Машинное обучение (Machine Learning)

Глубокое обучение: Архитектуры сверточных нейронных сетей

Уткин Л.В.

Санкт-Петербургский политехнический университет Петра Великого

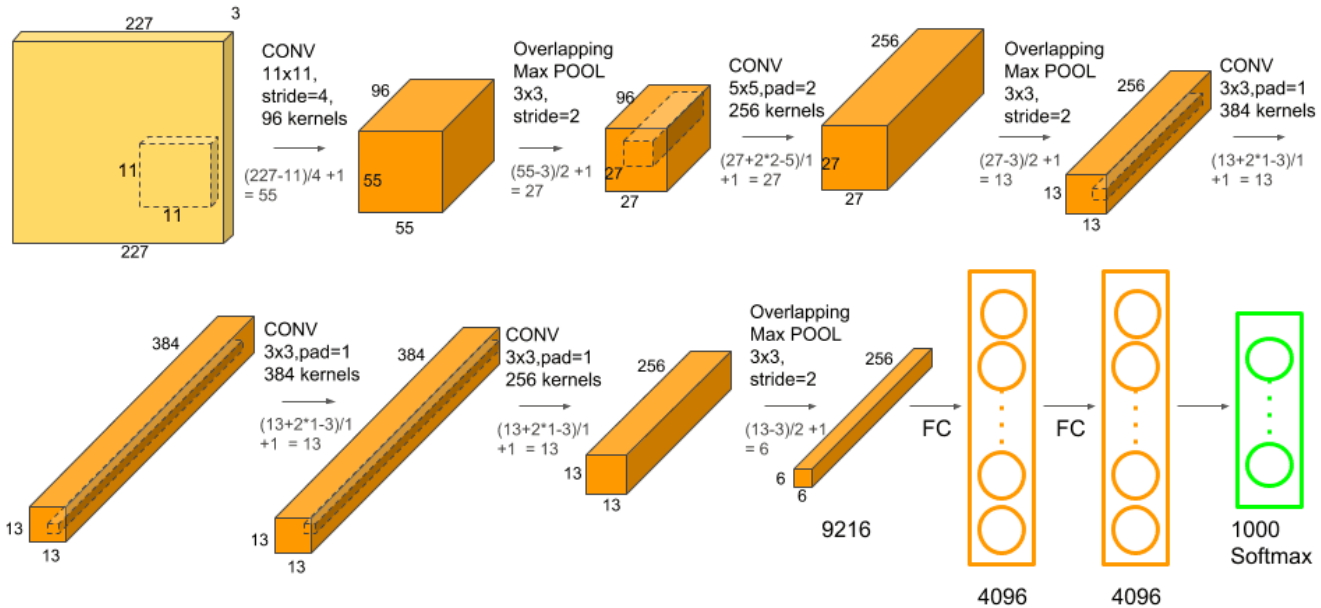
LeNet-5

- Одна из первых сетей (1998 г.)
- На входе - черно-белое изоб-ие 32x32 пикселя, 7 слоев:
  1. Свертка: каналов - 6, ядро–5x5, шаг–1.
  2. Пулинг: ядра - 2x2.
  3. Свертк: каналов - 16, ядро - 5x5, шаг - 1. Некоторые соединения опущены, чтобы убрать симметричность в сети и уменьшить количество параметров.
  4. Пулинг, аналогичный второму слою.
  5. Свертка: каналов - 120, ядро - 5x5.
  6. Полносвязный слой из 84 нейронов.
  7. Полносвязный слой из 10 нейронов, после которого идет Softmax.

LeNet-5

Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	32x32	-	-	-
1	Convolution	6	28x28	5x5	1	tanh
2	Average Pooling	6	14x14	2x2	2	tanh
3	Convolution	16	10x10	5x5	1	tanh
4	Average Pooling	16	5x5	2x2	2	tanh
5	Convolution	120	1x1	5x5	1	tanh
6	FC	-	84	-	-	tanh
Output	FC	-	10	-	-	softmax

AlexNet (Крижевский, 2012 г.)

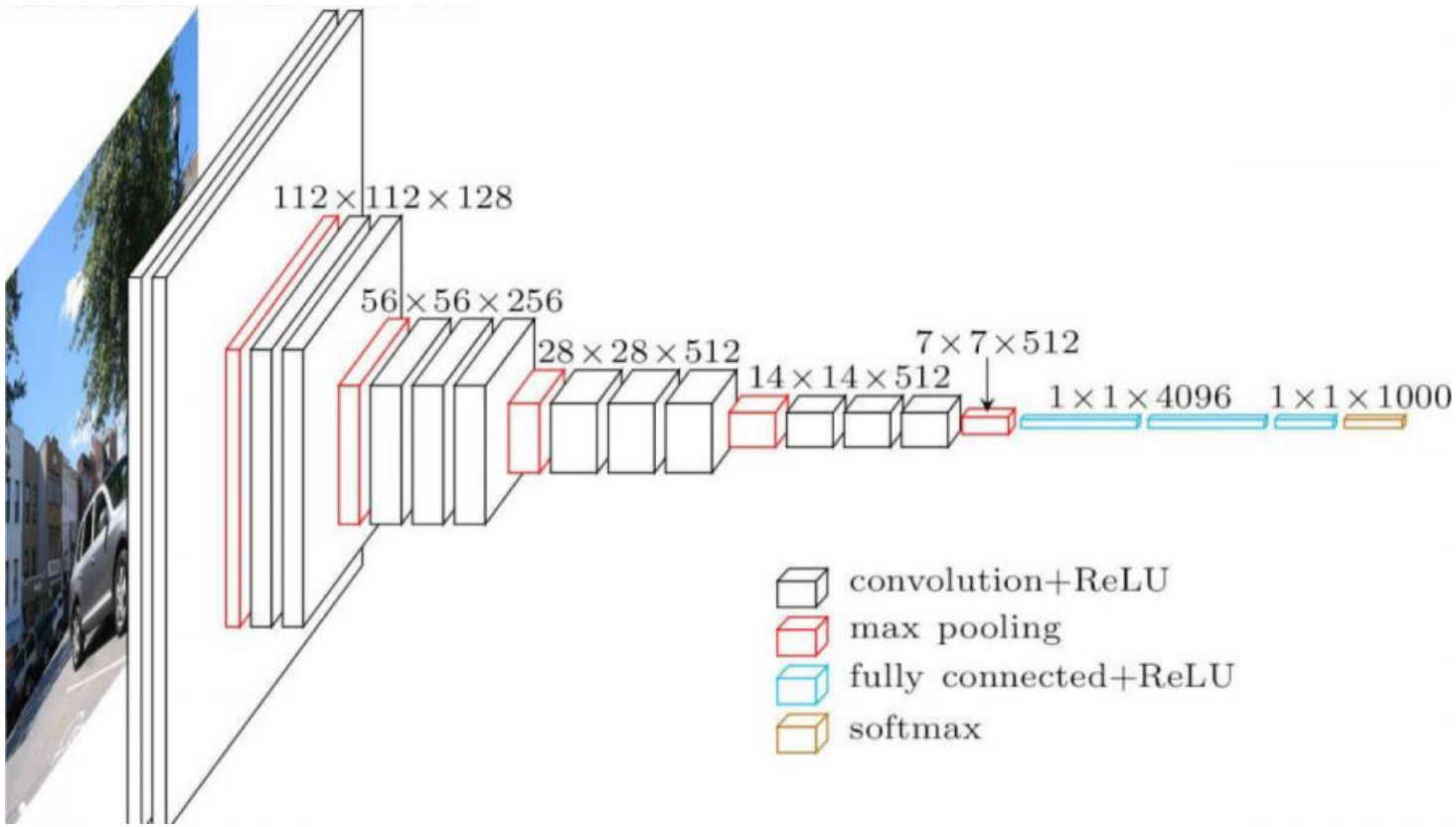


AlexNet

- Сеть имеет два вытянутых параллельных участка, чтобы обучать нейросеть параллельно на двух видеокартах Nvidia Geforce GTX 580
- Использовался стох. град. спуск (SGD), learning rate 0.01
- Аугментации данных (data augmentation)
- Сеть обучалась по батчам размера 128 и имела 60 миллионов параметров

VGG16

- K. Simonyan и A. Zisserman, точность 92.7% на ImageNet

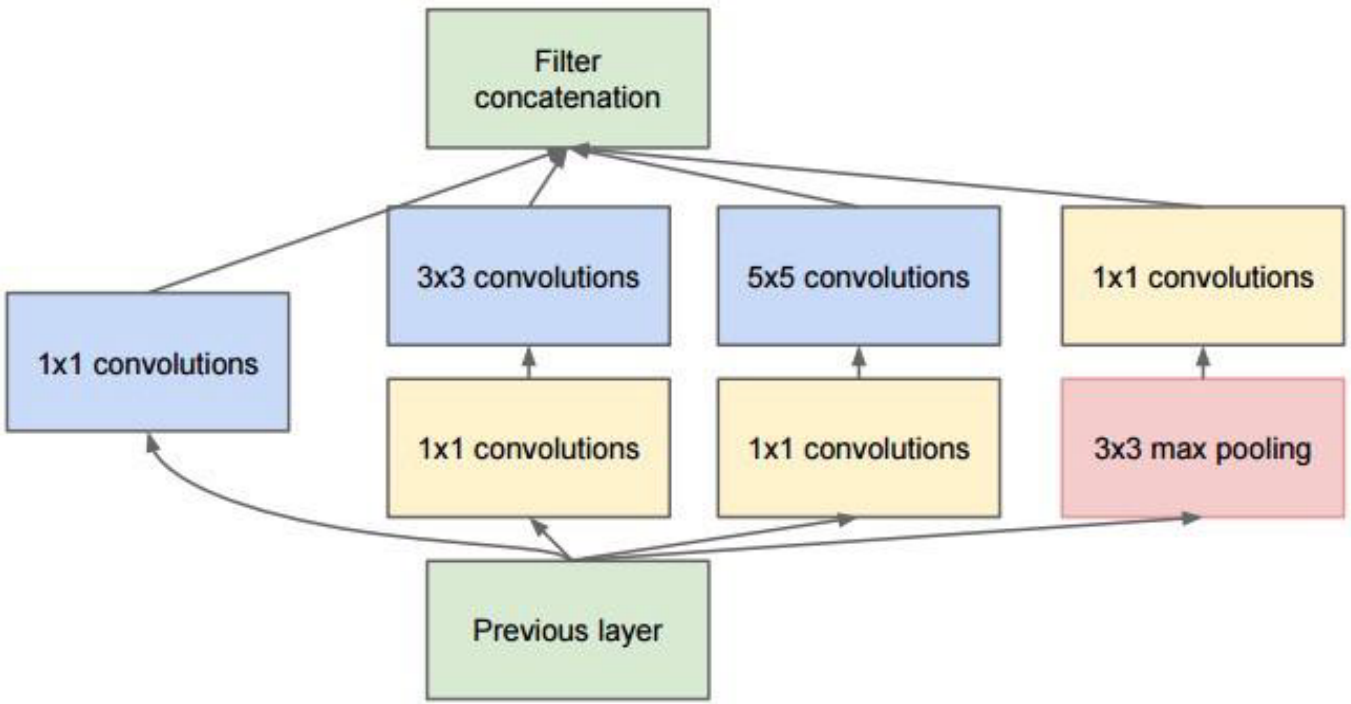


VGG16

- VGG использует свертки с малым размером ядра (3x3).
- Несколько сверток 3x3, объединенных в последовательность, могут эмулировать более крупные рецептивные поля, например, 5x5 или 7x7, а число обучаемых параметров меньше.
- Добавляются нулевые пиксели (padding)
- Уменьшение размера изображения только через max-pooling с размером ядра 2 и таким же шагом.
- Классификатор - 3 полносвязных слоев с Softmax.
- Размер сети более 130 миллионов параметров

GoogLeNet (C. Szegedy, 2014)

- Первая архитектура Inception (модуль Inception)



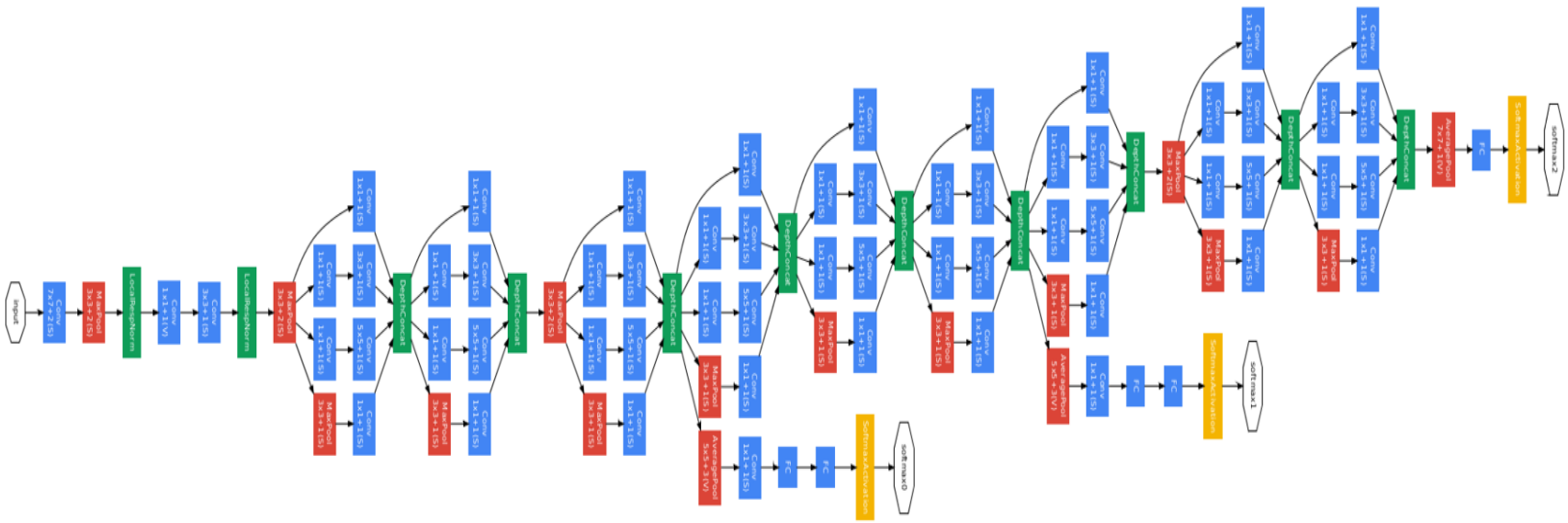
GoogLeNet (1)

- Сеть в виде конструктора, который собирается по блокам.
- Данные в блоке - по параллельным путям, которые затем конкатенируются, что позволяет выбирать наилучшее строение слоев самой сети. В результате обучения наиболее полезные пути станут вносить больший вклад в предсказание.

GoogLeNet (2)

- Использование сверток с ядром 1x1 (лин. комбинация карт признаков). Карты часто коррелированы между собой, поэтому ядра уменьшают число каналов, сохранив пространственные размеры. Свертки с ядром 1x1 ставят перед обычными сверточными слоями, что позволяет снизить количество параметров.
- Используются 2 дополн. выхода на более ранних слоях (с весом 0.3 к общей ошибке) для борьбы с затуханием градиента, так как сеть очень глубокая - 22 слоя. При тестировании эти пути удаляются.

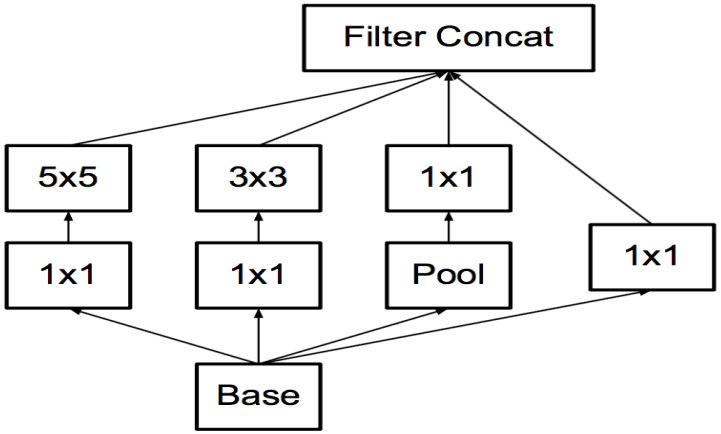
GoogLeNet

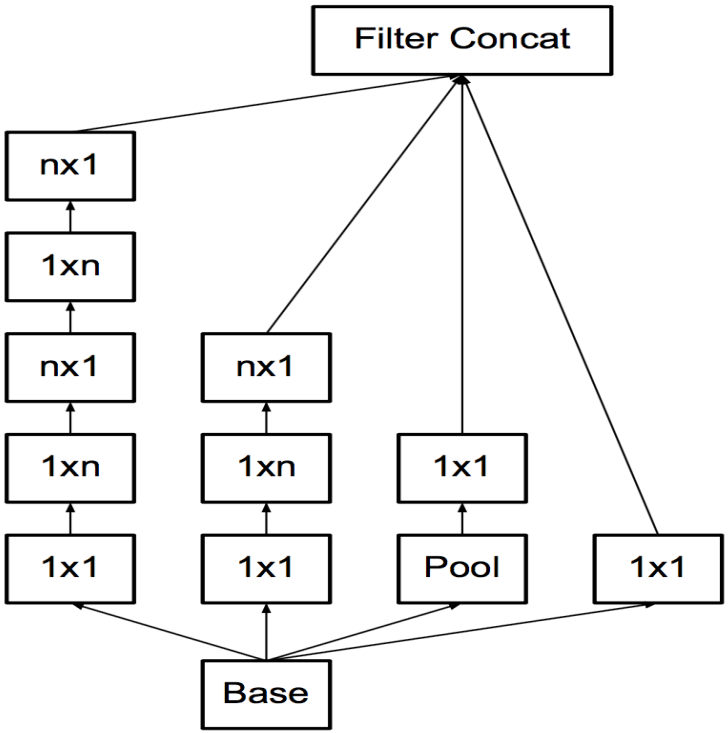


Inception v2 и v3

1. Следует избегать 1x1 сверток с сильным уровнем сжатия.
2. Следует соблюдать баланс между глубиной и шириной сети.
3. Можно заменить одну свертку с большими ядрами на несколько сверток с маленькими практически без потери качества.
4. На доп. выходах добавили батч-нормализацию как средство регуляризации.
5. Техника label-smoothing. Замена 1 и 0 на выходе смесью: softmax и распределение классов в датасете.

Inception v2 и v3

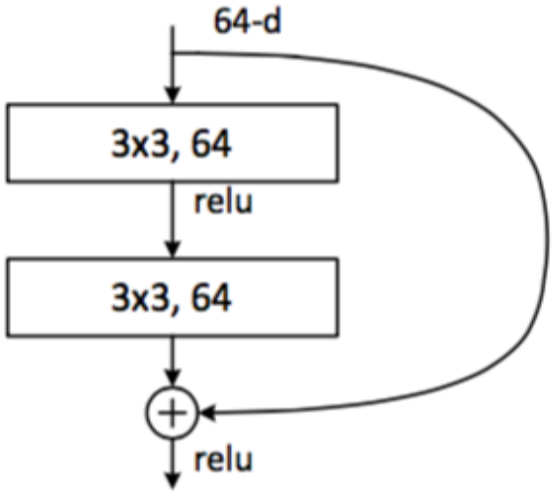


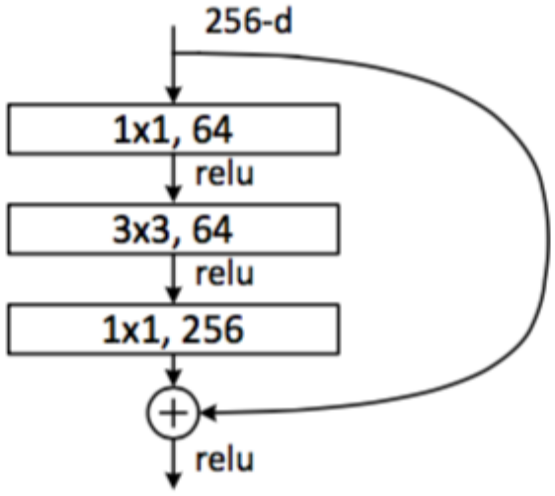


ResNet

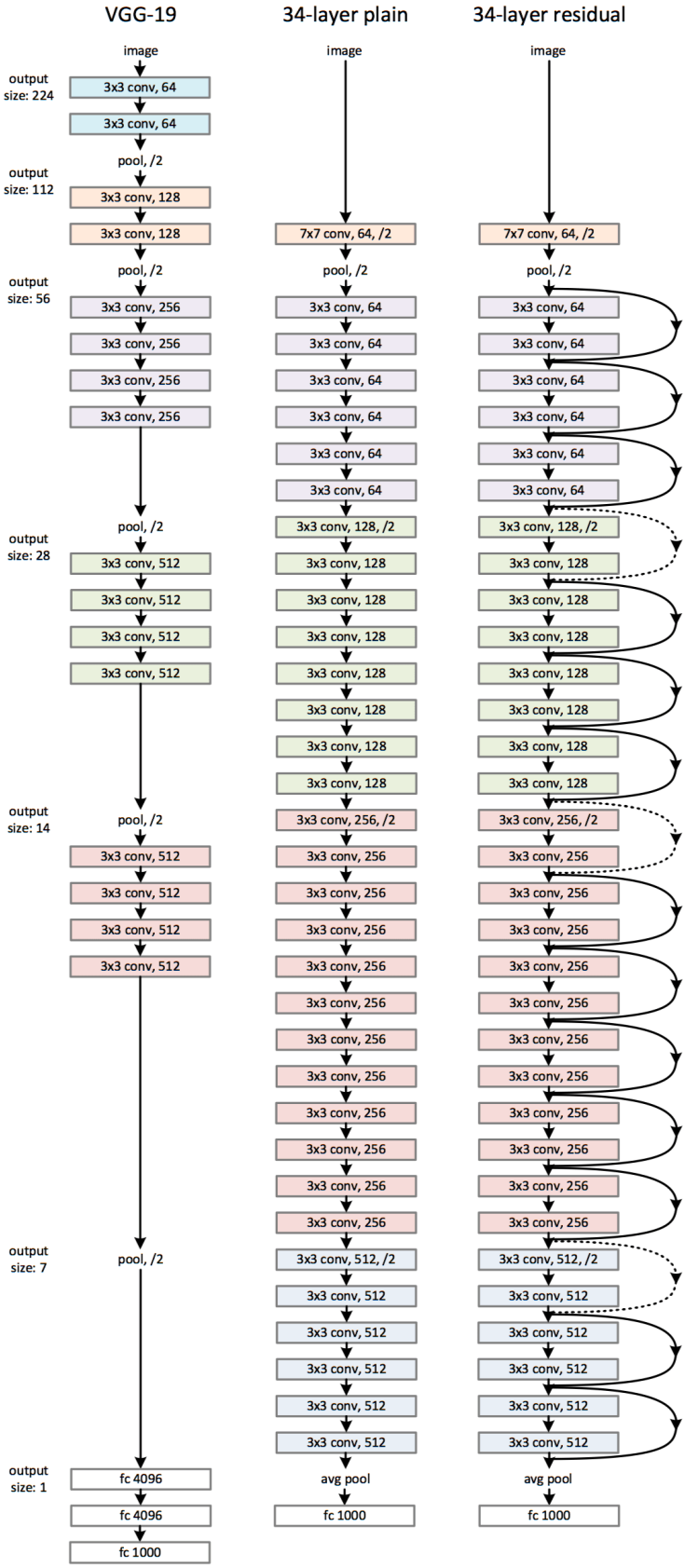
- Идея : пустить данные параллельно модулю через тождественный слой (identity layer) и затем просуммировать выходы.
- Блоки можно описать рекурсивной формулой $y_{i+1} = f_i(x_i) + x_i$.
- Градиент через skip/shortcut connections вычисляется умножением на единичную матрицу.
- Размерность тензора может поменяться при прохождении через Residual block, в таком случае делаются свертки и батч-нормализация.

ResNet


64-d
3x3, 64
relu
3x3, 64
relu

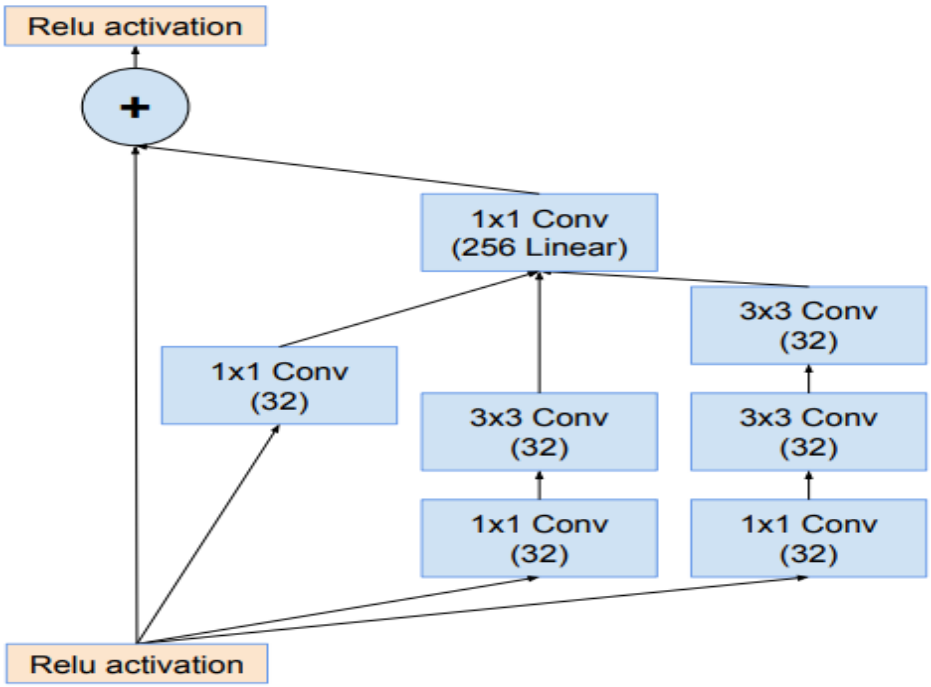

256-d
1x1, 64
relu
3x3, 64
relu
1x1, 256
relu

ResNet


VGG-19
34-layer plain
34-layer residual
image
output size: 224
3x3 conv, 64
3x3 conv, 64
pool, /2
output size: 112
3x3 conv, 128
3x3 conv, 128
pool, /2
output size: 56
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
pool, /2
output size: 28
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
pool, /2
output size: 14
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
pool, /2
output size: 7
fc 4096
fc 4096
fc 1000
output size: 1
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 512, /2
3x3 conv, 512
avg pool
fc 1000

Inception-v4 (Inception-ResNet)

В Inception-v3 добавили к существующей архитектуре shortcut connections

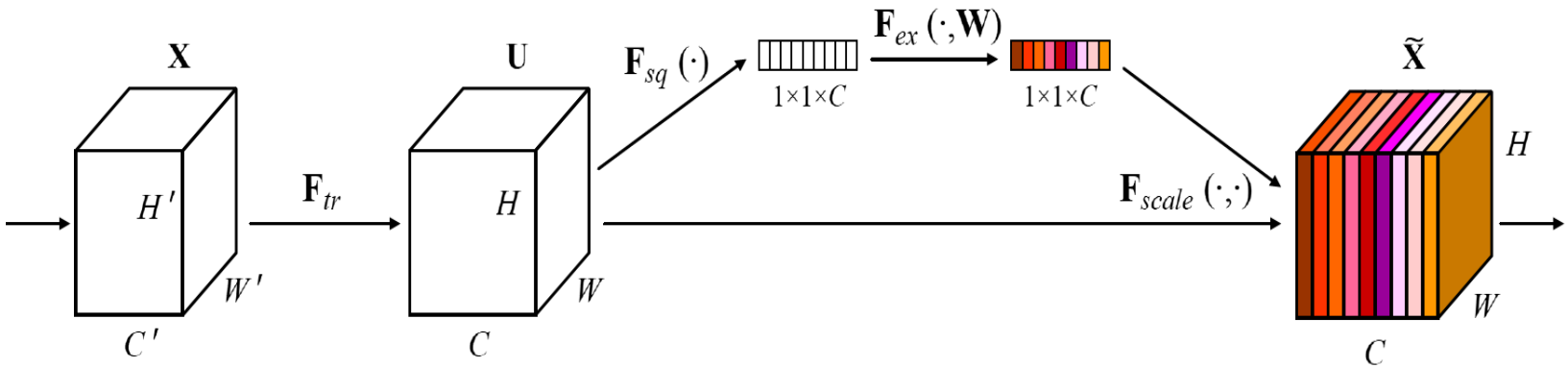


SENet

- Новое - адаптивная калибровка:
  - глобальный average-pooling с coxp-ем размерности по каналам, получаем одномерный вектор.
  - он пропускается через отдельную небольшую нейросеть, состоящую из линейного слоя, ReLU, линейного слоя и сигмоиды.
  - на выходе - вектор той же размерности, что и на входе, с элементами от 0 до 1.
  - далее каждый канал тензора карт признаков умножается на соответствующую компоненту полученного вектора.
  - т.о выполняется масштабирование каждого канала в зависимости от его значимости: полезные каналы умножаются на числа близкие к 1, а не особо важные на числа близкие к 0.

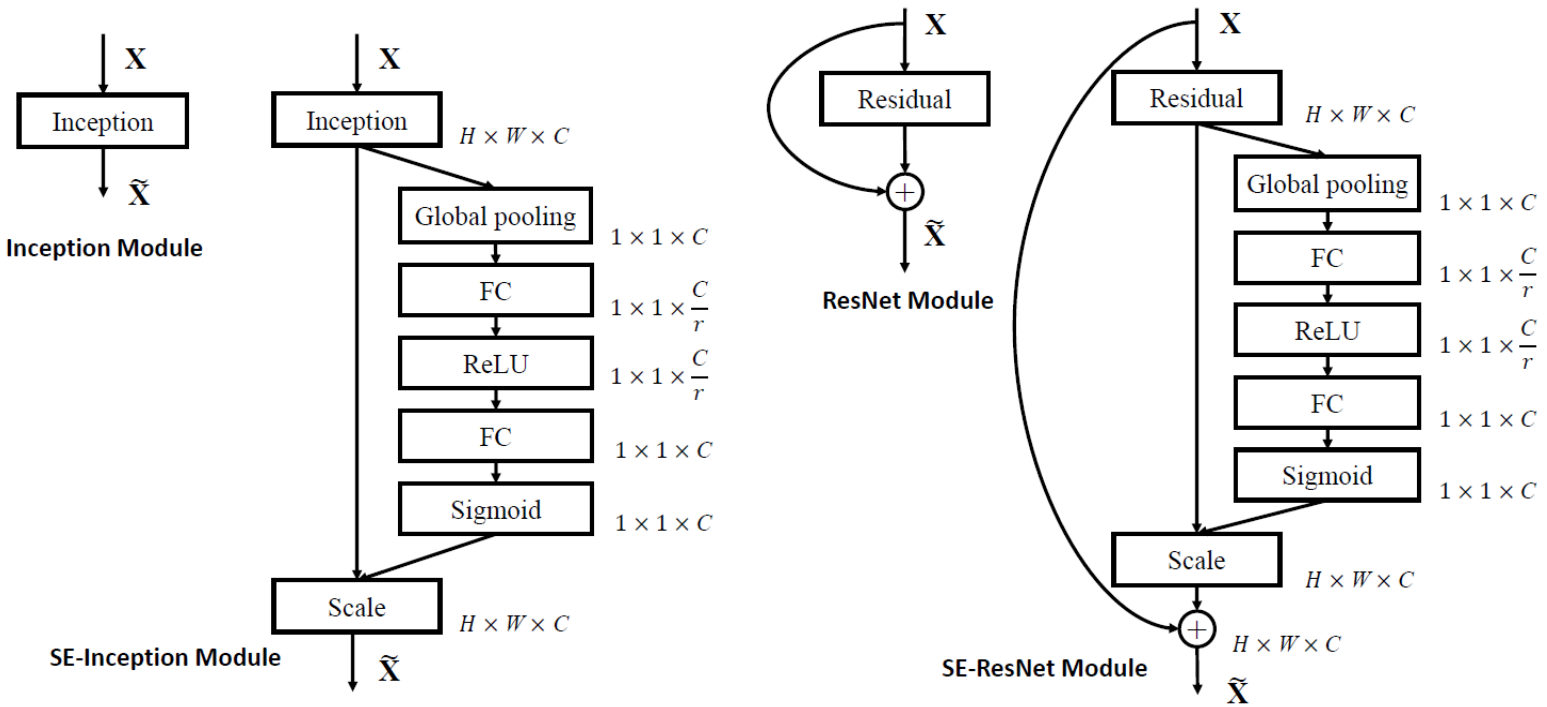
SENet

Squeeze-and-Excitation (SE) block



SENet

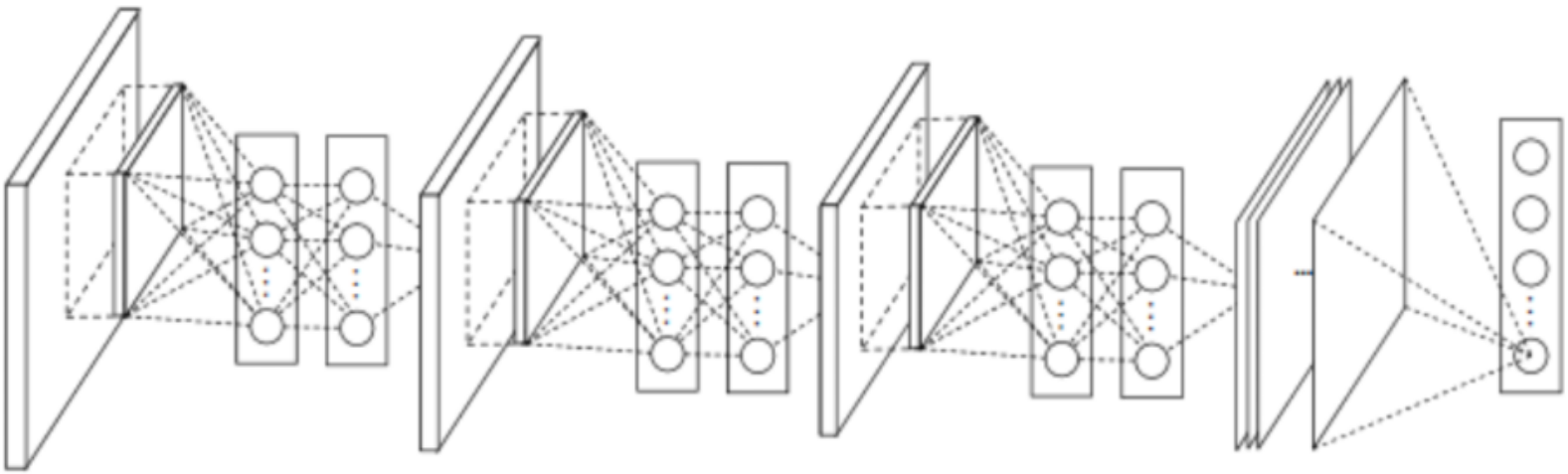
SE-блоки можно применять к другим сетям (SE-Inception модуль и SE-ResNet модуль)



Network in Network (1)

- Усложнение свертки добавлением внутрь нее небольшой нейронной сети на выход непосредственного применения ядра свертки к картам признаков.
- Особенность - расширение возможностей сети для глубокого распознавания сверточным преобразованием исходных образов и дополнительным учетом этих локальных структур для более качественного отображения набора признаков на следующие этапы.

Network in Network (2)



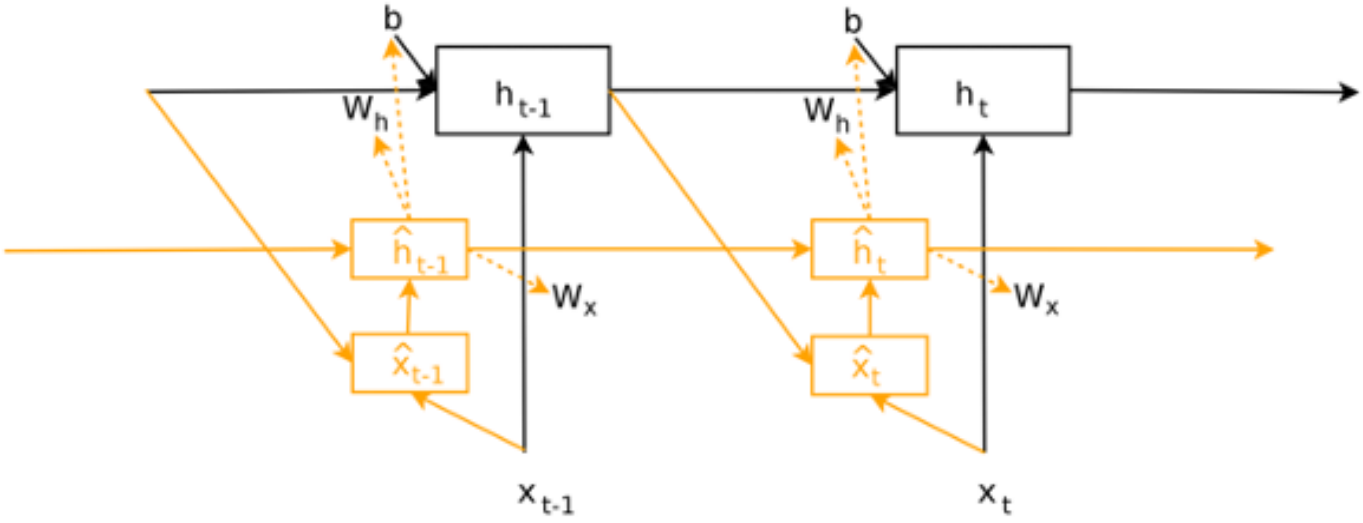
Network in Network (3)

- Преобразования тензоров становятся более сложными.
- Внутренние сети позволяют извлекать дополнительные полезные признаки внутри внешнего блока.
- В качестве добавленной внутренней структуры выступает многослойный персептрон, допускающий заметное увеличение сложности и гибкости схемы.
- В эту подсеть подаются небольшие наборы признаков, однако благодаря новым параметрам подсетей архитектура усложняется на качественном уровне.

HyperNets (1)

- **Идея**: перевод операций сети к динамическому выполнению и рег-ке сверток как рез-т работы другой нейронной сети - **гиперсети**
- **Статические гиперсети**: внешняя двухслойная сеть генерирует фильтр свертки из принятого эмбеддинга очер. слоя внутренней сети. Эмбеддинги настраиваются, и в процессе работы обученной сети берутся как аргументы для генерации сверток на тестировании
- **Динамические гиперсети**: надстройки для RNN. Сеть на каждом шаге t принимает на вход конкатенацию входного вектора x_t и скрытое состояние с предыдущего временного слоя рекуррентной сети h_{t-1} и генерирует следующее скрытое состояние, с помощью которого формируются веса модели на текущем временном шаге.

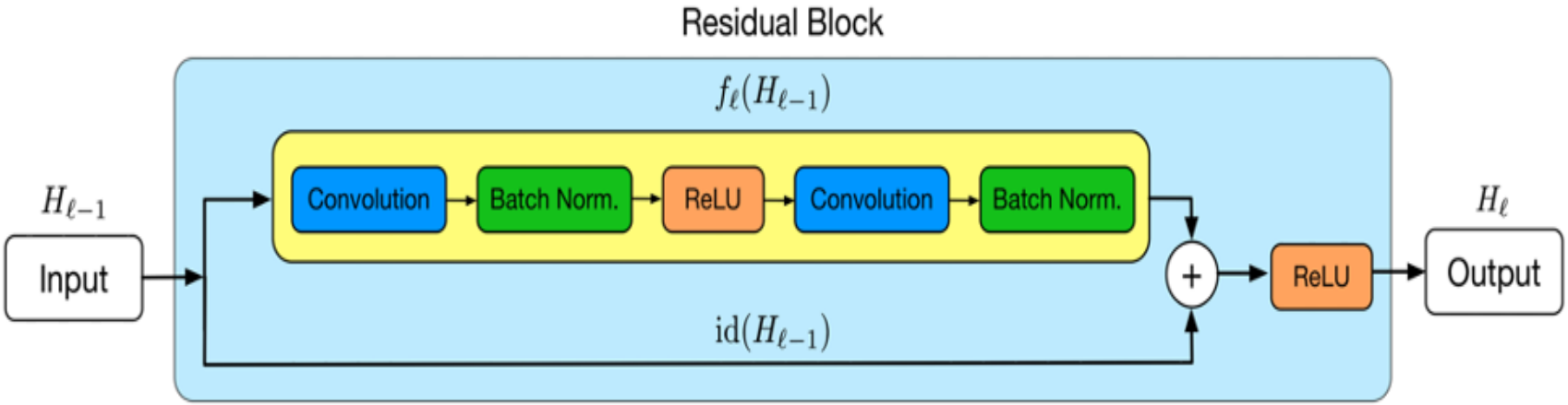
HyperNets (2)



Deep Networks with Stochastic Depth (1)

- Проблемы затухания градиента или снижение роли полезных признаков в процессе прямого распространения.
- На этапе обучения для каждого батча выкидывается большая часть слоев сети и производится настройка параметров оставшейся неглубокой части модели. Выкинутые слои заменяются тождественными преобразованиями набора признаков.
- На этапе теста используются все обученные таким образом слои.

Deep Networks with Stochastic Depth (2)



Deep Networks with Stochastic Depth (3)

- В качестве решающего правила выкидывания блока исходной архитектуры - значение с.в. с распределением Бернулли с параметром p_k
- Для реализации используют ResNet и исключают по такому распределению ее блоки, следующие за первым блоком Conv-BN-ReLU в сети
- Выход k-го слоя в таком случае можно выразить следующим образом: $H_k = \text{ReLU}(b_k f_k(H_{k1}) + id(H_{k1}))$
- Тест: $H_k^{Test} = \text{ReLU}\left(b_k f_k\left(H_{k-1}^{Test}, W_{k-1}\right) + H_{k-1}^{Test}\right)$

Deep Networks with Stochastic Depth (4)

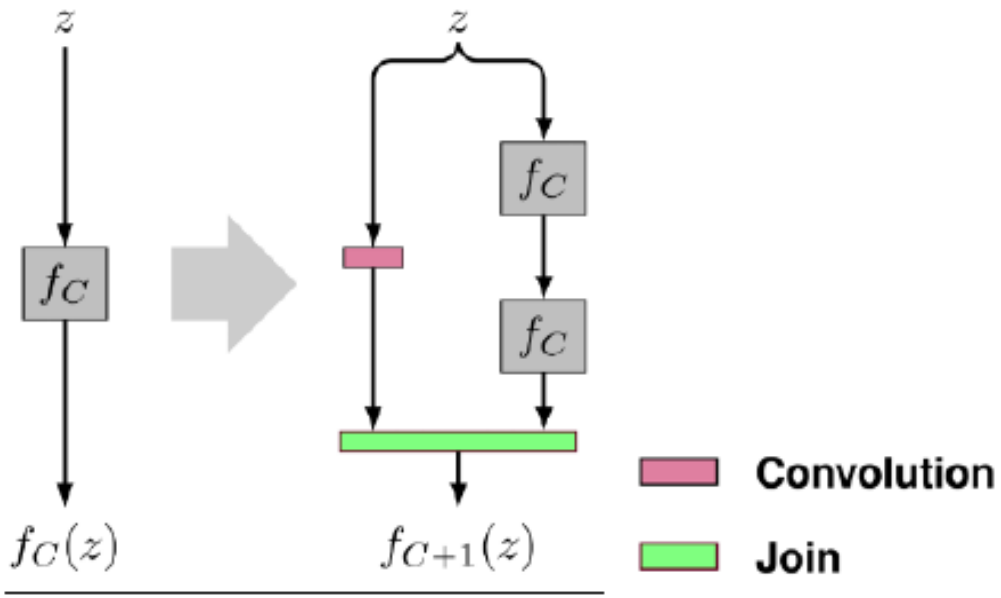
- Выход k-го слоя в таком случае можно выразить следующим образом: $H_k = \text{ReLU}(b_k f_k(H_{k1}) + id(H_{k1}))$
- Тест: $H_k^{Test} = \text{ReLU}\left(b_k f_k\left(H_{k-1}^{Test}, W_{k-1}\right) + H_{k-1}^{Test}\right)$
- Если L – общая глубина исходной сети, то $p_k = 1 - k/(1 - p_L)$
- Выборочное выкидывание слоев - эффективная модификация архитектур сверточных нейронных сетей.
- Позволяет добиться существенного ускорения обучения моделей.
- Позволяет сохранить и превзойти точность исходных моделей.

FractalNet (1)

- **Идея**: заданная фрактальная структура нейросети может достаточно хорошо связывать признаки разного уровня преобразований. В итоге это приводит к тому, что сеть можно отдельно настраивать на совместную обработку важных признаков и иметь возможность усложнять в процессе обучения архитектуру – эта идея соответствует подходу Fractal of FractalNet

FractalNet (2)

Правило рекуррентного усложнения архитектуры для достижения большей гибкости



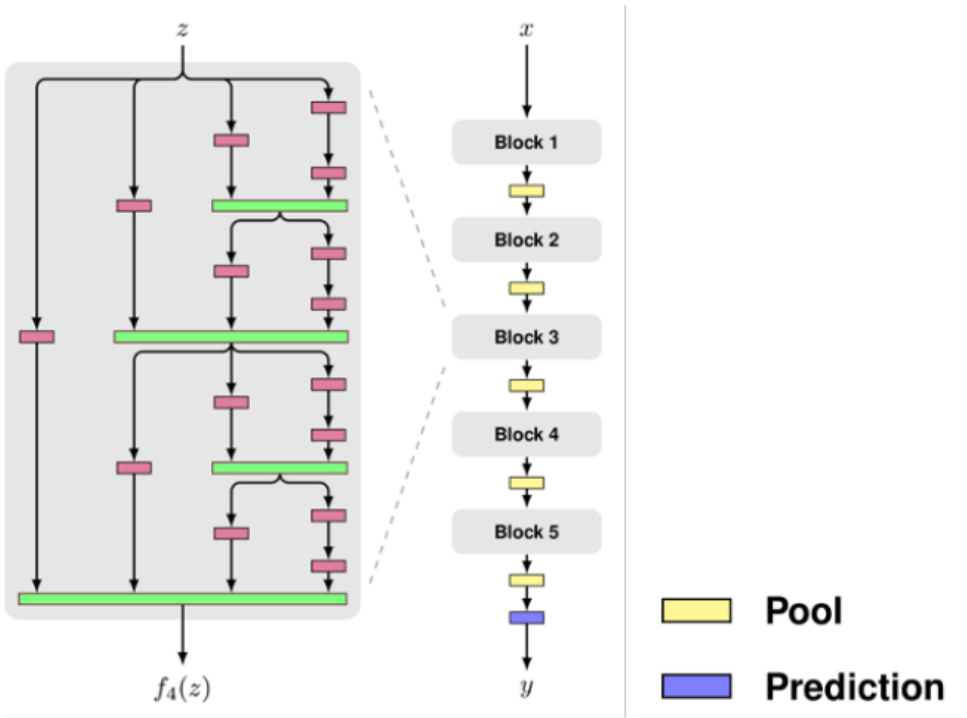
FractalNet (3)

- Формально его можно записать в следующем виде:

$$f_1(z) = conv(z)$$
$$f_{C+1}(z) = [(f_C * f_C)(z)] + [conv(z)]$$

- В итоге получается глубокая нейронная сеть с аналогичными результатами, как у ResNet
- Архитектура проста и не требует дополнительного прокидывания связей, как в ResNet

FractalNet (4)



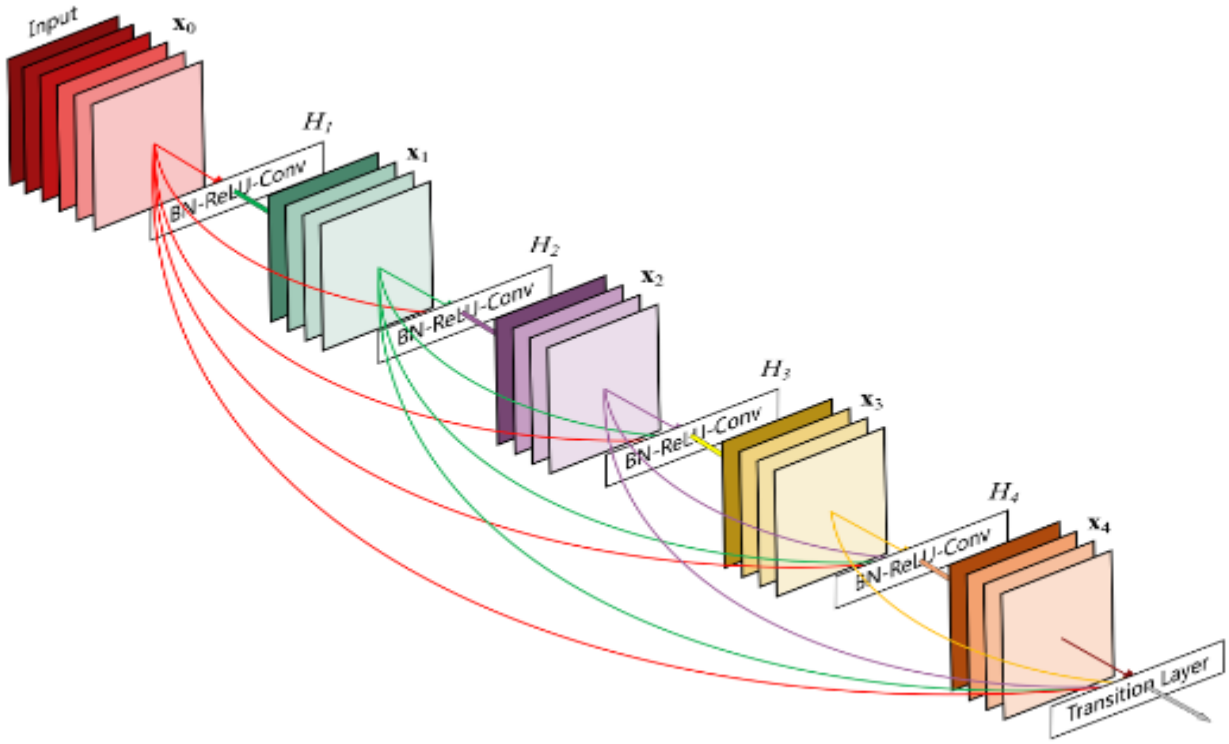
FractalNet (drop-path)

- Drop-path в FractalNet - аналог dropout в обычных сетях
- Заключается в исключении путей некоторых блоков
- **Локальная реализация**: Из блока сети каждая связь выбрасывается в соответствии с фиксированной вероятностью, при этом необходимо гарантировать, что по крайней мере один путь до выхода сети сохранится
- **Глобальная реализация**: из всей сети случайно выбирается единственный полноценный путь от входа до выхода, при этом выполняется условие, что он охватывает одинаковые пути по уровню фрактальной структуры.

Densely Connected Convolutional Networks (DenseNet) (1)

- Основной принцип архитектуры - в полном дополнении всех попарных связей между слоями сети. При объединении признаков, пришедших в один слой сети DenseNet, производится конкатенация, что способствует линейному росту признаков, обрабатываемой в слое. Это приводит к возможности сокращения параметров сети и вычислительным объемам

Densely Connected Convolutional Networks (DenseNet) (2)



Densely Connected Convolutional Networks (DenseNet) (3)

- Для обработки очередной карты признаков на k-том слое: $x_k = H_k([x_0, x_1, ..., x_{k-1}])$, т.е. каждый набор признаков получается преобразованием всех предыдущих. В сравнении с правилами ResNet-архитектур данное выражение кажется гораздо сложнее, но в реализации DenseNet используются определенные свойства, позволяющие избегать большие вычислительные затраты. Каждый слой состоит из комбинации блоков: BN + ReLU + 3x3 Convolution + dropout.

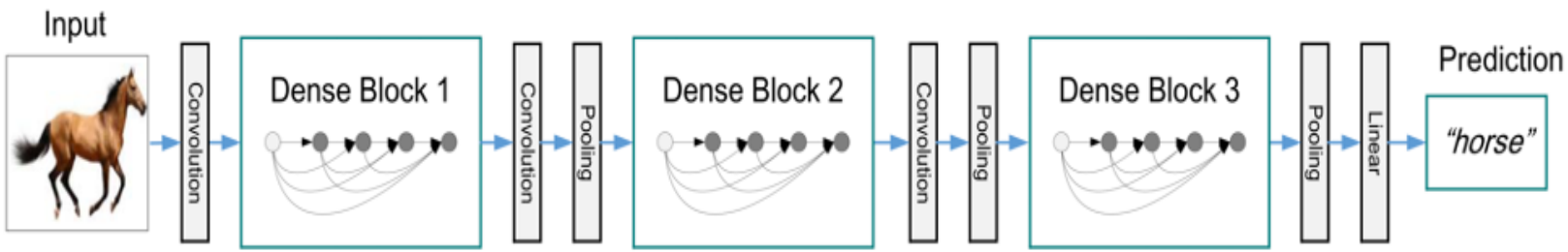
Densely Connected Convolutional Networks (DenseNet) (4)

- Для нейронной сети с L слоями требуется провести $L(L+1)/2$ связей.
- Каждый слой получает обработанную информацию со всех предыдущих.
- Одним из свойств метода является снижение требуемых параметров модели для обучения, так как получаемые на каждом слое данные аккумулируются, и появляется возможность из полученных слоем множества карт признаков получать фиксированное число карт (как правило это число равно 12 и является гиперпараметром модели), а оставшиеся – не подвергать изменениям.

Densely Connected Convolutional Networks (DenseNet) (5)

- Число параметров, настраиваемых сетью гораздо меньше, чем стандартной ResNet
- Улучшение информационного потока между слоями, входными и выходными данными. Так как каждый слой имеет непосредственную вычислительную связь с началом и концом сети, то как прямое, так и обратное распространение можно производить путем прямого доступа (до входных данных и до значения функции потерь на батче соответстсвенно).

Densely Connected Convolutional Networks (DenseNet) (6)



Densely Connected Convolutional Networks (DenseNet) (2)

- Для повышения эффективности DenseNet используют дополнительно два решения:
  - Для повышения выч. эффективности можно перейти к представлению слоя: [BN + ReLU + Conv(1x1)] + [BN + ReLU + Conv(3x3)]. Основные вычисления остаются во второй половине слоя. Добавление свертки 1x1 снижает количество карт признаков в слое
  - Регулируют долю карт признаков, переходящих от одного dense-блока к другому. За это отвечает гиперпараметр модели. На последующий dense-блок передается половина полученных карт признаков на текущем блоке. Это также позволяет повысить эффективность обучения.

Программное обеспечение Deep Learning: Torch

- Torch основан на библиотеке Lua
- Обработка естественного языка с помощью глубоких нейронных сетей
- Используется в Facebook и Twitter Research для исследований и разработки систем глубокого обучения

Программное обеспечение Deep Learning: MxNet

- MxNet - мощная библиотека, поддерживающая различные языки программирования: Python, Scala, R
- Одна из самых эффективных по быстродействию и по использованию памяти библиотек
- Простота использования нескольких графических процессоров (GPU)

Программное обеспечение Deep Learning: Theano

- Theano - Python-библиотека
- Объединяет Keras и Lasagne
- Охватывает не только глубокое обучение, но и различные методы машинного обучения: рекуррентная нейронная сеть, ограниченная машина Больцмана, глубокие сети доверия, сверточные нейронные сети
- Проста для разработчиков
- Имеются скрипты для конвертации моделей Caffe

Программное обеспечение Deep Learning: Lasagne

- Lasagne - библиотека для построения и обучения нейронных сетей в Theano
- Проста в использовании, понимании и расширении
- Для установки требует сначала установитьPython и Theano

Программное обеспечение Deep Learning: Keras

- Keras - модульная библиотека для построения нейронных сетей для Python
- Запускается “поверх” либо TensorFlow, либо Theano

Программное обеспечение Deep Learning: Caffe

- Caffe - флагман глубокого обучения
- Первая успешная открытая реализация с мощной, но простой базой: нет необходимости знать код для использования Caffe, используются простые файлы описаний сети
- Не поддерживает GPU, отличные от Nvidia

Программное обеспечение Deep Learning: TensorFlow

- TensorFlow - открытая библиотека для анализа представления данных в виде графа
- Вершины графа - математические операции, крайние вершины - матрицы данных большой размерности (тензоры)
- TensorFlow разработана в Google Brain Team в целях проведения исследования в области машинного обучения и глубоких нейронных сетей

Программное обеспечение Deep Learning: Deeplearning4j

- Deeplearning4j (DL4j) - JVM-фреймворк (Java Virtual Machine) для решения задач, связанных с большими данными

Вопросы

?