

СВЁРТОЧНЫЕ НЕЙРОННЫЕ СЕТИ

Сергей Николенко

ИТМО — Санкт-Петербург

5 октября 2026 г.

Random facts:

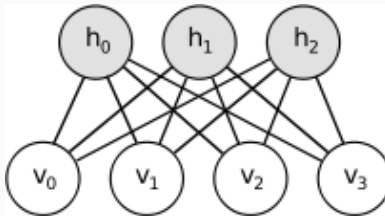
- 5 октября в ООН — Всемирный день учителей, а в России — ещё и День работников уголовного розыска
- 5 октября 1823 г. хирург Томас Уокли начал издавать первый медицинский журнал «The Lancet»
- 5 октября 1929 г. в Ленинграде состоялся первый в СССР сеанс звукового кино
- 5 октября 1962 г. в Великобритании вышли первый сингл The Beatles «Love Me Do» и первый фильм о Джеймсе Бонде «Dr. No», а ровно семь лет спустя на BBC — первый выпуск «Monty Python's Flying Circus»
- 5 октября 1988 г. на чилийском референдуме «Нет» набрало почти 56% при явке выше 88%, и Пиночету пришлось уйти после шестнадцати лет правления
- 5 октября 2000 г. массовые демонстрации в Сербии вынудили Слободана Милошевича уйти в отставку
- 5 октября 2021 г. вышла Windows 11; идея «Windows as a service» придётся подождать



- В прошлый раз мы разобрались, как вообще обучать нейронную сеть: функции активации, граф вычислений и backpropagation, варианты градиентного спуска от момента до Adam — и остановились на дропауте.
- Сегодня сначала добъём то, что осталось от «как обучать»: *инициализацию весов и нормализацию по мини-батчам*.
- А потом перейдём к главной архитектуре компьютерного зрения — *свёрточным сетям*: откуда они взялись, как устроены, что видят внутри и как выглядит их современный зоопарк.

ИНИЦИАЛИЗАЦИЯ ВЕСОВ

- Революция глубокого обучения началась с *предобучением без учителя* (unsupervised pretraining).
- Главная идея: добраться до хорошей области пространства весов, затем уже сделать fine-tuning градиентным спуском.
- Ограниченные машины Больцмана (restricted Boltzmann machines):



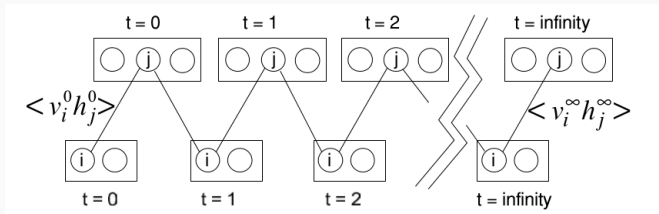
ПРЕДОБУЧЕНИЕ БЕЗ УЧИТЕЛЯ

- Это ненаправленная графическая модель, задающая распределение

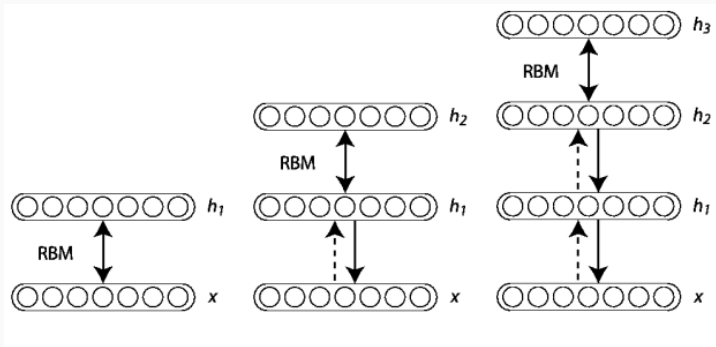
$$p(\mathbf{v}) = \sum_{\mathbf{h}} p(\mathbf{v}, \mathbf{h}) = \frac{1}{Z} \sum_{\mathbf{h}} e^{-E(\mathbf{v}, \mathbf{h})}, \text{ где}$$

$$E(\mathbf{v}, \mathbf{h}) = -\mathbf{b}^\top \mathbf{v} - \mathbf{c}^\top \mathbf{h} - \mathbf{h}^\top W \mathbf{v}.$$

- Обучают алгоритмом Contrastive Divergence (приближение к сэмплированию по Гиббсу).

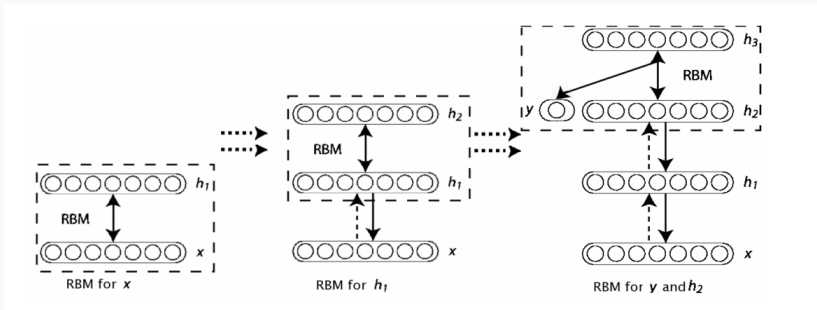


- Из RBM можно сделать глубокие сети, поставив одну на другую:



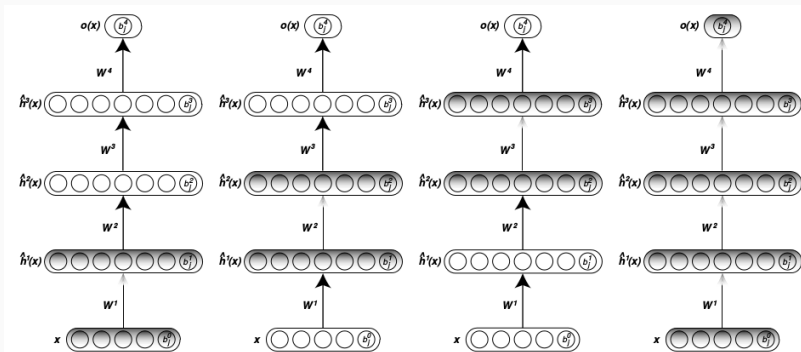
ПРЕДОБУЧЕНИЕ БЕЗ УЧИТЕЛЯ

- И вывод можно вести последовательно, уровень за уровнем:



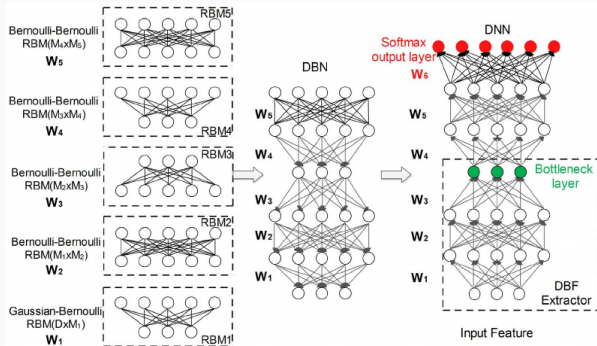
ПРЕДОБУЧЕНИЕ БЕЗ УЧИТЕЛЯ

- А потом уже дообучать градиентным спуском (fine-tuning).



ПРЕДОБУЧЕНИЕ БЕЗ УЧИТЕЛЯ

- Этот подход привёл к прорыву в распознавании речи.



- Но обучать глубокие сети из RBM довольно сложно, они хрупкие, и вычислительно тоже нелегко.
- И сейчас уже не очень-то и нужны сложные модели вроде RBM для того, чтобы попасть в хорошую начальную область.
- Инициализация весов — важная часть этого.

- *Xavier initialization* (Glorot, Bengio, 2010).
- Рассмотрим простой линейный нейрон:

$$y = \mathbf{w}^\top \mathbf{x} + b = \sum_i w_i x_i + b.$$

- Его дисперсия равна

$$\begin{aligned} \text{Var}[y_i] &= \text{Var}[w_i x_i] = \mathbb{E}[w_i^2 x_i^2] - (\mathbb{E}[w_i x_i])^2 = \\ &= \mathbb{E}[x_i]^2 \text{Var}[w_i] + \mathbb{E}[w_i]^2 \text{Var}[x_i] + \text{Var}[w_i] \text{Var}[x_i]. \end{aligned}$$

- Его дисперсия равна

$$\begin{aligned}\text{Var}[y_i] &= \text{Var}[w_i x_i] = \mathbb{E}[w_i^2 x_i^2] - (\mathbb{E}[w_i x_i])^2 = \\ &= \mathbb{E}[x_i]^2 \text{Var}[w_i] + \mathbb{E}[w_i]^2 \text{Var}[x_i] + \text{Var}[w_i] \text{Var}[x_i].\end{aligned}$$

- Для нулевого среднего весов

$$\text{Var}[y_i] = \text{Var}[w_i] \text{Var}[x_i].$$

- И если w_i и x_i инициализированы независимо из одного и того же распределения,

$$\text{Var}[y] = \text{Var}\left[\sum_{i=1}^{n_{\text{out}}} y_i\right] = \sum_{i=1}^{n_{\text{out}}} \text{Var}[w_i x_i] = n_{\text{out}} \text{Var}[w_i] \text{Var}[x_i].$$

- Иначе говоря, дисперсия на выходе пропорциональна дисперсии на входе с коэффициентом $n_{\text{out}} \text{Var}[w_i]$.

ИНИЦИАЛИЗАЦИЯ КСАВЬЕ

- До (Glorot, Bengio, 2010) стандартным способом инициализации было

$$w_i \sim U \left[-\frac{1}{\sqrt{n_{\text{out}}}}, \frac{1}{\sqrt{n_{\text{out}}}} \right].$$

- См., например, *Neural Networks: Tricks of the Trade*.
- Так что с дисперсиями получается

$$\text{Var}[w_i] = \frac{1}{12} \left(\frac{1}{\sqrt{n_{\text{out}}}} + \frac{1}{\sqrt{n_{\text{out}}}} \right)^2 = \frac{1}{3n_{\text{out}}}, \text{ и}$$

$$n_{\text{out}} \text{Var}[w_i] = \frac{1}{3},$$

и после нескольких уровней сигнал совсем умирает; аналогичный эффект происходит и в backprop.

- Инициализация Ксавье — давайте попробуем уменьшить изменение дисперсии, т.е. взять

$$\text{Var}[w_i] = \frac{2}{n_{\text{in}} + n_{\text{out}}};$$

для равномерного распределения это

$$w_i \sim U \left[-\frac{\sqrt{6}}{\sqrt{n_{\text{in}} + n_{\text{out}}}}, \frac{\sqrt{6}}{\sqrt{n_{\text{in}} + n_{\text{out}}}} \right].$$

- Но это работает только для симметричных активаций, т.е. не для ReLU...

- ...до работы (He et al., 2015). Вернёмся к

$$\text{Var} [w_i x_i] = \mathbb{E} [x_i]^2 \text{Var} [w_i] + \mathbb{E} [w_i]^2 \text{Var} [x_i] + \text{Var} [w_i] \text{Var} [x_i]$$

- Мы теперь можем обнулить только второе слагаемое:

$$\text{Var} [w_i x_i] = \mathbb{E} [x_i]^2 \text{Var} [w_i] + \text{Var} [w_i] \text{Var} [x_i] = \text{Var} [w_i] \mathbb{E} [x_i^2], \text{ и}$$

$$\text{Var} [y^{(l)}] = n_{\text{in}}^{(l)} \text{Var} [w^{(l)}] \mathbb{E} \left[(x^{(l)})^2 \right].$$

- Мы теперь можем обнулить только второе слагаемое:

$$\text{Var} [y^{(l)}] = n_{\text{in}}^{(l)} \text{Var} [w^{(l)}] \mathbb{E} \left[(x^{(l)})^2 \right].$$

- Предположим, что $x^{(l)} = \max(0, y^{(l-1)})$, и у $y^{(l-1)}$ симметричное распределение вокруг нуля. Тогда

$$\mathbb{E} \left[(x^{(l)})^2 \right] = \frac{1}{2} \text{Var} [y^{(l-1)}], \quad \text{Var} [y^{(l)}] = \frac{n_{\text{in}}^{(l)}}{2} \text{Var} [w^{(l)}] \text{Var} [y^{(l-1)}].$$

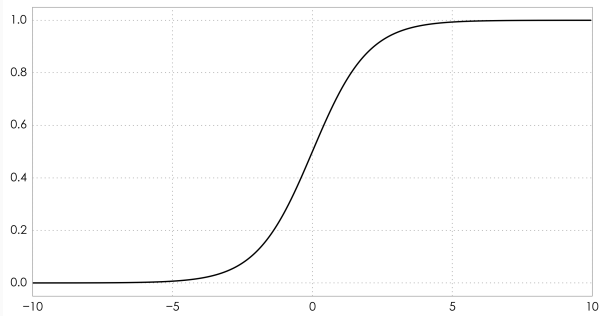
- И это приводит к формуле для дисперсии активации ReLU; теперь нет никакого n_{out} :

$$\text{Var}[w_i] = 2/n_{\text{in}}^{(l)}.$$

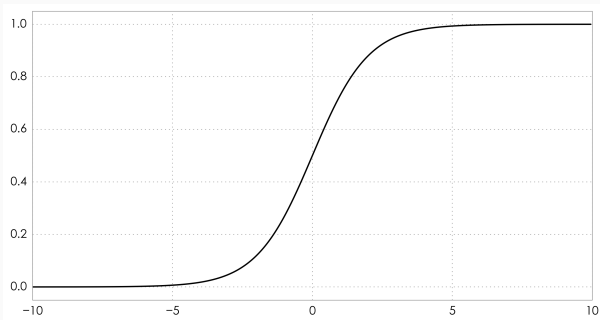
- Кстати, равномерную инициализацию делать не обязательно, можно и нормальное распределение:

$$w_i \sim N\left(0, \sqrt{2/n_{\text{in}}^{(l)}}\right).$$

- Кстати, о (Glorot, Bengio, 2010) – ещё одна важная идея.
- Эксперименты показали, что $\sigma(x) = \frac{1}{1+e^{-x}}$ работает в глубоких сетях довольно плохо.



- *Насыщение*: если $\sigma(x)$ уже «обучилась», т.е. даёт большие по модулю значения, то её производная близка к нулю и «поменять мнение» трудно.

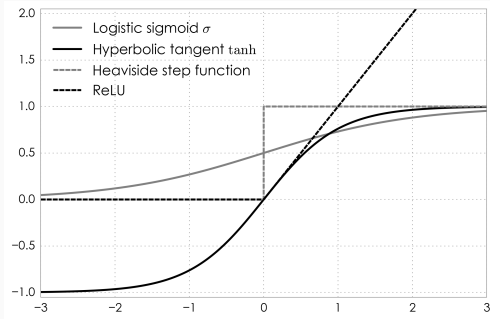


- Но ведь другие тоже насыщаются? В чём разница?

- Рассмотрим последний слой сети $h(W\mathbf{a} + \mathbf{b})$, где $\mathbf{a}$ — выходы предыдущего слоя, $\mathbf{b}$ — свободные члены, h — функция активации последнего уровня, обычно softmax .
- Когда мы начинаем оптимизировать сложную функцию потерь, поначалу выходы $\mathbf{h}$ не несут полезной информации о входах, ведь первые уровни ещё не обучены.
- Тогда неплохим приближением будет константная функция, выдающая средние значения выходов.
- Это значит, что $h(W\mathbf{a} + \mathbf{b})$ подберёт подходящие свободные члены $\mathbf{b}$ и постарается обнулить слагаемое $W\mathbf{h}$, которое поначалу скорее шум, чем сигнал.

О СИГМОИДАХ

- Иначе говоря, в процессе обучения мы постараемся привести выходы предыдущего слоя к нулю.
- Здесь и проявляется разница: у $\sigma(x) = \frac{1}{1+e^{-x}}$ область значений $(0, 1)$ при среднем $\frac{1}{2}$, и при $\sigma(x) \rightarrow 0$ будет и $\sigma'(x) \rightarrow 0$.
- А у $\tanh$ наоборот: когда $\tanh(x) \rightarrow 0$, $\tanh'(x)$ максимальна.

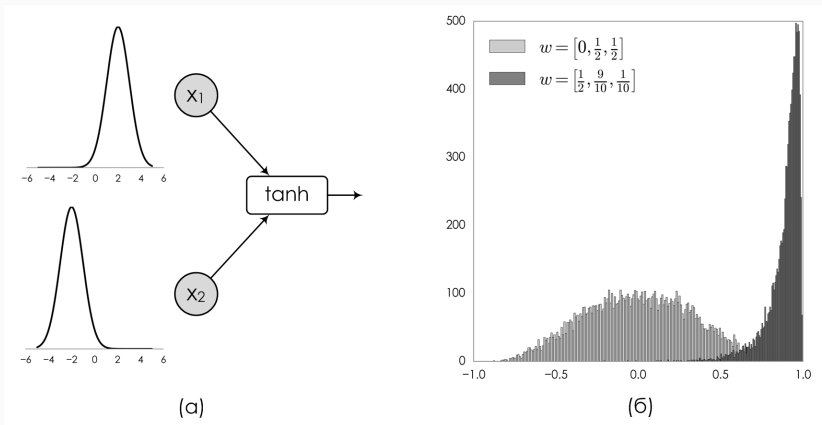


НОРМАЛИЗАЦИЯ ПО МИНИ-БАТЧАМ

- Ещё одна важная проблема в глубоких сетях: *внутренний сдвиг переменных* (internal covariate shift).
- Когда меняются веса слоя, меняется распределение его выходов.
- Это значит, что следующему уровню придётся всё начинать заново, он же не ожидал таких входов, не видел их раньше!
- Более того, нейроны следующего уровня могли уже и насытиться, и им теперь сложно быстро обучиться заново.
- Это серьёзно мешает обучению.

НОРМАЛИЗАЦИЯ ПО МИНИ-БАТЧАМ

- Вот характерный пример:



- Что делать?

НОРМАЛИЗАЦИЯ ПО МИНИ-БАТЧАМ

- Можно пытаться нормализовать входы каждого уровня.
- Не работает: рассмотрим для простоты уровень с одним только bias b и входами $\mathbf{u}$:

$$\hat{\mathbf{x}} = \mathbf{x} - \mathbb{E}[\mathbf{x}], \text{ где } \mathbf{x} = \mathbf{u} + b.$$

- На следующем шаге градиентного спуска получится $b := b + \Delta b \dots$
- ...но $\hat{\mathbf{x}}$ не изменится:

$$\mathbf{u} + b + \Delta b - \mathbb{E}[\mathbf{u} + b + \Delta b] = \mathbf{u} + b - \mathbb{E}[\mathbf{u} + b].$$

- Так что всё обучение сведётся к тому, что b будет неограниченно расти — не очень хорошо.

- Можно пытаться добавить нормализацию как отдельный слой:

$$\hat{\mathbf{x}} = \text{Norm}(\mathbf{x}, X).$$

- Это лучше, но теперь этому слою на входе нужен весь датасет X !
- И на шаге градиентного спуска придётся вычислить $\frac{\partial \text{Norm}}{\partial \mathbf{x}}$ и $\frac{\partial \text{Norm}}{\partial X}$, да ещё и матрицу ковариаций

$$\text{Cov}[\mathbf{x}] = \mathbb{E}_{\mathbf{x} \in X} [\mathbf{x} \mathbf{x}^\top] - \mathbb{E}[\mathbf{x}] \mathbb{E}[\mathbf{x}]^\top.$$

- Это точно не работает.

НОРМАЛИЗАЦИЯ ПО МИНИ-БАТЧАМ

- Решение в том, чтобы нормализовать каждый вход отдельно, и не по всему датасету, а по текущему кусочку; это и есть *нормализация по мини-батчам* (batch normalization).
- После нормализации по мини-батчам получим

$$\hat{x}_k = \frac{x_k - \mathbb{E}[x_k]}{\sqrt{\text{Var}[x_k]}},$$

где статистики подсчитаны по текущему мини-батчу.

- Ещё одна проблема: теперь пропадают нелинейности!
- Например, σ теперь практически всегда близка к линейной.

- Чтобы это исправить, нужно добавить гибкости уровню batchnorm.
- В частности, нужно разрешить ему обучаться иногда *ничего не делать* со входами.
- Так что вводим дополнительные параметры (сдвиг и растяжение):

$$y_k = \gamma_k \hat{x}_k + \beta_k = \gamma_k \frac{x_k - \mathbb{E}[x_k]}{\sqrt{\text{Var}[x_k]}} + \beta_k.$$

- γ_k и β_k — это новые переменные, тоже будут обучаться градиентным спуском, как веса.

НОРМАЛИЗАЦИЯ ПО МИНИ-БАТЧАМ

- И ещё добавим ϵ в знаменатель, чтобы на ноль не делить.
- Теперь мы можем формально описать слой батч-нормализации — для очередного мини-батча $B = \{\mathbf{x}_1, \dots, \mathbf{x}_m\}$:

- вычислить базовые статистики по мини-батчу

$$\mu_B = \frac{1}{m} \sum_{i=1}^m \mathbf{x}_i, \quad \sigma_B^2 = \frac{1}{m} \sum_{i=1}^m (\mathbf{x}_i - \mu_B)^2,$$

- нормализовать входы

$$\hat{\mathbf{x}}_i = \frac{\mathbf{x}_i - \mu_b}{\sqrt{\sigma_B^2 + \epsilon}},$$

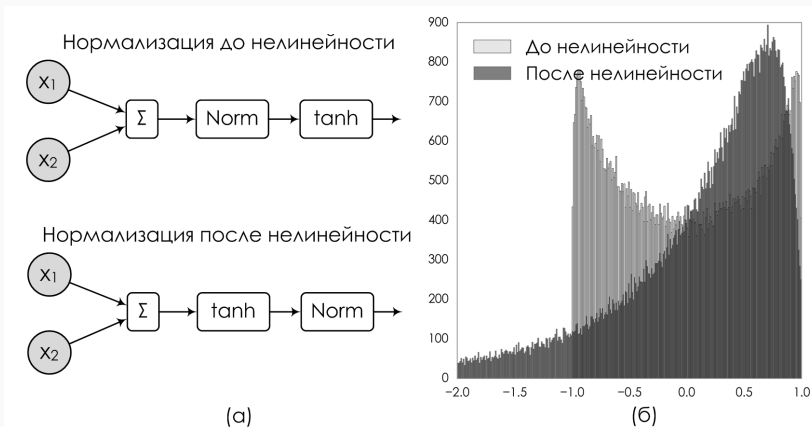
- вычислить результат

$$\mathbf{y}_i = \gamma \mathbf{x}_i + \beta.$$

- Через всё это совершенно стандартным образом пропускаются градиенты, в том числе по γ и β .

НОРМАЛИЗАЦИЯ ПО МИНИ-БАТЧАМ

- Последнее замечание: важно, куда поместить слой batchnorm.
- Можно до, а можно после нелинейности.



Что BN на самом деле делает

- Изначально идея была в том, чтобы сократить internal covariate shift.
- Но выяснилось, что от этого эффективность BN не особенно зависит, но BN помогает с регуляризацией, делает дропаут ненужным и т.д. Возможные причины:
 - BN сглаживает градиенты, уменьшает липшицеву константу (Santurkar et al., 2019);
 - BN разделяет обучение направления и длины векторов весов, что улучшает обучение;
 - есть результаты о том, что BN помогает добиться линейной сходимости в обычном градиентном спуске.

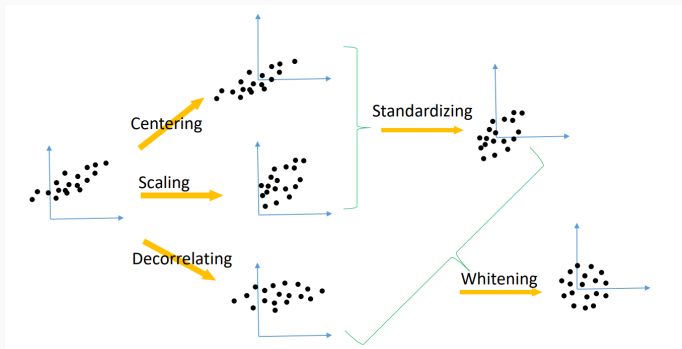
- Нормализация по мини-батчам сейчас стала фактически стандартом.
- Очередная очень крутая история, примерно как дропаут.
- Но мысль идёт и дальше:
 - (Laurent et al., 2016): BN не помогает рекуррентным сетям;
 - (Cooijmans et al., 2016): recurrent batch normalization;
 - (Salimans and Kingma, 2016): нормализация весов для улучшения обучения — давайте добавим веса как

$$\mathbf{h}_i = f \left(\frac{\gamma}{\|\mathbf{w}_i\|} \mathbf{w}_i^\top \mathbf{x} + b_i \right);$$

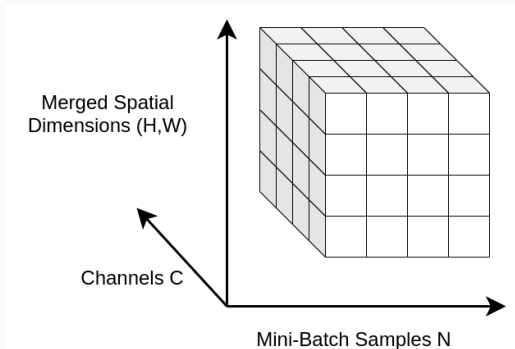
тогда мы будем перемасштабировать градиент, стабилизировать его норму и приближать его матрицу ковариаций к единичной, что улучшает обучение.

- Давайте разберёмся в ландшафте методов нормализации

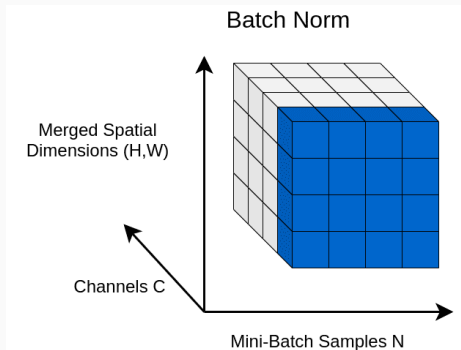
- В целом все эти методы используют одни и те же операции:



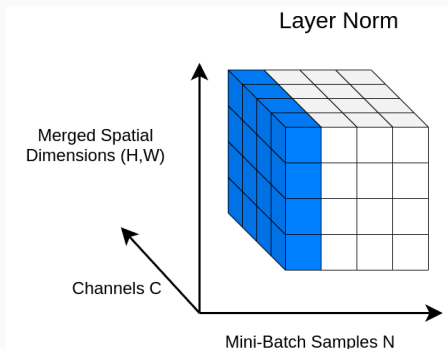
- И различаются набором операций и тем, по каким размерностям они применяются:



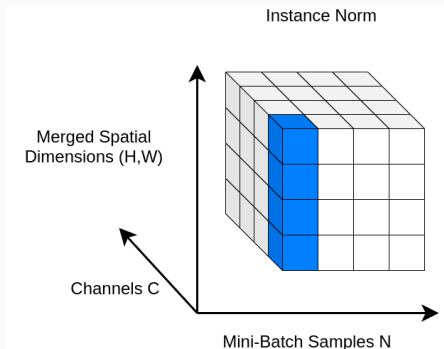
- Batch norm – по мини-батчу (часто используется для изображений):



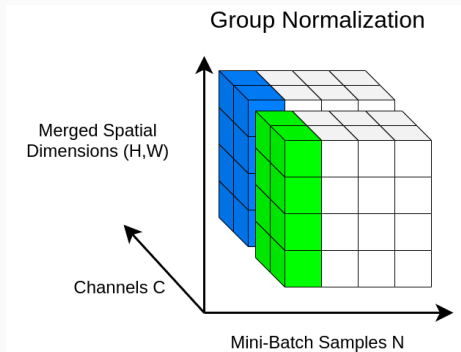
- Layer norm – по каналам и по размерности, т.е. по всем выходам слоя (используется в основном для векторов, в частности в трансформерах):



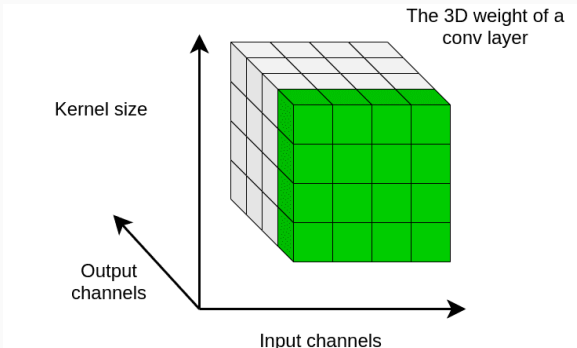
- Instance norm – только по размерности, т.е. каналы убираем (используется, например, в AdaIN для переноса стиля):



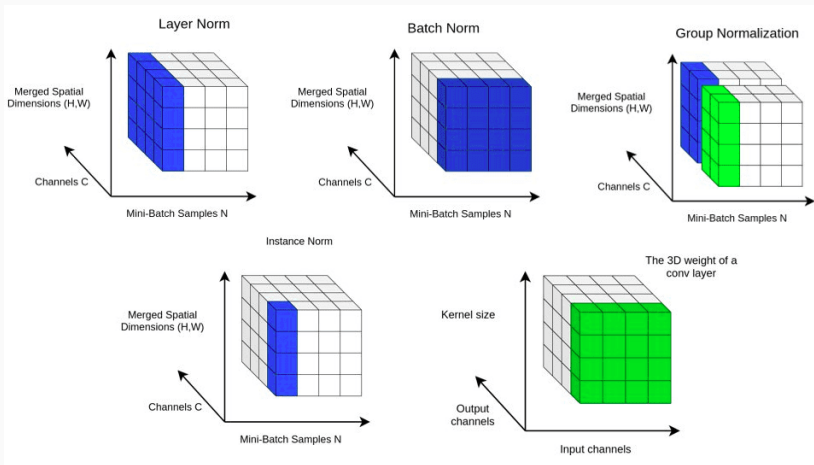
- Group norm – можно разделить каналы на группы:



- Weight standardization – усредним веса, а не активации, отдельно для каждого канала, т.е. в том числе нормализуем градиенты при backpropagation (group norm + weight standardization использовались в BiT – Big Transfer):



- Итого:



- Но вообще это, как всегда, большая наука (Huang et al., 2020):

Method	NAP	NOP	NRR	Published In
Local contrast normalization [108]	$\Pi_{IN}(\mathbf{X}) \in \mathbb{R}^{md \times hw}$; local spatial positions	Standardizing	No	ICCV, 2009
Local response normalization [75]	$\Pi_{PN}(\mathbf{X}) \in \mathbb{R}^{mhw \times d}$; local channels	Scaling	No	NeurIPS, 2012
Batch normalization (BN) [8]	$\Pi_{BN}(\mathbf{X}) \in \mathbb{R}^{d \times mhw}$	Standardizing	Learnable $\gamma, \beta \in \mathbb{R}^d$	ICML, 2015
Mean-only BN [21]	$\Pi_{BN}(\mathbf{X}) \in \mathbb{R}^{d \times mhw}$	Centering	No	NeurIPS, 2016
Layer normalization (LN) [20]	$\Pi_{LN}(\mathbf{X}) \in \mathbb{R}^{m \times dhw}$	Standardizing	Learnable $\gamma, \beta \in \mathbb{R}^d$	Arxiv, 2016
Instance normalization (IN) [67]	$\Pi_{IN}(\mathbf{X}) \in \mathbb{R}^{md \times hw}$	Standardizing	Learnable $\gamma, \beta \in \mathbb{R}^d$	Arxiv, 2016
L^p -Norm BN [90]	$\Pi_{BN}(\mathbf{X}) \in \mathbb{R}^{d \times mhw}$	Standardizing with L^p -Norm divided	Learnable $\gamma, \beta \in \mathbb{R}^d$	Arxiv, 2016
Divisive normalization [76]	$\Pi_{LN}(\mathbf{X}) \in \mathbb{R}^{m \times dhw}$; local spatial positions and channels	Standardizing	Learnable $\gamma, \beta \in \mathbb{R}^d$	ICLR, 2017
Conditional IN [68]	$\Pi_{IN}(\mathbf{X}) \in \mathbb{R}^{md \times hw}$	Standardizing	Side information	ICLR, 2017
Dynamic LN [101]	$\Pi_{LN}(\mathbf{X}) \in \mathbb{R}^{m \times dhw}$	Standardizing	Generated $\gamma, \beta \in \mathbb{R}^d$	INTERSPEECH, 2017
Conditional BN [107]	$\Pi_{BN}(\mathbf{X}) \in \mathbb{R}^{d \times mhw}$	Standardizing	Side information	NeurIPS, 2017
Pixel normalization [72]	$\Pi_{PN}(\mathbf{X}) \in \mathbb{R}^{mhw \times d}$	Scaling	No	ICLR, 2018
Decorrelated BN [34]	$\Pi_{BN}(\mathbf{X}) \in \mathbb{R}^{d \times mhw}$	ZCA whitening	Learnable $\gamma, \beta \in \mathbb{R}^d$	CVPR, 2018
Group normalization (GN) [22]	$\Pi_{GN}(\mathbf{X}) \in \mathbb{R}^{mg \times shw}$	Standardizing	Learnable $\gamma, \beta \in \mathbb{R}^d$	ECCV, 2018

- Правда большая:

Adaptive IN [69]	$\Pi_{IN}(\mathbf{X}) \in \mathbb{R}^{md \times hw}$	Standardizing	Generated $\gamma, \beta \in \mathbb{R}^d$	ECCV, 2018
L^1 -Norm BN [91], [92]	$\Pi_{BN}(\mathbf{X}) \in \mathbb{R}^{d \times mhw}$	Standardizing with L^1 -Norm divided	Learnable $\gamma, \beta \in \mathbb{R}^d$	NeurIPS, 2018
Whitening and coloring BN [79]	$\Pi_{BN}(\mathbf{X}) \in \mathbb{R}^{d \times mhw}$	CD whitening	Color transformation	ICLR, 2019
Generalized BN [93]	$\Pi_{BN}(\mathbf{X}) \in \mathbb{R}^{d \times mhw}$	General standardizing	Learnable $\gamma, \beta \in \mathbb{R}^d$	AAAI, 2019
Iterative normalization [81]	$\Pi_{BN}(\mathbf{X}) \in \mathbb{R}^{d \times mhw}$	ZCA whitening by Newton's iteration	Learnable $\gamma, \beta \in \mathbb{R}^d$	CVPR, 2019
Instance-level meta normalization [103]	$\Pi_{LN}(\mathbf{X})/\Pi_{IN}(\mathbf{X})/\Pi_{GN}(\mathbf{X})$	Standardizing	Learnable & generated $\gamma, \beta \in \mathbb{R}^d$	CVPR, 2019
Spatially adaptive denormalization [109]	$\Pi_{BN}(\mathbf{X}) \in \mathbb{R}^{d \times mhw}$	Standardizing	Generated $\gamma, \beta \in \mathbb{R}^{d \times h \times w}$	CVPR, 2019
Position normalization (PN) [71]	$\Pi_{PN}(\mathbf{X}) \in \mathbb{R}^{mhw \times d}$	Standardizing	Learnable $\gamma, \beta \in \mathbb{R}^d$	NeurIPS, 2019
Root mean square LN [97]	$\Pi_{LN}(\mathbf{X}) \in \mathbb{R}^{m \times dhw}$	Scaling	Learnable $\gamma \in \mathbb{R}^d$	NeurIPS, 2019
Online normalization [99]	$\Pi_{LN}(\mathbf{X}) \in \mathbb{R}^{m \times dhw}$	Scaling	No	NeurIPS, 2019
Batch group normalization [73]	$\Pi_{BGN}(\mathbf{X}) \in \mathbb{R}^{gmg \times smshw}$	Standardizing	Learnable $\gamma, \beta \in \mathbb{R}^d$	ICLR, 2020
Instance enhancement BN [105]	$\Pi_{BN}(\mathbf{X}) \in \mathbb{R}^{d \times mhw}$	Standardizing	Learnable & generated $\gamma, \beta \in \mathbb{R}^d$	AAAI, 2020
PowerNorm [96]	$\Pi_{BN}(\mathbf{X}) \in \mathbb{R}^{d \times mhw}$	Scaling	Learnable $\gamma, \beta \in \mathbb{R}^d$	ICML, 2020
Local context normalization [74]	$\Pi_{GN}(\mathbf{X}) \in \mathbb{R}^{mg \times shw}$: local spatial positions and channels	Standardizing	Learnable $\gamma, \beta \in \mathbb{R}^d$	CVPR, 2020
Filter response normalization [100]	$\Pi_{IN}(\mathbf{X}) \in \mathbb{R}^{md \times hw}$	Scaling	Learnable $\gamma, \beta \in \mathbb{R}^d$	CVPR, 2020
Attentive normalization [106]	$\Pi_{BN}(\mathbf{X})/\Pi_{IN}(\mathbf{X})/\Pi_{LN}(\mathbf{X})/\Pi_{GN}(\mathbf{X})$	Standardizing	Generated $\gamma, \beta \in \mathbb{R}^d$	ECCV, 2020

СВЁРТОЧНЫЕ СЕТИ: ИДЕЯ

- Зрительная кора давно известна.
- Но до середины XX века изучать могли только на очень «высоком» уровне: болезни, травмы, агнозии.
- Хьюбел и Визель: исследования отдельных нейронов, на что они активируются.



ЗРИТЕЛЬНАЯ КОРА

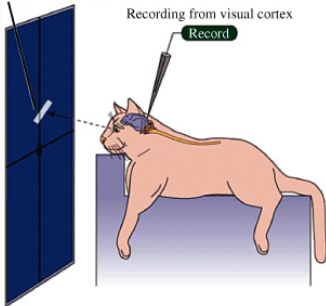
- Оказалось, что нейроны «ближних» к сетчатке уровней активируются на простые формы.
- Вообще оказалось, что зрительная кора делится на уровни: V1, V2, V3...
- Но от V3 и выше Хьюбел и Визель уже мудро не стали исследовать.

A Experimental setup

Light bar stimulus
projected on screen

Recording from visual cortex

Record



B

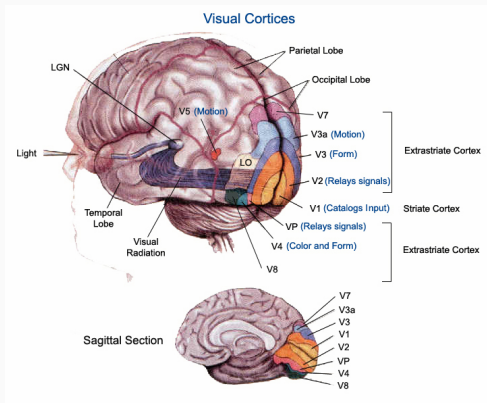
Stimulus
orientation

Stimulus
presented

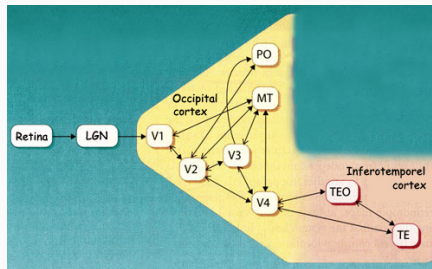


ЗРИТЕЛЬНАЯ КОРА

- Сетчатка → латеральное колленчатое тело (LGN, lateral geniculate nucleus) → зрительная кора.
- И там есть чёткая структура, организованная в уровни от V1 до V4-V6.

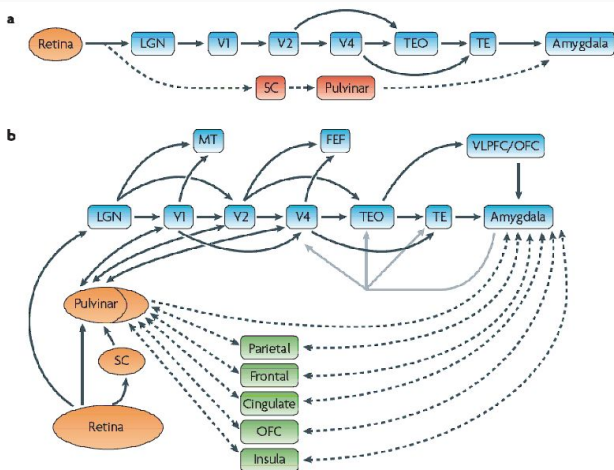


- Ну, вроде того. Мозг – сложная штука.



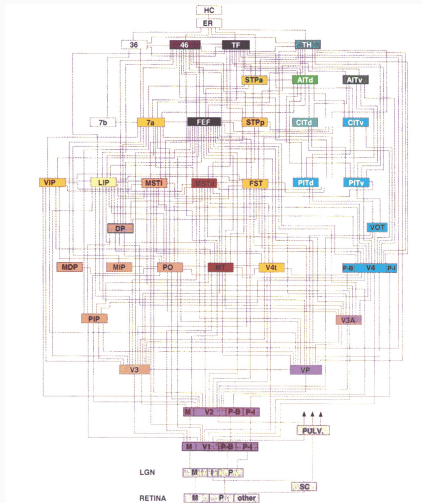
ЗРИТЕЛЬНАЯ КОРА

- Правда сложная.



ЗРИТЕЛЬНАЯ КОРА

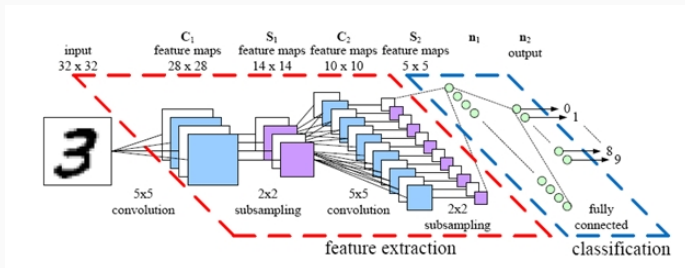
- Честно-честно.



- Но некие условные уровни там действительно есть:
 - V1 – локальные признаки, топографические карты;
 - V2 – больше локальных признаков, бинокулярное зрение;
 - V3 – цвет, текстура, первые результаты сегментации;
 - V4 – геометрические формы, силуэты; это самый важный уровень для *внимания*;
 - V5 – распознаёт движения объектов, сегментированных на V4;
 - V6 – обобщает данные со всей картинки, wide-field stimulation; например, обрабатывает наши собственные движения;
 - V7 (наверное) – распознаёт сложные объекты, например человеческие лица.
- What pathway: V2 – V4, распознавание форм и объектов.
- Where pathway: V2 – V5 – V6, распознавание движений, управление глазами и руками.

СВЁРТОЧНЫЕ НЕЙРОННЫЕ СЕТИ: ИДЕЯ

- Эти идеи нашли отражение в искусственных сетях.
- Свёртки впервые появились в *Neocognitron* (Fukushima, 1979; 1980).
- В современной форме – в группе ЛеКуна во второй половине 1980-х.
- Главная идея: хочется применить одну и ту же операцию к разным частям изображения.



СВЁРТОЧНЫЕ НЕЙРОННЫЕ СЕТИ: ИДЕЯ

- Разобьём изображение на окна:



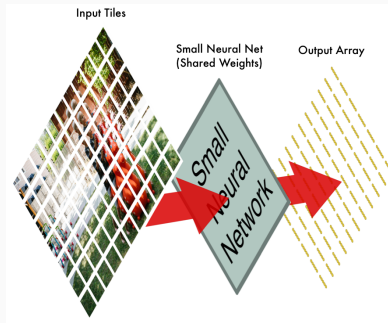
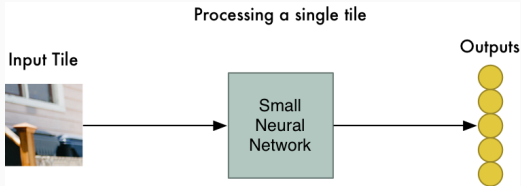
СВЁРТОЧНЫЕ НЕЙРОННЫЕ СЕТИ: ИДЕЯ

- Разобьём изображение на окна:



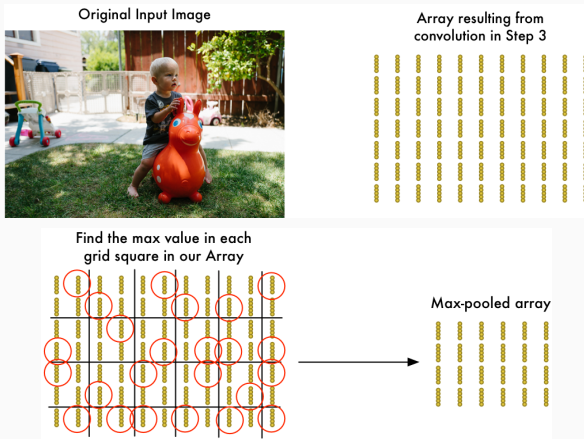
СВЁРТОЧНЫЕ НЕЙРОННЫЕ СЕТИ: ИДЕЯ

- Применим маленькую нейронную сеть к каждому окну:



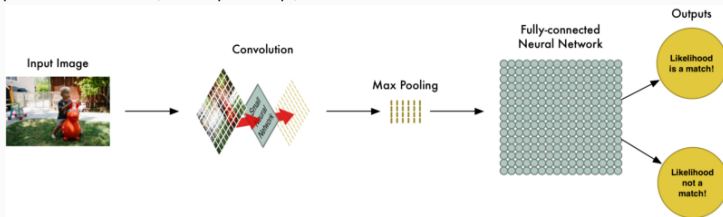
СВЁРТОЧНЫЕ НЕЙРОННЫЕ СЕТИ: ИДЕЯ

- Затем сожмём результат, сделав субдискретизацию (pooling; например, max-pooling) по маленьким окнам:

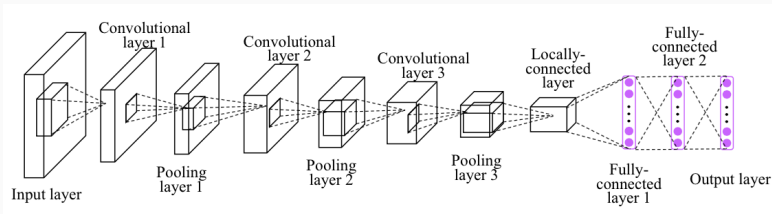


СВЁРТОЧНЫЕ НЕЙРОННЫЕ СЕТИ: ИДЕЯ

- А затем уже используем эти признаки, чтобы делать предсказания, например, полносвязной сетью:

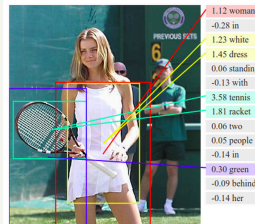
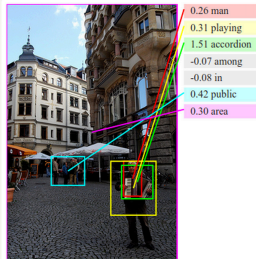
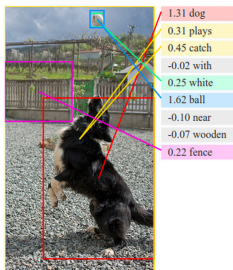


- Добавляя новые уровни, можно улучшить обобщающую способность, сделать признаки более общими:



СВЁРТОЧНЫЕ НЕЙРОННЫЕ СЕТИ: ИДЕЯ

- Более того, можно посмотреть, какие именно части картинки вызывают активации отдельных нейронов:

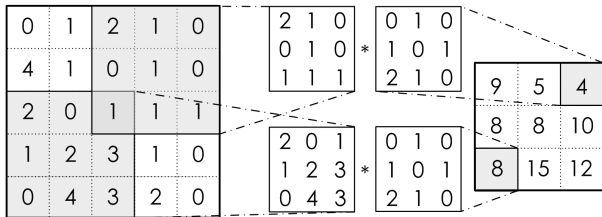


- Основная идея: мы применяем *одни и те же веса* по всей картинке.
- Это автоматически даёт устойчивость к переносам и т.п.
- И радикально сокращает число весов.
- Т.е. свёрточная структура – это такая радикальная форма регуляризации.

CNN ФОРМАЛЬНО

- Формально говоря, мы к каждому окну применяем одну и ту же маленькую матрицу.
- Если $\mathbf{x}^l$ – карта признаков на слое l , то двумерная свёртка размера $2d + 1$ с матрицей весов W размера $(2d + 1) \times (2d + 1)$ выглядит как

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



- Если $\mathbf{x}^l$ – карта признаков на слое l , то двумерная свёртка размера $2d + 1$ с матрицей весов W размера $(2d + 1) \times (2d + 1)$ выглядит как

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

- Затем обычно идёт нелинейность:

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

- В современных сетях – почти всегда ReLU.

- И затем max-pooling (иногда average-pooling, но обычно max):

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

0	1	2	1
4	1	0	1
2	0	1	1
1	2	3	1

(a)

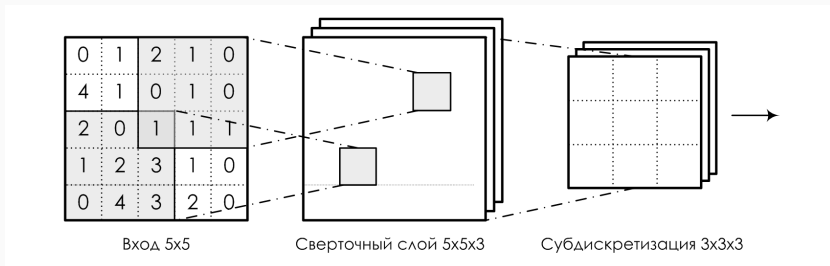
4	2	2
4	1	1
2	3	3

(б)

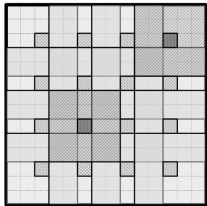
4	2
2	3

(в)

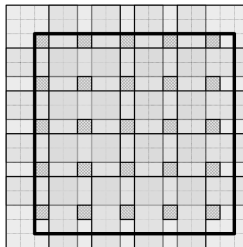
- Вот и получилась общая схема одного свёрточного слоя:



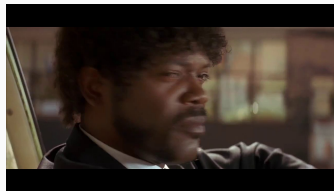
- Окнами можно покрывать по-разному; обычно всё-таки сохраняют исходный размер:



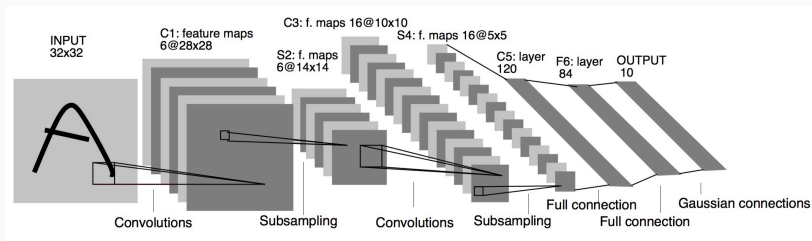
(а) окна 5x5 с шагом в 3 пикселя,
padding='VALID'



(б) окна 5x5 с шагом в 3 пикселя,
padding='SAME'



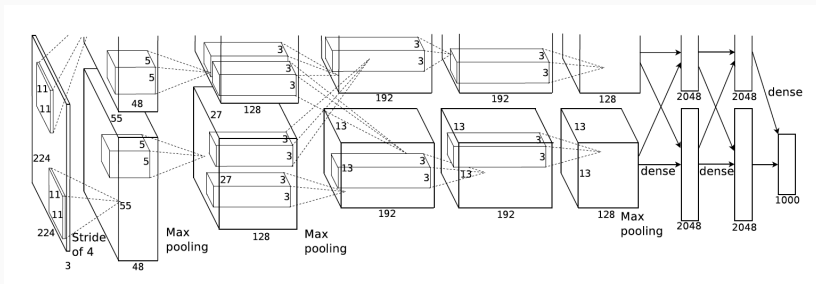
- Классическая архитектура *LeNet*:



- Современные CNN стали *очень* глубокими за счёт новых трюков – к ним мы скоро перейдём.

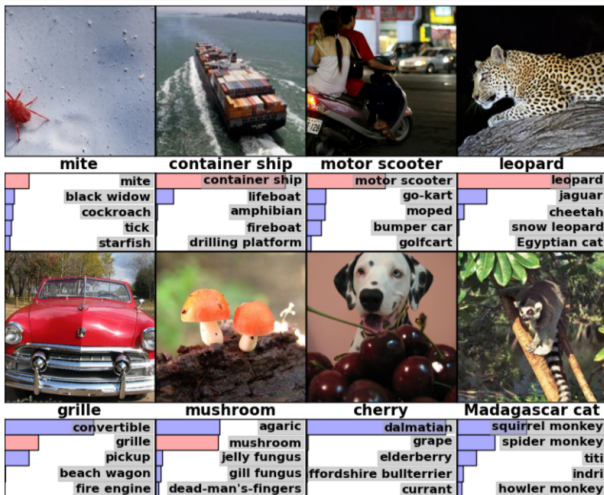
IMAGENET И ALEXNET

- ImageNet: > 15 миллионов картинок, ≈ 22000 категорий.
- Размечены руками (Amazon Mechanical Turk).
- Классическая сеть AlexNet (Krizhevsky, Sutskever, Hinton, 2012).



- С картинками можно делать data augmentation: изменять данные без изменения правильного ответа.
- (Krizhevsky, Sutskever, Hinton, 2012):
 - вырежем из 256×256 окна 224×224 (во время применения вырежем пять таких окон и усредним результаты);
 - отразим картинку по горизонтали;
 - поменяем интенсивности и освещённость (добавляя кратные главных компонент RGB-значений);
 - это увеличивает датасет в 2048 раз совершенно бесплатно. :)
- В AlexNet без data augmentation был бы оверфиттинг, это реально помогает.

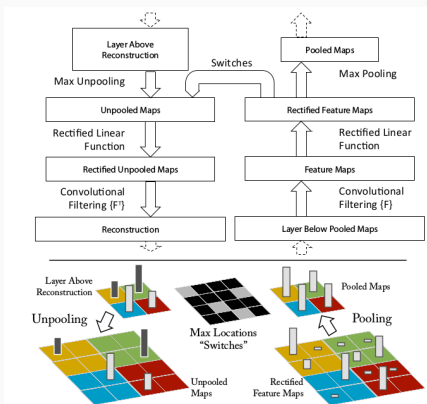
ALEXNET HA IMAGENET



ЧТО ВИДЯТ СВЁРТОЧНЫЕ СЕТИ

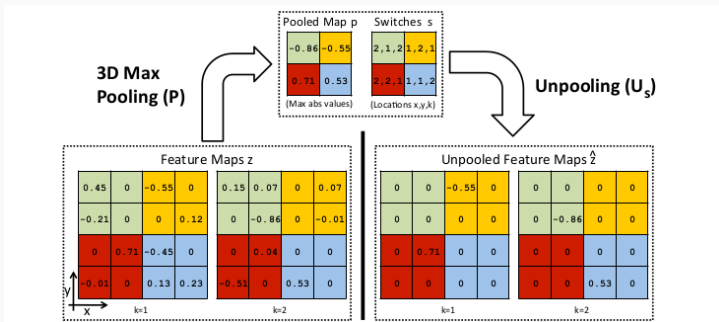
ДЕКОНВОЛЮЦИЯ

- (Zeiler et al., 2011; Zeiler, Fergus, 2014): деконволюция как обратная свёртке операция.
- Чтобы обратить свёртку, достаточно транспонировать.



ДЕКОНВОЛЮЦИЯ

- Интересный вопрос – как обратить субдискретизацию.
- Она, конечно, необратима, но можно запомнить положение максимумов и обратить примерно.

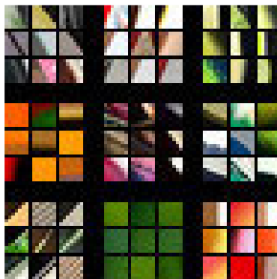


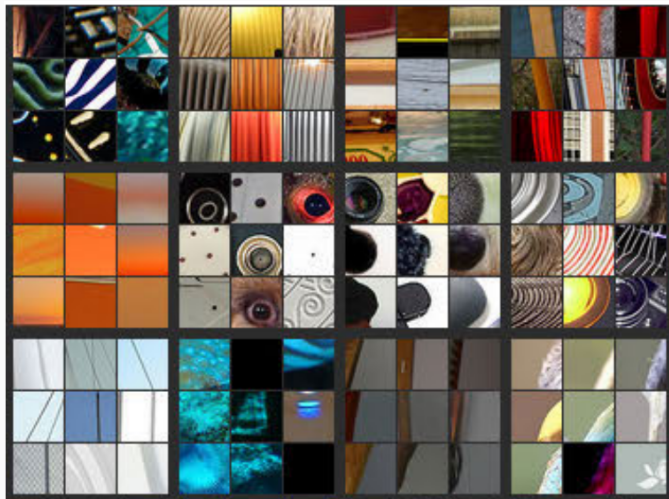
УРОВЕНЬ 1

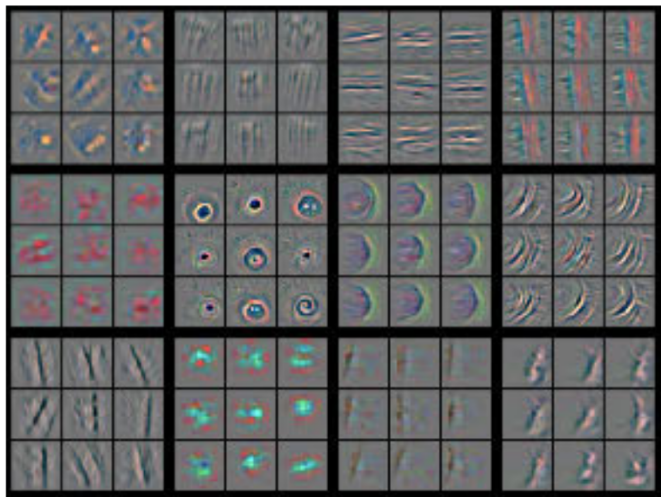
(Zeiler, Fergus, 2014): визуализируем признаки через деконволюцию. Это сеть, обученная на ImageNet.



Layer 1

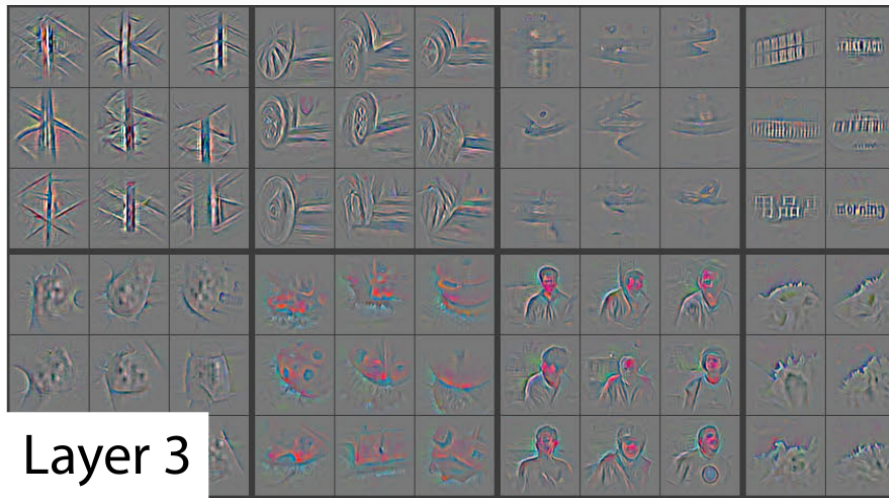


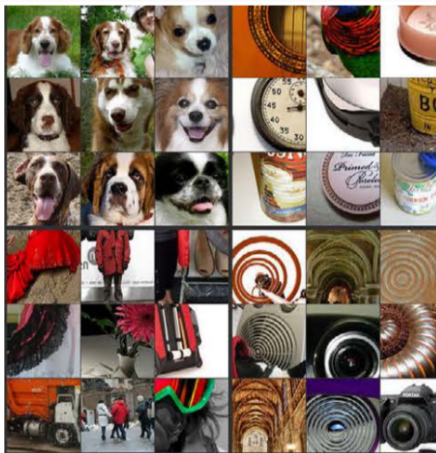
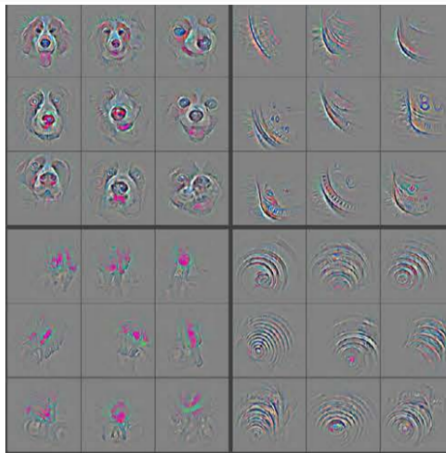




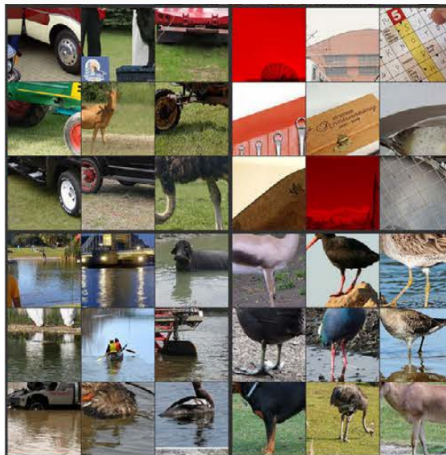
(ZEILER, FERGUS, 2014): УРОВЕНЬ 3



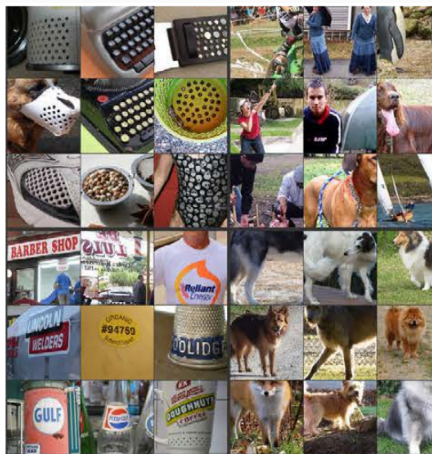
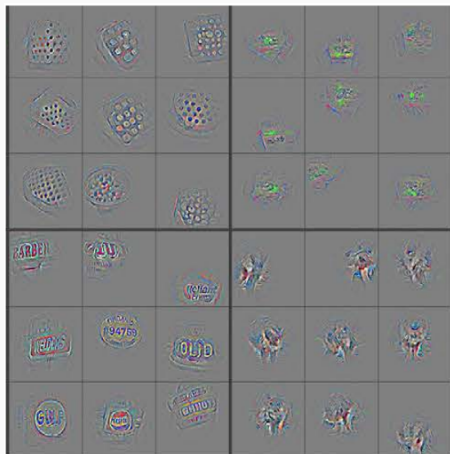


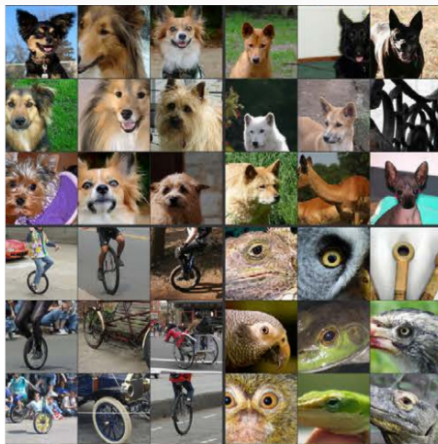
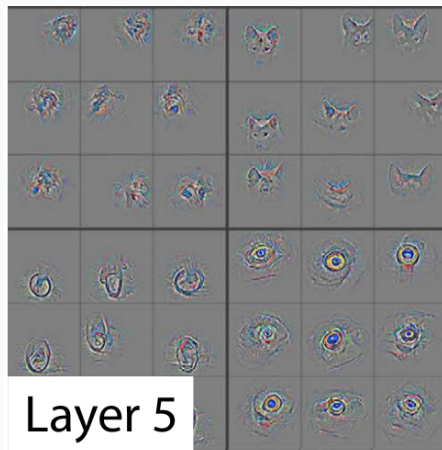


(ZEILER, FERGUS, 2014): УРОВЕНЬ 4



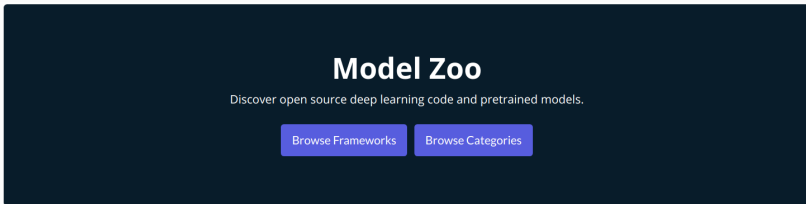
(ZEILER, FERGUS, 2014): УРОВЕНЬ 5



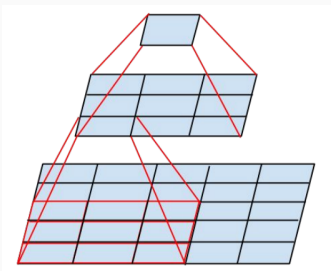


СОВРЕМЕННЫЕ СВЁРТОЧНЫЕ АРХИТЕКТУРЫ

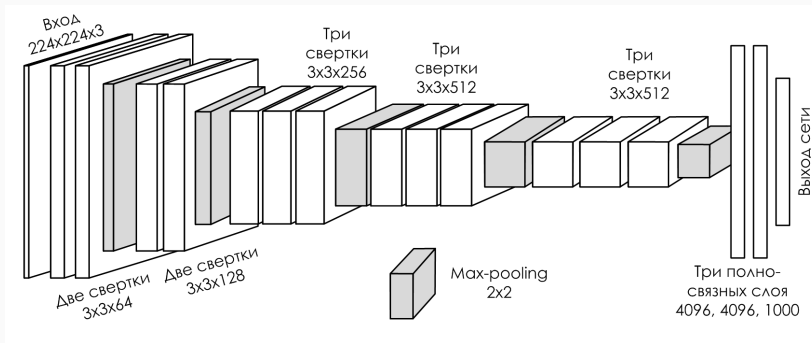
- Разных моделей, использующих CNN, очень много, и всё время появляются новые.
- Всё можно скачать, часто даже уже веса, предобученные на стандартных датасетах, есть.
- Но обычно они основаны на нескольких относительно стандартных идеях. Их-то мы и рассмотрим.



- VGG (Oxford Visual Geometry Group): давайте представлять большие свёртки как комбинации свёрток 3×3 .
- Это уменьшает число весов и делает сеть глубже.



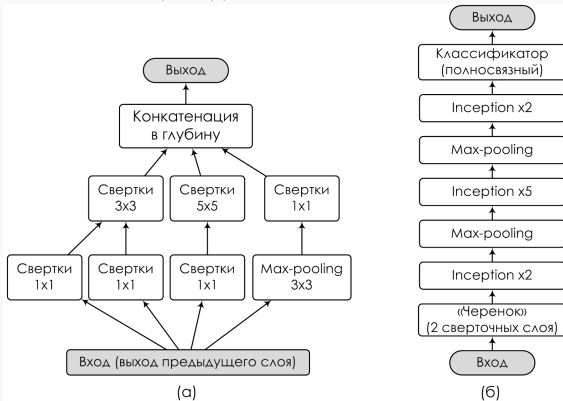
- Сеть VGG, использованная в ImageNet Large Scale Visual Recognition Competition (ILSVRC-2014).



- А сейчас NVIDIA специально оптимизирует GPU для свёрток 3×3 .

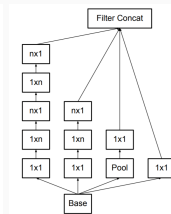
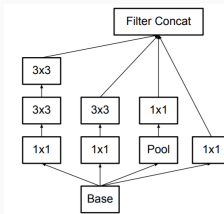
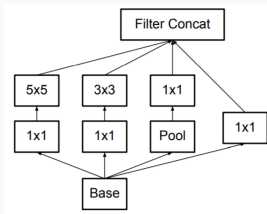
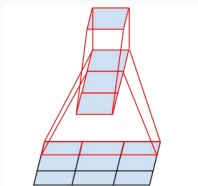
INCEPTION

- *Inception*, разработанная командой GoogLeNet: давайте используем идею “сеть в сети” (network in network), чтобы сократить веса ещё больше!
- И дополнительные классификаторы (просто лишние слагаемые в целевую функцию).

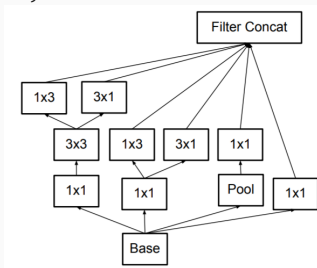


INCEPTION

- Во второй версии (Szegedy et al., 2015) свёртки $n \times n$ заменили на комбинации свёрток $n \times 1$ и $1 \times n$:



- А банки фильтров в Inception v2 сделали широкими, а не глубокими:



- В той же статье – Inception v3; то же самое, но ещё RMSProp, факторизованные свёртки 7×7 , batchnorm во вспомогательных классификаторах и...

- Label smoothing:
 - в обычном классификаторе мы обучаем результаты softmax с целевой переменной $q(k) = [k = y]$, т.е. аргументы softmax хотят расти неограниченно;
 - модели становятся слишком уверены в себе, оверфиттинг;
 - решение: давайте вместо этого оптимизировать

$$q'(k) = (1 - \epsilon)[k = y] + \epsilon u(k)$$

для какого-то априорного $u(k)$, например $u(k) = \frac{1}{K}$.

Network	Crops Evaluated	Top-5 Error	Top-1 Error
GoogLeNet [20]	10	-	9.15%
GoogLeNet [20]	144	-	7.89%
VGG [18]	-	24.4%	6.8%
BN-Inception [7]	144	22%	5.82%
PReLU [6]	10	24.27%	7.38%
PReLU [6]	-	21.59%	5.71%
Inception-v3	12	19.47%	4.48%
Inception-v3	144	18.77%	4.2%

- Остаточное обучение (residual learning): давайте обучим разности между очередным уровнем и предыдущим.
- Тогда градиенты смогут беспрепятственно проходить куда надо.
- Функция, реализуемая остаточным блоком, выглядит как

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

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

- И градиент будет проходить через этот блок беспрепятственно и не будет затухать:

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

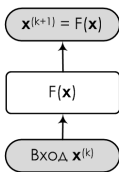
- Это привело к *очень* глубоким сетям.
- Другой аналогичный подход – *highway networks* (магистральные сети) Шмидхубера.
- Тоже представляем $\mathbf{y}^{(k)}$, выход слоя k , как линейную комбинацию входа этого слоя $\mathbf{x}^{(k)}$ и результата $F(\mathbf{x}^{(k)})$, веса которой управляются другими преобразованиями:

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

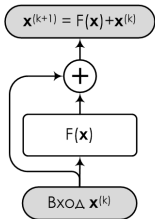
где C — это гейт переноса (carry gate), а T — гейт преобразования (transform gate); обычно комбинацию делают выпуклой, $C = 1 - T$.

- Практика показывает, что чем «прямее», тем лучше.

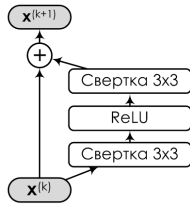
RESNET: ВАРИАНТЫ



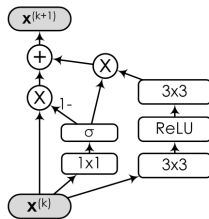
(a)



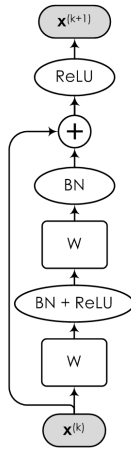
(б)



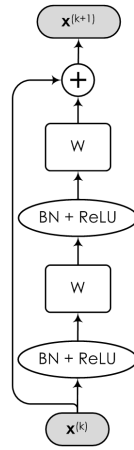
(B)



(r)



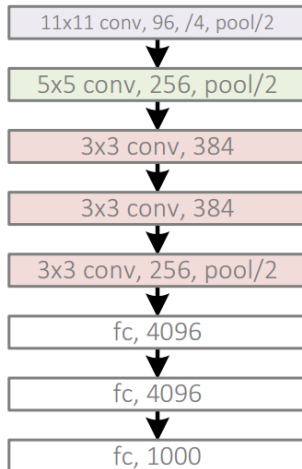
(Δ)



(e)

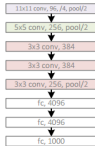
REVOLUTION OF DEPTH (KAIMING HE)

AlexNet, 8 layers
(ILSVRC 2012)

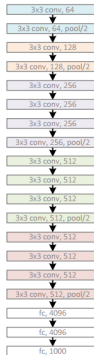


REVOLUTION OF DEPTH (KAIMING HE)

AlexNet, 8 layers
(ILSVRC 2012)



VGG, 19 layers
(ILSVRC 2014)



GoogleNet, 22 layers
(ILSVRC 2014)



ResNet led to the revolution of depth

AlexNet, 8 layers
(ILSVRC 2012)



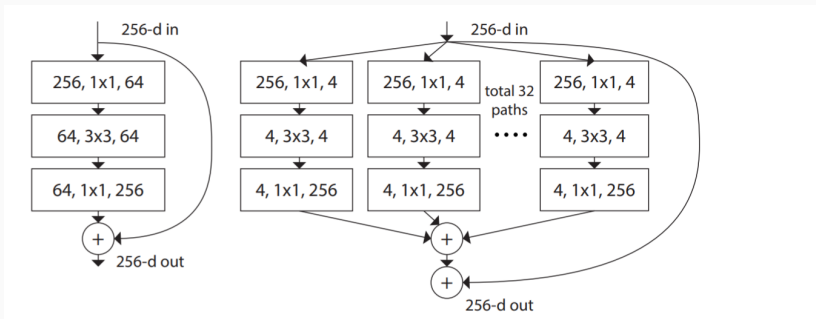
VGG, 19 layers
(ILSVRC 2014)



ResNet, 152 layers
(ILSVRC 2015)

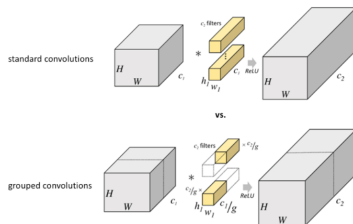


- ResNeXt (Xie et al., 2016): давайте заменим ResNet-блоки на "split-transform-merge" блоки, похожие на Inception.

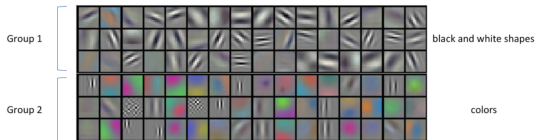


- Вход разбивается на блоки по каналам, к каждому блоку свои свёртки.

- Идея похожа на group convolutions, которые были ещё в AlexNet для параллелизации:

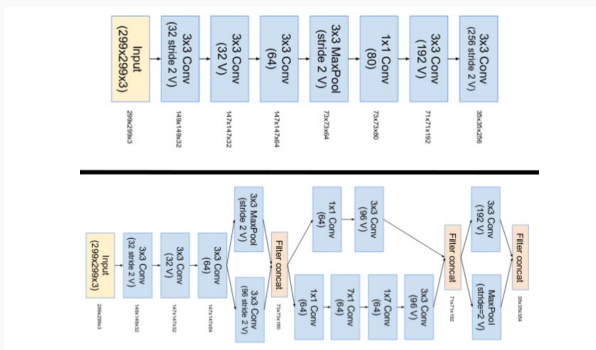


- И они дают специализацию в результатах:



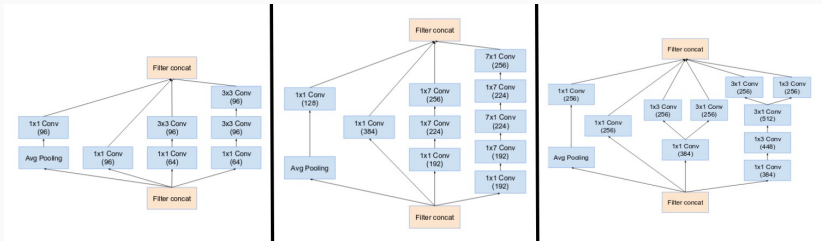
INCEPTION V4 И INCEPTION RESNET

- Ещё одна классическая статья (Szegedy et al., 2016): вводятся Inception v4 и Inception ResNet.
- Inception v4 – идея в том, чтобы всё стандартизировать, упростить модули. Во-первых, «черенок»:



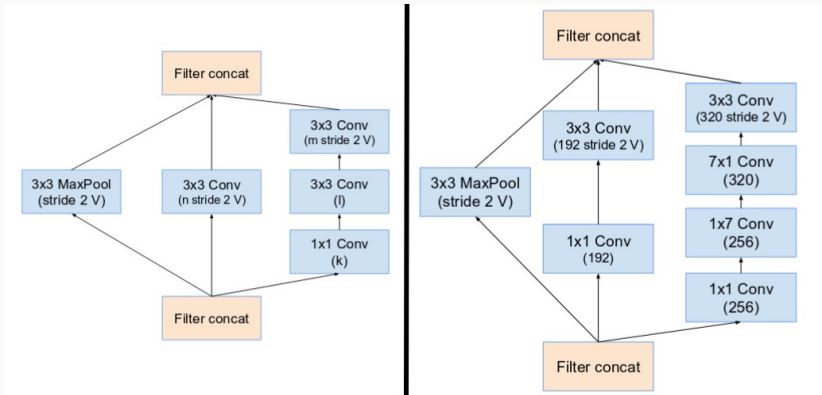
INCEPTION V4 и INCEPTION RESNET

- Во-вторых, теперь три базовых блока A, B, C:



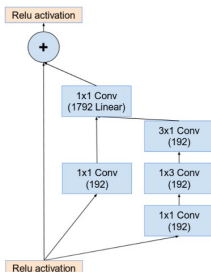
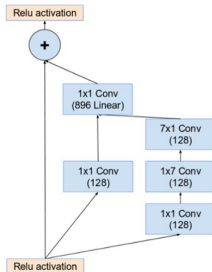
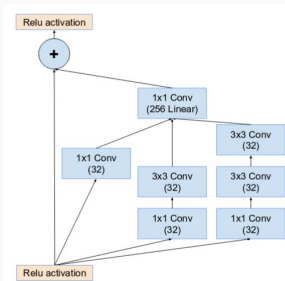
INCEPTION V4 И INCEPTION RESNET

- И специальные reduction blocks для сокращения размерности сетки:



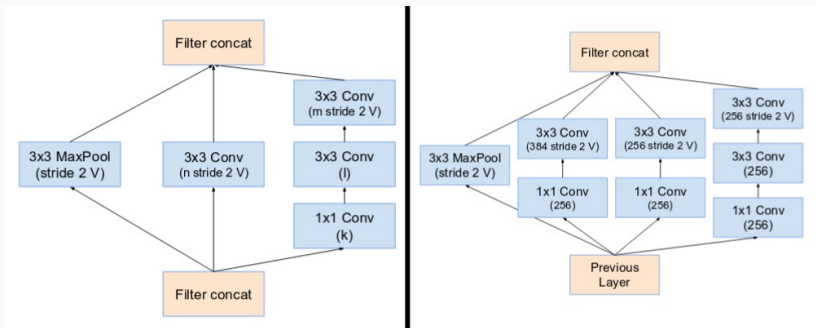
INCEPTION V4 и INCEPTION RESNET

- В Inception ResNet к тем же блокам добавляются остаточные СВЯЗИ:



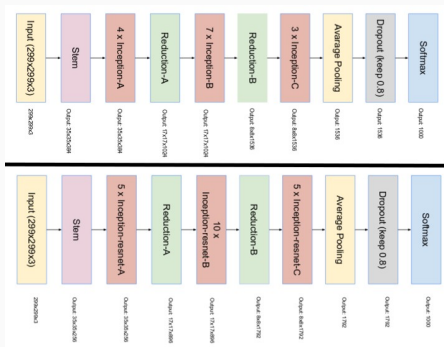
INCEPTION V4 И INCEPTION RESNET

- Субдискретизации теперь нет, но по-прежнему есть reduction blocks:



INCEPTION V4 И INCEPTION RESNET

- В итоге архитектура даже упростилась; сверху Inception v4, снизу Inception ResNet:



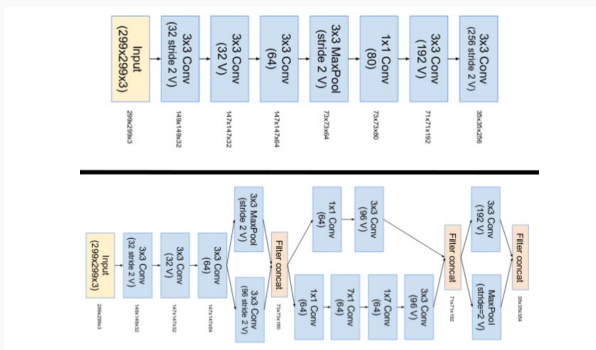
- И работает неплохо:

Network	Crops	Top-1 Error	Top-5 Error
ResNet-151 [5]	10	21.4%	5.7%
Inception-v3 [15]	12	19.8%	4.6%
Inception-ResNet-v1	12	19.8%	4.6%
Inception-v4	12	18.7%	4.2%
Inception-ResNet-v2	12	18.7%	4.1%

Network	Crops	Top-1 Error	Top-5 Error
ResNet-151 [5]	dense	19.4%	4.5%
Inception-v3 [15]	144	18.9%	4.3%
Inception-ResNet-v1	144	18.8%	4.3%
Inception-v4	144	17.7%	3.8%
Inception-ResNet-v2	144	17.8%	3.7%

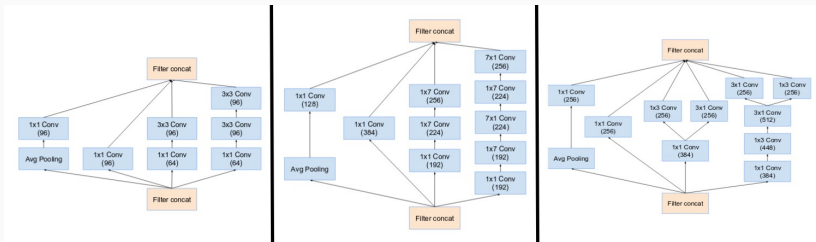
INCEPTION v4 и INCEPTION ResNet

- Ещё одна классическая статья (Szegedy et al., 2016): вводятся Inception v4 и Inception ResNet.
- Inception v4 – идея в том, чтобы всё стандартизировать, упростить модули. Во-первых, «черенок»:



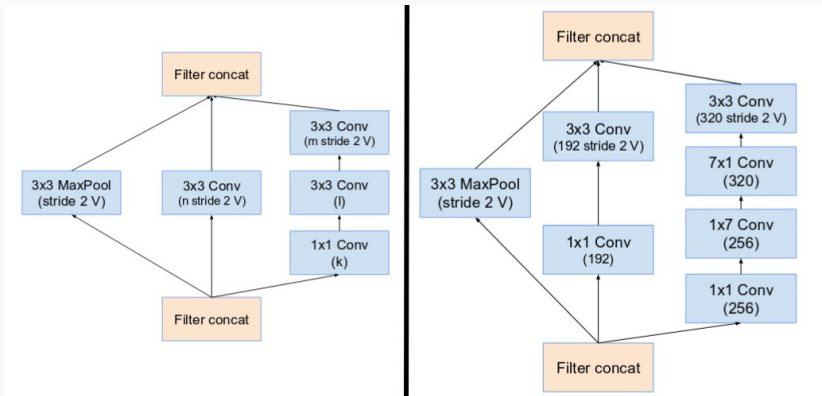
INCEPTION V4 и INCEPTION RESNET

- Во-вторых, теперь три базовых блока A, B, C:



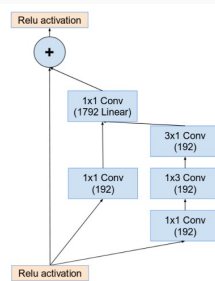
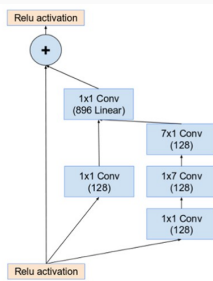
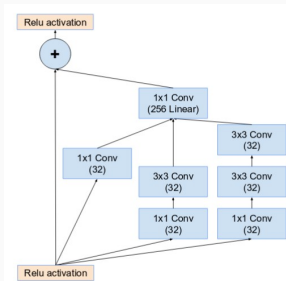
INCEPTION V4 И INCEPTION RESNET

- И специальные reduction blocks для сокращения размерности сетки:



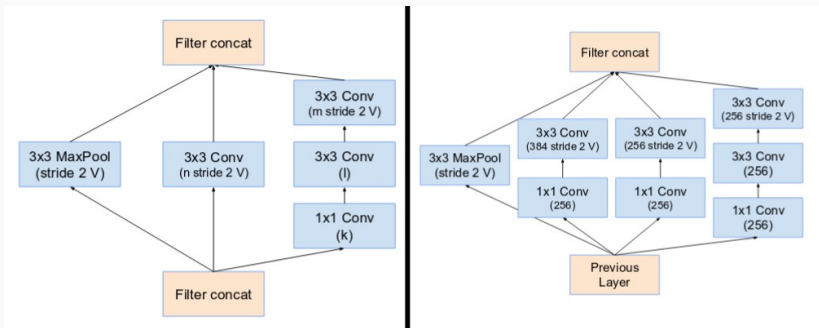
INCEPTION V4 И INCEPTION RESNET

- В Inception ResNet к тем же блокам добавляются остаточные СВЯЗИ:



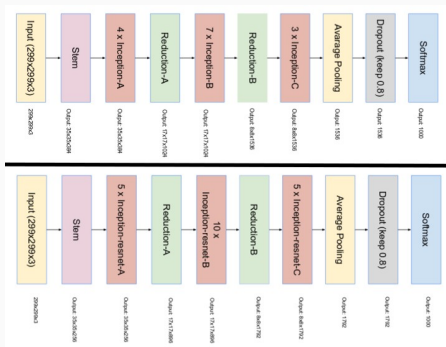
INCEPTION V4 И INCEPTION RESNET

- Субдискретизации теперь нет, но по-прежнему есть reduction blocks:



INCEPTION V4 И INCEPTION RESNET

- В итоге архитектура даже упростилась; сверху Inception v4, снизу Inception ResNet:

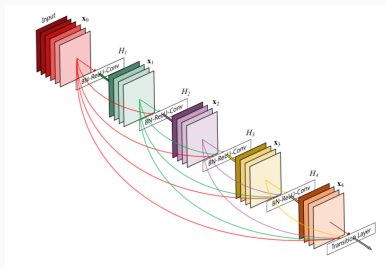


- И работает неплохо:

Network	Crops	Top-1 Error	Top-5 Error
ResNet-151 [5]	10	21.4%	5.7%
Inception-v3 [15]	12	19.8%	4.6%
Inception-ResNet-v1	12	19.8%	4.6%
Inception-v4	12	18.7%	4.2%
Inception-ResNet-v2	12	18.7%	4.1%

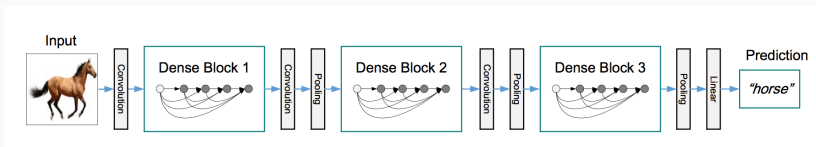
Network	Crops	Top-1 Error	Top-5 Error
ResNet-151 [5]	dense	19.4%	4.5%
Inception-v3 [15]	144	18.9%	4.3%
Inception-ResNet-v1	144	18.8%	4.3%
Inception-v4	144	17.7%	3.8%
Inception-ResNet-v2	144	17.8%	3.7%

- DenseNet (Huang et al., 2017, best paper at CVPR 2017) – полезно включать не только последовательные соединения между слоями:

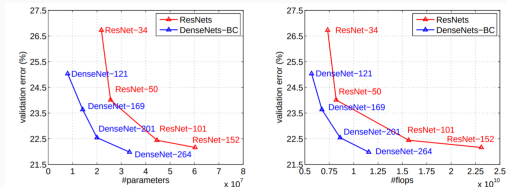


- Каждый с каждым! Выглядит странно – сколько ж весов нужно?..

- Но на самом деле нужно не так уж много; такие соединения позволяют мало фич использовать.
- Например, $k = 12$ фильтров на каждый уровень – это значит, что на уровне m будет $k_0 + k(m - 1)$ фич доступно.
- Такие конструкции объединяют в блоки:



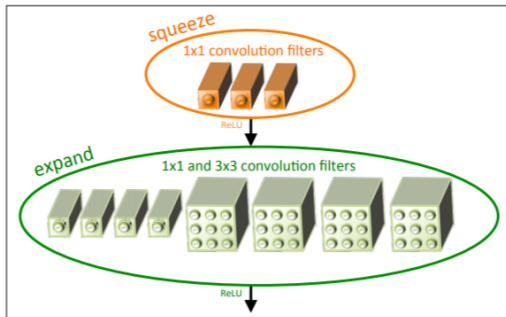
- В DenseNet меньше весов, обучается быстрее и лучше:



Method	Depth	Params	C10	C10+	C100	C100+	SVHN
Network in Network [22]	-	-	10.41	8.81	35.68	-	2.35
All-CNN [32]	-	-	9.08	7.25	-	33.71	-
Deeply Supervised Net [20]	-	-	9.69	7.97	-	34.57	1.92
Highway Network [34]	-	-	-	7.72	-	32.39	-
FractalNet [17]	21	38.6M	10.18	5.22	35.34	23.30	2.01
with Dropout/Drop-path	21	38.6M	7.33	4.60	28.20	23.73	1.87
ResNet [11]	110	1.7M	-	6.61	-	-	-
ResNet (reported by [13])	110	1.7M	13.63	6.41	44.74	27.22	2.01
ResNet with Stochastic Depth [13]	110	1.7M	11.66	5.23	37.80	24.58	1.75
	1202	10.2M	-	4.91	-	-	-
Wide ResNet [42]	16	11.0M	-	4.81	-	22.07	-
	28	36.5M	-	4.17	-	20.50	-
with Dropout	16	2.7M	-	-	-	-	1.64
ResNet (pre-activation) [12]	164	1.7M	11.26*	5.46	35.58*	24.33	-
	1001	10.2M	10.56*	4.62	33.47*	22.71	-
DenseNet ($k = 12$)	40	1.0M	7.00	5.24	27.55	24.42	1.79
DenseNet ($k = 12$)	100	7.0M	5.77	4.10	23.79	20.20	1.67
DenseNet ($k = 24$)	100	27.2M	5.83	3.74	23.42	19.25	1.59
DenseNet-BC ($k = 12$)	100	0.8M	5.92	4.51	24.15	22.27	1.76
DenseNet-BC ($k = 24$)	250	15.3M	5.19	3.62	19.64	17.60	1.74
DenseNet-BC ($k = 40$)	190	25.6M	-	3.46	-	17.18	-

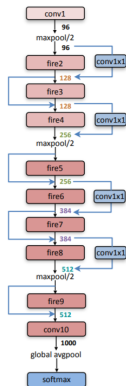
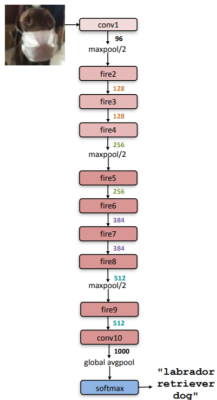
- SqueezeNet (Iandola et al., 2017) – как уменьшить число параметров:
 - заменять 3×3 фильтры на 1×1 ;
 - уменьшать число входов для 3×3 ;
 - делать downsampling как можно позже, чтобы карты активаций были побольше.

- Fire module:
 - squeeze convolutional layer (только 1×1);
 - expand layer (1×1 и 3×3).

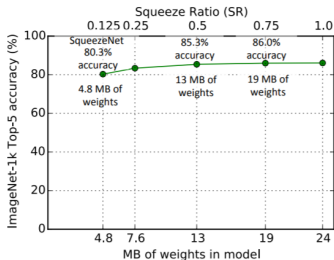


SQUEEZE NET

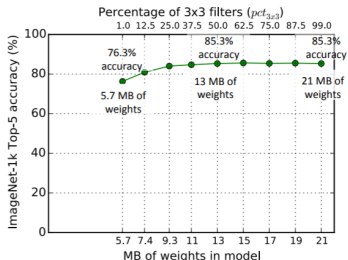
- И общая архитектура SqueezeNet:



- Получается в 50 раз меньше параметров, чем у AlexNet, но:



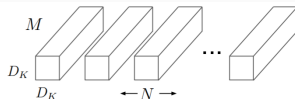
(a) Exploring the impact of the squeeze ratio (SR) on model size and accuracy.



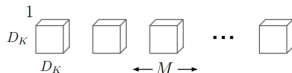
(b) Exploring the impact of the ratio of 3x3 filters in expand layers ($pct_{3 \times 3}$) on model size and accuracy.

Architecture	Top-1 Accuracy	Top-5 Accuracy	Model Size
Vanilla SqueezeNet	57.5%	80.3%	4.8MB
SqueezeNet + Simple Bypass	60.4%	82.5%	4.8MB
SqueezeNet + Complex Bypass	58.8%	82.0%	7.7MB

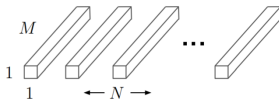
- MobileNet (Howard et al., 2017): сети для мобильных устройств.
- Depthwise separable convolutions: давайте разложим свёртку на depthwise (один фильтр к каждому каналу) и 1×1 .



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

- Тогда структура слоя будет чуть сложнее (но весов меньше), и общая структура не слишком глубокая:

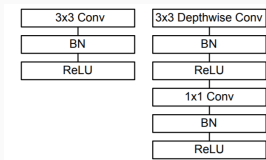


Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
5×	Conv dw / s1	$3 \times 3 \times 512$ dw
	Conv / s1	$1 \times 1 \times 512 \times 512$
		$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

- И получается, что можно сэкономить очень много параметров ценой небольшого ухудшения качества:

Table 8. MobileNet Comparison to Popular Models

Model	ImageNet	Million	Million
	Accuracy	Mult-Adds	Parameters
1.0 MobileNet-224	70.6%	569	4.2
GoogleNet	69.8%	1550	6.8
VGG 16	71.5%	15300	138

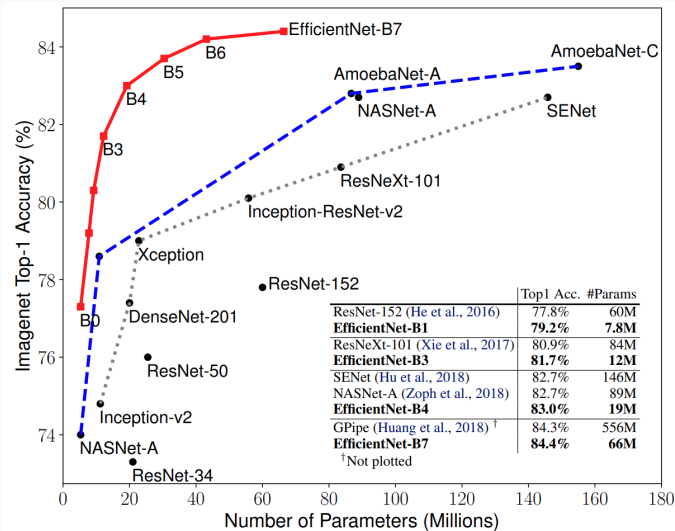
Table 9. Smaller MobileNet Comparison to Popular Models

Model	ImageNet	Million	Million
	Accuracy	Mult-Adds	Parameters
0.50 MobileNet-160	60.2%	76	1.32
Squeezenet	57.5%	1700	1.25
AlexNet	57.2%	720	60

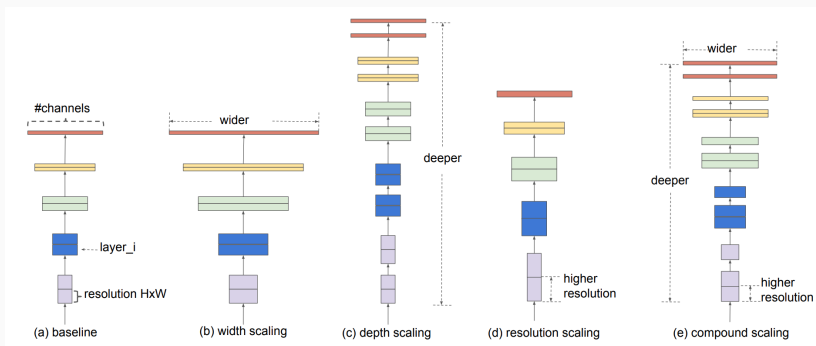
Table 10. MobileNet for Stanford Dogs

Model	Top-1	Million	Million
	Accuracy	Mult-Adds	Parameters
Inception V3 [18]	84%	5000	23.2
1.0 MobileNet-224	83.3%	569	3.3
0.75 MobileNet-224	81.9%	325	1.9
1.0 MobileNet-192	81.9%	418	3.3
0.75 MobileNet-192	80.5%	239	1.9

- EfficientNet (Tan, Le, 2019): а вот и у neural architecture search дошли руки до свёрточных архитектур



- Разные виды масштабирования для моделей:



СПАСИБО!

Спасибо за внимание!

