	glossy
3	3151251H	80×80×235	cylinder	pringles	mate
4	3044725H	175×65×170	pillow	lays	glossy

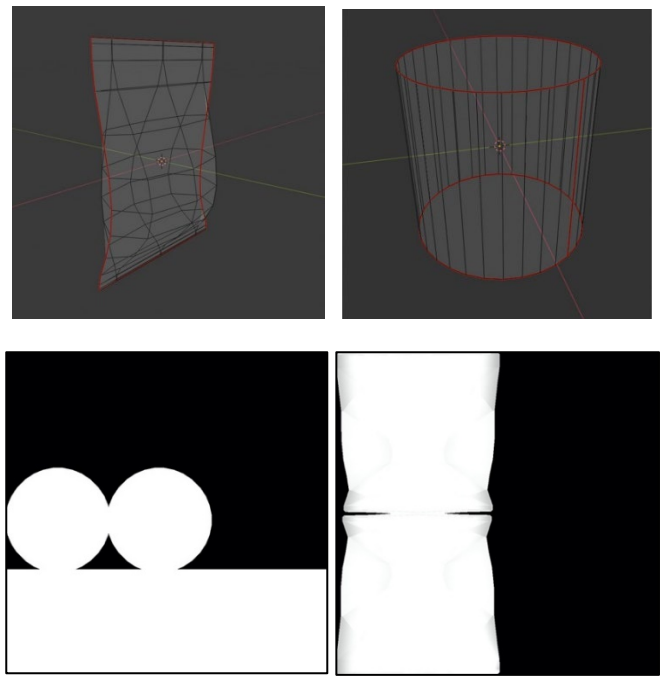


Рис.18. Примеры моделей и их разверток для шаблонов «цилиндр» и «подушка»

2.3.3. Генерация моделей

Генерацию модели можно разделить на следующие этапы:

1. Преобразование изображения объекта в текстуру.
2. Натяжение текстуры на UV-шаблон примитива.
3. Скалирование примитива на реальные размеры объекта.
4. Применение свойств материала к модели.

Преобразование в текстуру осуществляется путем кадрирования исходного изображения товарной единицы на белом фоне. Ниже приведена имплементация такой функции на языке python 3.7. Основная идея со-

стоит в том, чтобы итерироваться от краев изображения к центру до тех пор, пока внутри области кадрирования не останется пикселей, принадлежащих фону.

```
def crop(img):

    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)

    for i in range(0, gray.shape[0]):
        for j in range(0, gray.shape[1]):
            if (gray[i,j] != 255):
                gray[i,j] = 0

    cv.floodFill(gray,None,(0,0),128)[1]

    x = [int(img.shape[0]/2)]*2
    y = [int(img.shape[1]/2)]*2

    shift = 10

    flags = [True,True,True,True]

    while(flags[0]|flags[1]|flags[2]|flags[3]):

        if (flags[0]):
            x[0] -= 1
            for i in range(y[0],y[1]):
                if (gray[x[0] - shift,i] == 128):
                    flags[0] = False
                    break

        if (flags[1]):
            x[1] += 1
            for i in range(y[0],y[1]):
                if (gray[x[1] + shift,i] == 128):
                    flags[1] = False
                    break

        if (flags[2]):
            y[0] -= 1
            for i in range(x[0],x[1]):
                if (gray[i,y[0] - shift] == 128):
                    flags[2] = False
                    break

        if (flags[3]):
            y[1] += 1
            for i in range(x[0],x[1]):
                if (gray[i,y[1] + shift] == 128):
                    flags[3] = False
                    break

    return img[x[0]:x[1],y[0]:y[1]]
```

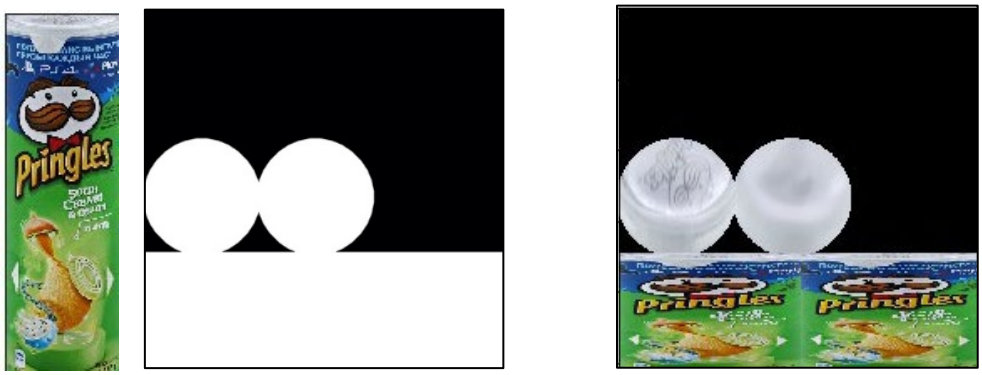
На вход данная функция принимает трехканальное изображение, а именно объект numpy.array размера (h,w,3). На рисунке 19 приведен пример работы такой функции:

Вход Выход

Рис.19. Результат работы функции кадрирования

После кадрирования необходимо отобразить текстуру на UV-шаблон примитива. В случае примера выше таким шаблоном будет развертка цилиндра. Обозначенное отображение различается в зависимости от формы объекта и должно быть имплементировано для каждого примитива отдельно. Ниже приведены функции отображения для примитивов цилиндра и подушки:

```
def pull_cylinder(img, patt):

    h = 0
    for i in range(patt.shape[0]):
        if (patt[patt.shape[0] - i - 1,0] == [0,0,0]).all():
            h = i
            break

    cap = img[0:50,:]
    cap = cv.resize(cap,(h,h))

    for i in range(h):
        for j in range(h):
            if(patt[patt.shape[0]-2*h+i,j] != [0,0,0]).all():
                patt[patt.shape[0]-2*h+i,j] = cap[i,j]

    cap = cv.medianBlur(cap,51)

    for i in range(h):
        for j in range(h):
            if(patt[patt.shape[0]-2*h+i,j+h] != [0,0,0]).all():
                patt[patt.shape[0]-2*h+i,j+h] = cap[i,j]

    img = cv.resize(img,(int(patt.shape[0]/2),h))

    patt[patt.shape[0]-h:patt.shape[0],0:int(patt.shape[0]/2)] = img
    patt[patt.shape[0]-h:patt.shape[0],int(patt.shape[0]/2):int(patt.shape[0])] = img

    return patt
```



Входные данные (текстура + UV-шаблон) Выходные данные (UV-текстура)

Рис.20. Пример работы функции отображения текстуры на развертку шаблона «цилиндр»

```
def pull_pillow(img, patt):

    h = int(patt.shape[0]/2)
    img = cv.resize(img,(h,h))

    blur = img
    blur = cv.flip(blur,0)

    for i in range(h):
        for j in range(h):
            if(patt[i,j] != [0,0,0]).all():
                patt[i,j] = img[i,j]

    for i in range(h):
        for j in range(h):
            if(patt[i+h,j] != [0,0,0]).all():
                patt[i+h,j] = blur[i,j]

    return patt
```

Входные данные (текстура + UV-шаблон) Выходные данные (UV-текстура)

Рис.21. Пример работы функции отображения текстуры на развертку шаблона «подушка»

Последним шагом на этапе генерации модели является скалирование каждой полученной модели на табличный размеры (рис. 22):

Исходная модель Нормированная модель

Рис.22. Скалирование модели

Таким образом можно большое количество различных моделей разной степени сложности. Главным преимуществом такого подхода является возможность эффективно генерировать модели, который можно приблизить простыми примитивами. Тем не менее, такой подход может быть применим даже для объектов со сложной геометрией. Для этого нужно смоделировать подходящий примитив и имплементировать

функцию генерации UV-текстуры для этого примитива.

2.3.4. Генерация сцен

Обычно под планограммой принято понимать план-схему выкладки товара на конкретном торговом оборудовании магазина. В более общем смысле под этим словом можно понимать и план-схему расстановки каких-либо объектов в среде.

К сожалению, задача генерации сцен очень зависит от специфики контекста. Решить такую задачу в общем виде для любых сцен и объектов не представляется возможным. Тем не менее, расстановку товарных единиц по продуктовым полкам магазина смоделировать довольно просто. Ниже приведен пример построения такого генератора.

Определим список _tree, который будет содержать данные об упаковке товаров. Каждый элемент списка содержит высоту "height" и ширину "width" упаковки (для планограммы достаточно фронтальной 2D проекции), а также цвет "color" для визуализации.

```
_tree = [
    {
        "width": 8,
        "height": 30,
        "color": (128, 128, 255)
    },
    . . .
    {
        "width": 20,
        "height": 30,
        "color": (128, 255, 255)
    }
]
```

Также определим структуру для самого стеллажа с товарами, содержащую ширину "width", список полок "height" с их высотами, а также толщину перегородки "wall_width":

```
_shelf = {
    "width": 150,
    "height": [
        40, 20, 20, 20, 20, 40
    ],
    "wall_width": 4
}
```

Функция genSchema() выполняет построение планограммы по параметрам shelf и tree. Сперва для каждой полки проходит проверка условия, по которому высота товара должны быть меньше высоты полки. После чего, функция getIndexes() каждой полке случайным образом приписывает товарные единицы, используя равномерное распределение.

```
def genSchema(shelf, tree):
    # Расчет полной высоты полки
    full_height = sum(shelf["height"]) + shelf["wall_width"] * len(shelf["height"])

    # Инициализация изображения планограммы
    schema = np.zeros((full_height, shelf["width"], 3), dtype='uint8')

    current_heigth = -shelf["wall_width"] // 2

    for n in range(len(shelf["height"])):

        current_heigth += shelf["height"][n] + shelf["wall_width"]
        current_width = 0

        filtered_tree = list(filter(lambda x: x["height"] < shelf["height"][n], tree))
        item_indexes = getIndexes(filtered_tree, shelf["width"])

        for item_index in item_indexes:

            # Отрисовка местоположения товарных единиц
            cv2.rectangle(schema,
                          (current_width,
                                        current_heigth - shelf["wall_width"] + 1),
                          (current_width + filtered_tree[item_index]["width"],
                           current_heigth - filtered_tree[item_index]["height"]),
                          filtered_tree[item_index]["color"], -1)

            current_width += filtered_tree[item_index]["width"] + random.randint(2, 4)

            if current_width > shelf["width"] - 10:
                break

        # Отрисовка стенок между партициями
        cv2.line(schema,
                 (0, current_heigth),
                 (shelf["width"], current_heigth),
                 (128, 128, 128),
                 shelf["wall_width"])

    return schema

def getIndexes(filtered_tree, shelf_width):
    probabilities = [random.randint(1, 5) for i in range(len(filtered_tree))]
    p_sum = sum(probabilities)
    capacities = list(map(lambda i: (shelf_width * (probabilities[i] / p_sum)) // filtered_tree[i]["width"], range(len(probabilities))))

    indexes = []
    for i in range(len(capacities)):
        for j in range(int(capacities[i])):
            indexes.append(i)

    return indexes
```

На рисунке ниже можно увидеть примеры работы такого генератора при разных параметрах

_shelf["width"] _shelf["height"]:

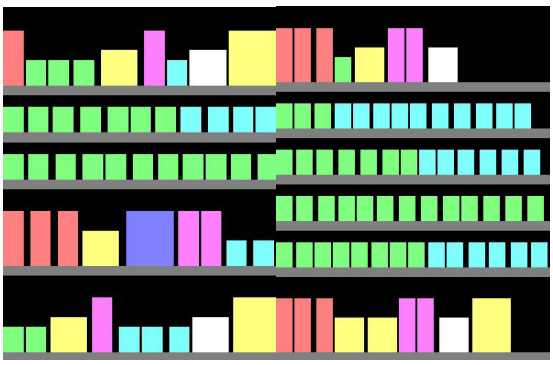


Рис.23. Визуализация генерации планограмм

2.3.5. Рендеринг

По полученным выше планограммам модели расставляются на сцене. Дополнительно выставляется начальное положение камеры, ограничения на перемещения, параметры освещения (положение и типа источников света, цветовая температура, интенсивность). После чего осуществляется рендеринг изображений и обрамляющих прямоугольников (рис. 24).

Рис.24. Пример рендера на движке Unreal Engine 4

3. Заключение

3.1. Обучение на сгенерированных данных

3.1.1. Описание архитектуры

В качестве целевой модели была выбрана сверточная нейронная сеть MobileNet-V1 с FPN надстройкой для семантической сегментации, так как модели семантических сегментаторов проще обучаются и отлаживаются в сравнении с детекторами, в которых проводится регрессия координат рамок.

На вход нейросеть принимает трехканальное изображение в виде тензора размера (224,224,3). Выход представляет собой тензор размера (56,56,C), где C=10 – количество классов. Единовременно за один шаг обучения на выход подаются C бинарных масок. Каждая маска соответствует своему классу и локализует объект этого класса на изображении. А во время прямого распространения сигнала нейросеть пытается эти маски предсказать.

Для подготовки обучающего набора данных был имплементирован класс DataGenerator (см. приложение Б). Этот класс содержит методы, которые из пар [изображение, список обрамляющих прямоугольников для каждого объекта на изображении] формирует пары [изображение, список масок для каждого класса]. Также класс содержит методы которые дополнительно проводят аугментацию, делая преобразования кадрирования и поворотов на небольшие углы, тем самым увеличивая количество примеров.

3.1.2. Результаты тестирования. Оценка качества

Введем основные метрики, которые были использованы для оценки качества работы обученной модели. Для простоты сперва рассмотрим классическую задачу бинарной классификации. В этом случае модель может дать лишь один из двух возможных ответов – класс обнаружен (**Positive**), класс не обнаружен (**Negative**) (рис. 25). Также при обучении мы можем оценить работу классификатора, и указать верно ли был дан ответ (**True Positive** или **True Negative**), либо неверно (**False Positive** или **False Negative**).

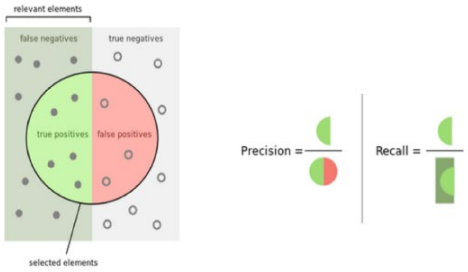


Рис.25. Метрики precision и recall

Точность (Precision) показывает долю верных ответов среди положительных и определяется как:

$$Precision = \frac{TP}{TP + FP}$$

Полнота (Recall) показывает долю всех обнаруженных положительных ответов среди всех возможных:

$$Recall = \frac{TP}{TP + FN}$$

Более общей метрикой является средняя точность или Average Precision (**AP**). Иногда ее называют PR-AUC (площадь под кривой «точность-полнота»). На множестве всех примеров валидационной выборки строится зависимость

$$Precision = Precision(Recall).$$

Тогда имеем:

$$AP = \int_0^1 p(r)dr$$

Теперь положим, что мы имеем задачу классификации на N классов. Очевидно, такую задачу можно свести к N подзадачам бинарной классификации. В таком случае можно посчитать AP для каждого класса,

после чего усреднить. Такая метрика называется **Mean Average Precision (mAP)**, она получила широкое распространение в оценке моделей компьютерного зрения, в особенности задач детектирования объектов:

$$mAP = \frac{1}{N}\sum_{i=1}^{N} AP_i$$

В задаче детектирования помимо ошибки классификации есть еще ошибка локализации, поэтому на расчет метрики дополнительно накладываются условия верной локализации. Считается, что объект класса C верно был задетектирован (случай True Positive), если выполнены следующие условия:

1. Спрогнозированная моделью вероятность $E[P_C]$ для класса C больше заданного порога T_P:
$$E[P_C] > T_P$$
2. Мера Жаккара IoU для истинного и предсказанного обрамляющих прямоугольников B_{true}, B_{pred} больше заданного порога T_{loc}:
$$IoU(B_{true}, B_{pred}) > T_{loc}$$

Выставление порога зависит от конкретной задачи и влияет на точность и полноту. Часто бывает так, что допускается неверный пропуск детектором некоторых объектов (FN) в угоду точности, как например в данной задачи детектирования товарных единиц на полках. В этом случае обычно пороги завышают.

Но бывают и другие задачи, например, связанные с медицинской диагностикой заболеваний, где более важна полнота. В этом случае часто лучше дать ложно-положительный ответ - болезнь есть (FP), чем пропустить ее (FN).

В настоящей работе производится попытка оценить не модель, а данные, на которых модель обучается. С этой целью на валидационном датасете, *полностью составленном из натуральных данных*, была посчитана метрика mAP(m,n), где m,n количество синтетических (сгенерированных) и натуральных примеров в обучающей выборке соответственно. Пороги были зафиксированы на уровнях $T_P = 0.95$, $T_{loc} = 0.5$. Результаты можно увидеть на рисунке 26.

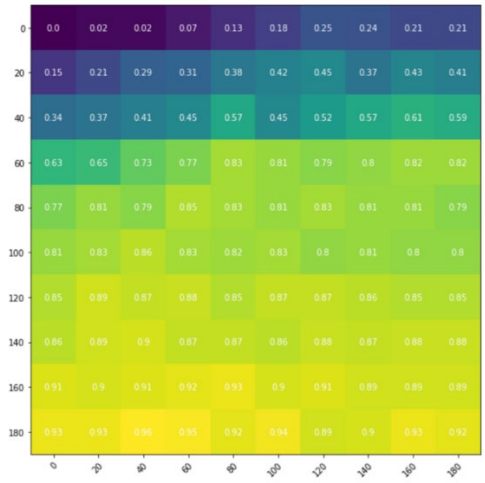


Рис.26. Тепловая карта метрики mAP на натуральном валидационном наборе в зависимости от количества сгенерированных примеров (ось X) и натуральных примеров (ось Y)

Полученные результаты можно интерпретировать следующим образом:

- При небольшом количестве примеров (m + n) качество работы модели мало зависит от соотношения синтетических и натуральных данных при обучении.
- С ростом количества примеров наблюдается появление зависимости от соотношения данных. Очевидно, метрика быстрее всего растет на выборке состоящей полностью из натуральных данных. Однако, на выборке, состоящей почти полностью из сгенерированных данных, метрика также растет, хотя и медленнее.
- При большом количестве примеров разница между натуральными данными и синтетическими становится все более заметной. На искусственных данных метрика выходит на плато примерно после 40-80 примеров, тогда как на натуральных данных продолжает расти, что можно объяснить малым разнообразием признаков сгенерированных примеров.

Очевидно, что обучение на правильно составленном натуральном наборе данных будет всегда лучше по качеству. Однако видно, что при довольно большой доле искусственных данных (60%) можно добиться значительного **mAP (80%)**. Во многих задачах этого может быть достаточно, чтобы составить начальный набор, на основе которого можно собрать и разметить большее количество уже натуральных данных.

3.1.3. Выводы

В данной работе были описан ряд современных методов решения задач детектирования объектов. Была выделена одна из ключевых проблем, которая связана с подготовкой обучающего набора данных, рассмотрены существующие подходы к решению данной проблемы, а также предложены алгоритмы автоматической генерации таких данных.

В ходе тестирования разработанных алгоритмов в контексте прикладной задачи детектирования товарных единиц на продуктовых полках выяснилось, что в ряде случаев предложенный подход может оказаться весьма эффективным и имеет потенциал к развитию.

Ниже перечислены факторы, при которых обучение на сгенерированных данных может оказаться целесообразным:

- количество классов большое или признаки классов часто меняются (проблема актуальности данных);
- объекты классов имеют простую геометрическую форму;
- объект класса представляет собой твердое недеформируемое тело;
- сцены плотно упакованы (большое количество объектов в кадре);
- объекты всегда находятся в одном контексте (полки магазинов, складские помещения и др.).

4. Благодарность

Исследование выполнено при финансовой поддержке РФФИ в рамках научных проектов 18-07-00225, 18-07-0909 и 18-07-01111.

Список информационных источников

[1] Liu, L., Ouyang, W., Wang, X. et al. Deep Learning for Generic Object Detection: A Survey. Int J Comput Vis 128, 261–318 (2020)
[2] Everingham, M., Eslami, S.M.A., Van Gool, L. et al. The PASCAL Visual Object Classes Challenge: A Retrospective. Int J Comput Vis 111, 98–136 (2015)
[3] Russakovsky, O., Deng, J., Su, H. et al. ImageNet Large Scale Visual Recognition Challenge. Int J Comput Vis 115, 211–252 (2015)
[4] Lin TY. et al. (2014) Microsoft COCO: Common Objects in Context. In: Fleet D., Pajdla T., Schiele B., Tuytelaars T. (eds) Computer Vision – ECCV 2014. ECCV 2014. Lecture Notes in Computer Science, vol 8693. Springer, Cham
[5] Felzenszwalb, P.F., Huttenlocher, D.P. Efficient Graph-Based Image Segmentation. International Journal of Computer Vision 59, 167–181 (2004)
[6] Achanta, R., et al.: SLIC superpixels. Epfl (2010)
[7] https://arxiv.org/abs/1406.2661
[8] https://neuromation.io/