Введение в CNN

Гончаров Павел **kaliostrogoblin3@gmail.com**
Нестереня Игорь **nesterione@gmail.com**

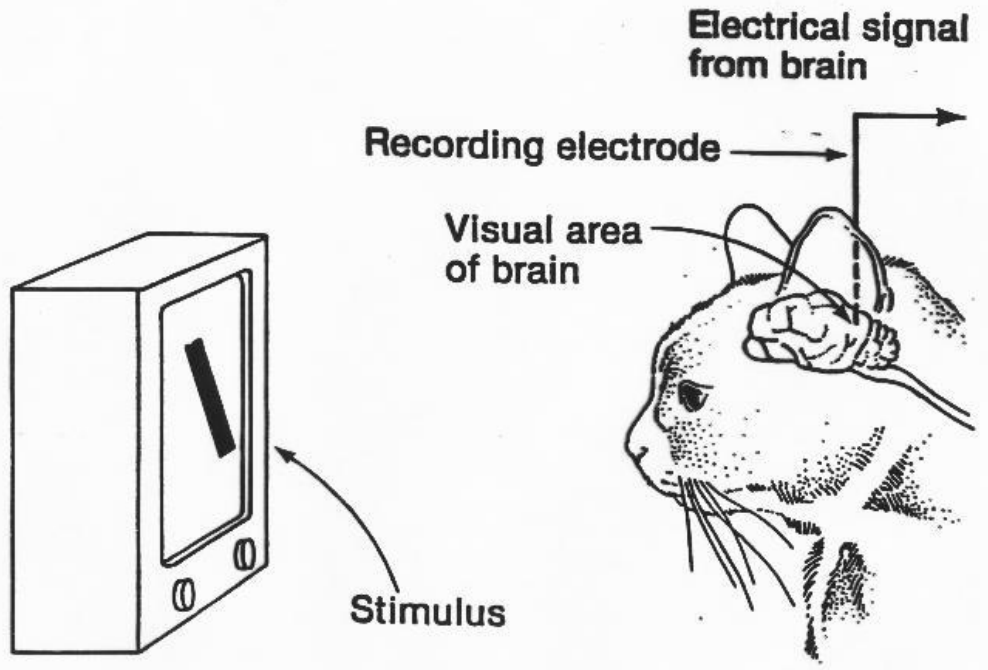
Мотивация перехода к CNN

- В полносвязной сети каждый нейрон одного слоя соединён с каждым нейроном следующего слоя. Каждая связь означает свой собственный вес. Затратно для обучения и для хранения всех значений. В свёрточной сети каждый нейрон связан с несколькими рядом расположенными нейронами на предыдущем слое. Меньшее количество связей делает свёрточные сети более дешёвыми по отношению к полносвязным.
- Преобразование изображения в вектор означает потерю информацию о расположении пикселей. При обработке визуальной информации обычно имеют значения некоторые признаки, как правило это набор локально расположенных пикселей.
- Большое количество связей в полносвязной сети ведёт к переобучению.

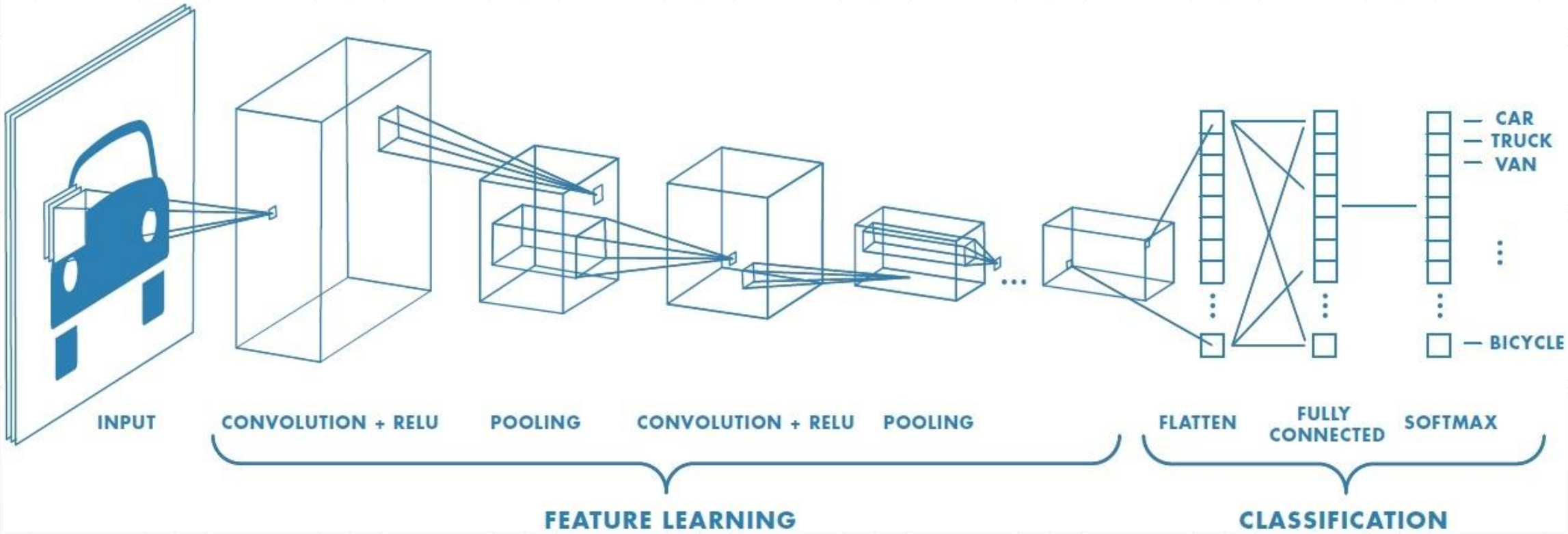
Идея свёрточный нейронных сетей

CNN (Convolutional Neural Network) – по своей идеи является прототипом зрительной коры головного мозга.

Зрительная кора имеет небольшие участки клеток, которые чувствительны к конкретным областям поля зрения. Эту идею детально рассмотрели с помощью потрясающего эксперимента Хьюбел и Визель в 1962 году, в котором показали, что отдельные мозговые нервные клетки реагировали (или активировались) только при визуальном восприятии границ определенной ориентации. Например, некоторые нейроны активировались, когда воспринимали вертикальные границы, а некоторые — горизонтальные или диагональные.

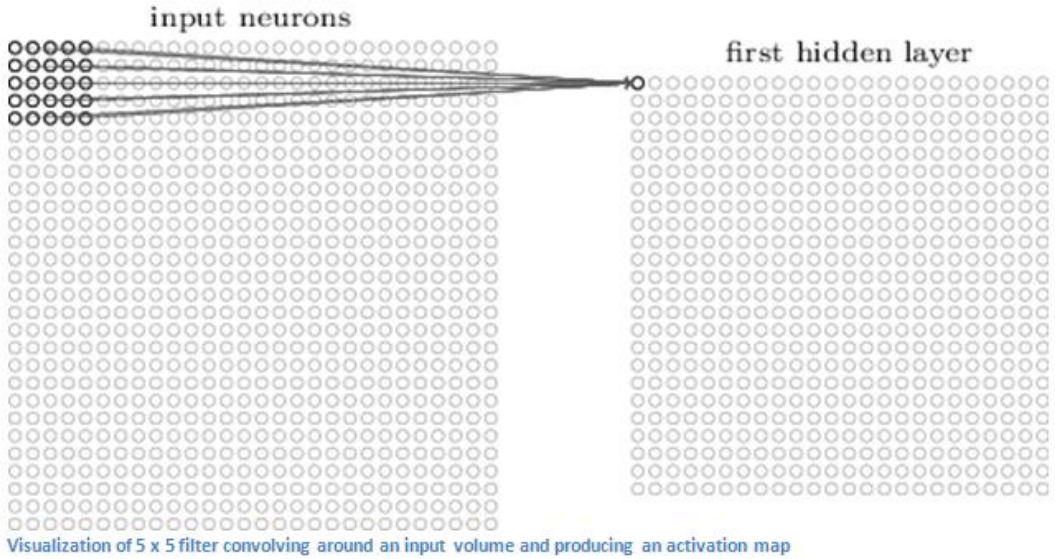


Общий вид свёрточной сети

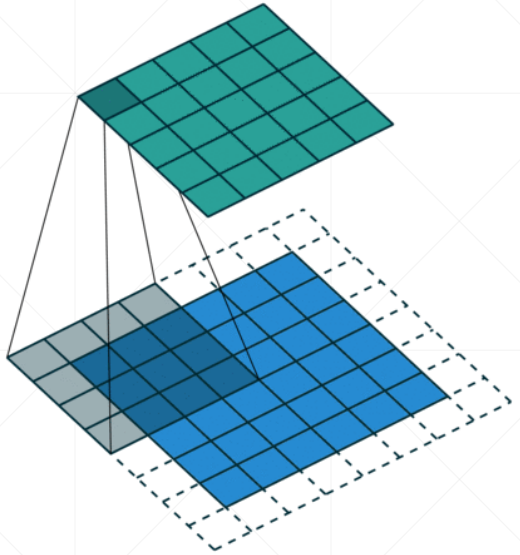


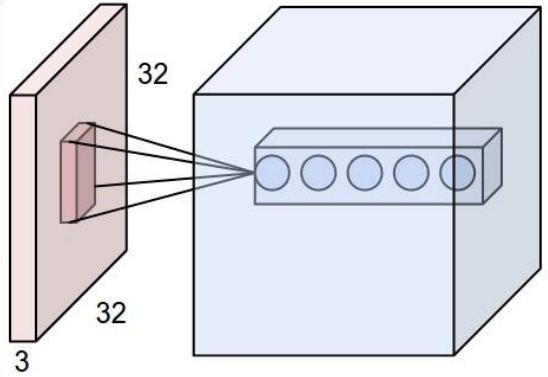
Input -> Conv -> ReLU -> Conv -> ReLU -> Pool -> ReLU -> Conv -> ReLU -> Pool ->Fully Connected

Как работает фильтр (свёртка)



Visualization of 5 x 5 filter convolving around an input volume and producing an activation map





После прохождения фильтра по всем позициям получается матрица 28 x 28 x 1, которую называют функцией активации или картой признаков.

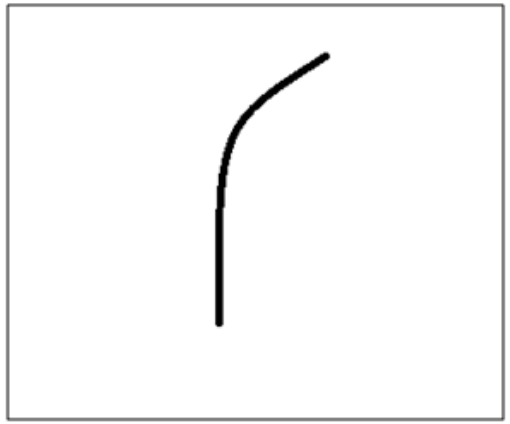
Допустим, теперь мы используем два 5 x 5 x 3 фильтра вместо одного. Тогда выходным значением будет 28 x 28 x 2.

http://cs231n.github.io/convolutional-networks/

Сверточный слой и рецептивное поле

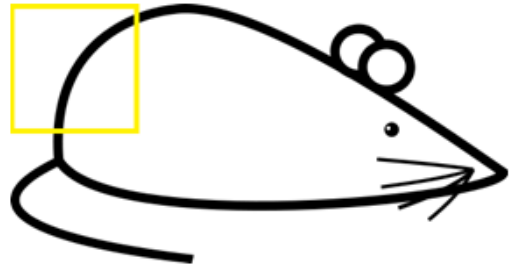
0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter



Visualization of a curve detector filter

Original image



Visualization of the filter on the image

Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 (A large number!)

Visualization of the filter on the image

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

Pixel representation of receptive field

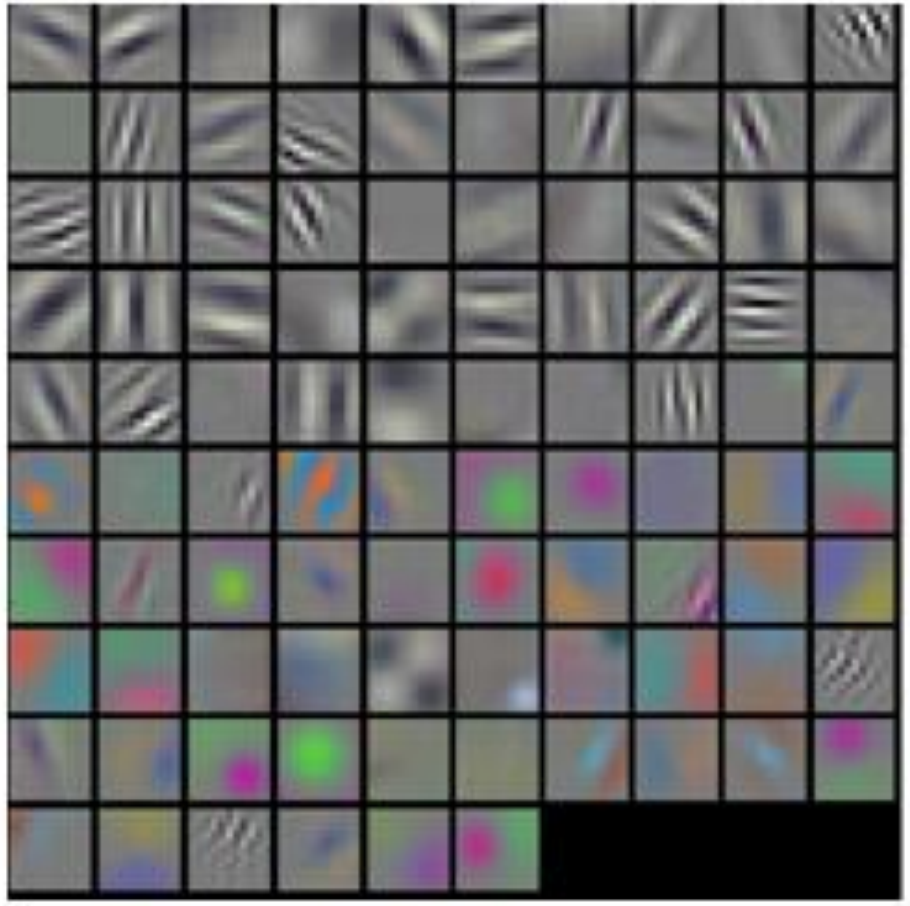
*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = 0

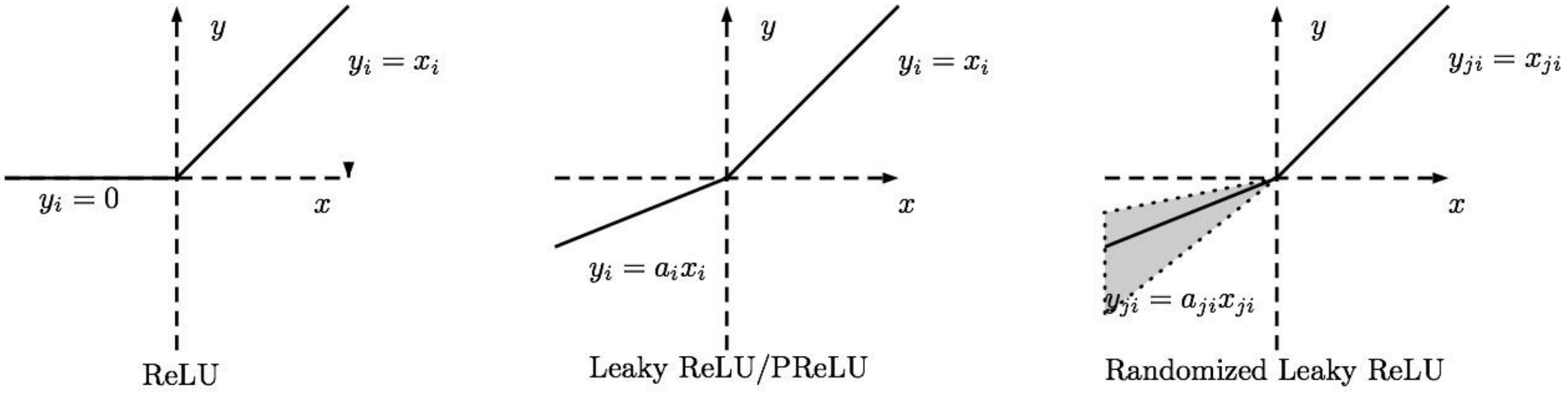
Фильтры первого слоя



Visualizations of filters

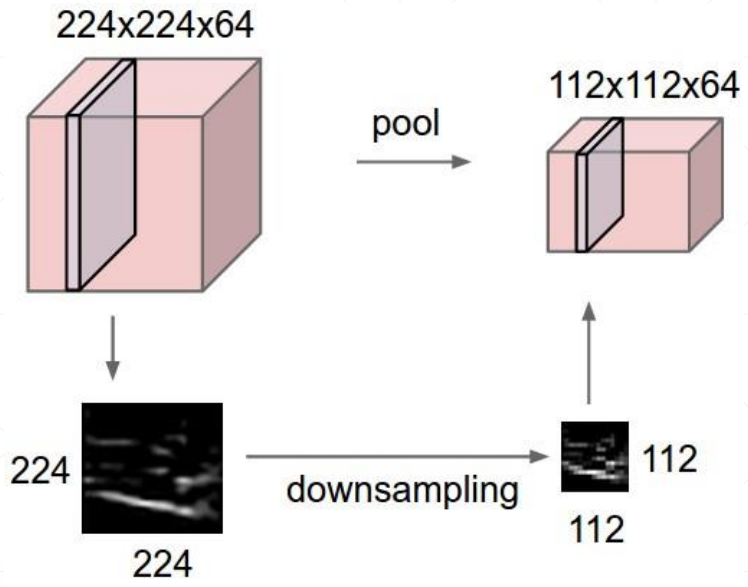
- Как можно себе представить, чтобы предположить какой тип объекта изображён на картинке, нам нужна сеть, способная распознавать свойства более высокого уровня, как например руки, лапы или уши.
- Когда вы двигаетесь вглубь сети, фильтры работают со все большим полем восприятия, а значит, они в состоянии обрабатывать информацию с большей площади первоначального изображения (простыми словами, они лучше приспособляются к обработке большей области пиксельного пространства)

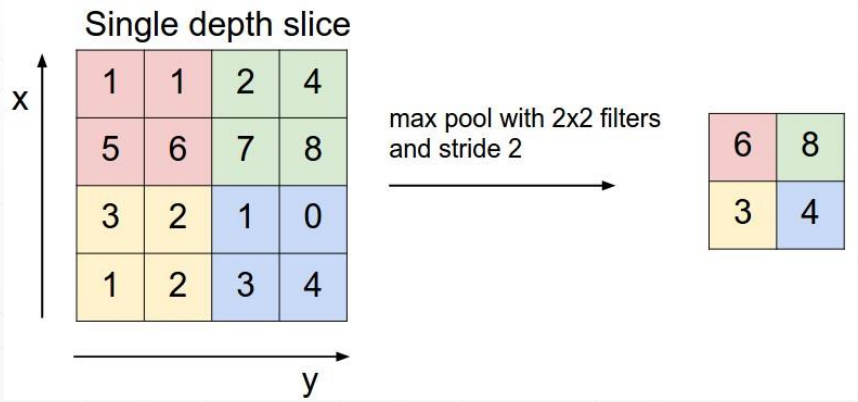
Функции активации



ReLU с «утечкой» (leaky ReLU, LReLU) представляет собой одну из попыток решить описанную выше проблему выхода из строя обычных ReLU. Обычный ReLU на интервале $x < 0$ дает на выходе ноль, в то время как LReLU имеет на этом интервале небольшое отрицательное значение (угловой коэффициент около 0,01). То есть функция для LReLU имеет вид $f(x) = \alpha x$ при $x < 0$ и $f(x) = x$ при $x \geq 0$, где α – малая константа. Некоторые исследователи сообщают об успешном применении данной функции активации, но результаты не всегда стабильны.

Max pooling

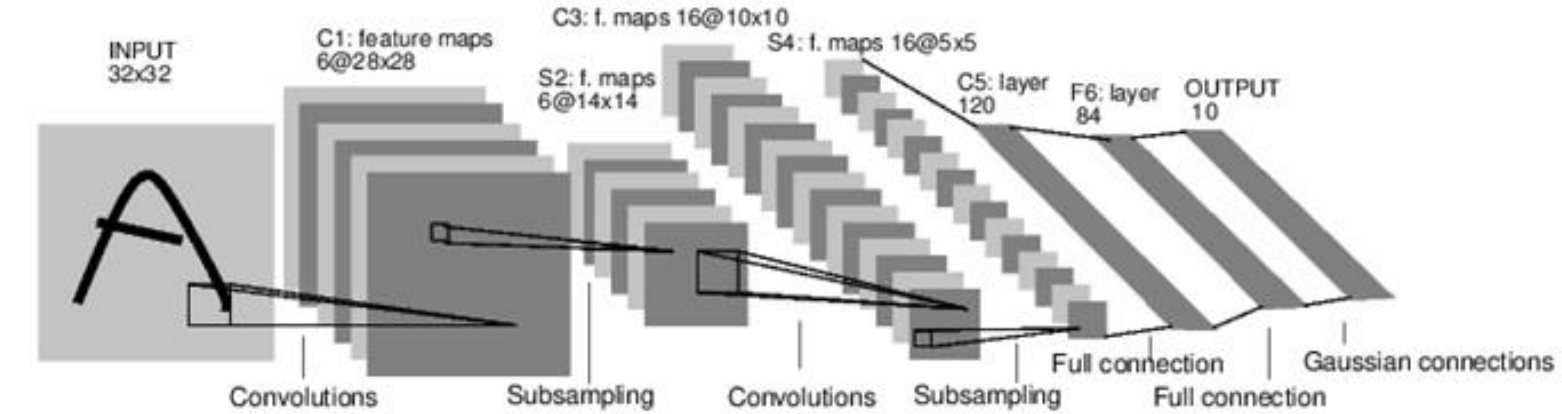




Популярный способ субдискретизации изображения — слой *подвыборки* (также называемый слоем *субдискретизации*, по-английски *downsampling* или *pooling* layer), который получает на вход маленькие отдельные фрагменты изображения (обычно 2x2) и объединяет каждый фрагмент в одно значение.

Существует несколько возможных способов агрегации, наиболее часто из четырех пикселей выбирается максимальный.

Полносвязные слои



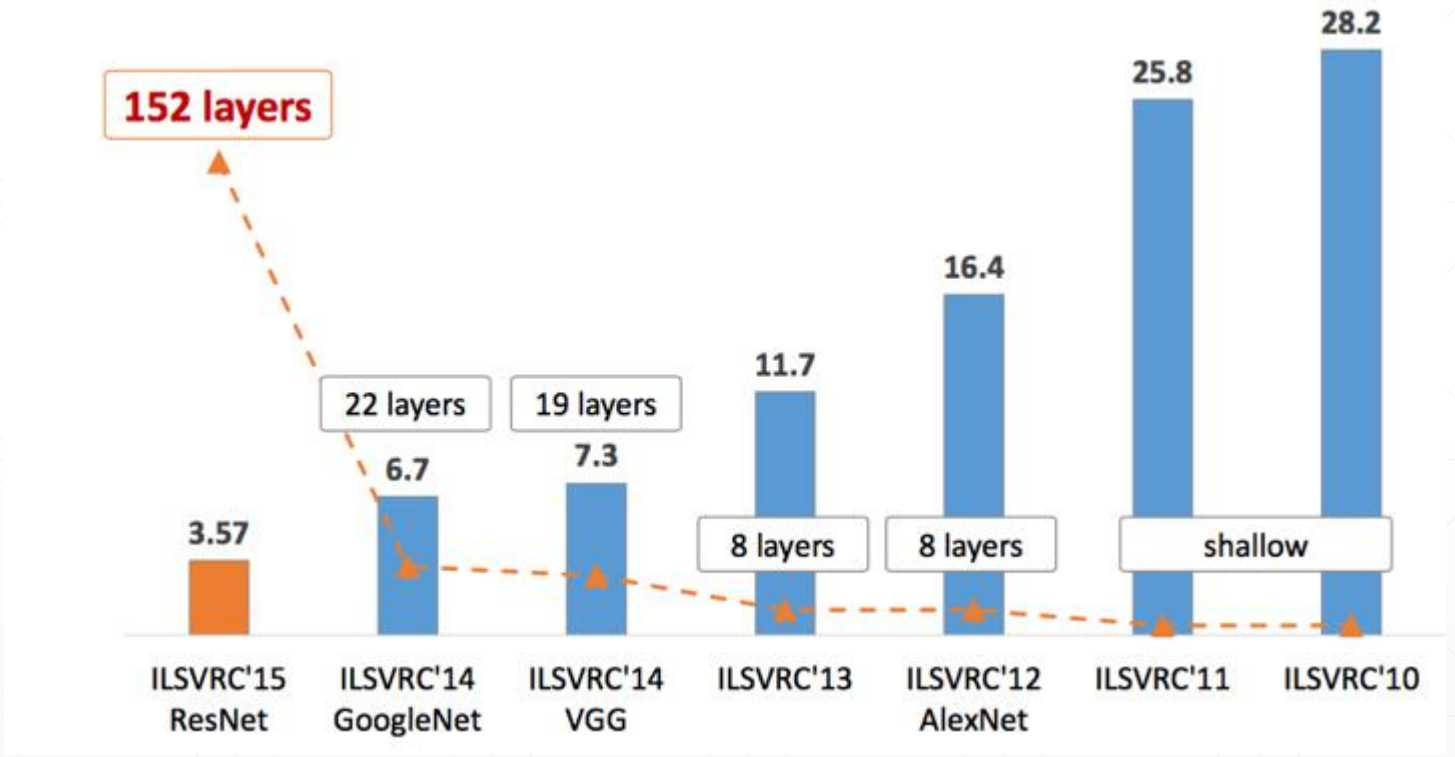
A Full Convolutional Neural Network (LeNet)

Способ, с помощью которого работает полносвязный слой — это обращение к выходу предыдущего слоя (который, как мы помним, должен выводить высокоуровневые карты свойств) и определение свойств, которые больше связаны с определенным классом.

Например, если программа предсказывает, что на каком-то изображении собака, у карт свойств, которые отражают высокоуровневые характеристики, такие как лапы или 4 ноги, должны быть высокие значения.

Архитектуры свёрточных сетей

- LeNet
- AlexNet
- ZF Net.
- GoogLeNet
- VGGNet
- ResNet



VGGNet

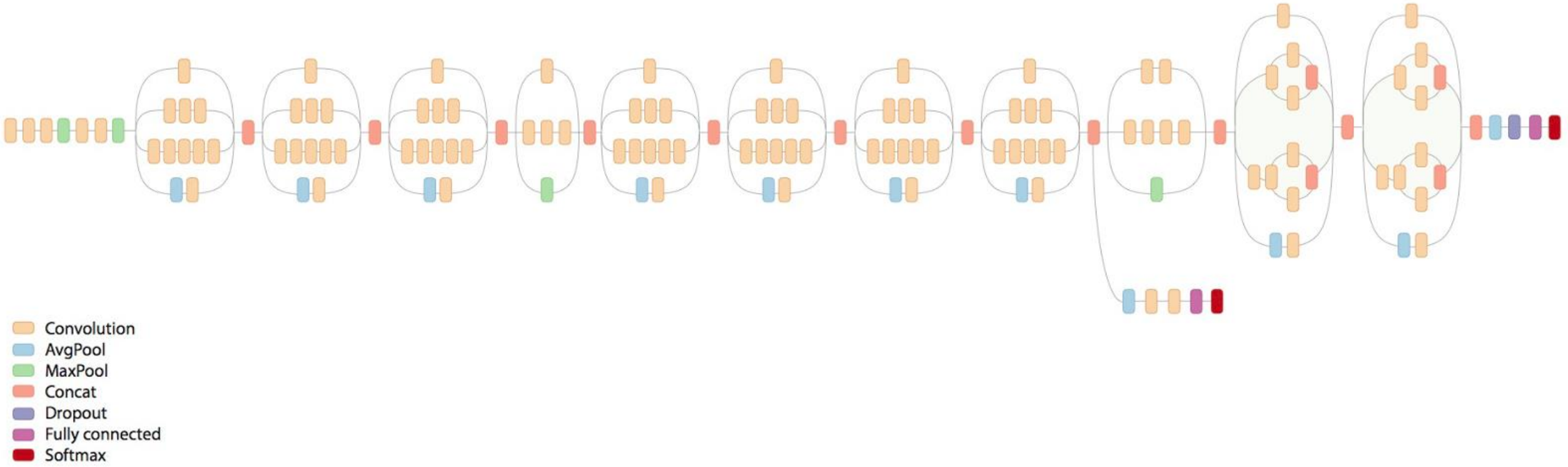
Победитель LSVRC 2014

Показали что лучший результат если использовать глубокие сети.

```
INPUT: [224x224x3]        memory:  224*224*3=150K   weights: 0
CONV3-64: [224x224x64]  memory:  224*224*64=3.2M   weights: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64]  memory:  224*224*64=3.2M   weights: (3*3*64)*64 = 36,864
POOL2: [112x112x64]  memory:  112*112*64=800K   weights: 0
CONV3-128: [112x112x128]  memory:  112*112*128=1.6M   weights: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128]  memory:  112*112*128=1.6M   weights: (3*3*128)*128 = 147,456
POOL2: [56x56x128]  memory:  56*56*128=400K   weights: 0
CONV3-256: [56x56x256]  memory:  56*56*256=800K   weights: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256]  memory:  56*56*256=800K   weights: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256]  memory:  56*56*256=800K   weights: (3*3*256)*256 = 589,824
POOL2: [28x28x256]  memory:  28*28*256=200K   weights: 0
CONV3-512: [28x28x512]  memory:  28*28*512=400K   weights: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512]  memory:  28*28*512=400K   weights: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512]  memory:  28*28*512=400K   weights: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512]  memory:  14*14*512=100K   weights: 0
CONV3-512: [14x14x512]  memory:  14*14*512=100K   weights: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512]  memory:  14*14*512=100K   weights: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512]  memory:  14*14*512=100K   weights: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512]  memory:  7*7*512=25K  weights: 0
FC: [1x1x4096]  memory:  4096  weights: 7*7*512*4096 = 102,760,448
FC: [1x1x4096]  memory:  4096  weights: 4096*4096 = 16,777,216
FC: [1x1x1000]  memory:  1000 weights: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters
```

Inception v3



https://habrahabr.ru/post/302242/

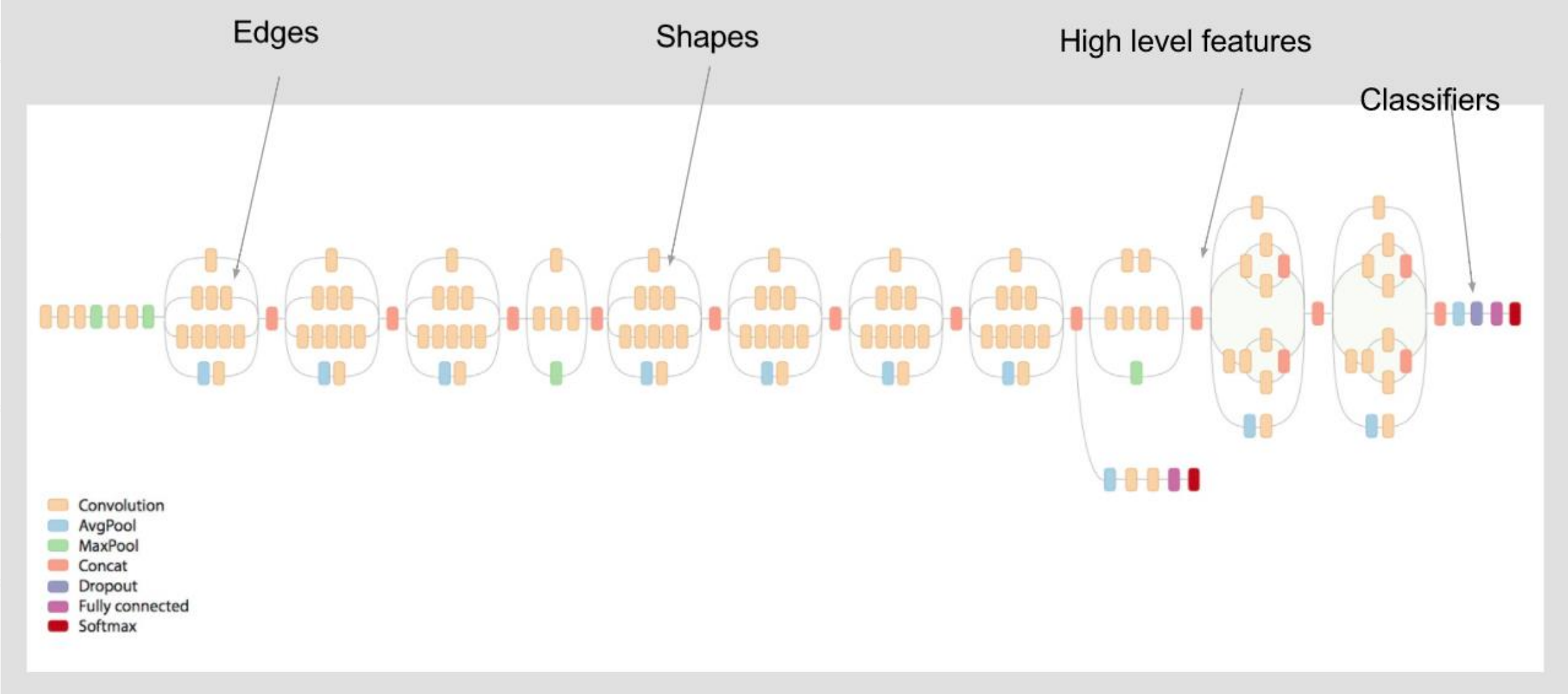
Дополнительная информация

- http://cs231n.github.io/convolutional-networks/
- https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/
- https://habrahabr.ru/post/309508/
- https://habrahabr.ru/company/wunderfund/blog/314872/
- http://datareview.info/article/eto-nuzhno-znat-klyuchevyie-rekomendatsii-po-glubokomu-obucheniyu-chast-2/

Transfer leaning

- На практике редко обучают свёрточные сети с нуля, потому что часто сложно собрать достаточный датасет. Можно использовать претренированную сеть в качестве инициализации весов.
- Глубокие нейронные сети дорого тренировать. Наиболее сложные модели могут занимать недели использую сотни GPU

Что запоминают слои сети?



Дополнительная информация

- https://towardsdatascience.com/transfer-learning-using-keras-d804b2e04ef8
- http://cs231n.github.io/transfer-learning/
- https://deeplearningsandbox.com/how-to-use-transfer-learning-and-fine-tuning-in-keras-and-tensorflow-to-build-an-image-recognition-94b0b02444f2
- https://github.com/yoavz/transfer-learning-keras
- https://habrahabr.ru/company/microsoft/blog/314934/