

Нейронный подход к глубокому обучению

Нейронный подход к глубокому обучению основан на двух главных идеях. Во-первых, мозг – доказательный пример того, что разумное поведение возможно. Поэтому концептуально прямой путь к построению искусственного интеллекта состоит в том, чтобы проанализировать с вычислительной точки зрения принципы работы мозга и воспроизвести его функциональность. Идею этого направления можно сформулировать следующим образом: Единственный объект, способный мыслить, – это человеческий мозг. Поэтому любое «мыслящее» устройство должно каким-то образом воспроизводить его структуру. Во-вторых, интересно понять, как работает мозг и какие принципы лежат в основе разума человека, поэтому модели машинного обучения, проливающие свет на эти фундаментальные научные вопросы, полезны вне зависимости от их применимости к конкретным инженерным задачам [1].

Современный термин «глубокое обучение» выходит за рамки нейробиологического взгляда на модели машинного обучения. В нем заложен более общий принцип обучения нескольких уровней композиции, применимый к системам машинного обучения, не обязательно устроенным по примеру нейронов (рис.1).



Рис. 1. Искусственный интеллект, машинное и глубокое обучение

Понять сущность глубокое обучение проще всего в исторических ключевых тенденциях [1]:

- у глубокого обучения долгая и богатая история, но оно фигурировало под разными названиями, отражающими различные философские воззрения, в связи с чем его популярность то усиливалась, то ослабевала;

- полезность глубокого обучения возросла, когда увеличился объем доступных обучающих данных;
- размер моделей глубокого обучения увеличивался по мере того, как совершенствовалась компьютерная инфраструктура (программная и аппаратная);
- со временем с помощью глубокого обучения удавалось решать все более сложные задачи со все большей точностью.

Таким образом, глубокое обучение определяется как особый раздел машинного обучения (рис.1): новый подход к поиску представления данных, делающий упор на изучение последовательных слоев все более значимых представлений. Под глубиной в глубоком обучении не подразумевается более глубокое понимание, достигаемое этим подходом; идея заключается в многослойном представлении. Количество слоев, на которые делится модель данных, называют глубиной модели.

Современное глубокое обучение часто вовлекает в процесс десятки и даже сотни последовательных слоев представления – и все они автоматически определяются под воздействием обучающих данных. Между тем другие подходы к машинному обучению ориентированы на изучении одного-двух слоев представления данных; по этой причине их иногда называют поверхностным обучением.

В сфере глубокого обучения за последние годы удалось добиться заметных успехов, надеемся, что ожидания на будущее десятилетие не превышают вероятных достижений. Уже сейчас многие значительные реализации находят применения в различных промышленных сферах и областях.

Глубокое обучение достигло уровня общественного внимания и инвестиций, невиданных прежде в истории искусственного интеллекта, особенный интерес представляют сверточные нейронные сети.

1.1. Биологическая мотивация

Идея сверточных нейронных сетей во многом мотивирована исследованиями о зрительной коре головного мозга. Человек с легкостью решает довольно сложные задачи распознавания образов, он без видимых усилий различает тысячи объектов, составляющих его окружение, несмотря на изменение расстояния, ракурса, перспективы и освещения.

При этом человечество давно задумывались о том, как именно устроено зрение. Глаз как оптический прибор изучали Леонардо да Винчи,

Иоганн Кеплер и многие другие великие физики, отмечавшие его выдающиеся оптические свойства.

Однако с точки зрения устройства сверточных нейронных сетей научный интерес представляет процесс, который происходит с прочитанным с сетчатки изображением.

Зрительный нерв (рис.2) – толстый пучок аксонов ганглионарных клеток, по которому информация с сетчатки доходит до мозга, – отмечали еще средневековые анатомы, и его важность для зрения была очевидной.



Рис.2. Строение зрительного нерва

Зрительный нерв входит в таламус, отдел мозга, обрабатывающий информацию от органов чувств, там первичная обработка происходит в так называемом латеральном колленчатом теле, а затем зрительная информация поступает в собственно зрительную кору (рис.3).

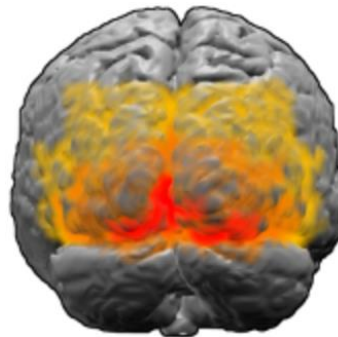


Рис.3. Мозг человека, вид сзади. Красным цветом обозначено первичная зрительная кора; оранжевым и жёлтым —экстрастриарная зрительная кора.

Зрительная кора расположена сзади, в затылочной доле головного мозга. Понятие зрительная кора включает первичную зрительную кору (также называемую зрительной зоной V1) и экстрастриарную зрительную кору – зоны V2, V3, V4, V5. Иногда выделяют зоны V6 и V7. Зоны отличаются друг от друга физиологией, архитектурой, обособленным положением в коре, и несомненно они различаются и по своим функциям. Хотя на самом деле функциональная специализация зон пока что до конца не ясна, но по нынешнему представлению [2]:

V1 – в этой зоне выделяются локальные признаки небольших участков считанного с сетчатки изображения;

V2 – продолжает выделять локальные признаки, слегка обобщая их и добавляя бинокулярное зрение (то есть стереоэффект от двух глаз);

V3 – зона распознается цвет, текстуры объектов, появляются первые результаты их сегментации и группировки;

V4 – распознает геометрические фигуры и очертания объектов, пока несложных; кроме того, именно здесь наиболее сильна модуляция посредством нашего внимания: активация нейронов в V4 не равномерна по всему полю зрения, а сильно зависит от того, на что мы осознанно или неосознанно обращаем внимание;

V5 – в основном занимается распознаванием движений, пытаюсь понять, куда и с какой скоростью передвигаются в зоне видимости те самые объекты, очертания которых выделились в зоне V4;

V6 – зона обобщает данные о всей картинке, она реагирует на изменения по всему полю зрения и изменения в картинке вследствие того, что передвигается сам человек;

V7 – в зоне происходит распознавание сложных объектов, в частности человеческих лиц.

Такая функциональная специализация – это первое замечание о зрительной коре, которое соответствует глубоким нейронным сетям: более высокие уровни нужны для того, чтобы выделять более общие признаки, соответствующие абстрактным свойствам входа, а на нижних уровнях признаки более конкретные. В сверточных сетях эффект будет тот же самый. Но есть и другие интересные свойства архитектуры зрительной коры, которые нашли отражение в машинном обучении.

При этом среди перечисленных зон иерархия не такая уж строгая: есть масса прямых связей, когда, например, нейроны из зоны V1 подаются на вход не только в зону V2, но и напрямую в зону V5; это находит отражение в сверточных архитектурах.

При этом между нейронами в мозге всегда присутствует очень сильная обратная связь от более высоких уровней к более низким. Например, современные исследования предполагают, что внимание, которое зарождается в зоне V4, потом переходит обратно к зонам V2 и V1, где также присутствует сильная модуляция на основе зрительного представления.

В искусственных нейронных сетях тоже вводят подобные механизмы. Однако в машинном обучении обратная связь находится на «низком» уровне, пока нет математической модели на подобии мозга управлять ею – одна из важнейших открытых задач нейробиологии. Пока многими специализациями искусственные нейронные сети не снабжены.

Из всей зрительной коры наиболее хорошо изучена зона V1, первичная зрительная кора: она ближе всего к собственно входам, достаточно просто устроена, и там проще понять, за что «отвечают» отдельные нейроны и как они друг с другом взаимодействуют; именно с зоной V1 были связаны самые значительные результаты. Зона V1 состоит из шести уровней нейронов, которые распознают [2]:

- *ориентацию*, то есть нейрон реагирует, например, на то, что освещенность вдоль диагонали изображения высокая, а по двум другим углам низкая; или на то, что освещенность нижней части изображения выше, чем верхней; таким образом нейроны распознают границы изображений;
- *пространственную частоту*, то есть то, насколько часто меняется освещенность в пределах рецептивного поля нейрона;
- *направление*: человеку интересна не только и не столько статичная картинка, сколько происходящие в ней движения; нейроны умеют «запоминать» и сравнивать предыдущие входы с последующими, распознавая таким образом направление не только в пространстве, но и во времени; аналогично распознается и временная частота;
- *различие между глазами*: в V1 у каждого нейрона два рецептивных поля, для каждого глаза, распознают различия между тем, что видят левый и правый глаз;
- *цвет*, относительно которого нейроны обычно распознают одно из трех основных направлений: красный – зеленый, синий – желтый и черный – белый.

Видно, обработка изображений в мозге человека устроена как глубокая нейронная сеть. И так только зоне V1 выделяют шесть уровней, а ведь сигнал по ходу движения проходит через сразу несколько зон, начиная с латерального колленчатого тела (в котором, кстати, тоже шесть уровней) и заканчивая высокоуровневыми зонами V6 и V7. Это нашло отражение в сверточных нейронных сетях для обработки изображений — это самые глубокие из существующих сетей.

1.2. Изображение

Обычно цветные картинки, подающиеся на вход нейронной сети, представлены в виде нескольких прямоугольных матриц, каждая из которых задает уровень одного из цветовых каналов в каждом пикселе изображения (рис. 3).

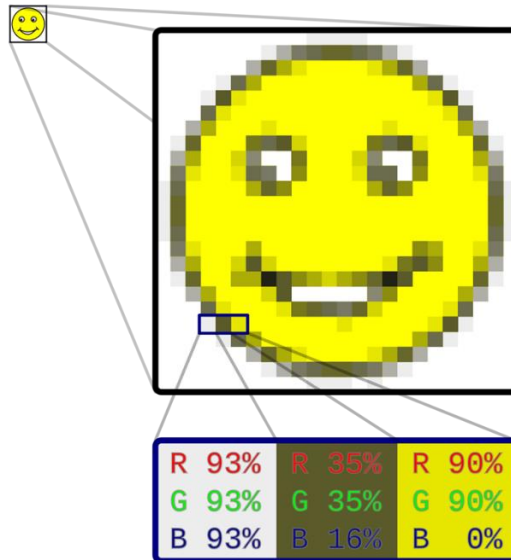


Рис. 3. Растровое изображение

Например, картинка размером 200×200 пикселей – это 120 000 чисел, три матрицы интенсивностей размером 200×200 каждая. Если изображение черно-белое, как, например, в MNIST (**эмнайэстей**) (Modified National Institute of Standards and Technology – объёмная база данных образцов рукописного написания цифр.), то такая матрица будет одна. А если это не простая картинка, а, скажем, результат изображающей масс-спектрометрии, когда в каждом пикселе находится целый спектр, то матриц может быть очень много. Но в любом случае мы будем предполагать, что в каждом пикселе входного изображения стоит некоторый тензор (обычно одномерный, то есть вектор чисел), и его компоненты называются каналами.

Все современные системы машинного обучения используют тензоры в качестве основной структуры данных. При этом тензоры являются фундаментальной структурой данных — настолько фундаментальной, что это отразилось на названии библиотеки Google TensorFlow (**Гугл Тензор Флей**) [3].

Фактически тензор — это контейнер для данных, практически всегда числовых. Другими словами, это контейнер для чисел. Вы уже знакомы с матрицами, которые являются двумерными тензорами: тензоры — это

обобщение матриц с произвольным количеством измерений (обратите внимание, что в терминологии тензоров измерения часто называют осями).

Приведем примеры тензоров [3]:

- Скаляр (тензором нулевого ранга) – тензор, содержащий единственное число. Количество осей тензора также называют его рангом.
- Вектор (тензор первого ранга) – одномерный массив чисел. Тензор первого ранга имеет единственную ось.
- Матрица (тензор второго ранга) – массив векторов. Матрица имеет две оси (часто их называют строками и столбцами). Матрицу можно представить, как прямоугольную таблицу с числами.
- Тензоры третьего и высшего рангов – упакованные матрицы в новый массив, то получится трехмерный тензор. Упаковав трехмерные тензоры в массив, получится четырехмерный тензор и т. д. В глубоком обучении чаще всего используются тензоры от нулевого ранга до четырехмерных, но иногда, например, при обработке видеоданных, дело может дойти и до пятимерных тензоров.

Данные, которыми приходится манипулировать, почти всегда будут относиться к одной из следующих категорий: векторные данные – двумерные тензоры с формой (*образцы, признаки*); временные ряды или последовательности – трехмерные тензоры с формой (*образцы, метки_времени, признаки*); изображения – четырехмерные тензоры с формой (*образцы, высота, ширина, цвет*) или с формой (*образцы, цвет, высота, ширина*); видео — пятимерные тензоры с формой (*образцы, кадры, высота, ширина, цвет*) или с формой (*образцы, кадры, цвет, высота, ширина*).

Изображения имеют три измерения: высоту, ширину и цвет. Даже при том, что черно-белые изображения (как в наборе данных MNIST) имеют только один канал цвета и могли бы храниться в двумерных тензорах, по соглашениям тензоры с изображениями всегда имеют три измерения, где для черно-белых изображений отводится только один канал цвета. Соответственно, пакет со 128 черно-белыми изображениями, имеющими размер 256×256 , можно сохранить в тензоре с формой (128, 256, 256, 1), а пакет со 128 цветными изображениями – в тензоре с формой (128, 256, 256, 3) (рис. 4).

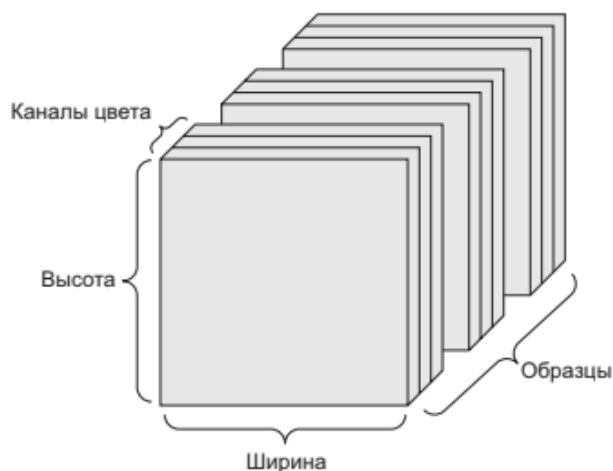


Рис. 4. Четырехмерный тензор с изображениями
(в соответствии с соглашением «канал следует первым»)

В отношении форм тензоров с изображениями существует два соглашения: соглашение канал следует последним (используется в TensorFlow) и соглашение канал следует первым (используется в Theano). Фреймворк машинного обучения TensorFlow, разработанный компанией Google, отводит для цвета последнюю ось: (образцы, высота, ширина, цвет). А библиотека Theano отводит для цвета ось, следующую сразу за осью пакетов: (образцы, цвет, высота, ширина). Если следовать соглашению, принятому в Theano, предыдущие примеры тензоров будут иметь форму (128, 1, 256, 256) и (128, 3, 256, 256). Фреймворк Keras поддерживает оба формата.

1.3. Операция свертки

Сверточная нейронная сеть, предложенная Яном Лекуном (LeCun) в 1988 – это специальный вид нейронной сети для обработки данных с сеточной топологией. Однако справедливости ради надо сказать, что первой настоящей сверточной сетью, позаимствовавшей для информатики воплощенные природой в зрительной коре идеи, был неоконитрон Кунихико Фукусимы, появившийся в 1979–1980 годах. Впрочем, Фукусима не использовал градиентный спуск и вообще обучение с учителем, а его работы были довольно прочно забыты.

Основная идея сверточной сети состоит в том, что обработка участка изображения очень часто должна происходить независимо от конкретного расположения этого участка. Эта ключевая идея наделяет сверточные нейронные сети двумя важными свойствами [4]:

1. Шаблоны, которые они изучают, являются инвариантными (неизменными) в отношении переноса. После изучения определенного

шаблона в правом нижнем углу картинке сверточная нейронная сеть сможет распознавать его повсюду: например, в левом верхнем углу. Полносвязной сети пришлось бы изучить шаблон заново, если он появляется в другом месте. Это увеличивает эффективность сверточных сетей в задачах обработки изображений (потому что видимый мир по своей сути является инвариантным в отношении переноса): таким сетям требуется меньше обучающих образцов для получения представлений, обладающих силой обобщения.

2. Они могут изучать пространственные иерархии шаблонов (рис. 5). Первый сверточный слой будет изучать небольшие локальные шаблоны, такие как края, второй – более крупные шаблоны, состоящие из признаков, возвращаемых первым слоем, и т. д. Это позволяет сверточным нейронным сетям эффективно изучать все более сложные и абстрактные визуальные представления (потому что видимый мир по своей сути является пространственно-иерархическим).

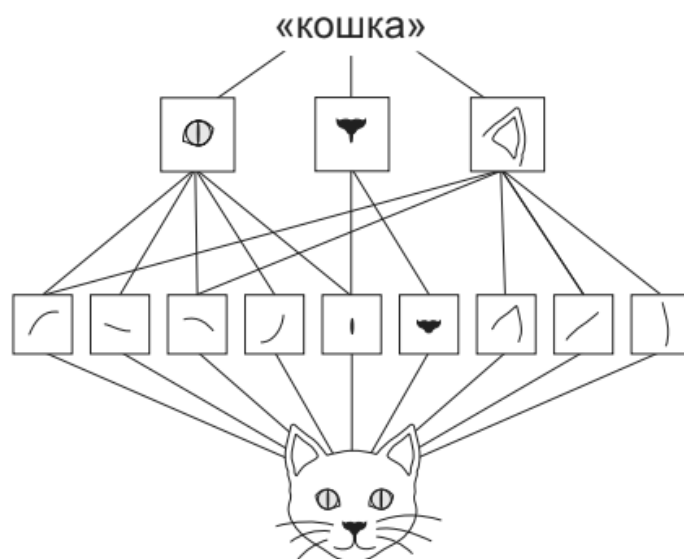


Рис. 5. Видимый мир формируется пространственными иерархиями видимых модулей.

На рис.5 гиперлокальные края объединяются в локальные объекты, такие как глаза или уши, которые, в свою очередь, объединяются в понятия еще более высокого уровня, такие как «кошка»

Свертка — это всего лишь линейное преобразование входных данных особого вида. Если x^l — карта признаков в слое под номером l , то результат двумерной свертки с ядром размера $2d + 1$ и матрицей весов W размера $(2d + 1) \times (2d + 1)$ на следующем слое будет таким [4]:

$$y_{i,j}^l = \sum_{-d \leq a,b \leq d} W_{a,b} x_{i+a,j+b}^l$$

где $y_{i,j}^l$ – результат свертки на уровне l , а $x_{i,j}^l$ – ее вход, то есть выход всего предыдущего слоя. Иначе говоря, чтобы получить компоненту (i,j) следующего уровня, применяется линейное преобразование к квадратному окну предыдущего уровня, то есть скалярно умножаем пиксели из окна на вектор свертки. На рис. 6 приведен пример свертки в применении к двумерному тензору.

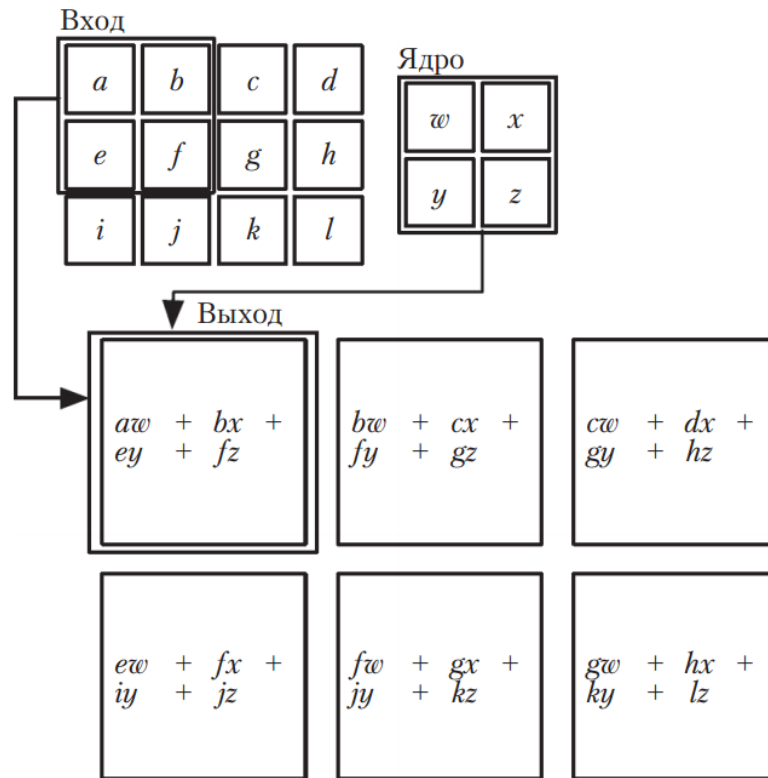


Рис. 6. Пример двумерной свертки.

На рис.6. выход ограничен только теми позициями, для которых ядро целиком укладывается в изображение; в некоторых контекстах такая свертка называется «допустимой». Прямоугольник с указывающими на него стрелками показывает, как левый верхний элемент выходного тензора образуется путем применения ядра к соответствующей левой верхней области входного тензора.

Приведем более конкретный пример, применим свертку с матрицей весов W размера 3×3 к матрице X размера 5×5 . Операция свертки обычно обозначается звездочкой *

$$\begin{pmatrix} 0 & 1 & 2 & 1 & 0 \\ 4 & 1 & 0 & 1 & 0 \\ 2 & 0 & 1 & 1 & 1 \\ 1 & 2 & 3 & 1 & 0 \\ 0 & 4 & 3 & 2 & 0 \end{pmatrix} * \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 2 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 9 & 5 & 4 \\ 8 & 8 & 10 \\ 8 & 15 & 12 \end{pmatrix}$$

Умножение подматрицы исходной матрицы X , соответствующей окну, и матрицы весов W – это не умножение матриц, а просто скалярное произведение соответствующих векторов. А всего окно умещается в матрице X девять раз, и получается (рис.7).

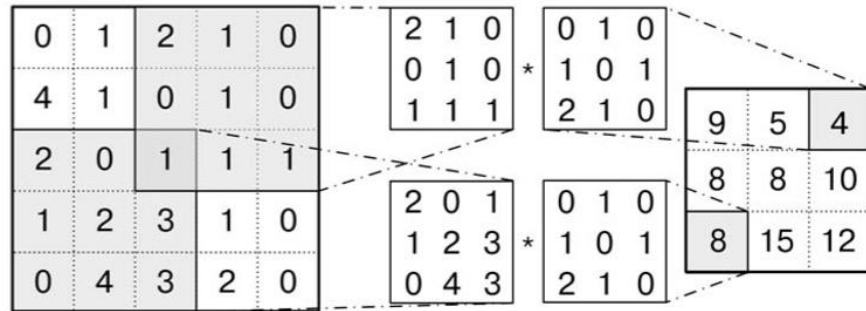


Рис. 7. Пример подсчета результата свертки: два примера подматрицы и общий результат

В терминологии сверточных сетей первый аргумент (в нашем примере функция X) называется входом, а второй (функция W) – ядром. Выход иногда называют картой признаков.

Если размер свертки будет выражаться четным числом, то в одном из случаев в пределах суммирования неравенство станет строгим. Это преобразование обладает следующими свойствами [1]:

- свертка сохраняет структуру входа (порядок в одномерном случае, взаимное расположение пикселей в двумерном и т.д.), так как применяется к каждому участку входных данных в отдельности;
- операция свертки обладает свойством разреженности, так как значение каждого нейрона очередного слоя зависит только от небольшой доли входных нейронов (а, например, в полносвязной нейронной сети каждый нейрон зависел бы от всех нейронов предыдущего слоя);
- свертка многократно переиспользует одни и те же веса, так как они повторно применяются к различным участкам входа.

Почти всегда после свертки в нейронной сети следует нелинейность, которая записывается следующим образом:

$$z_{i,j}^l = h(y_{i,j}^l).$$

В качестве функции h часто используют ReLU, особенно в очень глубоких сетях, но и классические гиперболические функции также используются.

Свойство разреженной связностью (разреженными весами) достигается это за счет того, что ядро меньше входа. Например, входное изображение может содержать тысячи или миллионы пикселей, но небольшие значимые признаки. Например, границы, можно обнаружить с помощью ядра, охватывающего всего десятки или сотни пикселей. Следовательно, нужно хранить меньше параметров, а это снижает требования модели к объему памяти и повышает ее статистическую эффективность. Кроме того, для вычисления выхода потребуется меньше операций. Все вместе обычно намного повышает эффективность сети [1].

Графически разреженная связность иллюстрируется на рис. 8.

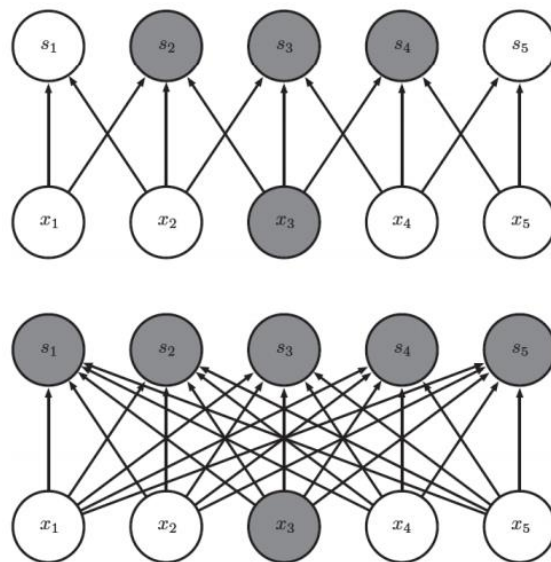


Рис. 8. Разреженная связность.

На рис.8. выделен один входной блок x_3 и выходные блоки в слое s , на которые этот блок влияет. Вверху. Если s образован сверткой с ядром ширины 3, то x_3 влияет только на три выхода. Внизу. Если s образован умножением на матрицу, то связность уже не разреженная, поэтому x_3 влияет на все выходы. В глубокой сверточной сети блоки нижних уровней могут косвенно взаимодействовать с большей частью сети, как показано на рис. 9. Это дает сети возможность эффективно описывать сложные взаимодействия между многими переменными путем составления из простых строительных блоков, каждый из которых описывает только разреженные взаимодействия [1].

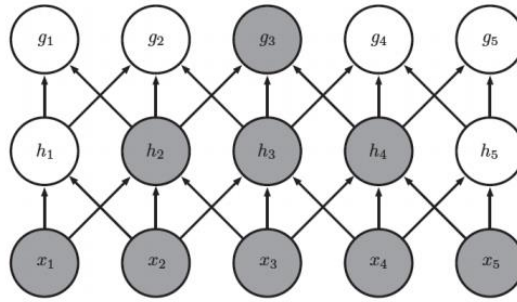


Рис. 9. Рецептивное поле блоков в глубоких слоях сверточной сети больше рецептивного поля в слоях, близких к поверхности.

Этот эффект усиливается, если сеть включает такие архитектурные особенности. Это означает, что прямые связи в сверточной сети действительно очень разрежены, блоки в глубоких слоях могут быть косвенно связаны со всем входным изображением или с большей его частью.

Под разделением параметров понимают, что один и тот же параметр используется в нескольких функциях модели. В традиционной нейронной сети каждый элемент матрицы весов используется ровно один раз при вычислении выхода слоя. Он умножается на один элемент входа, и больше мы к нему никогда не возвращаемся. Вместо употребления термина «разделение параметров» можно сказать, что в сети присутствуют **связанные веса**, поскольку значение веса, примененного к одному входу, связано со значением веса, примененного где-то еще. В сверточной нейронной сети каждый элемент ядра применяется к каждой позиции входа (за исключением, быть может, некоторых граничных пикселей – в зависимости от того, как решено обрабатывать границу). Разделение параметров означает, что вместо обучения отдельного набора параметров для каждой точки мы должны обучить только один набор [5].

Таким образом, свертка многократно эффективнее умножения матриц с точки зрения требований к памяти и статистической эффективности. Механизм разделения параметров наглядно изображен на рис. 10, где черными стрелками показаны связи, в которых участвует конкретный параметр в двух разных моделях. Вверху. Черными стрелками показано использование центрального элемента 3-элементного ядра в сверточной модели. Благодаря разделению параметров этот параметр используется для всех элементов входа. Внизу. Одиночная черная стрелка показывает использование центрального элемента матрицы весов в

полносвязной модели. Здесь никакого разделения параметров нет, поэтому параметр используется только один раз [1].

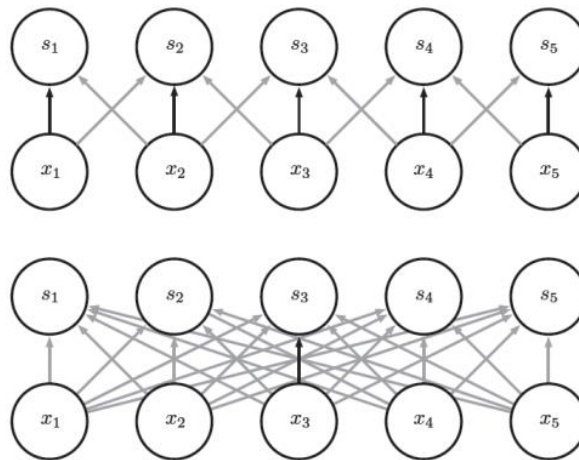


Рис. 10. Разделение параметров.

Практическое применение принципов (разреженная связность и разделение параметров) кардинально улучшают эффективность линейной функции при обнаружении границ в изображении.

В случае свертки специальный вид разделения параметров наделяет слой свойством, которое называется эквивариантностью относительно параллельного переноса. При работе с изображениями свертка создает двумерную карту появления определенных признаков во входном изображении. Если переместить объект во входном изображении, то его представление на выходе переместится на такую же величину. Это бывает нужно, когда мы знаем, что некоторая функция от небольшого числа пикселей полезна при применении к нескольким участкам входа. Иногда, в случае обработки изображений полезно обнаруживать границы в первом слое сверточной сети. Одни и те же границы встречаются более-менее везде в изображении, поэтому имеет смысл разделять параметры по всему изображению. Но в некоторых случаях такое глобальное разделение параметров нежелательно. Например, если обрабатываются изображения, которые были кадрированы, так чтобы в центре оказалось лицо человека, то, наверное, хотелось выделять разные признаки в разных точках – часть сети будет обрабатывать верхнюю часть лица в поисках бровей, а другая часть – искать подбородок в нижней части лица.

Свертка не эквивариантна относительно некоторых преобразований, например, масштабирования или поворота. Для обработки таких преобразований нужны другие механизмы. Существуют типы данных, которые нельзя обработать с помощью нейронных сетей, определяемых

путем умножения на матрицу фиксированной формы. Свертка позволяет обрабатывать некоторые данные такого рода.

1.4. Эффекты границ, дополнение и шаг свертки

При обсуждении свертки в контексте нейронных сетей обычно имеется в виду не стандартная операция дискретной свертки в том смысле, в каком она понимается с математической точки зрения. Практически используемые функции немного отличаются. Опишем эти различия и выделим некоторые полезные свойства функций, применяемых в нейронных сетях. Прежде всего, под сверткой в нейронных сетях понимается операция, состоящая из многих параллельных вычислений свертки (рис. 11).

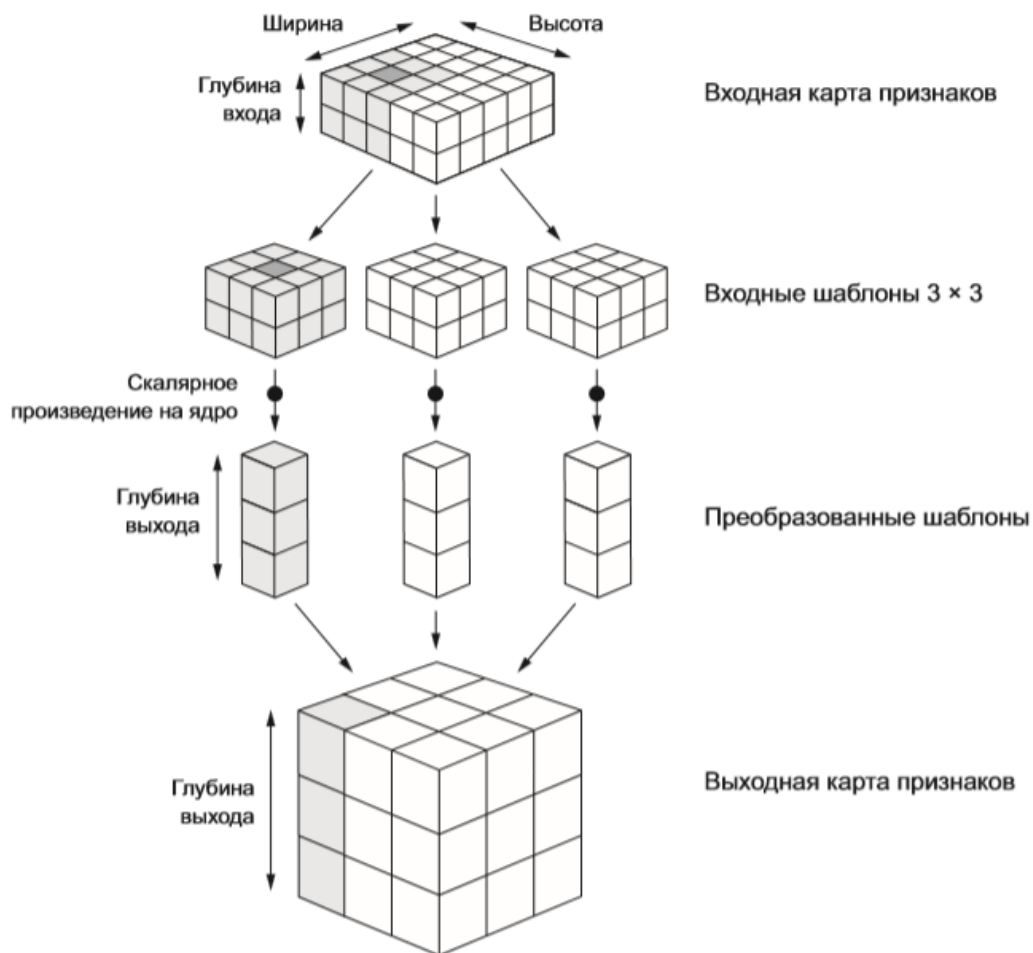


Рис. 11. Принцип действия свертки

Примечательно, что выходные ширина и высота могут отличаться от входных. На то есть две причины [4]:

- эффекты границ, которые могут устраняться дополнением входной карты признаков;
- использование шага свертки.

Раскроем более подробнее эти понятия.

Рассмотрим карту признаков 5×5 (всего 25 клеток). Существует всего 9 клеток, в которых может находиться центр окна 3×3 , образующих сетку 3×3 (рис. 12). Следовательно, карта выходных признаков будет иметь размер 3×3 . Она получилась сжатой: ровно на две клетки вдоль каждого измерения.

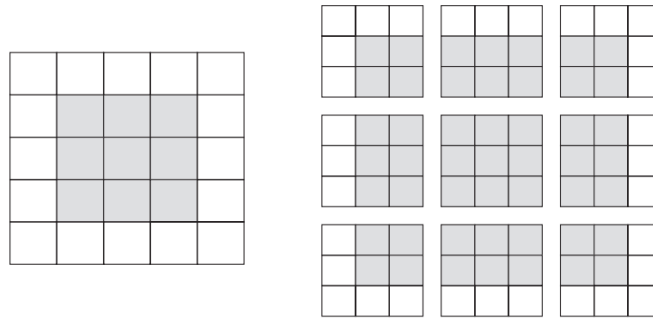


Рис. 12. Допустимые местоположения шаблонов 3×3 во входной карте признаков 5×5

Чтобы получить выходную карту признаков с теми же пространственными размерами, что и входная карта, можно использовать дополнение. Это важное свойство любой реализации сверточной сети – возможность неявно дополнять нулями вход V , чтобы расширить его (рис.13).

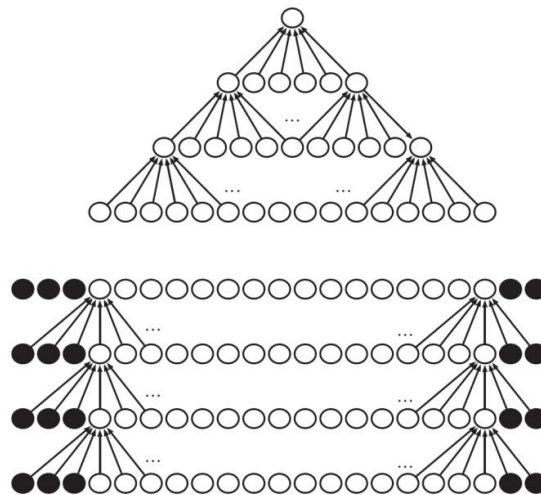


Рис. 13. Влияние дополнения нулями на размер сети.

Без этого ширина представления уменьшается на ширину ядра без одного пикселя в каждом слое. Дополнение входа нулями позволяет управлять шириной ядра и размером выхода независимо. Отсутствие дополнения, принуждает выбирать между быстрым уменьшением пространственной протяженности сети и использованием малых ядер – то

и другое значительно ограничивает выразительную мощность сети.

Дополнение заключается в добавлении соответствующего количества строк и столбцов с каждой стороны входной карты признаков, чтобы можно было поместить центр окна свертки в каждую входную клетку [5]. Для окна 3×3 нужно добавить один столбец справа, один столбец слева, одну строку сверху и одну строку снизу. Для окна 5×5 нужно добавить две строки (рис. 14).

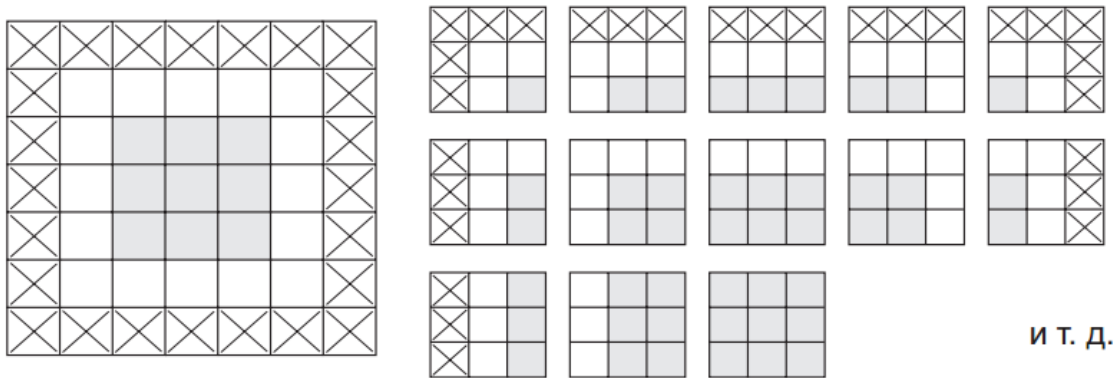


Рис. 14. Дополнение входной карты признаков 5×5 , чтобы получить 25 шаблонов 3×3

Заслуживают упоминания три частных случая дополнения нулями. Первый называется корректной сверткой – когда дополнение не используется вовсе, а ядру свертки разрешено занимать только те позиции, где ядро целиком уместается внутри изображения.. В таком случае каждый выходной пиксель является функцией одного и того же числа входных, поэтому поведение выходных пикселей несколько более регулярно. Но размер выхода при этом уменьшается на каждом слое. Если ширина входного изображения равна m , а ширина ядра – k , то ширина выхода будет равна $m - k + 1$. Скорость такого сжатия может быть очень велика, если используются большие ядра. Поскольку сжатие больше 0, число сверточных слоев в сети ограничено. По мере добавления слоев пространственная протяженность сети рано или поздно снизится до 1×1 , после чего новые слои нельзя считать сверточными.

Другой частный случай называется конгруэнтной сверткой – дополнение столькими нулями, чтобы размер выхода был равен размеру входа. В этом случае сеть может содержать столько сверточных слоев, сколько может поддержать оборудование, поскольку операция свертки не изменяет архитектурных возможностей следующего слоя. Однако входные пиксели в окрестности границы оказывают меньшее влияние на выходные,

чем пиксели, расположенные ближе к центру. Из-за этого граничные пиксели будут недостаточно представлены в модели.

Этот недостаток послужил основанием для еще одного крайнего случая, называемого полной сверткой, когда добавляется столько нулей, чтобы каждый пиксель посещался k раз в каждом направлении. В результате размер выходного изображения становится равен $m + k - 1$. В таком случае выходные пиксели вблизи границы являются функциями меньшего числа входных пикселей, чем пиксели в центре. Из-за этого трудно обучить единственное ядро, которое вело бы себя хорошо во всех позициях сверточной карты признаков. Обычно оптимальная степень дополнения нулями лежит между «корректной» и «конгруэнтной» сверткой.

Другой фактор, который может влиять на размер выходной карты признаков, – шаг свертки. В объяснениях предполагалось, что центральная клетка окна свертки последовательно перемещается в смежные клетки входной карты. Однако в общем случае расстояние между двумя соседними окнами является настраиваемым параметром, который называется шагом свертки и по умолчанию равен 1. Также имеется возможность определять свертки с пробелами (strided convolutions) – свертки с шагом больше 1. На рис. 15 можно видеть, как извлекаются шаблоны 3×3 сверткой с шагом 2 из входной карты 5×5 (без дополнения) [5].

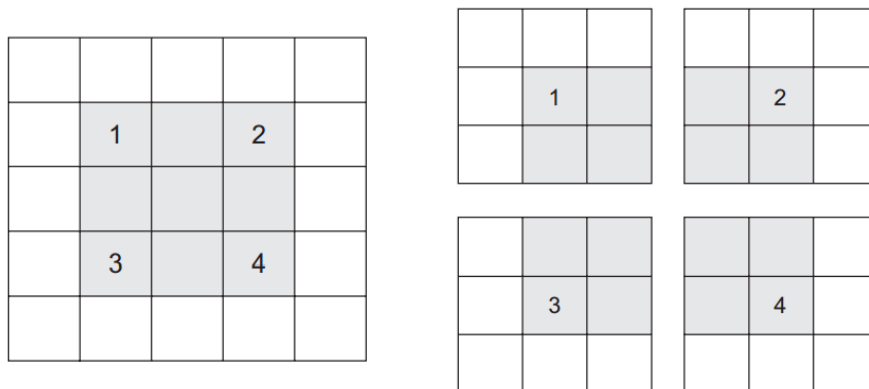


Рис. 15. Шаблоны 3×3 свертки с шагом 2×2

Использование шага 2 означает уменьшение ширины и высоты карты признаков за счет уменьшения разрешения в два раза (в дополнение к любым изменениям, вызванным эффектами границ). Таким образом, будем называть s шагом свертки пониженного разрешения. Можно также определить разные шаги по каждому направлению движения (рис. 16) [4].

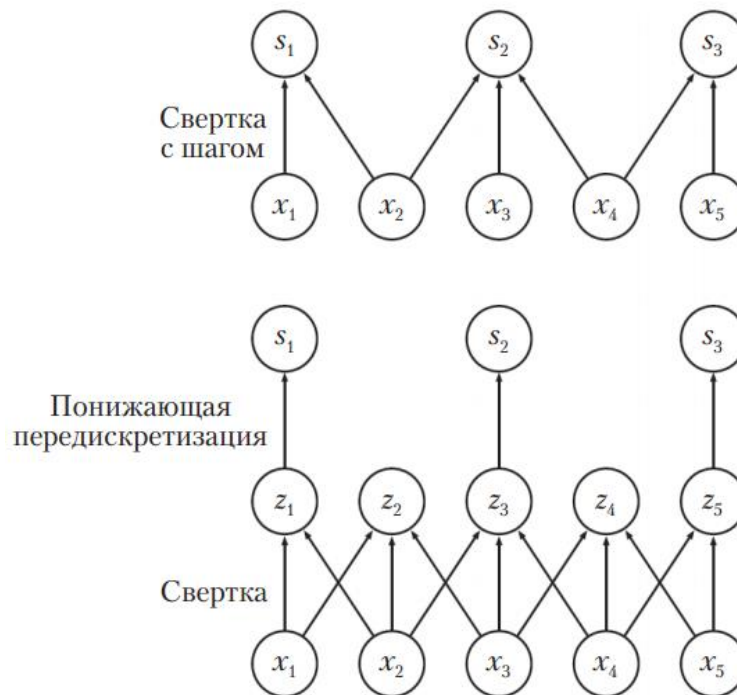


Рис. 16. Свертка с шагом.

В этом примере шаг равен 2. (Вверху) Свертка с шагом 2, реализованная в виде одной операции. (Внизу) Свертка с шагом больше 1 пикселя математически эквивалентна свертке с шагом 1, за которой следует понижающая передискретизация. Очевидно, что такой двухэтапный подход вычислительно расточителен, поскольку многие вычисленные значения затем отбрасываются

Свертки с пробелами редко используются на практике, хотя могут пригодиться в моделях некоторых типов, поэтому желательно знать и помнить об этой возможности.

1.5. Выбор максимального значения из соседних

В классическом сверточном слое, кроме линейной свертки и следующей за ней нелинейности, есть и еще одна операция: субдискретизация (pooling – по-русски ее иногда называют еще операцией «подвыборки», от альтернативного английского термина subsampling; встречается и слово «пулинг»)

Смысл субдискретизации прост: в сверточных сетях обычно исходят из предположения, что наличие или отсутствие того или иного признака гораздо важнее, чем его точные координаты. Например, при распознавании лиц сверточной сетью гораздо важнее понять, есть на фотографии лицо и с какого конкретно пикселя оно начинается и в каком заканчивается. Поэтому обобщают выделяемые признаки, потеряв часть информации об

их местоположении, но зато сократив размерность.

Обычно в качестве операции субдискретизации к каждой локальной группе нейронов применяется операция взятия максимума (max-pooling). Иногда встречаются и другие операции субдискретизации например взятие среднего, а не максимума. Однако именно максимум встречается на практике чаще всего и для большинства практических задач дает хорошие результаты [5].

$$x_{i,j}^{l+1} = \max_{-d \leq a \leq d, -d \leq b \leq d} z_{i+a,j+b}^l$$

Здесь d – это размер окна субдискретизации. Как правило, шаг субдискретизации и размер окна совпадают, то есть получаемая на вход матрица делится на непересекающиеся окна, в каждом из которых выбирается максимум; для $d = 2$ эта ситуация проиллюстрирована на рис.17, в.

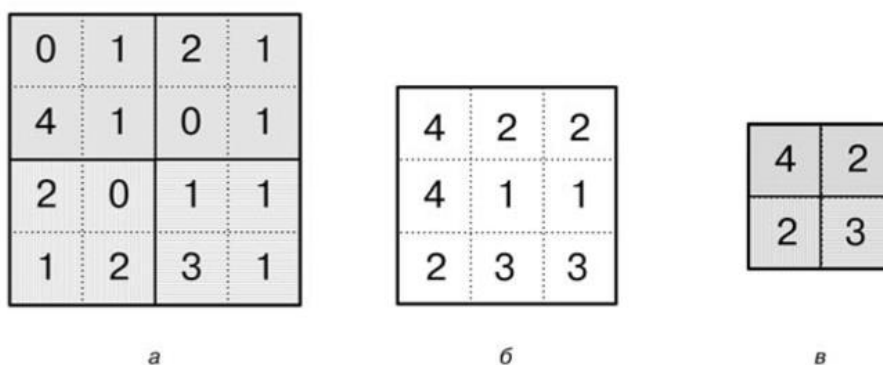


Рис. 17. Пример субдискретизации с окном размера 2×2 :

а – исходная матрица; б – матрица после субдискретизации с шагом 1;

в – матрица после субдискретизации с шагом 2.

Хотя в результате субдискретизации действительно теряется часть информации, сеть становится более устойчивой к небольшим трансформациям изображения в роде сдвига или поворота.

Субдискретизация позволяет сделать представление приблизительно инвариантным относительно малых параллельных переносов входа. Инвариантность относительно параллельного переноса означает, что если сдвинуть вход на небольшую величину, то значения большинства подвергнутых пулингу выходов не изменятся (рис. 18). Инвариантность физической величины означает её неизменность при изменении физических условий или по отношению к некоторым преобразованиям [3].

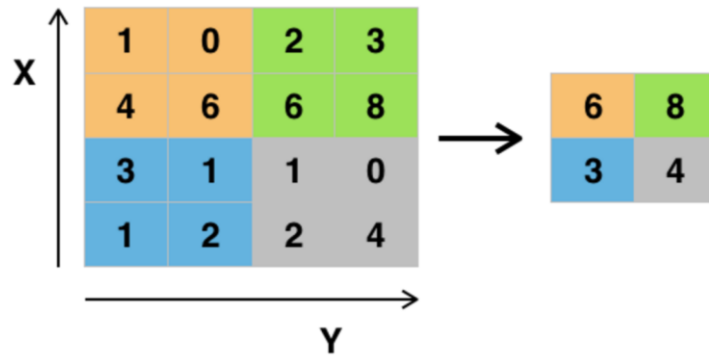


Рис. 18. Пример max-пулинга

Локальная инвариантность относительно параллельного переноса полезна, если нас больше интересует сам факт существования некоторого признака, а не его точное местонахождение (рис.19) [2].

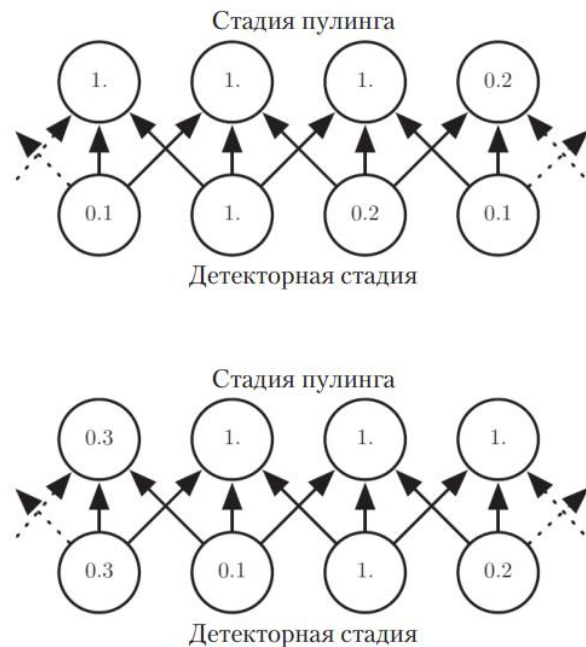


Рис. 19. Мах-пулинг приносит инвариантность.

Вверху на рис.19 изображена середина выходного слоя сверточной сети. В нижней строке показаны выходы нелинейности, а в верхней – выходы max-пулинга с шагом в один пиксель между областями пулинга, каждая из которых имеет ширину три пикселя. Внизу. Та же самая сеть, сдвинутая вправо на один пиксель. В нижней строке изменились все значения, а в верхней – только половина, потому что блоки max-пулинга чувствительны лишь к максимальному значению в своей окрестности, а не точному положению этого значения.

Когда определяется, присутствует в изображении лицо, не важно положение глаз с точностью до пикселя, нужно только знать, есть ли глаз слева и глаз справа. В других ситуациях важнее сохранить

местоположение признака. Например, если ищется угловая точка, образованная пересечением двух границ, ориентированных определенным образом, то необходимо сохранить положение границ настолько точно, чтобы можно было проверить, пересекаются ли они.

Пулинг можно рассматривать как добавление бесконечно сильного априорного предположения, что обучаемая слоем функция должна быть инвариантна (неизменна) к малым параллельным переносам. Если это предположение правильно, то оно может существенно улучшить статистическую эффективность сети [3].

Пулинг по пространственным областям порождает инвариантность к параллельным переносам, но если он производится по выходам сверток с различными параметрами, то признаки могут обучиться, к каким преобразованиям стать инвариантными (рис.20).

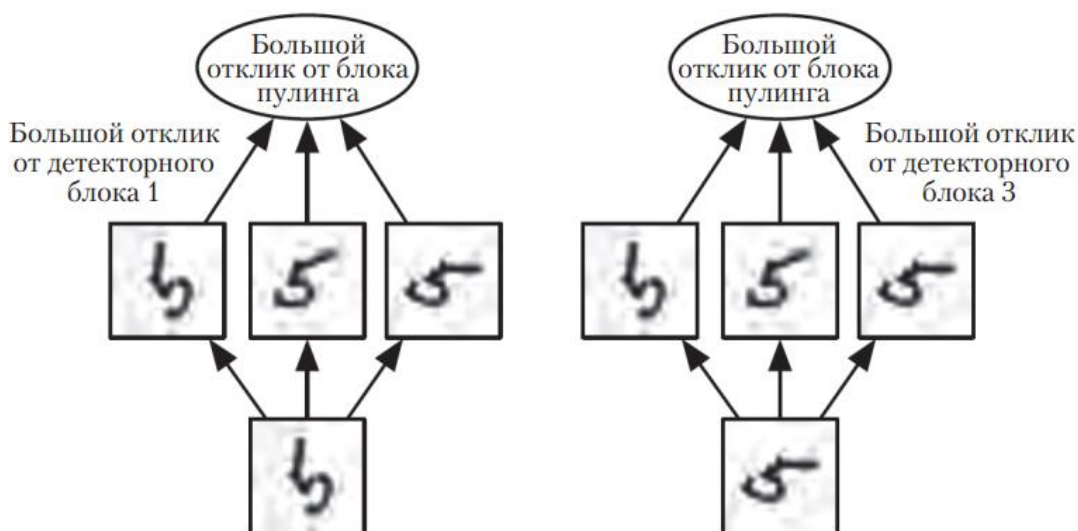


Рис. 20. Пример обученной инвариантности

Блок, выполняющий пулинг по нескольким признакам, обученным с разными параметрами, может обучиться инвариантности к преобразованиям входа. На рис. 20 видно набор из трех обученных фильтров и блок max-пулинга, который обучился инвариантности к вращению. Все три фильтра предназначены для распознавания рукописной цифры 5. Каждый фильтр настроен на свою ориентацию пятерки. Если на входе появляется цифра 5, то соответствующий фильтр распознает ее, что даст большой отклик на детекторный блок. Тогда блок max-пулинга даст большой отклик независимо от того, какой детекторный блок был активирован. На рис. 20 показано, как сеть обрабатывает два разных входа, активирующих разные детекторные блоки. В обоих случаях выход блока

пулинга примерно одинаков. Мах-пулинг по пространственной области обладает естественной инвариантностью к параллельным переносам; такой многоканальный подход необходим только для обучения другим преобразованиям.

Поскольку пулинг агрегирует отклики по целой окрестности, количество блоков пулинга можно сделать меньшим, чем количество детекторных блоков, если агрегировать статистику по областям, отстоящим друг от друга на $k > 1$ пикселей. Пример приведен на рис. 21.

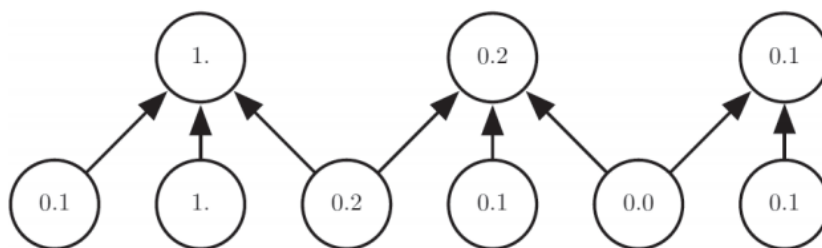


Рис. 21. Пулинг с понижающей передискретизацией.

Здесь мах-пулинг используется с пулом ширины 3 и шагом 2 между пулами. В результате размер представления уменьшается вдвое, что снижает вычислительную и статистическую нагрузки на следующий слой. Отметим, что размер самой правой области пулинга меньше остальных, но ее все равно необходимо включить, если мы не хотим игнорировать некоторые детекторные блоки [2].

Тем самым повышается вычислительная эффективность сети, поскольку следующему слою предстоит обработать примерно в k раз меньше входов. Если число параметров в следующем слое – функция от размера входа (например, когда следующий слой полносвязный и основан на умножении матриц), то уменьшение размера входа также может повысить статистическую эффективность и уменьшить требования к объему памяти для хранения параметров.

В большинстве задач пулинг необходим для обработки входов переменного размера. Например, если классифицировать изображения разного размера, то код слоя классификации должен иметь фиксированный размер. Обычно это достигается за счет варьирования величины шага между областями пулинга, так чтобы слой классификации всегда получал одинаковый объем сводной статистики независимо от размера входа. Так, можно определить финальный слой пулинга в сети, так чтобы он выводил четыре сводных статистических показателя, по одному на каждый

квадрант изображения, вне зависимости от размера самого изображения.

1.6. Обучение

Таким образом стандартный слой сверточной сети (рис.22) состоит из трех компонентов [1]:

- свертка в виде линейного отображения, выделяющая локальные признаки;
- нелинейная функция, примененная покомпонентно к результатам свертки;
- субдискретизация, которая обычно сокращает геометрический размер получающихся тензоров.

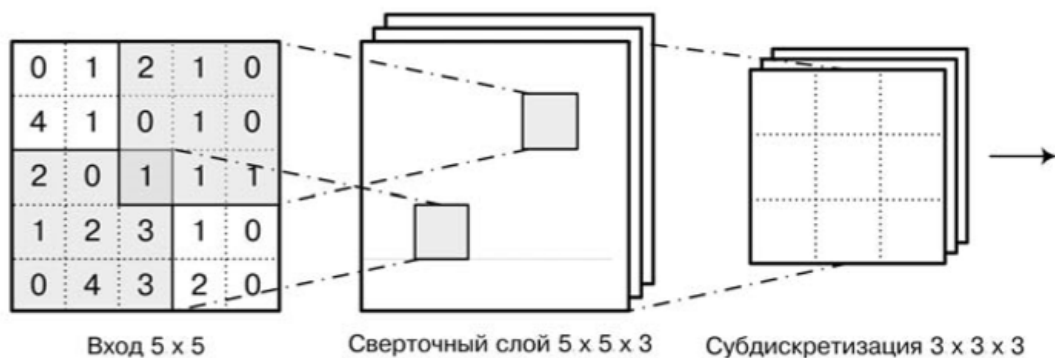


Рис. 22. Схема одного слоя сверточной сети: свертка, за которой следует субдискретизация

По сравнению с «картинкой» на входе размерность тензора увеличилась: сверточная сеть обычно обучает сразу несколько карт признаков на каждом слое (на рис. 22 таких карт три).

Теперь необходимо обучить сверточную сеть. Предположим, что необходимо оптимизировать некоторую функцию ошибки E , при этом уже известно значения на выходах нашего сверточного слоя. Чтобы провести итерацию обучения, нужно понять, как через них выражаются значения градиентов функции ошибки от весов. Через функцию взятия максимума ошибка проходит без изменений, слой субдискретизации ничего не обучает. Однако он делает проходящие по графу вычисления градиенты разреженными, ведь из всех элементов окна субдискретизации $z_{i,j}^l$ частная производная $\partial E / \partial x_{i,j}^{l+1}$ относится только к одному, максимальному, а остальные получают нулевой градиент, и на этом их обучение можно закончить.

Пропуская через нелинейность получится [4]:

$$\frac{\partial E}{\partial y_{i,j}^l} = \frac{\partial E}{\partial z_{i,j}^l} \frac{\partial z_{i,j}^l}{\partial y_{i,j}^l} = \frac{\partial E}{\partial z_{i,j}^l} h(y_{i,j}^l)$$

На сверточном уровне появляются веса, которые нужно обучить. Некоторая сложность здесь состоит в том, что все веса делятся, и каждый участвует во всех выходах; так что сумма получится достаточно большая:

$$\frac{\partial E}{\partial \omega_{a,b}^l} = \sum_i \sum_j \frac{\partial E}{\partial y_{i,j}^l} \frac{\partial y_{i,j}^l}{\partial \omega_{a,b}^l} = \sum_i \sum_j \frac{\partial E}{\partial z_{i+a,j+b}^{l-1}}$$

где индексы i и j пробегают все элементы картинки на промежуточном слое $y_{i,j}^l$, то есть после свертки, но до субдискретизации. Для полноты картины осталось только пропустить градиенты на предыдущий слой.

$$\frac{\partial E}{\partial x_{i,j}^l} = \sum_i \sum_j \frac{\partial E}{\partial y_{i-a,j-b}^l} \frac{\partial y_{i-a,j-b}^l}{\partial x_{i,j}^l} = \sum_i \sum_j \frac{\partial E}{\partial y_{i-a,j-b}^l} \omega_{a,b}$$

Таким образом, адаптируется процедура обратного распространения ошибки, она же обратный проход по графу вычислений, для сверточного слоя. Обратный проход для свертки оказался очень похож на, опять же, свертку с теми же весами $\omega_{a,b}$, только вместо $i + a$ и $j + b$ теперь $i - a$ и $j - b$. В случае, когда изображение дополняется нулями по необходимости и размерности сохраняются, обратный проход можно считать в точности такой же сверткой, что и в прямом проходе, только с развернутыми осями.

Для построения глубокой сети из таких слоев необходимо использовать выход очередного слоя как вход для следующего, а разные карты признаков будут служить каналами. Размер слоя за счет субдискретизации будет постепенно сокращаться, и в конце концов последние слои сети смогут «окинуть взглядом» весь вход, а не только маленькое окошечко из него.

Минимальный размер, при котором окно называемым фильтром имеет центр и выражает такие отношения, как «слева»/«справа» или «сверху»/«снизу», – это размер 3×3 . Именно такой размер фильтров используется в большинстве современных сверточных архитектур. Обычно, в фильтрах, упоминается только две размерности, отвечающие за «рецептивное поле» фильтра. Но на самом деле фильтр задается четырехмерным тензором, последние две размерности которого обозначают число каналов предшествующего и текущего слоя. Так,

например, при работе с цветными изображениями на вход получается три слоя, передающие соответственно красный, зеленый и синий цвета. Когда в первом сверточном слое «фильтр размера 5×5 », это значит, что в первом слое есть несколько наборов весов, переводящих тензор размером $5 \times 5 \times 3$ («параллелепипед весов») в скаляр. Фильтры размером 1×1 – это просто линейные преобразования из входных каналов в выходные с последующей нелинейностью [5].

Теперь, познакомившись со всеми видами сверток, мы можем привести полноценный пример, который упрощенно, но вполне адекватно отражает то, что происходит при реальной обработке изображений сверточными сетями (рис. 23).

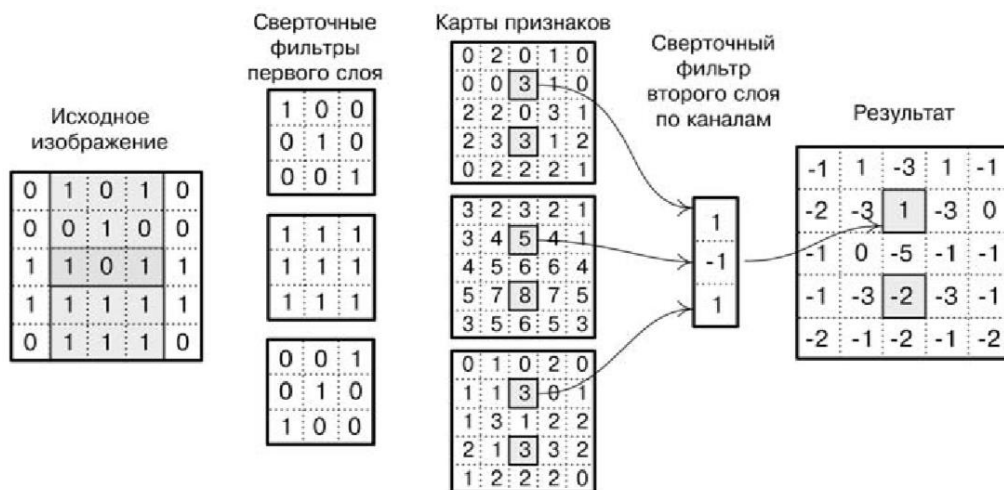


Рис. 22. Пример выделения признаков на двух сверточных слоях.

Поясним рис. 22 [1]:

- каждая карта признаков первого слоя выделяет некий признак в каждом окне исходного изображения; в частности, первая карта признаков «ищет» диагональную линию из единиц, точнее, из пикселей высокой интенсивности (это значит, что она сильнее всего активируется, когда пиксели на диагонали присутствуют), вторая просто активируется на закрашенное окно изображения (чем больше пикселей с высокой интенсивностью, тем лучше), а третья аналогична первой ищет другую диагональ;
- нелинейность в сверточном слое и слой субдискретизации мы в этом примере решили пропустить;
- карта признаков на втором слое (для простоты одна) пытается найти на картинке крестик из двух диагональных линий; для этого она объединяет признаки, выделенные на первом слое, то есть свертка второго

слоя – это одномерная свертка по каналам (признакам), а не по окнам в изображении; вот что свертка второго слоя «хочет увидеть» в том или ином наборе признаков:

- как можно более ярко выраженную диагональную линию из левого верхнего в правый нижний угол, то есть сильно активированный первый признак первого слоя;

- ярко выраженную линию из левого нижнего в правый верхний угол, то есть сильно активированный третий признак первого слоя;

- и как можно меньше других активированных пикселей, то есть суммарная активация, выраженная во втором признаке первого слоя, играет для этой свертки в минус;

- в результате такая сеть действительно находит крестик и на исходной картинке; обратите внимание на разницу между двумя выделенными на рис. 22 окнами – в одном действительно получается крестик, а в другом кроме крестика есть еще много «мусора», и за счет второго признака нейрон, отвечающий за крестик, получает отрицательную активацию;

- а если бы субдискретизация была нетривиальной, она находила бы ещё и «псевдокрестики», в которых диагональные линии находятся рядом друг с другом, а не обязательно в одном и том же окне.

1.7. Архитектуры сверточных сетей

Одной из самых популярных глубоких сверточных архитектур (представленная в 2014 году) является модель, которую принято называть VGG (*виджиджей*). Название происходит от того, что эта модель была разработана в Оксфордском университете в группе визуальной геометрии (Visual Geometry Group). VGG – это сразу две конфигурации сверточных сетей, на 16 и 19 слоев. Основным нововведением стала идея использовать фильтры размером 3×3 с единичным шагом свертки вместо использовавшихся в лучших моделях предыдущих лет сверток с фильтрами 7×7 с шагом 2 и 11×11 с шагом 4 [1]. Причем это хорошо аргументированное предложение:

Во-первых, рецептивное поле трех подряд идущих сверточных слоев размером 3×3 имеет размер 7×7 , в то время как весов у них будет всего 27, против 49 в фильтре 7×7 . Аналогично обстоит дело с фильтрами 11×11 . Это значит, что VGG может стать более глубокой, то есть содержать больше слоев, при этом одновременно уменьшая общее число весов.

Конечно между соответствующими сверточными слоями не должно быть слоев субдискретизации. Во-вторых, наличие дополнительной нелинейности между слоями позволяет увеличить «разрешающую способность» по сравнению с единственным слоем с большей сверткой. Этот же аргумент можно использовать как мотивацию для того, чтобы ввести в сеть свертки размером 1×1 ; такие слои тоже позволяют добавить дополнительную нелинейность в сеть, не меняя размер рецептивного поля. Схема одной из VGG-сетей показана на рис.23 [2].

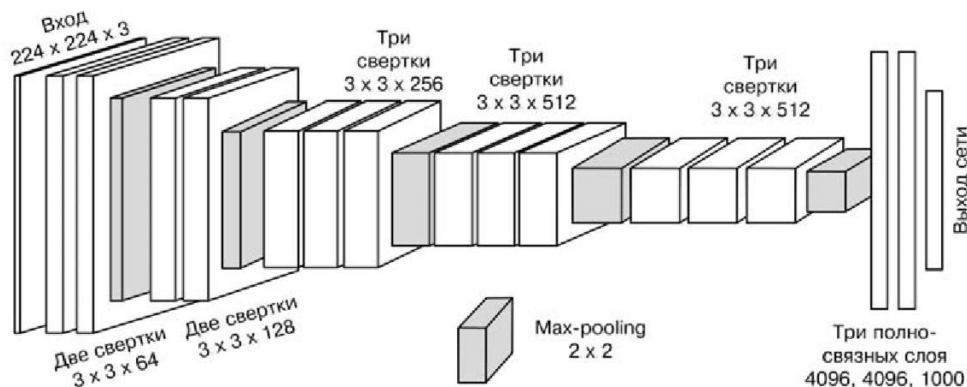


Рис. 23.Схема сети VGG-16

На рис. 23 воплощены три общих принципа [3]:

во-первых, как по две-три свертки 3×3 следуют друг за другом без субдискретизации;

во-вторых, как число карт признаков постепенно растет на более глубоких уровнях сети;

в-третьих, как в конце полученные признаки окончательно «сплюсываются» в одномерный вектор и на нем работают последние, уже полносвязные слои. Все это стандартные методы создания архитектур глубоких сверточных сетей, и если разрабатывать свою архитектуру, стоит следовать этим общим принципам.

Следующая важная сверточная архитектура – это архитектура Inception (начало) [1]. Она была разработана в Google и появилась практически одновременно с VGG. Авторы вдохновились идеями использовать в качестве строительных блоков для глубоких сверточных сетей не просто последовательность «свертка—нелинейность—субдискретизация», как это обычно делается, а более сложными конструкциями.

В Inception (*инсепшен*) такой компонент «собирают» из небольших сверточных конструкций. Так что в сети развивается идея «вложенной»

архитектуры. «Строительными блоками» Inception (*инсепшен*) являются модули, комбинирующие свертки размером 1×1 , 3×3 и 5×5 , а также max-pooling субдискретизацию. Каждый блок представляет собой объединение четырех «маленьких» сетей, выходы которых объединяются в выходные каналы и передаются в следующий слой. Выбор набора свертки и субдискретизации обусловлен скорее удобством, чем необходимостью, и при желании можно использовать другие конфигурации. Использование сверточных слоев 1×1 не столько в качестве дополнительной нелинейности, сколько для понижения размерности между слоями. Свертки 3×3 и тем более 5×5 между слоями с большим числом каналов (в Inception-модулях каналов может быть вплоть до 1024), оказываются крайне ресурсоемкими, несмотря на малые размеры отдельно взятых фильтров. Фильтры 1×1 могут помочь сократить число каналов, прежде чем подавать их на фильтры большего размера. Эта идея отражена на рис. 24, а, на котором показана структура одного блока из исходной работы [1]. На рис. 24, б представлена очень общая и высокоуровневая схема всей сети: она начинается с двух «обычных» сверточных слоев, а затем идут 11 Inception-модулей, дважды перемежаемых субдискретизацией, которая понижает размерность; после этого сеть завершается традиционными полносвязными слоями, дающими уже собственно выход классификатора.

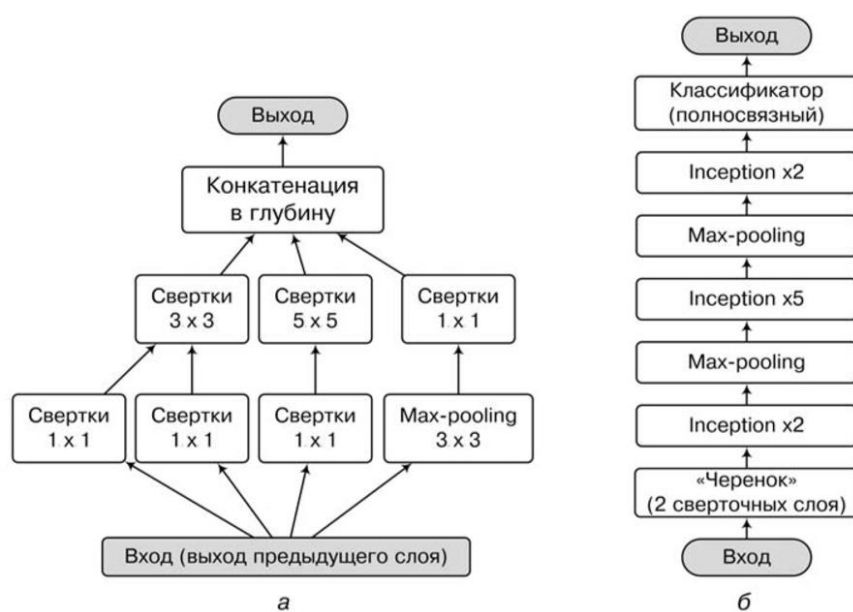


Рис. 24. Inception (*инсепшен*): а – схема одного Inception-модуля; б –общая схема сети GoogLeNet (*Гугленет*)

С учетом достаточно глубокой архитектуры сети GoogLeNet (*Гугленет*) – в общей сложности она содержит порядка 100 различных слоев с общей глубиной в 22 параметризованных слоя, или 27 слоев с учетом субдискретизаций – эффективное распространение градиентов по ней вызывает сомнения. Чтобы решить эту проблему, авторы предложили добавить вспомогательные классифицирующие сети поверх некоторых промежуточных слоев. Иначе говоря, добавляются две новые небольшие полносвязные сети, делающие предсказания на основе промежуточных признаков.

Как показала практика, стало возможно глубокие архитектуры обучать эффективно, однако те решения, к которым сходились нейронные сети большой глубины, часто оказывались хуже, чем у менее глубоких моделей. И эта «деградация» не была связана с переобучением, как можно было бы предположить. Оказалось, что это более фундаментальная проблема: с добавлением новых слоев ошибка растет не только на тестовом, но и на тренировочном множестве.

Для решения проблемы деградации команда из Microsoft Research (Майкрософт Ресеч) разработала новую идею: глубокое остаточное обучение (*deep residual learning*), которое легло в основу сети ResNet (*резнет*) [1]. В базовой структуре новой модели нет ничего нового: это слои, идущие последовательно друг за другом. Отдельные уровни, составные блоки сети тоже выглядят достаточно стандартно, это просто сверточные слои, обычно с дополнительной нормализацией. Разница в том, что в остаточном блоке слой из нейронов можно «обойти»: есть специальная связь между выходом предыдущего слоя $x^{(k)}$ и следующего слоя $x^{(k+1)}$, которая идет напрямую, не через вычисляющий слой. Базовый слой нейронной сети на рис. 25, а превращается в остаточный блок с обходным путем на рис. 25, б. Математически происходит очень простая вещь: когда два пути, «сложный» и «обходной», сливаются обратно, их результаты просто складываются друг с другом. И остаточный блок выражает такую функцию:

$$y^{(k)} = F(x^{(k)}) + x^{(k)},$$

где $x^{(k)}$ – входной вектор слоя k , $F(x)$ – функция, которую вычисляет слой нейронов, а $y^{(k)}$ – выход остаточного блока, который потом станет входом следующего слоя $x^{(k+1)}$.

Получается, что если блок в целом должен аппроксимировать функцию $H(x)$, то это достигается тогда, когда $F(x)$ аппроксимирует остаток $H(x) - x$, отсюда и название остаточные сети (residual networks). В остаточном блоке обучается слой нейронов воспроизводить изменения входных значений, необходимые для получения итоговой функции.

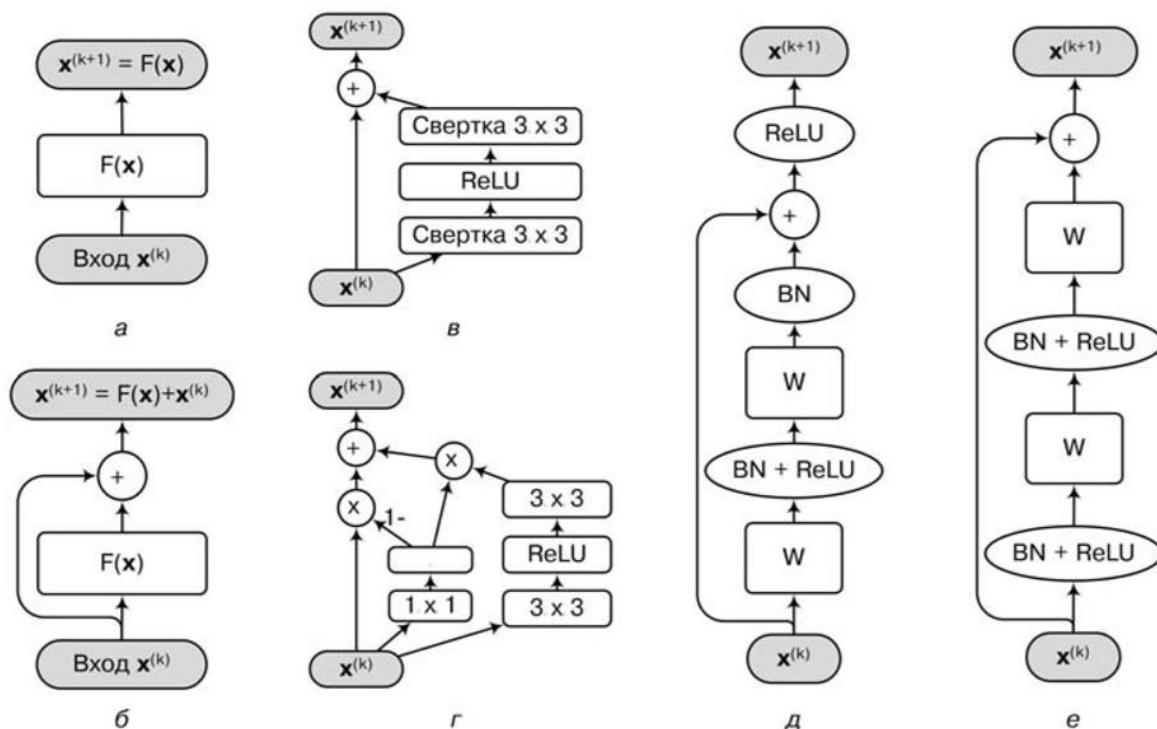


Рис. 25. Блоки сетей для остаточного обучения: а – базовый блок; б – такой же блок с остаточной связью; в – простой блок с остаточной связью; г – блок с остаточной связью, контролирующийся гейтом; д – блок с остаточной связью исходной сети ResNet; е – более поздняя модификация.

Преимущества [1]: Во-первых, часто получается, что обучить «остаточную» функцию проще, чем исходную; Оказывается, что с помощью двухслойной нелинейной нейронной сети выучить тождественную функцию достаточно сложно; в то же время в остаточной форме от сети требуется просто заполнить все веса нулями, а «обходной путь» сделает всю работу сам. Главная же причина состоит в том, что градиент во время обратного распространения может проходить через этот блок беспрепятственно, градиенты не будут затухать, ведь всегда есть возможность пропустить градиент напрямую:

$$\frac{\partial y^{(k)}}{\partial x^{(k)}} = 1 + \frac{\partial F(x^{(k)})}{\partial x^{(k)}}$$

Это значит, что даже насыщенный и полностью обученный слой F , производные которого близки к нулю, не мешает обучению.

Примеры таких «обходных путей», которые использовались в разных вариантах сети ResNet и других исследованиях этой группы авторов, показаны на рис. 25. Проводилось подробное сравнение нескольких вариантов остаточных блоков. Остаточные блоки, которые теоретически должны быть более выразительными, на практике оказываются хуже, чем самые простые варианты.

На рис. 25, в изображен очень простой вариант остаточного блока, в котором $y^{(k)} = F(x^{(k)}) + x^{(k)}$, а на рис. 25, г – вариант посложнее:

$$y^{(k)} = \left(1 - \sigma(f(x^{(k)}))\right)x^{(k)} + \sigma(f(x^{(k)}))F(x^{(k)})$$

где f – другая функция входа, реализованная через свертки 1×1 . Это значит, что $F(x^{(k)})$ и $x^{(k)}$ суммируются не с равными весами, а с весами, управляемыми дополнительным «гейтом». Казалось бы, простой вариант является частным случаем сложного: достаточно просто обучить гейты так, чтобы веса были равными, то есть чтобы всегда выполнялось $f(x^{(k)}) = 0$. Однако эксперименты показали, что простота в данном случае важнее выразительности и важно обеспечить максимально свободное и беспрепятственное течение градиентов. На рис. 25, д показан вариант остаточного блока, а на рис. 25, е – улучшенный вариант. Обратите внимание, что разница, по сути, только в том, что из «обходного пути» убрали ReLU-нелинейность, последнее «препятствие» на пути значений с предыдущего слоя [1].

Правда, вне которых приложениях от них отказываются ради скорости и экономии ресурсов: большая сверточная сеть с остаточными связями никак не поместится в смартфон. Если ресурсы важны, стоит посмотреть в сторону моделей, которые показывают немного более слабые результаты в собственно распознавании, но имеют при этом на порядок меньше весов; выделим, в частности, MobileNets (*мобайлнетс*) и SqueezeNet (*скуизнет*).

В заключении отметим, что идея про гейт, управляющий остаточными блоками, как на рис. 25, г, все-таки оказалась довольно плодотворной. Именно она легла в основу так называемых магистральных сетей (highway networks), предложенных группой Юргена Шмидхубера [1]. Идея магистральных сетей именно такая, представляем $y^{(k)}$, выход слоя k ,

как линейную комбинацию входа этого слоя $x^{(k)}$ и результата $F(x^{(k)})$, веса которой управляются другими преобразованиями [1]:

$$y^{(k)} = C(x^{(k)})x^{(k)} + T(x^{(k)})F(x^{(k)})$$

где C – это гейт переноса, а T – гейт преобразования; обычно комбинацию делают выпуклой, $C = 1 - T$. Магистральные сети тоже позволили обучать очень глубокие сети с сотнями уровней, а затем эта конструкция была адаптирована для рекуррентных сетей. Что здесь победит — простота или выразительность — вопрос пока открытый, но в любом случае варианты этих архитектур уже позволили сделать беспрецедентно глубокие сети, и останавливаться на достигнутом исследователи не собираются.

1.8. Библиотеки

В последние годы обучение нейронных сетей превратилось в высшей степени инженерную дисциплину. Появилось несколько очень удобных библиотек, которые позволяют буквально в пару строк кода построить модель нейронной сети (сколь угодно глубокой), сформировать для нее граф вычислений, автоматически подсчитать градиенты и выполнить процесс обучения, причем сделать это можно как на процессоре, так и — совершенно прозрачным образом, изменив пару строк или пару ключей при запуске — на видеокарте, что обычно ускоряет обучение в десятки раз. С каждым годом эти библиотеки становятся все удобнее, выходят новые версии существующих, а также абсолютно новые программные продукты. Есть библиотеки общего назначения, которые могут создать любой граф вычислений, и специализированные надстройки, которые реализуют разные компоненты нейронных сетей.

Библиотеки, которые используются для реализации сверточных нейронных сетей, имеют основной интерфейс к языку программирования Python. Этот язык стал фактически стандартом в современном машинном обучении и обработке данных.

Keras (*керас*) (<https://keras.io>) – это фреймворк поддержки глубокого обучения для Python, обеспечивающий удобный способ создания и обучения практически любых моделей глубокого обучения. Первоначально Keras разрабатывался для исследователей с целью дать им возможность быстро проводить эксперименты. Keras обладает следующими ключевыми характеристиками [4]:

- позволяет выполнять один и тот же код на CPU (*сипию*) или GPU (*джипию*);
- имеет дружелюбный API (*эйпиай*), упрощающий разработку прототипов моделей глубокого обучения;
- включает в себя встроенную поддержку сверточных сетей (для распознавания образов), рекуррентных сетей (для обработки последовательностей) и все возможных их комбинаций;
- включает в себя поддержку произвольных сетевых архитектур: моделей с множественными входами или выходами, совместное использование слоев, совместное использование моделей и т. д. Это означает, что Keras подходит для создания практически любых моделей глубокого обучения, от порождающих состязательных сетей до нейронной машины Тьюринга.

Фреймворк Keras распространяется на условиях свободной лицензии и может бесплатно использоваться в коммерческих проектах. Он совместим с любыми версиями Python. Keras используется в Google, Netflix, Uber, CERN, Yelp, Square и сотнях стартапов, решающих широкий круг задач.

Keras – это библиотека уровня модели, предоставляющая высокоуровневые строительные блоки для конструирования моделей глубокого обучения. Она не реализует низкоуровневые операции, такие как манипуляции с тензорами и дифференцирование, – для этого используется специализированная и оптимизированная библиотека поддержки тензоров. При этом Keras не полагается на какую-то одну библиотеку поддержки тензоров, а использует модульный подход (рис. 26); то есть к фреймворку Keras можно подключить несколько разных низкоуровневых библиотек [5].

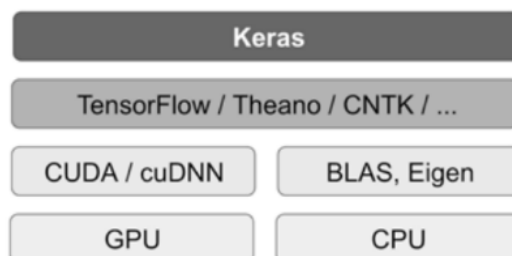


Рис. 26. Программно-аппаратный стек поддержки глубокого обучения

В настоящее время поддерживаются три такие библиотеки: TensorFlow, Theano и Microsoft Cognitive Toolkit (CNTK). В будущем Keras, скорее всего, будет расширен еще несколькими низкоуровневыми механизмами поддержки глубокого обучения.

TensorFlow (*тензорфлей*), CNTK (*сиэммикей*) и Theano (*сиано*) – это одни из ведущих платформ глубокого обучения в настоящее время. Theano (<http://deeplearning.net/software/theano>) разработана в лаборатории MILA Монреальского университета, TensorFlow (www.tensorflow.org) разработана в Google, а CNTK (*сиэммикей*) (<https://github.com/Microsoft/CNTK>) разработана в Microsoft. Любой код, использующий Keras, можно запускать с любой из этих библиотек без необходимости менять что-либо в коде: можно легко переключаться между ними в процессе разработки, что часто оказывается полезно, например, если одна из библиотек показывает более высокую производительность при решении данной конкретной задачи. Рекомендуется по умолчанию использовать библиотеку TensorFlow как наиболее распространенную, масштабируемую и высококачественную [5].

Среди библиотек общего назначения, которые способны строить граф вычислений и проводить автоматическое дифференцирование, долгое время бесспорным лидером была Theano (*сиано*). Однако в ноябре 2015 года Google выпустила (с открытым исходным кодом) библиотеку TensorFlow (*тензорфлей*) [3], предназначенную для таких же целей. TensorFlow стала вторым поколением библиотек глубокого обучения в Google. В TensorFlow реализован полный набор операций над тензорами из NumPy с поддержкой матричных вычислений над массивами разной формы и конвертирования между этими формами. Библиотеки Theano(*сиано*) и TensorFlow на данный момент остаются двумя бесспорными лидерами в этой области, и сложно уверенно рекомендовать одну из них [5].

Используя TensorFlow (Theano или CNTK), Keras может выполнять вычисления и на CPU (*сипию*), и на GPU (*джипию*). При выполнении на CPU(*сипию*) TensorFlow (*тензорфлей*) сама использует низкоуровневую библиотеку специализированных операций с тензорами, которая называется Eigen (*айген*) (<http://eigen.tuxfamily.org>).