Машинное обучение, ФКН ВШЭ
Семинар №10

23 ноября 2017 г.

1 Backpropagation

Backpropagation – метод обратного распространения ошибки, позволяющий эффективно вычислять градиент функции потерь для нейронной сети.

Рассмотрим следующую сеть:

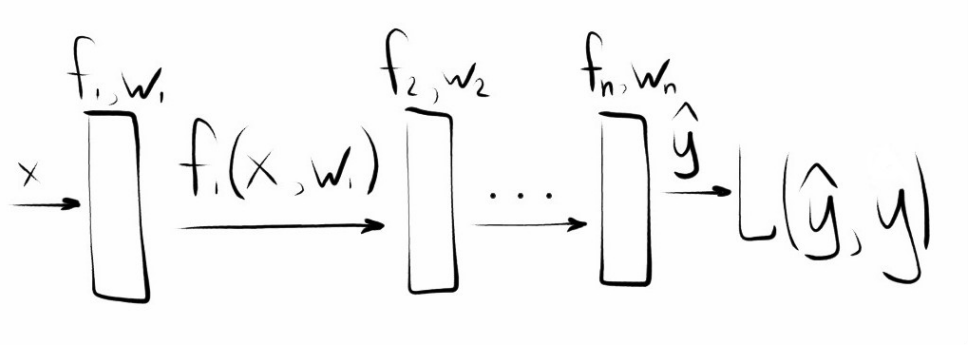

Рис. 1. Простейшая нейронная сеть

Это функция от объекта x из нашей выборки, которая представляется композицией элементарных функций f_i. Мы хотим вычислять градиенты функции потерь для предсказаний сети по её параметрам w, чтобы в дальнейшем обучать сеть с помощью градиентного спуска или другого метода оптимизации, использующего градиент фукнции.

§1.1 Скалярный случай

Для начала рассмотрим случай, когда все величины являются скалярами. Он обобщается на случай векторных величин. Введём обозначения: $x_1 = f_1(x, w_1),\ x_2 = = f_2(x_1, w_2) \ldots, a(x) = f_n(x_{n-1}, w_n)$.

Посчитаем производную по w_n для последнего слоя.

$$L(y, z) = L\Big(y, f_n(x_{n-1}, w_n)\Big)$$

$$\frac{\partial}{\partial w_n} L\Big(y, f_n(x_{n-1}, w_n)\Big) = \frac{\partial}{\partial z} L(y, z)\Big|_{z=f_n(x_{n-1}, w_n)} \cdot \frac{\partial}{\partial w_n} f_n(x_{n-1}, w_n)$$

Заметим, что мы можем вычислить производную, используя знания только о последнем слое. Аналогичным образом можем посчитать производную по x_{n-1}. Она нам ещё пригодится.

$$\frac{\partial}{\partial x_{n-1}} L\Big(y, f_n(x_{n-1}, w_n)\Big) = \frac{\partial}{\partial z} L(y, z)\Big|_{z=f_n(x_{n-1}, w_n)} \cdot \frac{\partial}{\partial x_{n-1}} f_n(x_{n-1}, w_n)$$

Попробуем сделать то же самое для предпоследнего слоя:

$$L(y, z) = L\Big(y, f_n(f_{n-1}(x_{n-2}, w_{n-1}), w_n)\Big)$$

$$\frac{\partial}{\partial w_{n-1}} L\Big(y, f_n(f_{n-1}(x_{n-2}, w_{n-1}), w_n)\Big) =$$
$$= \frac{\partial}{\partial z} L(y, z)\Big|_{z=f_n(x_{n-1}, w_n)} \cdot \frac{\partial}{\partial x_{n-1}} f_n(x_{n-1}, w_n) \cdot \frac{\partial}{\partial w_{n-1}} f_{n-1}(x_{n-2}, w_{n-1})$$

Здесь можно воспользоваться рассчитанной ранее производной по x_{n-1}:

$$\frac{\partial}{\partial w_{n-1}} L\Big(y, f_n(f_{n-1}(x_{n-2}, w_{n-1}), w_n)\Big) = \frac{\partial}{\partial x_{n-1}} L\Big(y, f_n(x_{n-1}, w_n)\Big) \cdot \frac{\partial}{\partial w_{n-1}} f_{n-1}(x_{n-2}, w_{n-1})$$

То же самое можем сделать для производной по x_{n-2}:

$$\frac{\partial}{\partial x_{n-2}} L\Big(y, f_n(f_{n-1}(x_{n-2}, w_{n-1}), w_n)\Big) = \frac{\partial}{\partial x_{n-1}} L\Big(y, f_n(x_{n-1}, w_n)\Big) \cdot \frac{\partial}{\partial x_{n-2}} f_{n-1}(x_{n-2}, w_{n-1})$$

Как видим, мы можем вычислять производные по весам нашей фунции, используя знания только о текущем слое и производных для слоёв с бОльшим индексом. Выпишем формулу для произвольного i-го слоя.

Для i-го слоя мы знаем:

- f_i – функция нашего слоя,
- $\partial L / \partial x_i$ – производная по выходу i-го слоя, т.к. её мы вычислили ранее для слоя $i + 1$.

Тогда мы можем записать следующие формулы:

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial x_i} \frac{\partial f_i(x_{i-1}, w_i)}{\partial w_i}$$

$$\frac{\partial L}{\partial x_{i-1}} = \frac{\partial L}{\partial x_i} \frac{\partial f_i(x_{i-1}, w_i)}{\partial x_{i-1}}$$

Производную по весу w_i мы используем для обновления наших весов градиентным спуском, а производную по входам x_{i-1} передаём на слой $i - 1$.

§1.2 Векторный случай

При переходе к векторным величинам, в предыдущих рассуждениях поменяется только формула производной сложной функции.

Пусть $x \in \mathbb{R}^n, y \in \mathbb{R}^m, z \in \mathbb{R}$, и заданы функции $z = g(f(x)), y = f(x)$.

Легко показать, что $\nabla_x g = \left(\frac{dy}{dx}\right)^T \nabla_y g$, где $\frac{dy}{dx}$ – якобиан:

$$\frac{dy}{dx} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_2}{\partial x_n} \\ \cdots & & & \\ \frac{\partial y_m}{\partial x_1} & \frac{\partial y_m}{\partial x_2} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}.$$

Теперь рассмотрим некоторый слой и выпишем для него формулы.

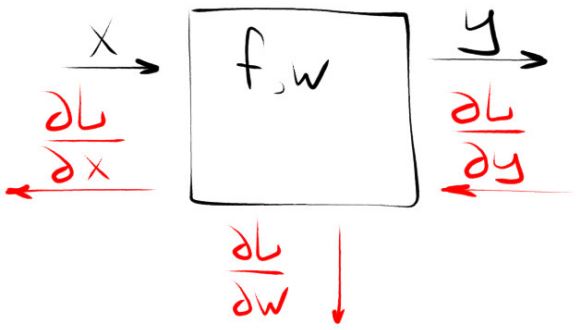

Рис. 2. Прямой и обратный проход по нейронной сети для одного слоя

$$\frac{\partial L}{\partial w} = \left(\frac{dy}{dw}\right)^T \frac{\partial L}{\partial y},$$

$$\frac{\partial L}{\partial x} = \left(\frac{dy}{dx}\right)^T \frac{\partial L}{\partial y}.$$

Здесь $\frac{dy}{dx}$ и $\frac{dy}{dw}$ соответствующие якобианы, а $\frac{\partial L}{\partial y} = \nabla_y L$, $\frac{\partial L}{\partial x} = \nabla_x L$, $\frac{\partial L}{\partial w} = \nabla_w L$ – градиенты.

Возникает вопрос – что делать с более сложными структурами входов и весов? Ведь на вход может поступать не вектор, а матрица (чёрно-белое изображение) или тензор (многомерный аналог матрицы). На самом деле, запись в виде тензора используется для более удобного представления формул, мы, в свою очередь, можем векторизовать все наши формулы и использовать формулу сложной производной для векторов.

2 Полносвязный слой

Теперь, когда мы знаем как вычислять градиенты по параметрам нашей функции, рассмотрим какими элементарными функциями пользуются для построения алгоритмов.

Полносвязный слой (Dense layer или Fully-Connected layer) представляет собой домножение на матрицу и добавление сдвигов:

$$f_i(x_{i-1}, w_i) = W_i x_{i-1} + b_i,$$

здесь $w_i = (W_i, b_i)$.

Название этого слоя произошло от следующей картинки:

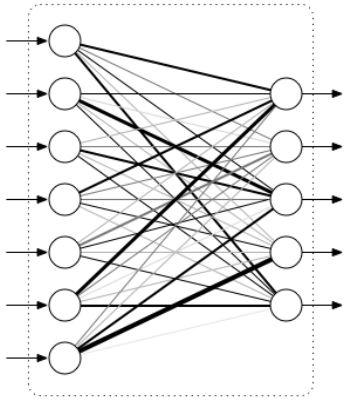

Рис. 3. Полносвязный слой

На картинке слева входной вектор x_{i-1}, справа выходной вектор $W_i x_{i-1}$, толщина линий соответствует весам в матрице W_i.

§2.1 Нелинейности

Посмотрим что будет на выходе двух подряд идущих полносвязных слоёв.

$$f_i(x_{i-1}, w_i) = W_i f_{i-1}(x_{i-2}, w_{i-1}) + b_i = W_i(W_{i-1}x_{i-2} + b_{i-1}) + b_i = Ax_{i-2} + \beta,$$

где $A = W_i W_{i-1},\ \beta = W_i b_{i-1} + b_i$. Получается, что композицию двух полносвязных слоёв можно представить одним полносвязным слоем.

Этот факт вызывает некоторые трудности – если мы будем использовать только полносвязные слои, то ничего, сложнее линейной модели, построить не удастся. Поэтому добавляют нелинейности (нелинейные функции), которые применяют к выходам слоёв. Это позволяет обогатить семейство элементарных функций. Некоторые из них:

- ReLU (Rectified Linear Unit)

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

- Leaky ReLU
$$f(x) = \max(\alpha x, x),\ \alpha < 0.5$$
- sigmoid
$$f(x) = \frac{1}{1 + \exp(-x)}$$
- tanh
$$f(x) = \frac{2}{1 + \exp(-2x)} - 1$$

3 Свёрточный слой

Свёрточный слой (Convolutional layer) – одна из самых важных компонент нейронной сети при работе с изображениями. Для того, чтобы понять как устроены свёрточные слои, разберёмся, что такое операция свёртки и как она применяется к картинкам.

3.0.1 Операция свёртки

В свёртке участвуют две функции:

- $f(x)$ – исходная функция, или фильтруемая
- $g(x)$ – фильтр, или фильтрующая функция.

Сначала рассмотрим одномерную свёртку для обычных функций

$$(f * g)(t) = \int_{-\infty}^{+\infty} f(x)g(-x + t)dx$$

Что происходит в этой формуле? Мы оставляем исходную функцию $f(x)$ без изменений, а вот функцию $g(x)$ инвертируем относительно аргумента, сдвигаем на параметр t, а потом считаем площадь под произведением получившихся функций.

Рассмотрим двумерный дискретный случай для изображений. Есть две матрицы I, K – изображение размера $h \times w$ и свёртка размера $k_1 \times k_2$. Двигая матрицу свёртки относительно изображения, считаем сумму их произведений, точно так же, как в непрерывном одномерном случае. Формально это записывается следующим образом:

$$(I * K)_{ij} = \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} I(i + m, j + n)K(m, n), 0 \leq i \leq h - k_1, 0 \leq j \leq w - -k_2$$

Важно заметить, что мы не инвертируем матрицу K, как делалось в одномерном случае с функцией $g(x)$. И у этого момента есть некоторая история. В «классическом» Computer Vision, для операции свёртки, во всех фильтрах инвертировали строки и столбцы, таким образом, мы перемножали левый нижний элемент матрицы

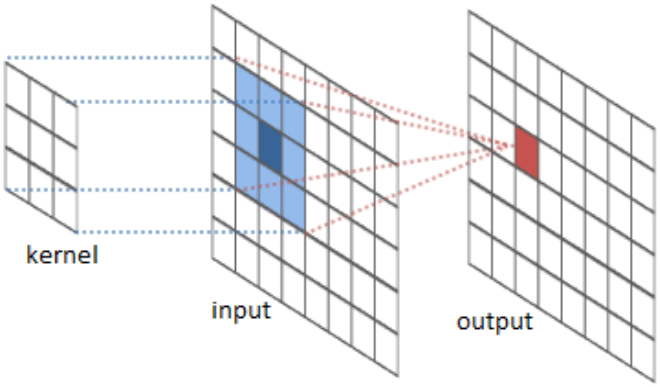

Рис. 4. Применение свёртки на изображении

G с правым верхним элементом под матрицей I. И это действительно является каноническим определением операции свёртки. Но распространение нейронных сетей изменило наш лексикон. Дело в том, что в нейросетях абсолютно неважно инвертируем ли мы строки и столбцы, или же нет, поэтому, для сокращения времени выполнения операции, инвертирование не делают. Поэтому сейчас, когда говорят о нейросетях, под свёрткой подразумевают свёртку без инвертирования, но когда речь заходит о «классическом» Computer Vision, то подразумевают каноническое определение свёртки, т.е. с инвертированием.

Кроме этого, можно заметить, что размер конечного изображения уменьшается по сравнению с исходным. Но в некоторых ситуациях мы не хотим чтобы размер конечного изображения отличался. Чтобы избежать уменьшения размеров, исходное изображение дополняют нулями по краям (но это, разумеется, не единственый способ борьбы с этой проблемой). Заметьте, что количество нулей зависит от размера фильтра!

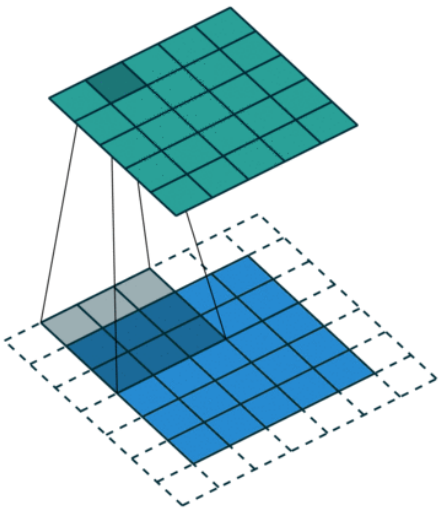

Рис. 5. Дополнение изображения нулями

§3.1 Фильтры

В «классическом» Computer Vision до активного применения нейронных сетей для свёртки использовались фильтры, определённые вручную, а не обученные. Посмотрим на один из самых простых — фильтр Собеля. Этот фильтр позволяет находить границы на изображениях, считая производную в каждой точке. В качестве примера посмотрим на одномерный случай:

$$\text{diff} = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix},$$

Такой фильтр приближает производную попарными разностями:

$$\left.\frac{\partial f}{\partial x}\right|_{x=x_i} = f(x_{i+1}) - f(x_{i-1}).$$

Теперь посмотрим на устройство фильтра Собеля в двумерном случае:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * A, \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * A,$$

здесь A – фильтруемое изображение. Сначала отфильтруем наше изображение двумя разными фильтрами – один отвечает за производную по x, другой за производную по y. В обоих случаях мы дополнительно считаем производные в соседних пикселях и усредняем их с весами. После того, как мы знаем проекции производной на каждую из осей, можем рассчитать результирующую производную по евклидовой метрике:

$$G = \sqrt{G_x^2 + G_y^2},$$

Здесь возведение в степень и квадратный корень – поэлементные операции.

Обратите внимание, что фильтр Собеля не симметричен, т.к. зависит от того, какое направление по x и по y мы считаем положительным. В данном примере мы считали, что x увеличивается слева направо, а y сверху вниз.

Существуют разнообразные фильтры (например, фильтр Лапласа или Гаусса), но с началом активного применения нейронных сетей вместо готовых фильтров стали использовать обучаемым, где матрица, использозуемая для свёртки, обучается вместе со всей сетью. В таком случае фильтр становится одним из слоёв нейронной сети.

Обученные свёрточные слои нейронной сети можно визуализировать (либо сами матрицы, либо выдавамые значения на конкретных изображениях). На рис. (7) можно заметить, что первый слой учится выделять границы на изображении, а следующие слои находят более сложные структуры, опираясь на признаки предыдущего уровня.

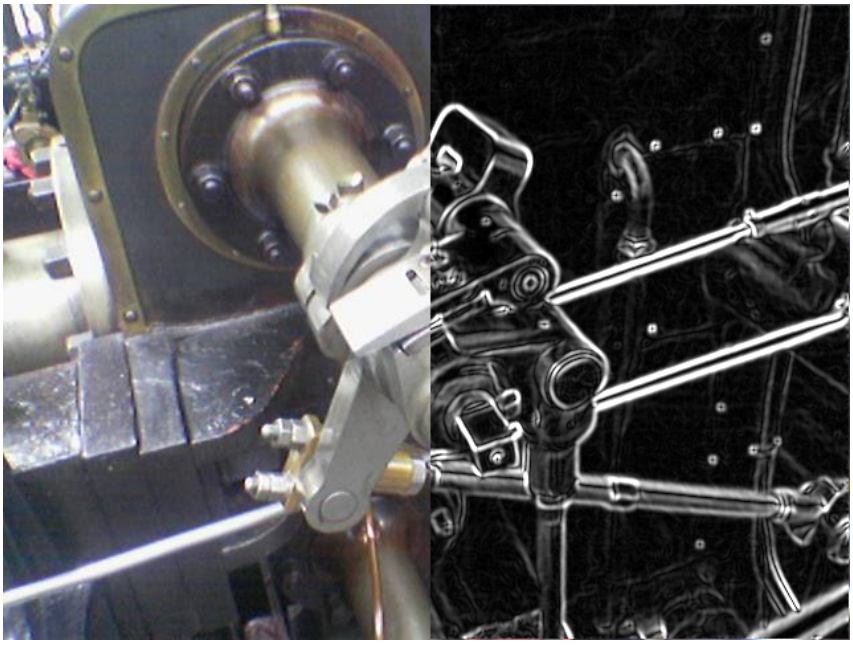

Рис. 6. Оригинал и применённый фильтр Собеля

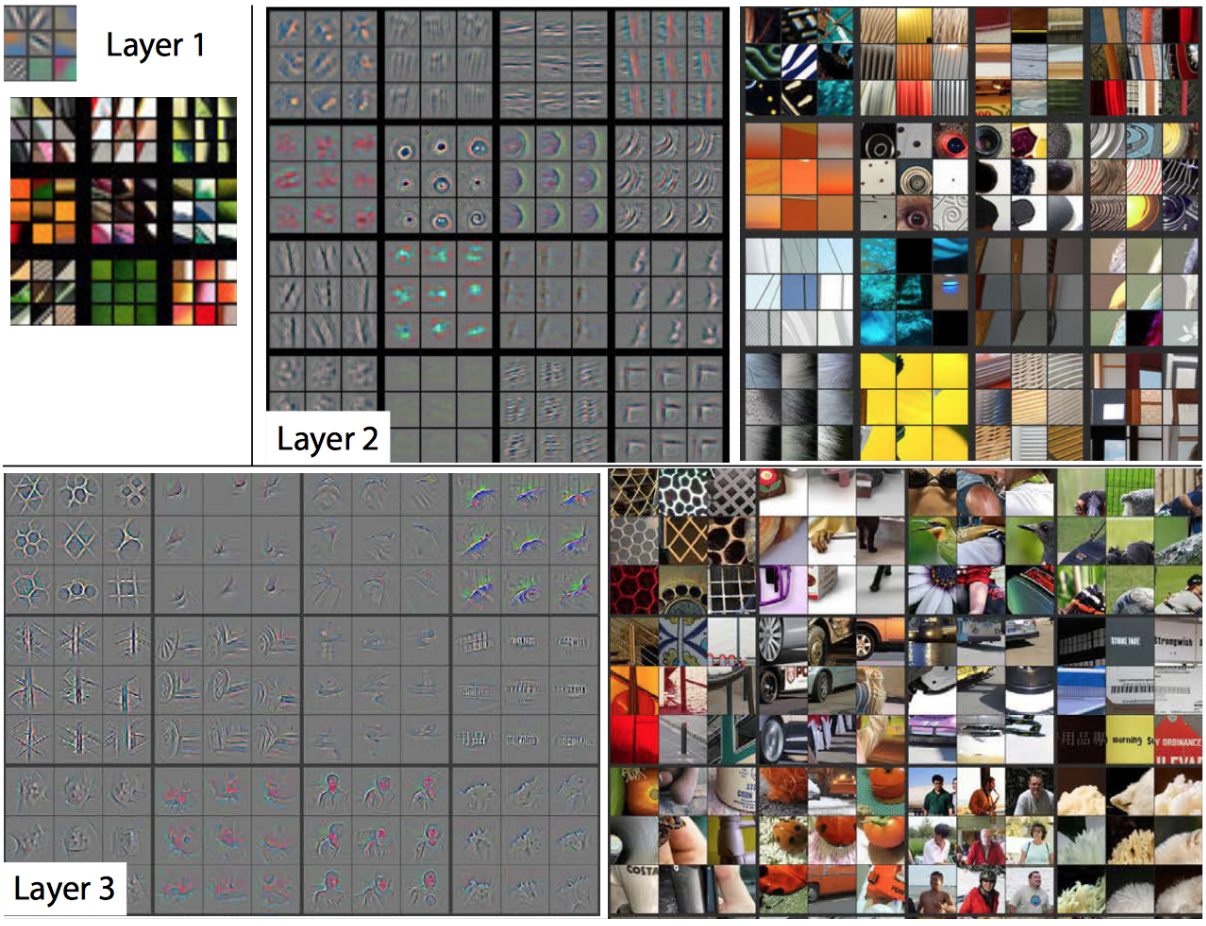

Рис. 7. Визуализация активация со скрытых слоёв

Определение двумерной свёртки для серых картинок легко обобщается на случай, когда каждому пикселю соответствует не одно значение, а целый вектор.

В общем случае, на вход свёрточному слою подаётся трёхмерный тензор (можно сказать, что матрица, у которой не две размерности, а три) размера $H \times W \times D$, где первые два измерения соответствуют координатам пикселей, а третье измерение соответствует признаковому описанию пикселей. Фильтр тоже является трёхмерным тензором размера $h \times w \times D$. Обратите внимание, что третья размерность фильтра и входного тензора совпадает.

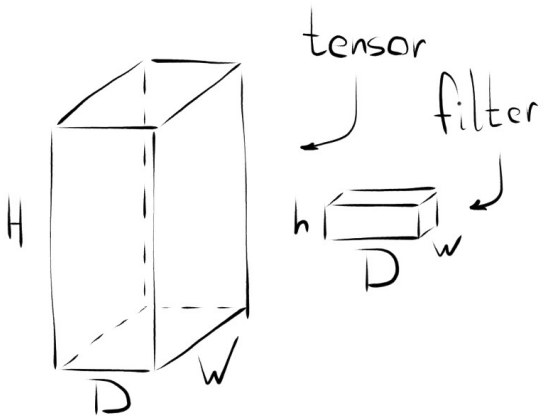

Рис. 8. Изображение и фильтр в случае нескольких цветовых каналов

Свёртка происходит аналогичным образом, мы накладываем фильтр на тензор, считаем сумму произведения пикселей с весами, при этом двигать фильтр мы, как и раньше, можем только по координатам пикселей. На выходе такой свёртки мы получаем матрицу.

Свёрточный слой обычно имеет несколько фильтров, веса которых, являются параметрами нашего алгоритма. Сворачивая исходный тензор размера $H \times W \times D$ с D' фильтрами размера $h \times w \times D$, мы получаем результирующий тензор рамера $(H - h+1) \times (W - w+1) \times D'$. Таким образом, свёрточный слой задаётся параметрами h, w и количеством фильтров D'. Также, после свёртки, к каждому фильтру добавляется сдвиг b_d (d – индекс фильтра), который тоже обучается.

4 Свёрточные нейронные сети

- Архитектура с множеством свёрточных слоёв.
- Обычно используется для изображений, но есть и другие случаи – char-CNN.

§4.1 ImageNet

Успех и популярность свёрточных сетей обусловлен многими факторами, и один из них это появление в 2010 году базы картинок ImageNet, в которой содержится около 1.2 миллиона картинок, размеченных на 1000 различных классов. Ежегодно

на этом датасете проводится соревнование на лучшее решение по различным задачам (классификация, детектирование, ...).

4.1.1 Метрика качества

Алгоритм должен выдать 5 классов $l_j,\ j = 1, \ldots, 5$, для каждой картинки, также у нас есть истинные классы, которые на ней присутствуют $g_k,\ k = 1, \ldots, n$, где n – кол-во классов на данной картинке. Тогда ошибка на одной картинке это

$$e = \frac{1}{n} \sum_k \min_j d(l_j, g_k),$$

где $d(x, y) = 0$, при $x = y$ и $d(x, y) = 1$ в остальных случаях.

Идея такого подсчёта заключается в том, чтобы не штрафовать «более умный» алгоритм, который может обнаружить объекты, которые на картинке есть, но не размечены.

Разберём несколько примеров архитектур, выигрывавших классификацию на ImageNet и их основные идеи, которые используются и в других архитектурах при решении других задач.

4.1.2 VGG19

- Почти выиграла ImageNet в 2014 году
- Имеет 19 слоёв (есть 16-слойная версия)
- Идеи:
  - последовательное использование свёрточных слоёв (друг за другом) позволяет уменьшить количество настраиваемых параметров по сравнению со свёртками больших размеров, имеющих такую же рецепторную область (область изображения, которую «видит» каждый фильтр)
  - увеличение количества фильтров в два раза после каждого пулинга «для компенсации».

Архитектуру сети VGG можно увидеть на рис. (9).

4.1.3 GoogLeNet (inception block)

- Выиграла ImageNet в 2014 году
- Идеи:
  - использование фильтров меньшего размера для уменьшения числа настраиваемых параметров, но увеличение глубины
  - уменьшение толщины каждого слоя свёртками 1 на 1 (из большой толщины делаем меньшую)

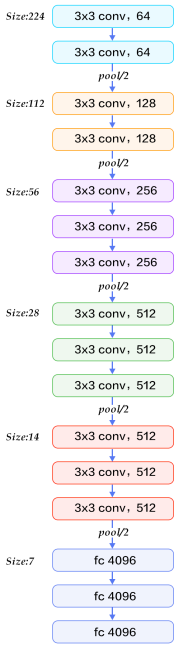

Рис. 9. 16-слойная сеть VGG

– использование фильтров разного размера для извлечения признаков разного масштаба
– упаковка всего в единый в блок, который использование как кирпичик для построения сети
– отсутствие скрытых полносвязных выходов из-за их «тяжёлости» (чрезмерно большое число настраиваемых параметров).

Схему одного блока можно увидеть на рис. (10).

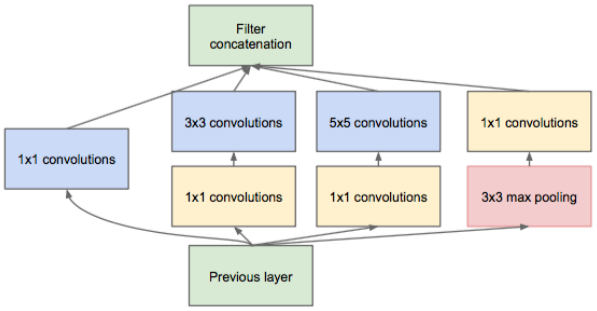

Рис. 10. Изображение и фильтр в случае нескольких цветовых каналов

4.1.4 ResNet

- Выиграла ImageNet в 2015 году

- Идея: добавление shortcut-соединений в сеть (соединение между слоями напрямую без умножения на матрицы), позволяющее делать и обучать сеть любой глубины, так как решается проблема затухающих градиентов + добавление любого количества слоёв с нулевыми матрицами не изменяют показаний сети из-за прямых соединений (например, VGG уже нельзя делать глубже, потому что он становится только хуже).

  Схему shortcut-соединений можно увидеть на рис. (11).

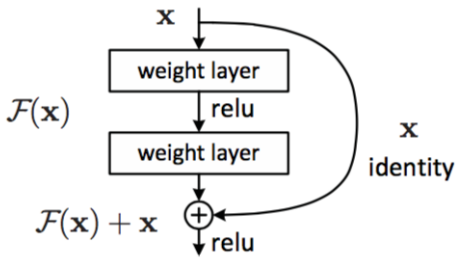

Рис. 11. Изображение и фильтр в случае нескольких цветовых каналов

Можно строить сети, объединяя различные идеи, например, есть различные варианты Inception-ResNet.