

# Глубокое обучение

## Лекция 6: Продвинутые модели глубокого машинного зрения

Авторы:

- Сергей Вячеславович Макрушин, e-mail: [s-makrushin@yandex.ru](mailto:s-makrushin@yandex.ru) (<mailto:s-makrushin@yandex.ru>)
- Леонид Алексеев Владимирович

Финансовый университет, 2023 г.

При подготовке лекции использованы материалы:

- ...
- V 0.2

15.11.2023

```
In [1]: # загружаем стиль для оформления презентации
from IPython.display import HTML
from urllib.request import urlopen
html = urlopen("file:./lec_v2.css")
HTML(html.read().decode('utf-8'))
```

Out[1]:

### Разделы:

- [Сверточные сети](#)
- 
- [к оглавлению](#)

[https://colab.research.google.com/drive/1AghjX\\_mmTDirPwPWdSuepSq7sTLG1uhp?usp=sharing#scrollTo=qPKry0D1y2gk](https://colab.research.google.com/drive/1AghjX_mmTDirPwPWdSuepSq7sTLG1uhp?usp=sharing#scrollTo=qPKry0D1y2gk)  
[\(https://colab.research.google.com/drive/1AghjX\\_mmTDirPwPWdSuepSq7sTLG1uhp?usp=sharing#scrollTo=qPKry0D1y2gk\)](https://colab.research.google.com/drive/1AghjX_mmTDirPwPWdSuepSq7sTLG1uhp?usp=sharing#scrollTo=qPKry0D1y2gk)

In [9]:

```
import math
import numpy as np
import tqdm.notebook
import matplotlib.pyplot as plt
import seaborn as sns

import torch
import torch.nn as nn
import torch.nn.functional as F

# import Lightning as L
# L.seed_everything(0)

from transformers import AutoImageProcessor, ResNetForImageClassification
from transformers import AutoFeatureExtractor, VanForImageClassification
from datasets import load_dataset
import evaluate
```

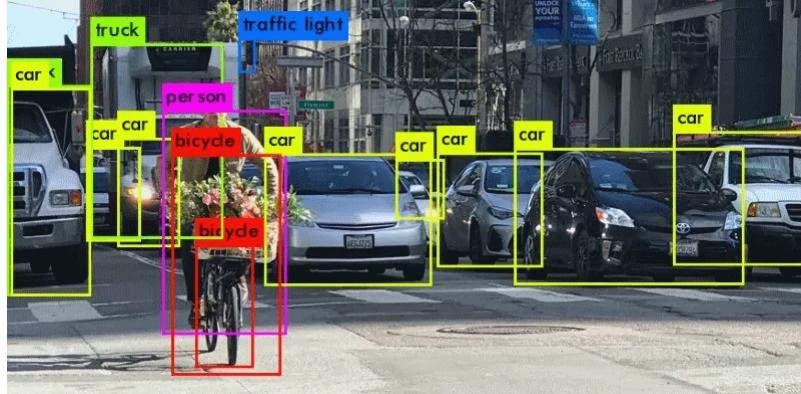
```
-----  
ModuleNotFoundError Traceback (most recent call last)  
~\AppData\Local\Temp\ipykernel_3460\2451718000.py in <module>  
      12 # L.seed_everything(0)  
      13  
---> 14 from transformers import AutoImageProcessor, ResNetForImageClassification  
      15 from transformers import AutoFeatureExtractor, VanForImageClassification  
      16 from datasets import load_dataset  
  
ModuleNotFoundError: No module named 'transformers'
```

## Постановки задач

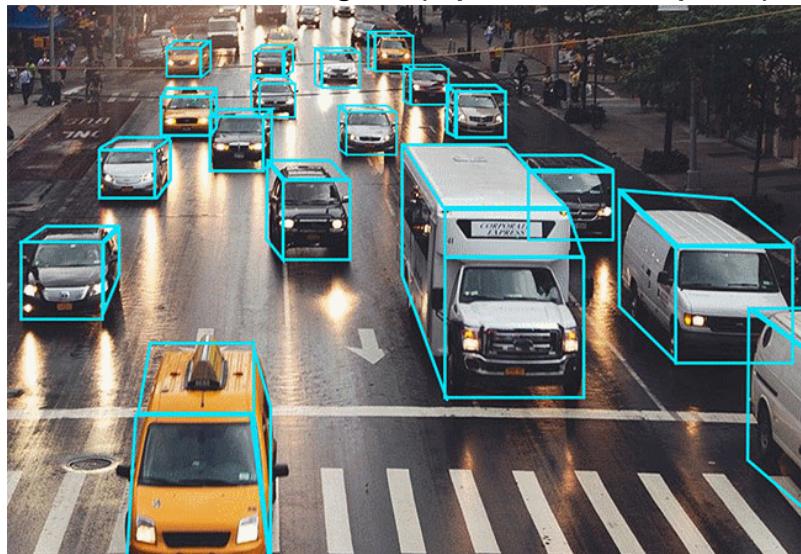
### Типы задач машинного зрения решаемые с помощью глубоких моделей

- **Классификация изображений:** определяется к какому классу или категории относится изображение (например, определение видов животных, классификация цифр, определение объектов на фотографии). Это одна из наиболее распространенных задач.
  - **Семантическая сегментация** - разделение изображения на сегменты и присвоение каждому сегменту метки класса. Это позволяет понимать, какие части изображения принадлежат к различным объектам (например, сегментация дорог, неба, автомобилей на фотографии).
  - **Локализация объектов** - поиск и определение позиции объектов на изображении с выделением их ограничивающих рамок без привязки к конкретным классам (например, поиск и выделение всех лиц на групповой фотографии).
  - **Обнаружение объектов** - задача выявления и локализации объектов на изображении, размещение ограничивающих рамок вокруг этих объектов и присвоение им меток классов (например, детектирование автомобилей на дороге, лиц на фотографиях).
  - **Повышение качества изображений и суперразрешение (superresolution)** - уменьшения шума и улучшение качества изображений, повышение разрешения изображений.
- 
- **Генерация контента** - создание новых изображений, основанных на обучении на больших наборах данных, что может быть полезно в задачах синтеза изображений или стилизаций.
  - **Распознавание действий и сцен** - определение активностей, сцен или последовательностей событий на основе анализа видеоданных.

### Ограничивающие рамки



**Расстановка bounding box (ограничивающих рамок)**



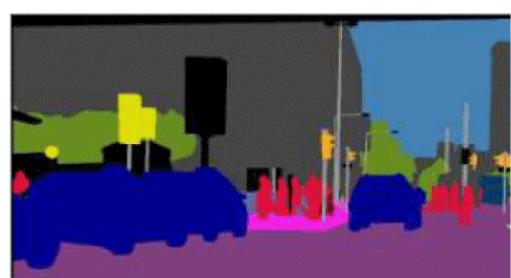
**Расстановка 3D bounding box (трехмерных ограничивающих рамок)**

**Bounding box (ограничительная рамка)** в задачах машинного зрения - это прямоугольник или параллелепипед, который описывает положение и размер объекта на изображении или в видео. Он задается координатами (обычно левый верхний угол и правый нижний угол) и может быть использован для выделения, обозначения или ограничения области, содержащей интересующий объект. Эта техника широко используется в обнаружении и распознавании объектов,

#### Виды задач сегментации изображения



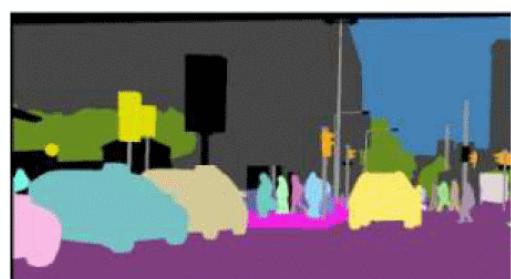
a. Исходное изображение



b. Семантическая сегментация



c. Экземплярная сегментация



d. Паноптическая сегментация

#### Типы сегментации

**Семантическая сегментация (Semantic Segmentation)**: этот вид сегментации направлен на присвоение каждому пикселью изображения метки, отражающей его принадлежность к определенному классу или категории объектов. Например, на изображении сцены собаки, кошки и деревья каждый пиксель будет помечен как "собака", "кошка" или "дерево".

- Применение: Семантическая сегментация широко используется в автоматическом восприятии среды для самоуправляемых автомобилей, сегментации медицинских изображений и в различных задачах компьютерного зрения.

**Сегментация экземпляров (Instance Segmentation)**: этот тип сегментации не только определяет категорию объекта на изображении, но и **различает отдельные экземпляры объектов**, присваивая каждому уникальный идентификатор.

- Например, если на изображении присутствуют две собаки, то каждая из них будет помечена отдельно, даже если они относятся к одному классу.
- Применение: Инстансная сегментация полезна в областях, требующих точной локализации и различия между объектами, таких как трекинг объектов в видео, обнаружение людей на изображениях многолюдных сцен, а также в робототехнике и медицинском анализе.

**Паноптическая сегментация (Panoptic Segmentation)** : этот вид сегментации комбинирует особенности семантической и сегментации экземпляров. Он стремится к **полному покрытию изображения метками** для всех пикселей, независимо от того, являются ли они частью объекта или фона, и при этом различает отдельные экземпляры объектов. В паноптической сегментации различают:

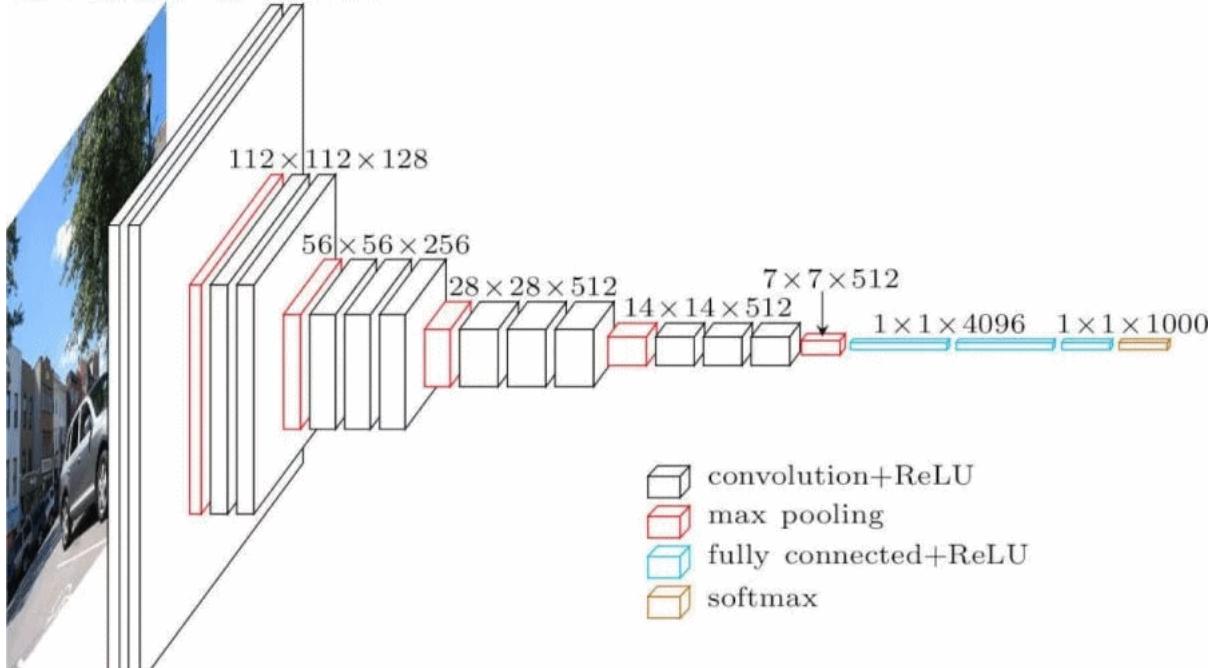
- "**Thing**" (**вещь**) - сегменты изображения, которые представляют объекты, имеющие четкую ограниченную форму и четко выделяемые границы, например, люди, автомобили, животные и другие конкретные объекты.
- "**Stuff**" (**вещество**) - сегменты, представляющие более абстрактные или неструктурированные области на изображении, такие как небо, дорога, трава, вода и другие фоновые или окружающие элементы. Эти области обычно не имеют четких границ или форм и чаще представляют собой большие участки изображения.
- Паноптическая сегментация стремится к объединению "thing" и "stuff", предоставляя **полную семантическую сегментацию всего изображения**, которая включает в себя как объекты "thing", так и контекстуальные области "stuff".
- Применение: Паноптическая сегментация полезна в задачах, где важно не только понимание содержимого изображения, но и различие между объектами и фоном для полного

## Популярные архитектуры

### Архитектура VGG

**VGG16** — модель сверточной нейронной сети, предложенная в 2014 г. K. Simonyan и A. Zisserman из Оксфордского университета

- Модель достигает **точности 92.7% — топ-5**, при тестировании на ImageNet в задаче распознавания объектов на изображении.
- VGG16 — одна из самых знаменитых моделей, отправленных на соревнование ILSVRC-2014.
- Она является **улучшенной версией AlexNet**, в которой **заменены большие фильтры** (размера 11 и 5 в первом и втором сверточном слое, соответственно) **на несколько фильтров размера 3x3**, следующих один за другим.
- Использовать предобученную VGG16 в PyTorch можно используя этот вызов:  
<https://pytorch.org/vision/main/models/generated/torchvision.models.vgg16.html>  
(<https://pytorch.org/vision/main/models/generated/torchvision.models.vgg16.html>)
- Модель предложена в статье “Very Deep Convolutional Networks for Large-Scale Image Recognition” см. <https://arxiv.org/pdf/1409.1556.pdf> (<https://arxiv.org/pdf/1409.1556.pdf>)



**Изображения двухмерные**

Архитектура:

- На вход слоя conv1 подаются RGB изображения размера 224x224.
- Далее изображения проходят через стек сверточных слоев, в которых используются фильтры с очень маленьким рецептивным полем размера 3x3 (который является наименьшим размером для получения представления о том, где находится право/лево, верх/низ, центр).
- В одной из конфигураций используется сверточный фильтр размера 1x1, который может быть представлен как линейная трансформация входных каналов (с последующей непропорциональностью).
- Сверточный шаг фиксируется на значении 1 пиксель. Пространственное дополнение (padding) входа сверточного слоя выбирается таким образом, чтобы пространственное разрешение сохранялось после свертки, то есть дополнение равно 1 для 3x3 сверточных слоев.
- Пространственный пулинг осуществляется при помощи пяти max-pooling слоев, которые следуют за одним из сверточных слоев (не все сверточные слои имеют последующие max-pooling). Операция max-pooling выполняется на окне размера 2x2 пикселей с шагом 2.
- После стека сверточных слоев (который имеет разную глубину в разных архитектурах) идут три полно связанных слоя: первые два имеют по 4096 каналов, третий — 1000 каналов (так как в соревновании ILSVRC требуется классифицировать объекты по 1000 категориям; следовательно, классу соответствует один канал).
- Все скрытые слои снабжены ReLU. Отметим также, что сети (за исключением одной) не содержат слоев нормализации (Local Response Normalisation).
- Последним идет soft-max слой. Конфигурация полно связанных слоев одна и та же во всех нейросетях.

Недостатки VGG:

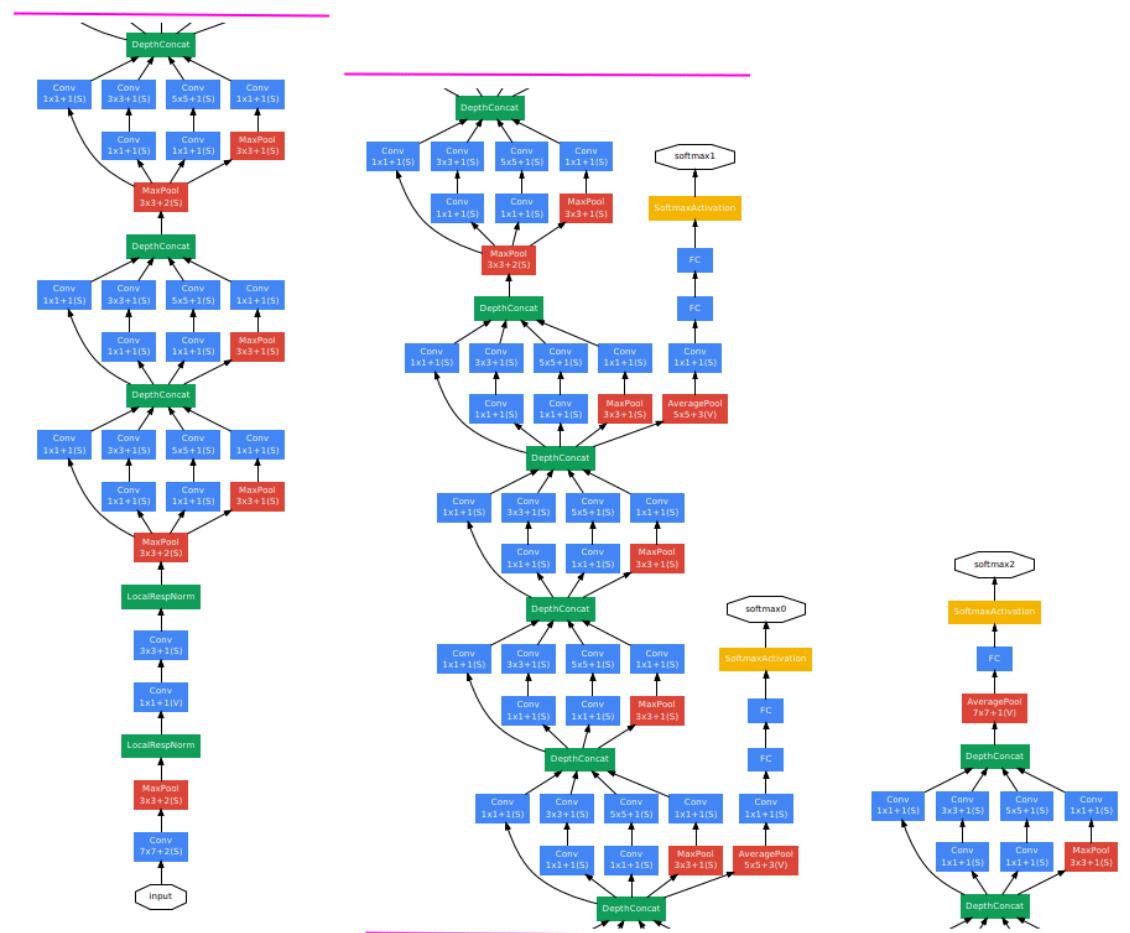
- Очень медленная скорость обучения.
- Сама архитектура сети весит слишком много (появляются проблемы с дисковым и пропускной способностью) Из-за глубины и количества полно связанных узлов, VGG16 весит более 533 МБ.
- Хотя VGG16 и используется для решения многих проблем классификации при помощи нейронных сетей, меньшие архитектуры более предпочтительны (SqueezeNet, GoogLeNet и другие).

Несмотря на недостатки, данная архитектура является отличным строительным блоком для обучения, так как её легко реализовать.

## GoogLeNet

**GoogLeNet** - глубокая нейронная сеть, представленная исследователями компании Google в 2014 году. Она известна своей эффективностью и особенной архитектурой, основанной на модулях, называемых **Inception modules**.

- Ее основное преимущество не точность, а **эффективность в размере модели и необходимом количестве вычислений**.



## Изображения двухмерные

## Основные черты архитектуры GoogLeNet:

- **Inception Modules:** Основная особенность GoogLeNet - использование Inception modules, которые являются блоками с **параллельными сверточными операциями ядрами разного размера (1x1, 3x3, 5x5) и операцией пулинга**. Это позволяет сети эффективно извлекать **признаки различных масштабов на разных уровнях**.
  - **Глубина и ширина сети:** GoogLeNet имеет большую глубину, но при этом **содержит относительно меньше параметров** по сравнению с другими глубокими сетями, разработанными в тот период. Это достигается за счет **использования 1x1 сверток для уменьшения размерности признаковых карт перед 3x3 и 5x5 свертками** в Inception модулях.
  - **Reduction Blocks:** Они используются для **уменьшения размерности признаковых карт** перед сверточными операциями, чтобы уменьшить вычислительную сложность и объем параметров. В этих блоках используются комбинации сверток и операций пулинга для сокращения размерности.
  - **Auxiliary Classifiers:** GoogLeNet также содержит вспомогательные классификаторы (**вспомогательные выходы**) во время обучения, **помогающие в борьбе с проблемой затухания градиента**. Эти вспомогательные классификаторы помогают передавать градиенты назад для обучения на ранних этапах сети.
  - **Использование среднего пулинга (average pooling) вместо полно связных слоев:** В конце сети GoogLeNet использует глобальный средний пулинг для преобразования признаковых карт в одномерный вектор перед финальным классификатором.

- <http://arxiv.org/abs/1409.4842> (<http://arxiv.org/abs/1409.4842>).

## Свертка 1x1

**Свертка с ядром 1x1** используется для изменения размерности каналов тензора слоя признаковой карты изображения в сверточных нейронных сетях.

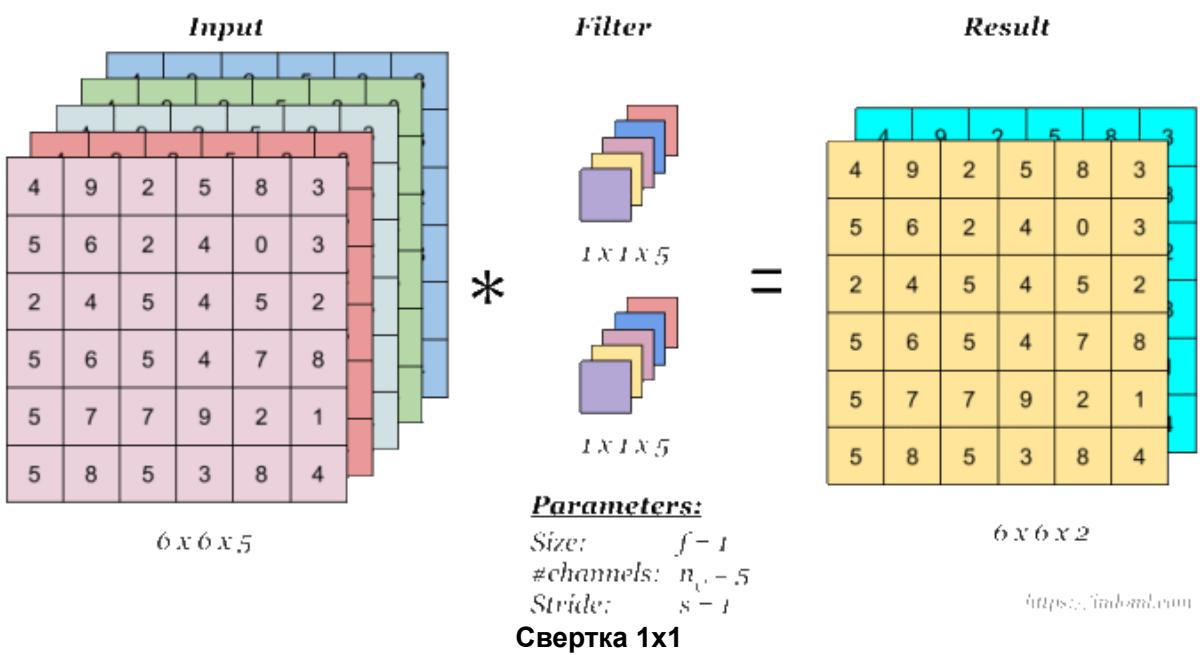
Допустим, у нас есть входной тензор размерности  $(N, F_0, H, W)$ , где:

- $N$  - размер батча
- $F_0$  - количество сверточных фильтров
- $H, W$  - пространственные размерности

Если мы применяем слой свертки с  $F_1$  фильтрами пространственной размерности 1x1, тогда новая размерность выходного тензора будет  $(N, F_1, H, W)$ . Здесь  $F_1$  - новое количество сверточных фильтров.

- Пространственная размерность (высота и ширина) при этом не изменяется.

Таким образом, **операция свертки 1x1 используется для изменения размерности фильтров (F)** в сверточном слое. Если  $F_1 > F_0$ , то мы увеличиваем размерность (количество фильтров), а если  $F_1 < F_0$ , то мы уменьшаем размерность (количество фильтров) в пространстве фильтров.



## Ключевые эффекты от применения сверточных операций с ядром 1x1:

- **Уменьшение размерности признаковых карт** Одно из главных преимуществ сверточных операций с ядром 1x1 - это уменьшение размерности признаковых карт. Это происходит путем снижения количества каналов (глубины) признаковой карты, что **помогает уменьшить вычислительную сложность и количество параметров в нейронной сети**.
- **Увеличение вычислительной эффективности** Использование ядер свертки размером 1x1 может значительно снизить количество вычислений по сравнению с большими ядрами, что ускоряет процесс обучения и инференса в нейронной сети.
- **Комбинирование признаков** Свертка с ядром 1x1 **выполняет линейную комбинацию признаков** на каждом пикселе входной карты. Даже несмотря на то, что эта операция использует ядро с минимальным размером, она позволяет **моделировать нелинейные комбинации признаков из предыдущего слоя**, что улучшает способность сети **извлекать более сложные и абстрактные признаки**.

- Регуляризация и сокращение размерности Операция с ядром 1x1 также может служить методом регуляризации и сокращения размерности, что способствует **предотвращению переобучения и улучшает обобщающую способность** модели.

Следует отметить, что использование сверток с ядром 1x1 является важной частью архитектуры

### Inception Module

Основная идея модуля Inception состоит в том, чтобы использовать **несколько параллельных сверточных операций с различными размерами ядер** и объединить их результаты.

Inception модуль состоит из нескольких слоев, каждый из которых **выполняет операции свертки и пулинга параллельно**. Это позволяет модели **одновременно извлекать информацию на разных уровнях детализации** и обнаруживать признаки различных размеров.

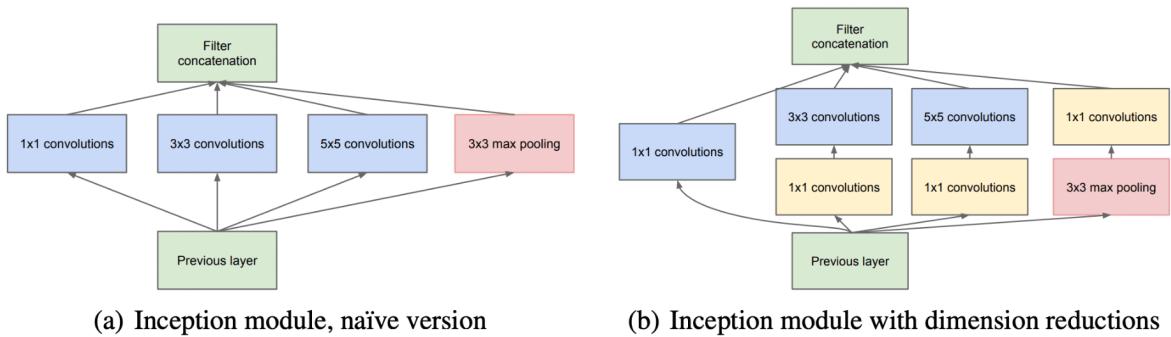
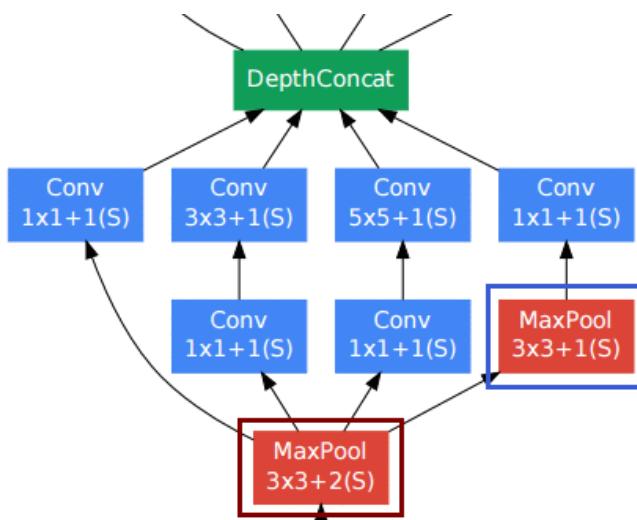


Figure 2: Inception module  
Изображения двухмерные

Основные компоненты, которые включаются в Inception модуль:

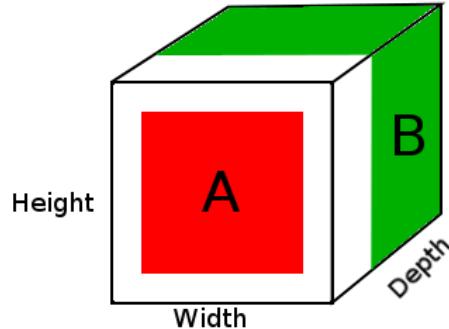
- **1x1 свертка:** этот слой используется для **уменьшения размерности признаковых карт**. 1x1 свертка помогает уменьшить количество каналов, что позволяет сократить вычислительные затраты.
- **3x3 свертка:** Этот слой обеспечивает извлечение признаков **на среднем уровне детализации**. Он позволяет модели улавливать более сложные пространственные паттерны на изображениях.
- **5x5 свертка:** Этот слой помогает модели **обнаруживать более крупные и абстрактные особенности** на изображении.



Изображения двухмерные

- **Пулинг:** Max pooling слои также используются параллельно с сверточными операциями для **уменьшения пространственной размерности** и извлечения ключевых признаков из различных областей изображения.

- В разных частях модуля Inception используются пулинг с разным шагом (stride). Пулинг с шагом 1 не приводит к изменению пространственного размера признаковой карты.



**Схема работы DepthConcat**

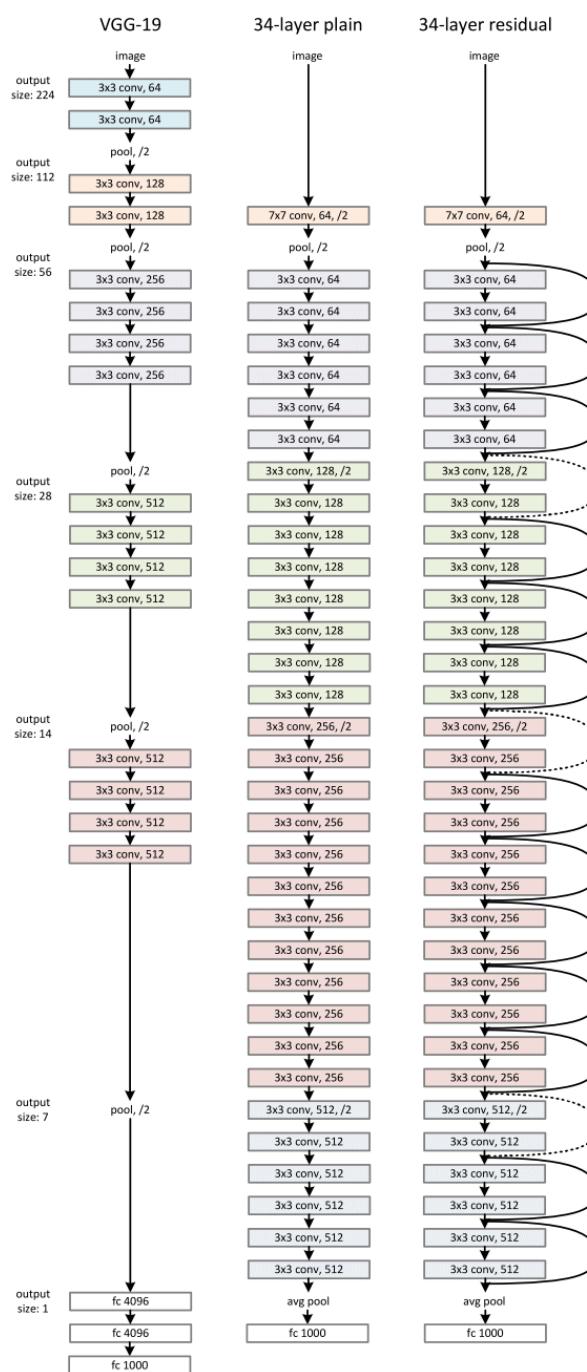
- Concatenation (объединение): Результаты параллельных операций (результаты сверток и пулинга) объединяются в одну выходную признаковую карту путем конкатенации по оси каналов. Это позволяет модели использовать информацию, извлеченную из разных типов операций, для более точного и полного представления признаков изображения.
  - При этом модуль DepthConcat выбирает наибольшее пространственное разрешение поступающих пространственных карт, укладывает все карты вдоль слоя глубины тензора, а карты с меньшей размерностью центрирует и заполняет (padding) нулями края этих карт для выравнивания размерности во всем тензоре признаковых карт.

Inception модули повторяются в сети GoogLeNet с различными параметрами и количеством параллельных операций для извлечения более разнообразных и богатых признаков изображений

## Глубокие архитектуры со skip connections (residual connections)

По мере увеличения глубины ИНС все сильнее проявляется проблема затухающего градиента: когда градиенты, проходящие через сеть, уменьшаются, что приводит к медленной сходимости и ухудшению производительности.

Кроме перечисленных выше подходов для решения этой проблемы используют **архитектуры со skip connections** (residual connections).

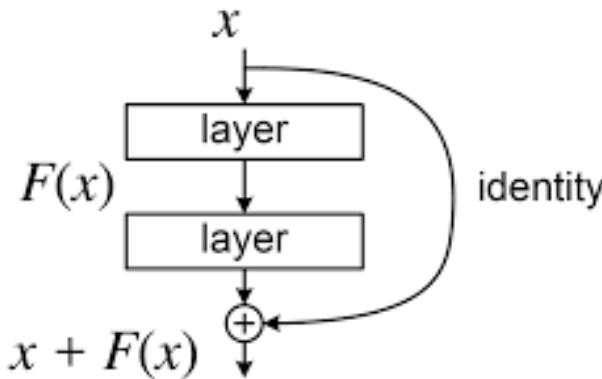


### Архитектура ResNet с residual connections

Мотивация для архитектур с residual connections:

- Глубокие нейронные сети строят иерархические представления, строя стек из множества слоев. Каждый слой захватывает все более абстрактные особенности входных данных.
- Однако по мере увеличения глубины сети градиенты, распространяющиеся назад во время обучения, становятся крайне маленькими, что затрудняет обучение сети затруднительным

### Знакомство с residual connections



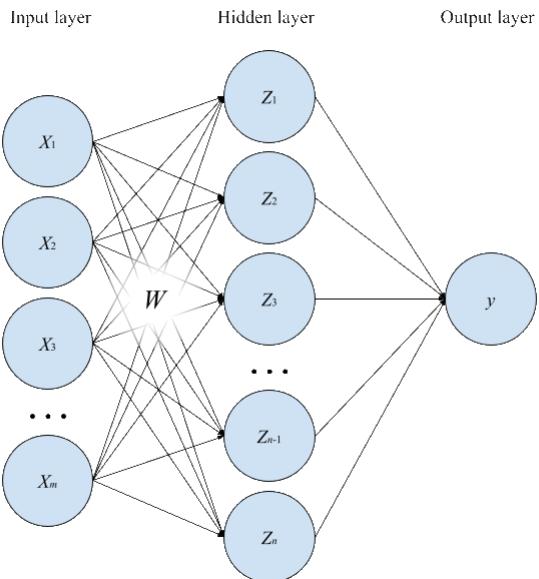
### Блок ИНС с residual connections

Residual connections работают, напрямую соединения слои с пропуском некоторых промежуточных, что позволяют градиенту напрямую переходить из одного слоя в другой без изменений.

- В отличие от традиционных прямых соединений, резидуальные соединения добавляют исходный вход (идентичное отображение) к выходу последующего слоя.
- Основная идея резидуальных соединений заключается в том, что сеть учится резидуальному отображению, которое представляет собой разницу между желаемым выходом  $F(x)$  и входом  $x$ . Добавляя резидуальное отображение к входу, сеть может сосредотачиваться на изучении различий или изменений, необходимых для получения желаемого выхода, вместо того чтобы учить всё отображение с нуля.

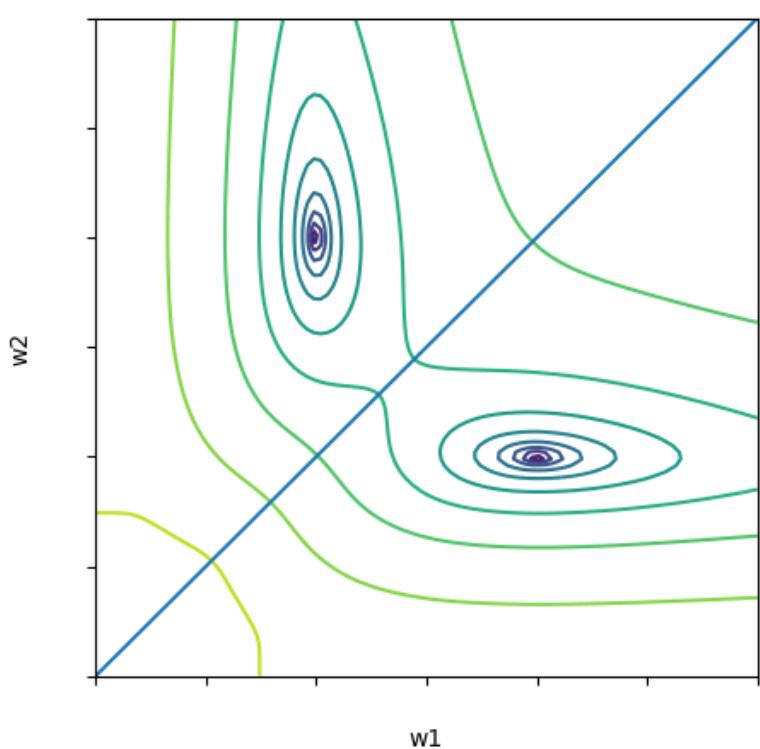
Skip connection в residual архитектурах помогают модели выучить тождественное преобразование - таким образом, имея возможность пропускать сигнал без трансформаций, модель должна, по крайней мере, быть не хуже, чем ее менее глубокая версия, следовательно, деградировать точность не может.

### Свойство симметрии в ИНС



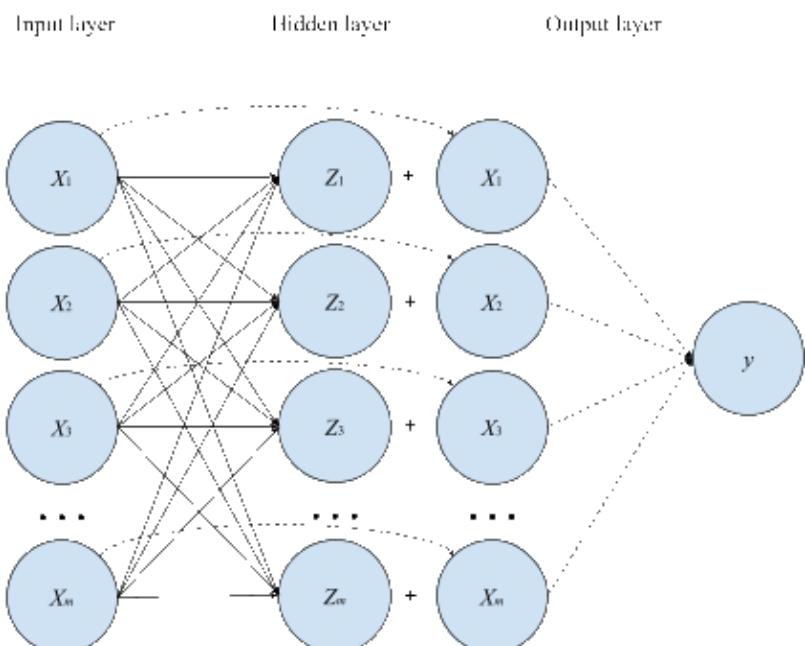
Простая полносвязная ИНС

- Для простоты рассуждений возьмем полносвязную сеть с одним скрытым слоем . Рассмотрим, для начала, случай, когда все веса выходного линейного слоя фиксированы и равны  $1/n$ , т.е. последний слой выполняет роль простого арифметического усреднения выходов нейронов скрытого слоя.
- Что произойдет с результатом работы сети, если мы поменяем местами, скажем, первый и второй нейроны скрытого слоя? Очевидно, что при такой перестановке результат работы сети не изменится.
- Т.е. в худшем случае у нас есть  $n!$  симметричных вариантов в параметрическом пространстве. Это приводит к появлению сложного оптимизационного ландшафта, особенно в глубоких сетях.



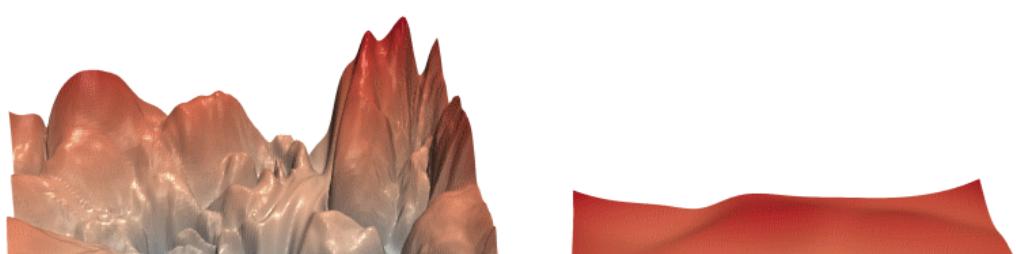
**Ландшафт квадратичной функции потерь для модели с двумя параметрами; синяя прямая - 1d плоскость симметрии параметрического пространства**

- Добавление skip connections нарушает симметрию в параметрическом пространстве и упрощает задачу поиска минимума.

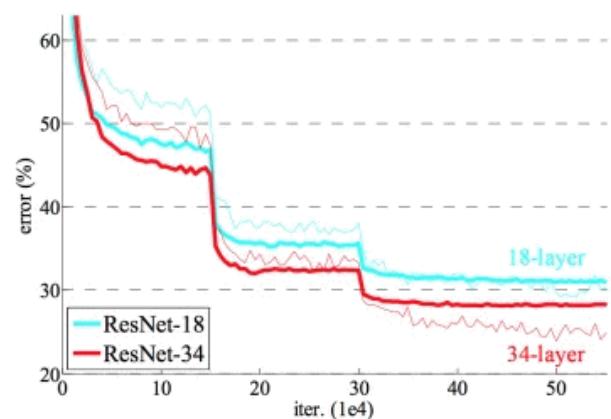
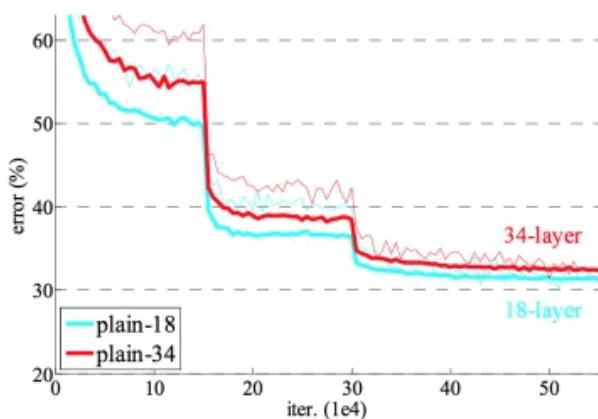


**Полносвязная ИНС с добавленными skip connections от входного слоя к скрытому сло**

- Добавление skip connections нарушает симметрию в параметрическом пространстве и упрощает задачу поиска минимума.



**Улучшение метрик сверточных моделей после добавления skip connection**



### Сравнение обучения resnet и обычной сверточной сети

Слева: простые сети из 18 и 34 слоев. Справа: ResNets из 18 и 34 слоев.

Тонкие кривые обозначают ошибку обучения, а жирные кривые обозначают ошибку валидации.

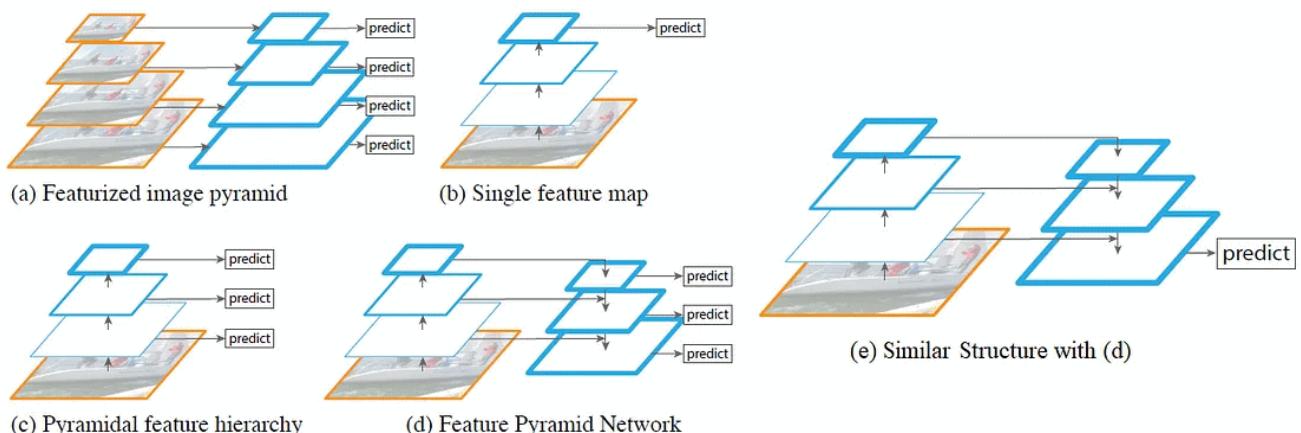
- С увеличением глубины сверточной сети точность **сначала увеличивается, а затем быстро ухудшается**. Более глубокая сеть имеет большую ошибку обучения и, следовательно, ошибку тестирования.

## Архитектуры использующие skip-connection

### Feature Pyramid Network

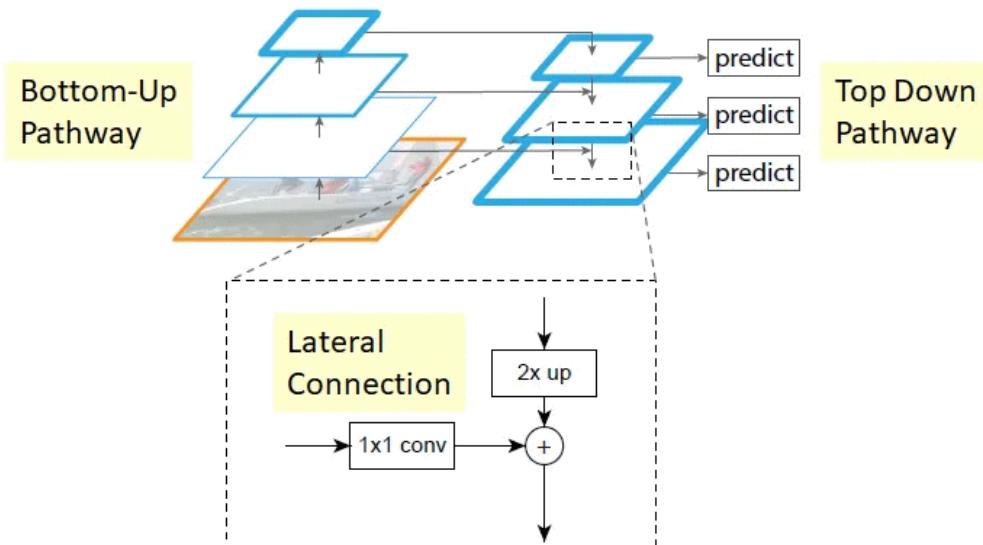
**Feature Pyramid Network (сеть с пирамидой признаков, FPN)**, — нейронная сеть с особым механизмом извлечения извлечения признаков, которое принимает одномасштабное изображение произвольного размера в качестве входных данных и выводит карты признаков пропорционального размера на нескольких заданных уровнях.

- Этот подход **не зависит от базовых архитектур (backbone)**. Таким образом, он действует как универсальное решение для построения пирамид признаков внутри глубоких сверточных сетей, которые будут использоваться в различных задачах.
- Feature Pyramid является одним из главных базовых механизмов многокомпонентных систем распознавания для обнаружения объектов в различных масштабах.



- (a) **Featurized image pyramid** - подход широко использовался в эпоху создания ручных признаков.
- (b) **Single feature map** - это стандартное решение сверточной архитектуры на одном входном изображении, где предсказание происходит в конце сети.
- (c) **Pyramidal Feature Hierarchy** - На каждом уровне делается предсказание. Архитектура повторно использует многомасштабные карты признаков с разных уровней, вычисленные на прямом проходе, и поэтому обходится бесплатно (т.к. при выполнении свертки эти вычисления итак выполняются).

- Однако тут упускается возможность повторного использования карт признаков более высокого разрешения в иерархии, следовательно, пропускается обнаружение маленьких объектов.
- (d) **Feature Pyramid Network** - архитектура объединяет семантически насыщенные признаки низкого разрешения с семантически слабыми признаками высокого разрешения через верхний и боковой путь (a top-down pathway and lateral connections). Эта пирамида признаков имеет богатую семантику на всех уровнях и быстро создается из одного входного изображения.
- (e) **Similar Architecture** - многие архитектуры (такие как TDM, SharpMask, RED-Net, U-Net) используют схожие top-down pathway и lateral connections, однако они делают предсказания только на последнем этапе.



111

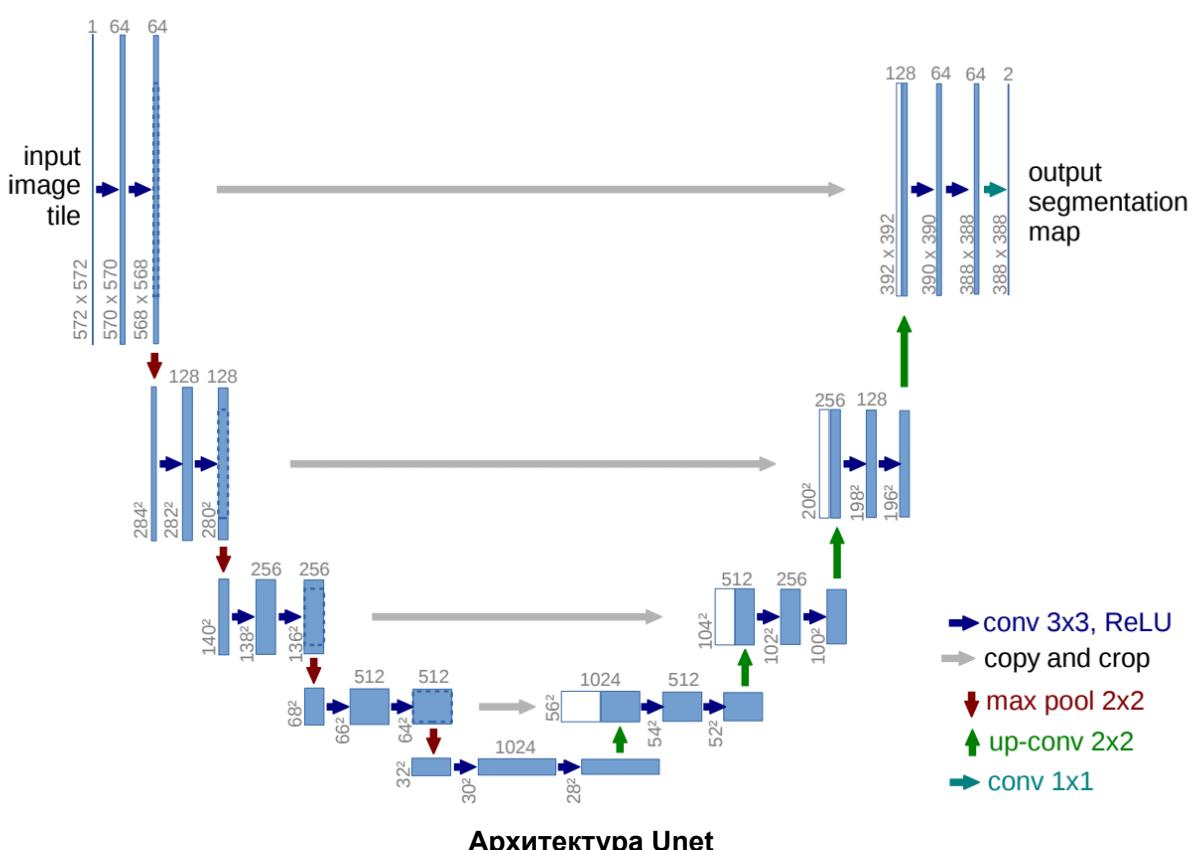
Ссылки:

- <https://towardsdatascience.com/review-fpn-feature-pyramid-network-object-detection-262fc7482610> (<https://towardsdatascience.com/review-fpn-feature-pyramid-network-object-detection-262fc7482610>)
- <https://paperswithcode.com/method/fpn> (<https://paperswithcode.com/method/fpn>)
- <https://paperswithcode.com/methods/category/feature-pyramid-blocks> (<https://paperswithcode.com/methods/category/feature-pyramid-blocks>).

## Unet

**UNet** - сверточная нейронная сеть, разработанная для **сегментации изображений**, которая эффективно решает задачи **семантической сегментации**. Она была предложена Ольфом Роннебергером и другими исследователями в 2015 году.

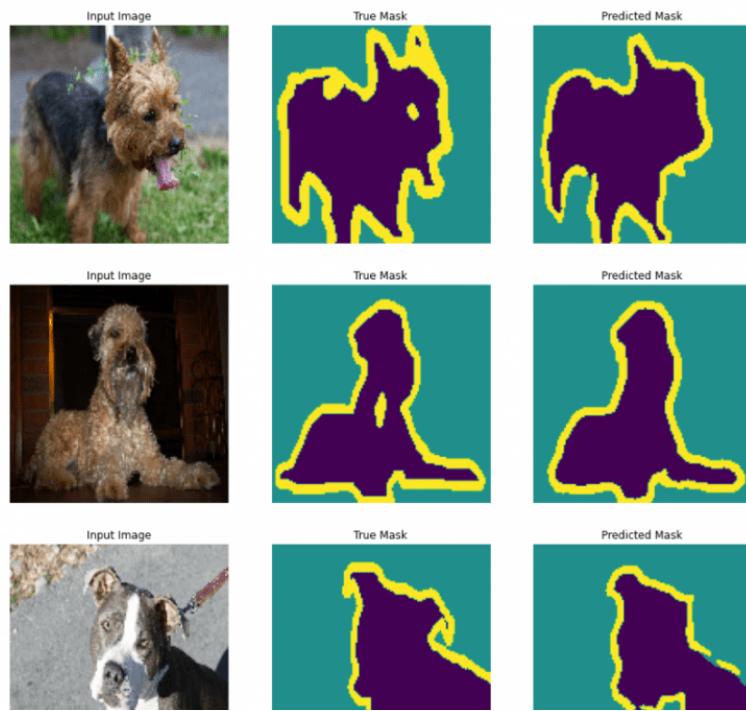
- Особенность UNet: **архитектура энкодера-декодера со skip connections**. Она помогает сохранить пространственную информацию в процессе уменьшения размерности изображения и в последующем его увеличении для лучшего восстановления деталей на выходе сети.



Архитектура Unet

Вот основные компоненты и характеристики модели UNet:

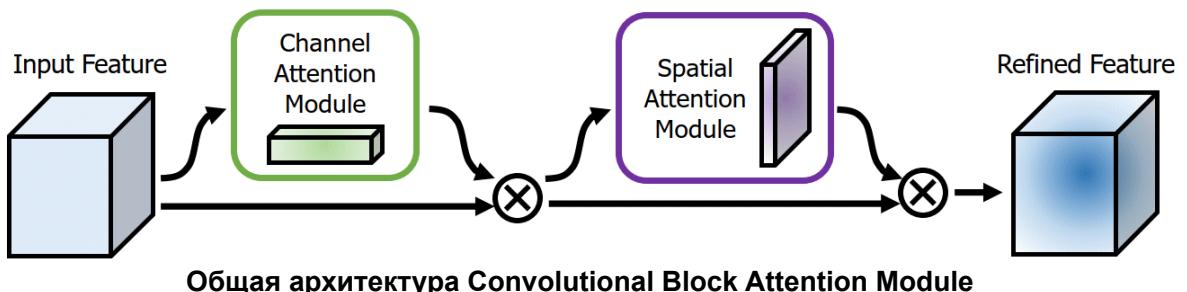
- **Энкодер (Encoder)** - состоит из последовательности сверточных слоев и пулинга, которые помогают извлекать признаки из изображения и уменьшают его размерность.
- **Декодер (Decoder)** - декодер состоит из транспонированных сверточных слоев (обычно Conv2DTranspose или UpSampling2D), которые увеличивают размерность изображения. Этот этап восстанавливает пространственную информацию к размеру исходного изображения.
- **Skip Connections** - позволяют передавать информацию о пространственной структуре изображения из более глубоких слоев энкодера к соответствующим слоям декодера. Это помогает восстановить более точные детали и контекст изображения.



## Attention

**Механизм внимания (attention)** в современных сверточных нейронных сетях применяется для улучшения способности сетей к **выделению значимых признаков из входных данных**. Этот механизм позволяет **определять важность различных частей входных данных** при выполнении задачи и **сосредотачиваться** на наиболее информативных областях.

### Convolutional Block Attention Module



Механизм внимания может быть применен в сверточных нейронных сетях в нескольких формах:

- **Spatial Attention (Пространственное внимание)**: Этот тип внимания сосредотачивается на определенных **областях в пространстве входных данных**. В контексте сверточных сетей, применение пространственного внимания позволяет модели выбирать наиболее значимые области изображения, что может быть полезно для задач, таких как **детектирование объектов или сегментация**.
- **Channel Attention (Канальное внимание)**: Этот тип внимания фокусируется на **каналах признаковых карт**. Он позволяет модели выделять наиболее информативные каналы, что помогает **усилить или подавить определенные признаки** на различных уровнях абстракции. Это может быть полезным для **извлечения ключевых признаков** из изображений или других типов входных данных.

Ссылки:

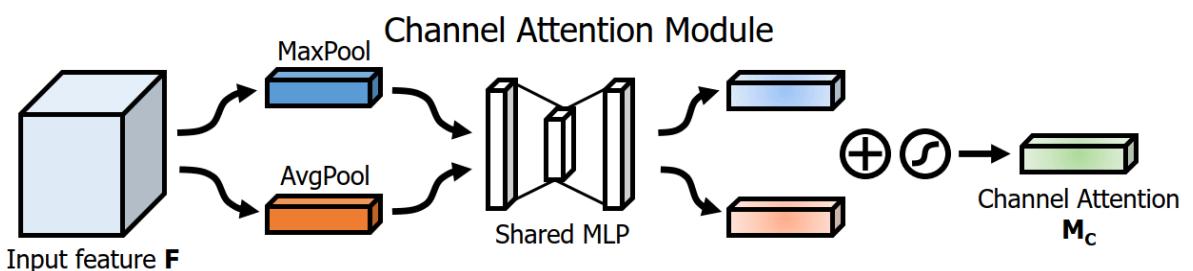
- <https://arxiv.org/abs/1807.06521> (<https://arxiv.org/abs/1807.06521>)

### Channel Attention

**Модуль внимания к каналам (Channel Attention Module)** создает карту канального внимания в сверточных нейронных сетях, используя межканальные взаимосвязи признаков. Поскольку **каждый канал карты признаков рассматривается как отдельный детектор признаков**, канальное внимание сосредотачивается на тех **признаках**, которые являются значимыми при заданном во входном изображении. Для эффективного вычисления канального внимания мы **сжимаем пространственное измерение** входной карты признаков (до 1x1).

$$\begin{aligned} M_c(F) &= \sigma(MLP(\text{AvgPool}(F)) + MLP(\text{MaxPool}(F))) \\ &= \sigma(W_1(W_0(F_{\text{avg}}^c)) + W_1(W_0(F_{\text{max}}^c))), \end{aligned} \quad (2)$$

where  $\sigma$  denotes the sigmoid function,  $W_0 \in \mathbb{R}^{C/r \times C}$ , and  $W_1 \in \mathbb{R}^{C \times C/r}$ . Note that the MLP weights,  $W_0$  and  $W_1$ , are shared for both inputs and the ReLU activation function is followed by  $W_0$ .



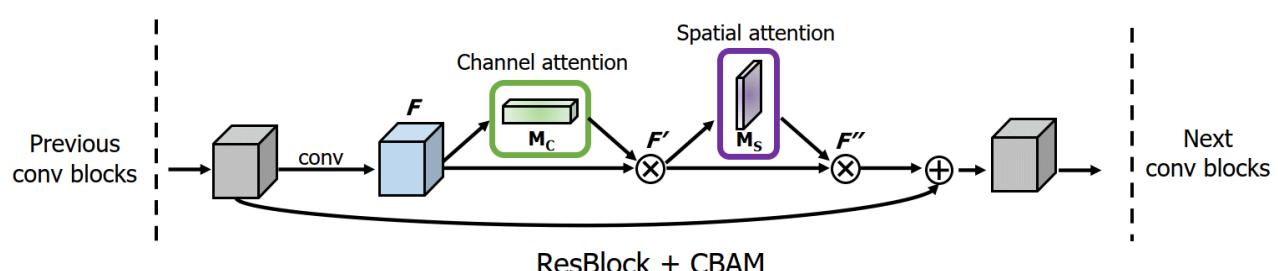
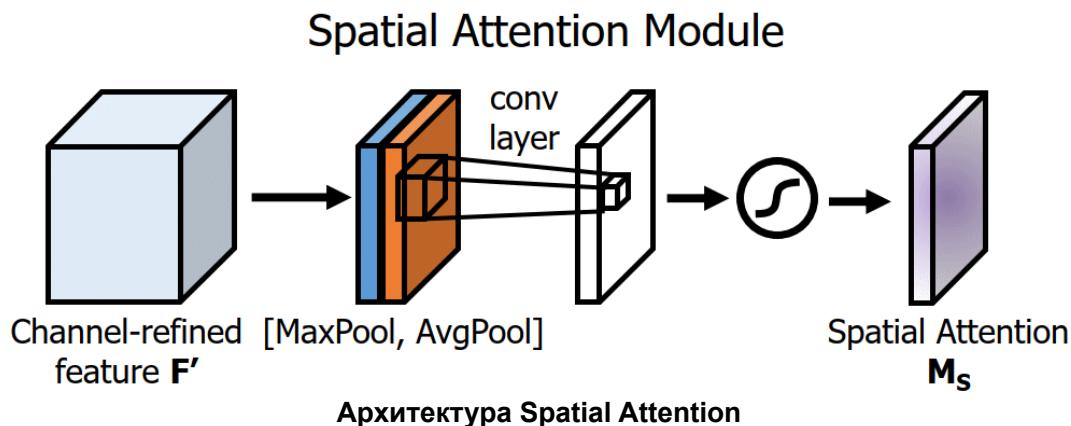
## Spatial Attention

Модуль пространственного внимания (Spatial Attention Module) создает карту пространственного внимания, используя взаимосвязь между пространственными признаками в картах признаков. В отличие от канального внимания, **пространственное внимание фокусируется на том, где** находится информативная часть, что **дополняет канальное внимание**.

Для вычисления пространственного внимания сначала применяются операции усреднения и максимума по оси каналов и выполняется их объединение. Таким образом создается эффективное пространственное описание признаков. На объединенном описании признаков применяется сверточный слой для создания карты пространственного внимания, которая кодирует, где усиливать или подавлять признаки.

$$\begin{aligned} \mathbf{M}_s(\mathbf{F}) &= \sigma(f^{7 \times 7}([\text{AvgPool}(\mathbf{F}); \text{MaxPool}(\mathbf{F})])) \\ &= \sigma(f^{7 \times 7}([\mathbf{F}_{\text{avg}}^s; \mathbf{F}_{\text{max}}^s])), \end{aligned} \quad (3)$$

where  $\sigma$  denotes the sigmoid function and  $f^{7 \times 7}$  represents a convolution operation with the filter size of  $7 \times 7$ .



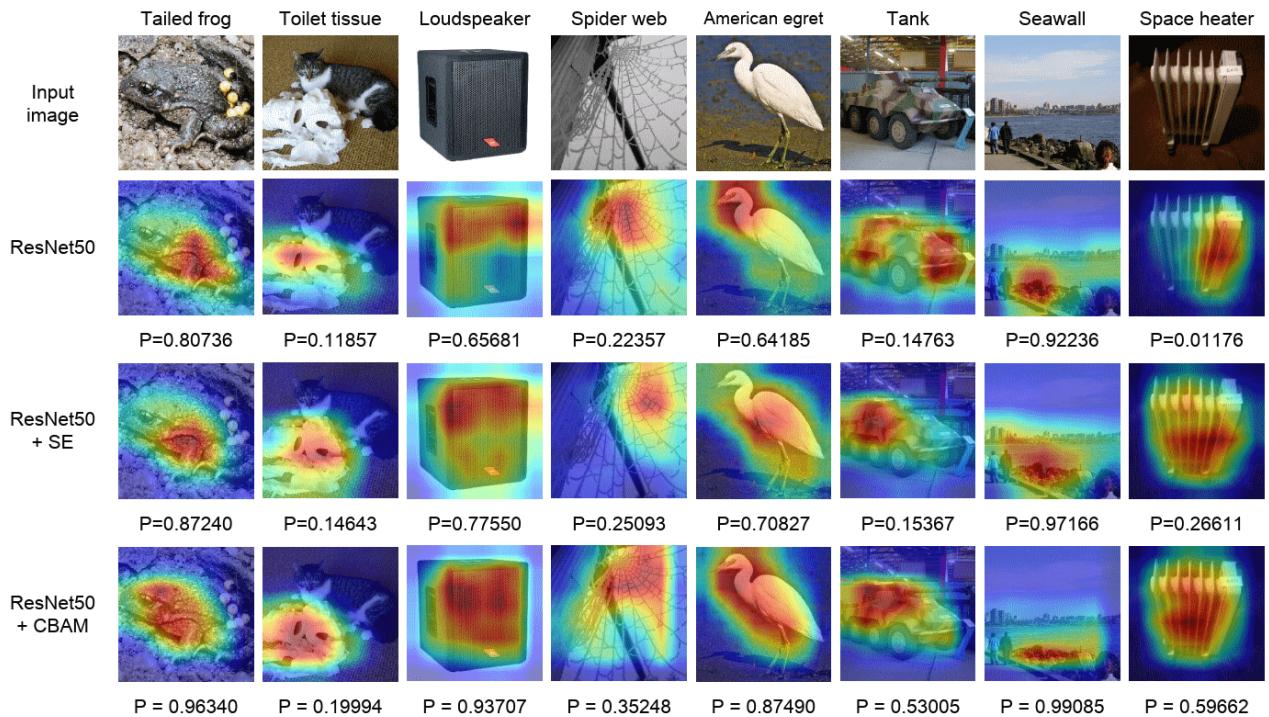
Интеграция блоков архитектуры Convolutional Block Attention Module в сверточную сеть со Skip Connection

**Grad-CAM (Gradient-weighted Class Activation Mapping)** - метод визуализации, который позволяет понять, **какие области** изображения входа в сверточной нейронной сети были **наиболее активны** при принятии решения о классификации. Он помогает исследовать, **как модель сосредотачивается на определенных областях изображения** для принятия окончательного вывода.

Принцип работы Grad-CAM основан на **градиентной информации, полученной во время процесса обучения сети**. Он использует **градиенты выходного слоя (softmax)** по отношению к **активациям сверточного слоя**, чтобы определить важность каждого пикселя в сверточных картах признаков.

- **Прямой проход (Forward Pass):** Сначала изображение проходит через сверточную нейронную сеть и вычисляются активации на последнем сверточном слое перед классификационным слоем.
- **Вычисление градиентов:** Затем с помощью обратного распространения ошибки (backpropagation) вычисляются градиенты выходного слоя по отношению к активациям последнего сверточного слоя.
- **Комбинирование градиентов:** Градиенты усредняются по каждому каналу сверточной карты признаков, получая веса для каждого канала.
- **Взвешенная сумма признаков:** Для каждого канала сверточной карты признаков вычисляется взвешенная сумма, учитывая веса, полученные из градиентов. Это позволяет определить, какие области изображения сильнее всего влияют на окончательное решение модели.
- **Проекция на изображение:** Полученные взвешенные суммы активаций объединяются с исходным изображением, создавая визуализацию, которая подсвечивает области, на которые сеть смотрела при принятии решения.

Grad-CAM позволяет интерпретировать, какие части изображения были наиболее значимыми для классификации, обеспечивая интерпретируемость сверточных нейронных сетей и помогая понять их принятие решений.



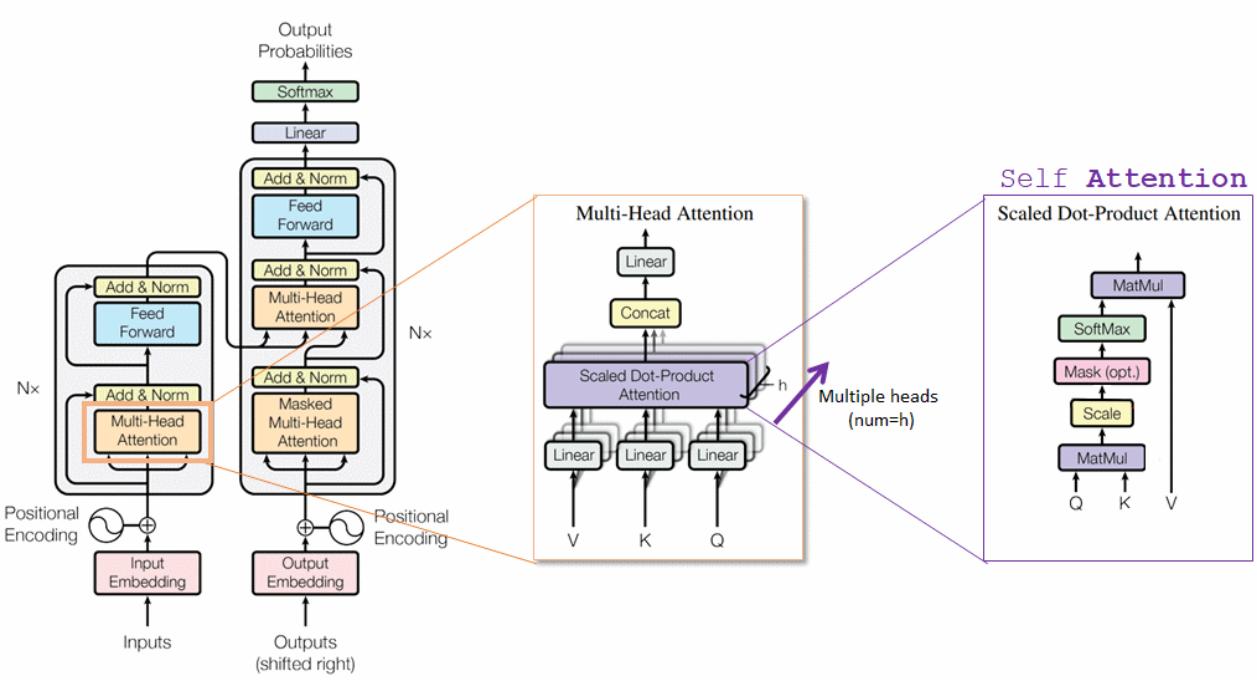
**Примеры работы ResNet (визуализация Grad-CAM) в разных конфигурациях, в том числе с Convolutional Block Attention Module**

Ссылки:

- <https://glassboxmedicine.com/2020/05/29/grad-cam-visual-explanations-from-deep-networks/> (<https://glassboxmedicine.com/2020/05/29/grad-cam-visual-explanations-from-deep-networks/>).

## Применение логики Transformer в CV

Архитектура Transformer и блоки Attention

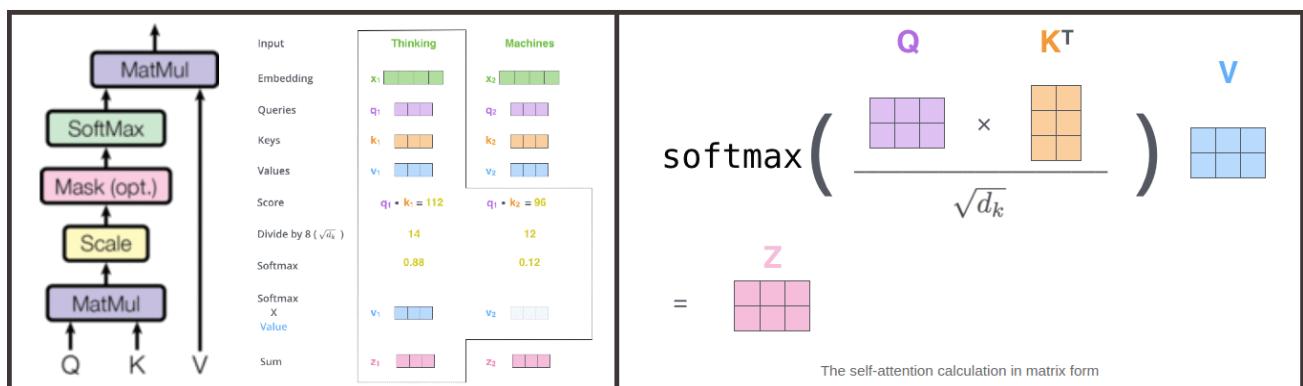


### Архитектура transformer и блок self-attention

На первом шаге вычисляются тензоры Query, Key и Value. Для этого тензор X умножается на обученные тензоры, обозначенные как  $WQ$ ,  $WK$ ,  $WV$ .

$$\begin{array}{ccc}
 X & W^Q & Q \\
 \begin{matrix} \text{green grid} \end{matrix} & \times & \begin{matrix} \text{purple grid} \end{matrix} = \begin{matrix} \text{purple grid} \end{matrix} \\
 \\ 
 X & W^K & K \\
 \begin{matrix} \text{green grid} \end{matrix} & \times & \begin{matrix} \text{orange grid} \end{matrix} = \begin{matrix} \text{orange grid} \end{matrix} \\
 \\ 
 X & W^V & V \\
 \begin{matrix} \text{green grid} \end{matrix} & \times & \begin{matrix} \text{blue grid} \end{matrix} = \begin{matrix} \text{blue grid} \end{matrix}
 \end{array}$$

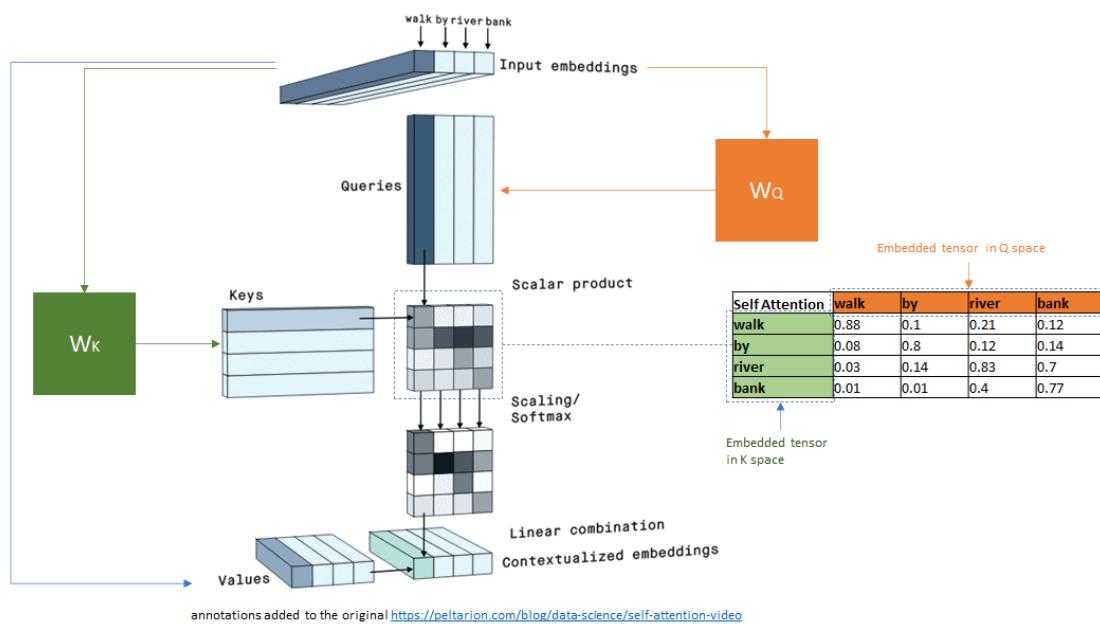
### Подготовка векторов для self-attention



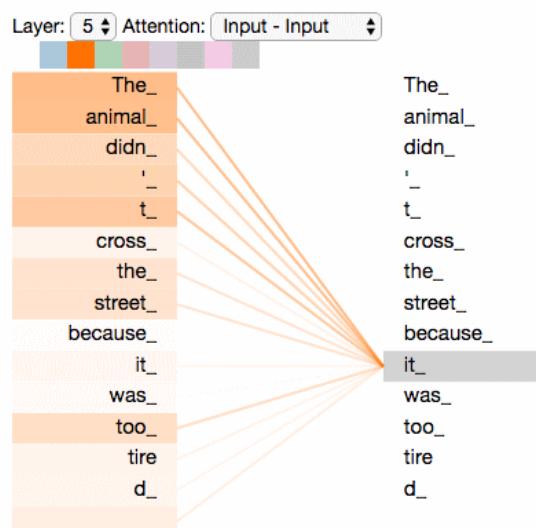
### Рассчеты в модуле self-attention

The power of embedding (transfer/project into another tensor space)

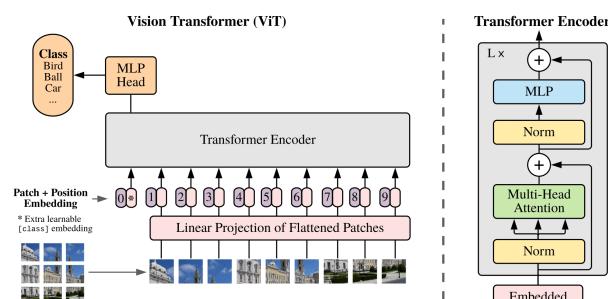
The same input sentence (sequence of word+positional embeddings) is projected to 2 different vector tensor spaces Q, K.  
Each W (W<sub>Q</sub>, W<sub>K</sub>) is a trained model that generates a contextual embedding in a different space.



### Результаты self-attention как матрица коэффициентов внимания для слов предложения.



### Пример результатов для self-attention



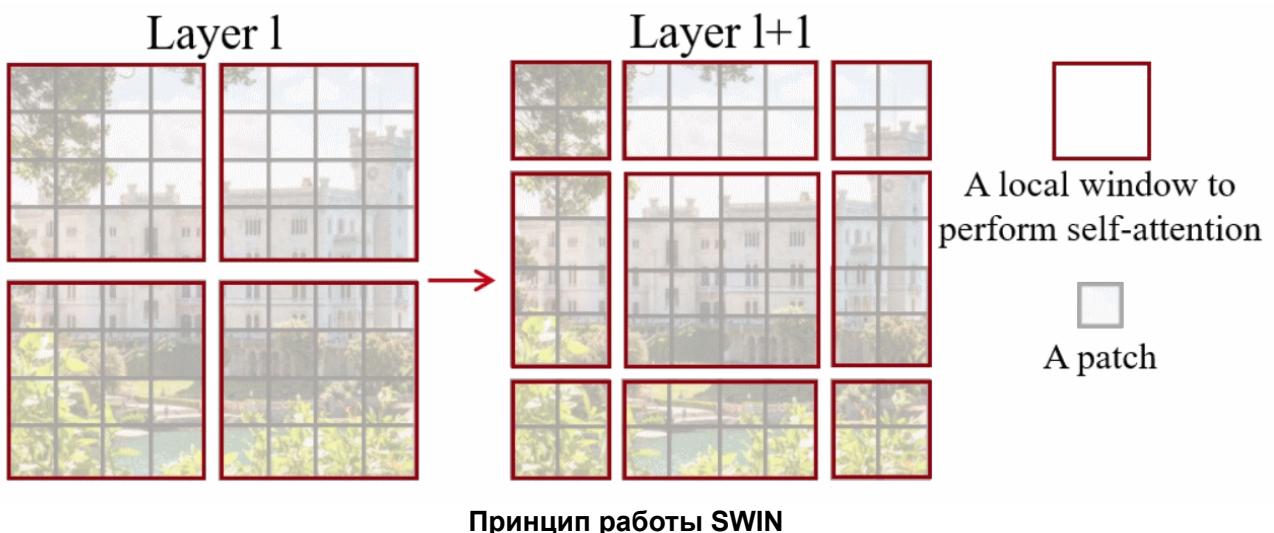
### SWIN - (Shifted) Window Attention

Проблемой Multi-Head Attention является его квадратичная сложность, что вызывает много проблем с вычислительной сложностью при применении на картинках высокого разрешения. Решение представленное авторами Swin Transformer — для каждого токена считать Attention не со всеми другими токенами, а только с находящимися в некотором окне фиксированного размера (Window Multi-Head Attention). Если размерность токенов —  $C$ , а размер окна —  $M \times M$ , то сложности для Multi-Head Self Attention обычного и Window-based получаются следующие:

$$\Omega(\text{MSA}) = 4hwC^2 + 2(hw)^2C,$$

$$\Omega(\text{W-MSA}) = 4hwC^2 + 2(M)^2hwC$$

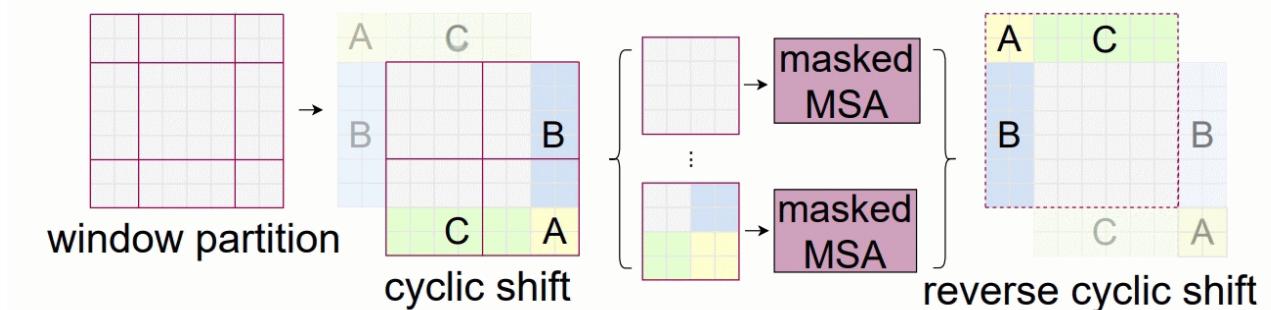
То есть Attention теперь работает за линейное по  $hw$  время. Однако такой подход уменьшает общую репрезентативную способность сети, так как токены из различных окон никак не будут взаимодействовать. Чтобы исправить ситуацию, после каждого блока с Window Multi-Head Attention добавлен аналогичный слой, со смещёнными по диагонали окнами Attention:



### Принцип работы SWIN

Это вернуло взаимодействие между токенами, оставив при этом линейную вычислительную сложность.

Как проиллюстрировано выше, сдвиг окон Attention увеличивает их количество. Это значит, что реализация этого слоя с наивным паддингом исходной «карты» признаков нулями обязет считать больше Attention-ов (9 вместо 4 в примере), чем мы посчитали бы без сдвига. Чтобы не производить лишних вычислений, авторы предложили перед подсчётом циклически сдвигать само изображение и вычислять уже маскированный Attention, чтобы исключить взаимодействие не соседних токенов. Такой подход вычислительно эффективнее наивного, так как количество вычисляемых Attention не увеличивается:



### Принцип работы SWIN

Также авторы использовали несколько другие positional embedding. Их заменили на обучаемую матрицу  $B$ , называемую relative position bias, которая прибавляется к произведению query и key под softmax:

$$\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} + B \right) V$$

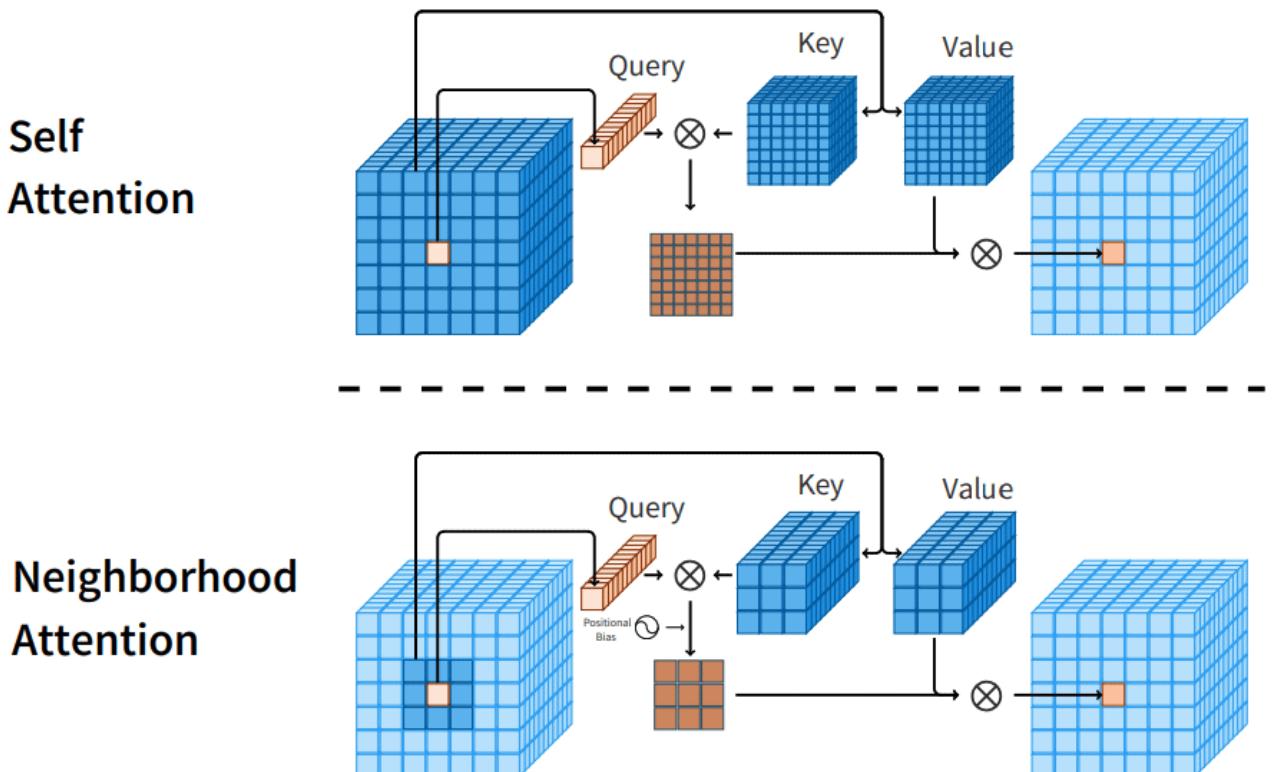
Как оказалось, подобный подод приводит к лучшему качеству.

Ссылки:

- <https://habr.com/ru/articles/599057/>
- <https://habr.com/ru/companies/otus/articles/743972/>
- <https://paperswithcode.com/paper/swin-transformer-hierarchical-vision>

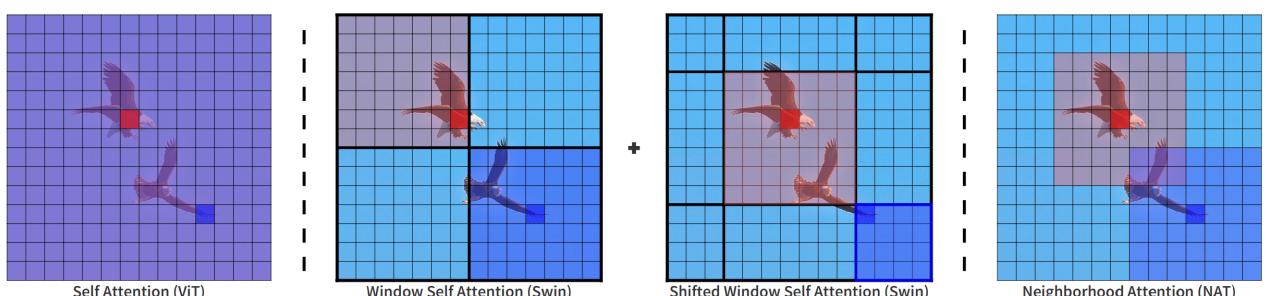
## Neighborhood Attention

Внимание к соседям (Neighborhood Attention) — это разновидность ограниченного Self-Attention, в котором рецептивное поле каждого токена ограничено ближайшими к нему соседними пикселями. Он был предложен в Neighborhood Attention Transformer в качестве альтернативы другим механизмам локального внимания.



Примеры работы ResNet (визуализация Grad-CAM) в разных конфигурациях, в том числе с Convolutional Block Attention Module

## Примеры работы механизмов внимания



Примеры работы внимания для подходов Self-Attention, Shifted Window Self Attention и Neighborhood Attention

## Позиционное кодирование

Многоголовый блок внимания не зависит от перестановок и **не может различить, предшествует ли вход другому входу в последовательности или нет**. Однако обычно в прикладных задачах **позиция (положение) очень важна для интерпретации признаков. Информация о местоположении может быть добавлена** с помощью специальных признаков во входных данных. Мы могли бы обучить эмбеддинг для каждой возможной позиции, но это не привело бы к обобщению на динамической длине входной последовательности. Лучшим вариантом является

использование **паттернов признаков**, которые сеть может идентифицировать по объектам и потенциально обобщать на более крупные последовательности. Конкретным **паттерном**, **используемым чаще всего, является синус и косинус функция различных частот**:

$$PE_{(pos,i)} = \begin{cases} \sin\left(\frac{pos}{10000^{i/d_{\text{model}}}}\right) & \text{if } i \bmod 2 = 0 \\ \cos\left(\frac{pos}{10000^{(i-1)/d_{\text{model}}}}\right) & \text{otherwise} \end{cases}$$

$PE_{(pos,i)}$  представляет кодировку позиции в  $pos$  в последовательности и скрытую размерность  $i$ . Эти значения, объединенные для всех скрытых измерений, добавляются к исходным входным объектам и представляют информацию о местоположении (см визуализацию архитектуры ниже).

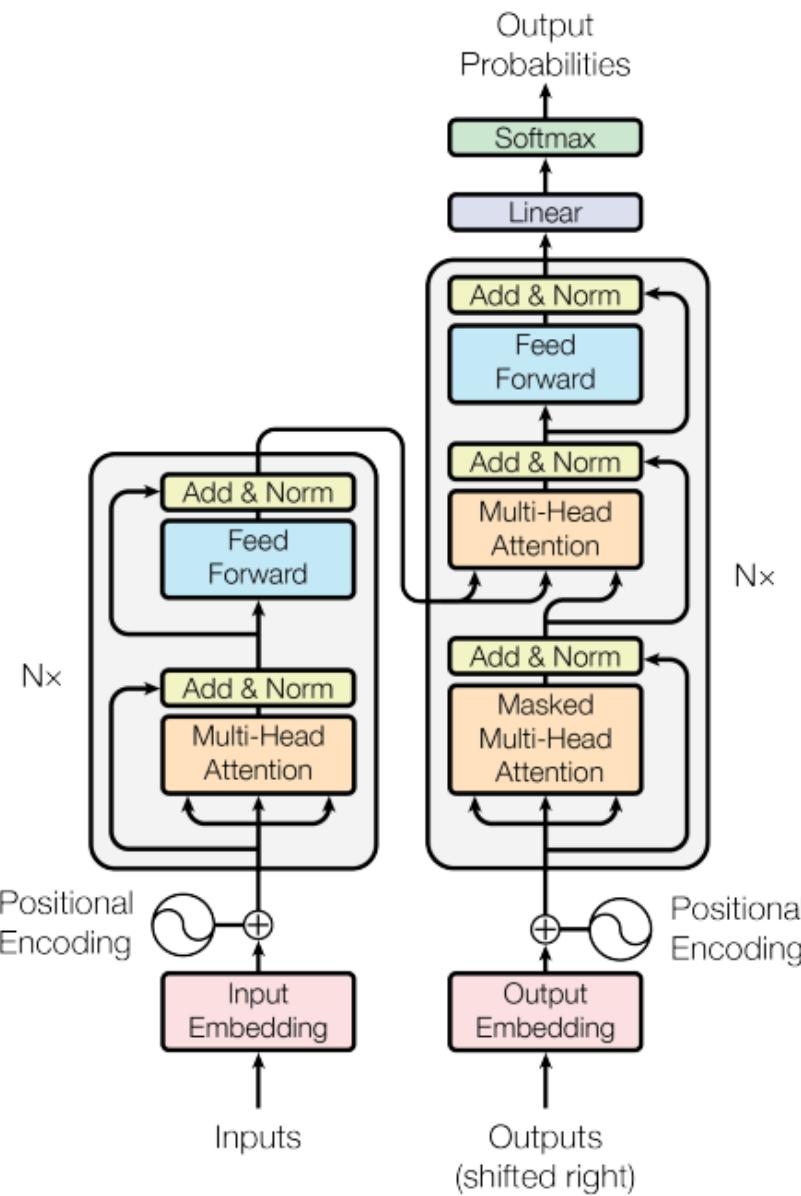


Figure 1: The Transformer - model architecture.

Мы различаем четные ( $i \bmod 2 = 0$ ) и нечетные ( $i \bmod 2 = 1$ ) скрытые размерности, где мы применяем синус или косинус соответственно. Идея, лежащая в основе этой кодировки, заключается в том, что можно представить  $PE_{(pos+k,:)}$  как линейную функцию из  $PE_{(pos,:)}$ , что могло бы позволить модели легко отслеживать относительные положения. Длины волн в различных измерениях варьируются от  $2\pi$  до  $10000 \cdot 2\pi$ .

Позиционное кодирование (Positional Encoding) также является одним из главных базовых механизмов многокомпонентных систем распознавания (обычно в Transformers). Ниже приведена имплементация механизма из руководства PyTorch Transformers в NLP с некоторыми исправлениями:

```
In [ ]: class PositionalEncoding(nn.Module):
    def __init__(self, d_model, max_len=5000):
        """Позиционное кодирование.

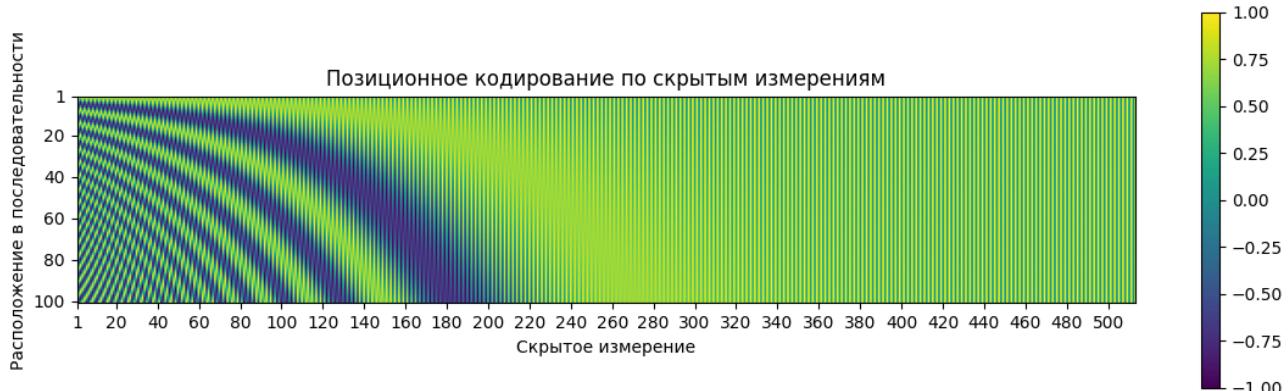
        Аргументы:
            d_model: Скрытая размерность входных данных.
            max_len: Максимальная ожидаемая длина последовательности.
        """
        super().__init__()

        # Создайте матрицу из [SeqLen, HiddenDim], представляющую позиционную кодировку
        pe = torch.zeros(max_len, d_model)
        position = torch.arange(0, max_len, dtype=torch.float).unsqueeze(1)
        div_term = torch.exp(torch.arange(0, d_model, 2).float() * (-math.log(10000.0)))
        pe[:, 0::2] = torch.sin(position * div_term)
        pe[:, 1::2] = torch.cos(position * div_term)
        pe = pe.unsqueeze(0)

        # register_buffer => Тензор, который не является параметром, но должен быть читаемым
        # Используется для тензоров, которые должны находиться на том же устройстве,
        # persistent=False сообщает PyTorch не добавлять буфер в state dict (например,
        self.register_buffer("pe", pe, persistent=False)

    def forward(self, x):
        x = x + self.pe[:, : x.size(1)]
        return x
```

## Визуализация позиционного кодирования



**Сгенерированное изображение позиционной кодировки по скрытой размерности и расположению в последовательности**

На иллюстрации каждый пиксель представляет изменение входных признаков, которое выполняется для кодирования определенной позиции.

```
In [5]: import math
import numpy as np
import tqdm.notebook
import matplotlib.pyplot as plt
import seaborn as sns
```

```
In [10]: encod_block = PositionalEncoding(d_model=512, max_len=100)
pe = encod_block.pe.squeeze().cpu().numpy()

fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(14, 4))
pos = ax.imshow(pe, extent=(1, pe.shape[1] + 1, 1, pe.shape[0] + 1))
fig.colorbar(pos, ax=ax)
ax.set_xlabel("Скрытое измерение")
ax.set_ylabel("Расположение в последовательности")
ax.set_title("Позиционное кодирование по скрытым измерениям")
ax.set_xticks([1] + [i * 20 for i in range(1, 1 + pe.shape[1] // 20)])
ax.set_yticks([1] + [i * 20 for i in range(1, 1 + pe.shape[0] // 20)])
plt.show()

-----
NameError Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_3460\688563382.py in <module>
----> 1 encod_block = PositionalEncoding(d_model=512, max_len=100)
      2 pe = encod_block.pe.squeeze().cpu().numpy()
      3
      4 fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(14, 4))
      5 pos = ax.imshow(pe, extent=(1, pe.shape[1] + 1, pe.shape[0] + 1, 1))

NameError: name 'PositionalEncoding' is not defined
```

In [7]:

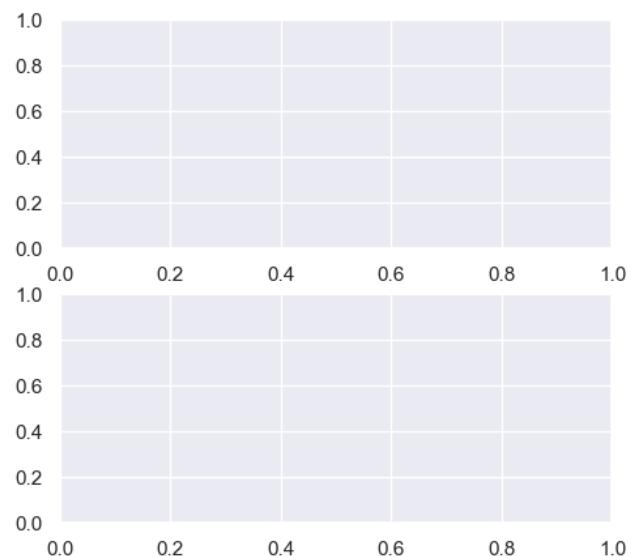
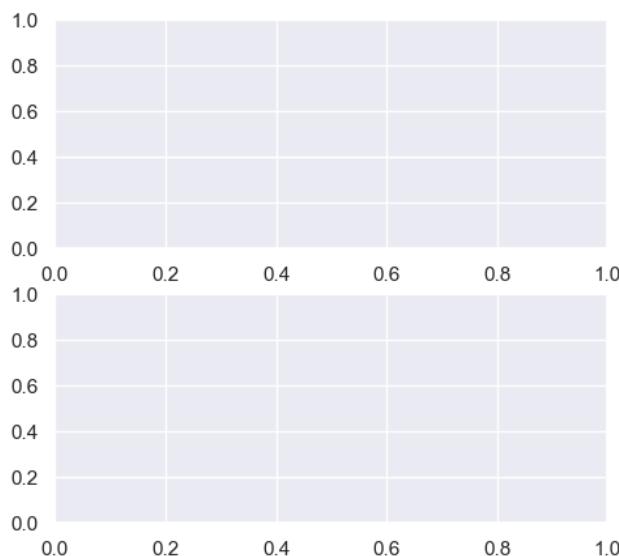
```
sns.set_theme()
fig, ax = plt.subplots(2, 2, figsize=(12, 5))
ax = [a for a_list in ax for a in a_list]
dims = [10, 11, 12, 13]
for i, d in enumerate(dims):
    ax[i].plot(np.arange(1, 17), pe[:16, d], color="C%i" % i, marker="o", markersize=6, markeredgewidth=2, markeredgecolor="black")
    ax[i].set_title("Кодирование в скрытом измерении %d" % (d + 1))
    ax[i].set_xlabel("Расположение в последовательности", fontsize=10)
    ax[i].set_ylabel("Позиционное кодирование", fontsize=10)
    ax[i].set_xticks(np.arange(1, 17))
    ax[i].tick_params(axis="both", which="major", labelsize=10)
    ax[i].tick_params(axis="both", which="minor", labelsize=8)
    ax[i].set_xlim(-1.2, 1.2)
fig.subplots_adjust(hspace=0.8)
sns.reset_orig()
plt.show()
```

NameError

Traceback (most recent call last)

```
~\AppData\Local\Temp\ipykernel_3460\1734999700.py in <module>
      4 dims = [10, 11, 12, 13]
      5 for i, d in enumerate(dims):
----> 6     ax[i].plot(np.arange(1, 17), pe[:16, d], color="C%i" % i, marker="o", ma
      7         rkersize=6, markeredgewidth=2, markeredgecolor="black")
      8         ax[i].set_title("Кодирование в скрытом измерении %d" % (d + 1))
      9         ax[i].set_xlabel("Расположение в последовательности", fontsize=10)
```

NameError: name 'pe' is not defined



## Спасибо за внимание!

### Технический раздел:

next **Q:** qs line

next **A:** an line

next **Note:** an line

next **Def:** df line

next **Ex:** ex line

next + pl line  
next - mn line  
next ± plmn line  
next ➔ hn line

---

---

```
In [1]: from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Convolution2D, MaxPooling2D
from keras.utils import np_utils
```

Using TensorFlow backend.

C:\ProgramData\Anaconda3\lib\site-packages\h5py\\_\_init\_\_.py:36: FutureWarning: Conversion of the second argument of issubdtype from `float` to `np.floating` is deprecated. In future, it will be treated as `np.float64 == np.dtype(float).type`.  
from .\_conv import register\_converters as \_register\_converters

```
In [2]: # загружаем данные MNIST
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

```
In [3]: # преобразуем каждую картинку в двумерный массив
batch_size, img_rows, img_cols = 64, 28, 28
X_train = X_train.reshape(X_train.shape[0], img_rows, img_cols, 1)
X_test = X_test.reshape(X_test.shape[0], img_rows, img_cols, 1)
input_shape = (img_rows, img_cols, 1)
```

```
In [4]: # приводим данные к типу float32 и нормализуем их от 0 до 1
X_train = X_train.astype("float32")
X_test = X_test.astype("float32")
X_train /= 255
X_test /= 255
```

```
In [5]: # переводим правильные ответы в one-hot представление
Y_train = np_utils.to_categorical(y_train, 10)
Y_test = np_utils.to_categorical(y_test, 10)
```

```
In [6]: # инициализируем модель  
model = Sequential()
```

```
# добавляем сверточные слои  
model.add(Convolution2D(32, 5, 5, border_mode="same", input_shape=input_shape))  
model.add(Activation("relu"))  
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), border_mode="same"))  
model.add(Convolution2D(64, 5, 5, border_mode="same", input_shape=input_shape))  
model.add(Activation("relu"))  
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), border_mode="same"))
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:5: UserWarning: Update your `Conv2D` call to the Keras 2 API: `Conv2D(32, (5, 5), input_shape=(28, 28, 1..., padding="same")`  
"""
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:7: UserWarning: Update your `MaxPooling2D` call to the Keras 2 API: `MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding="same")`
```

```
import sys
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:8: UserWarning: Update your `Conv2D` call to the Keras 2 API: `Conv2D(64, (5, 5), input_shape=(28, 28, 1..., padding="same")`
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:10: UserWarning: Update your `MaxPooling2D` call to the Keras 2 API: `MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding="same")`
```

```
# Remove the CWD from sys.path while we load stuff.
```

```
In [7]: # добавляем полносвязные слои
```

```
model.add(Flatten())  
model.add(Dense(1024))  
model.add(Activation("relu"))  
model.add(Dropout(0.5))  
model.add(Dense(10))  
model.add(Activation("softmax"))
```

```
WARNING:tensorflow:From C:\ProgramData\Anaconda3\lib\site-packages\keras-2.0.2-py3.6.egg\keras\backend\tensorflow_backend.py:1062: calling reduce_prod (from tensorflow.python.ops.math_ops) with keep_dims is deprecated and will be removed in a future version.
```

```
Instructions for updating:
```

```
keep_dims is deprecated, use keepdims instead
```

In [8]: # компилируем и запускаем обучение

```
model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
model.fit(X_train, Y_train, batch_size=batch_size, nb_epoch=10, verbose=1, validation_data=(X_test, Y_test), verbose=0)

# выводим результаты
print("Test score: %f" % score[0])
print("Test accuracy: %f" % score[1])
```

WARNING:tensorflow:From C:\ProgramData\Anaconda3\lib\site-packages\keras-2.0.2-py3.6.egg\keras\backend\tensorflow\_backend.py:2548: calling reduce\_sum (from tensorflow.python.ops.math\_ops) with keep\_dims is deprecated and will be removed in a future version.

Instructions for updating:

keep\_dims is deprecated, use keepdims instead

WARNING:tensorflow:From C:\ProgramData\Anaconda3\lib\site-packages\keras-2.0.2-py3.6.egg\keras\backend\tensorflow\_backend.py:1123: calling reduce\_mean (from tensorflow.python.ops.math\_ops) with keep\_dims is deprecated and will be removed in a future version.

Instructions for updating:

keep\_dims is deprecated, use keepdims instead

```
C:\ProgramData\Anaconda3\lib\site-packages\keras-2.0.2-py3.6.egg\keras\models.py:826: UserWarning: The `nb_epoch` argument in `fit` has been renamed `epochs`.
  warnings.warn('The `nb_epoch` argument in `fit` is deprecated and will be removed in a future version. Please use `epochs` instead.')
```

Train on 60000 samples, validate on 10000 samples

Epoch 1/10

```
60000/60000 [=====] - 26s - loss: 0.1243 - acc: 0.9619 -
```

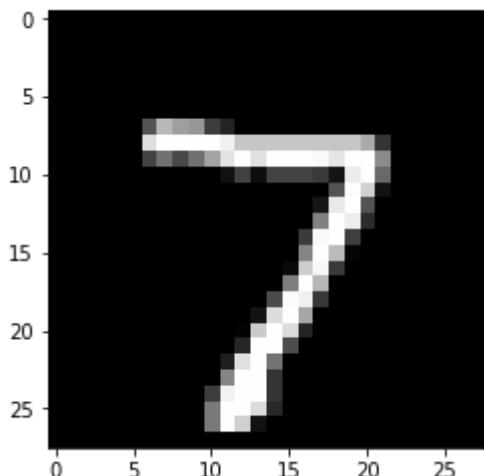
In [9]: from keras.preprocessing import image

```
import numpy as np
import matplotlib.pyplot as plt
from keras import models
```

In [10]: viz\_fig = X\_test[0]

```
pixels = viz_fig.reshape((28, 28))
print(viz_fig.shape, pixels.shape)
plt.imshow(pixels, cmap='gray')
plt.show()
```

(28, 28, 1) (28, 28)



```
In [11]: model.summary()
```

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 28, 28, 32)	832
activation_1 (Activation)	(None, 28, 28, 32)	0
max_pooling2d_1 (MaxPooling2D)	(None, 14, 14, 32)	0
conv2d_2 (Conv2D)	(None, 14, 14, 64)	51264
activation_2 (Activation)	(None, 14, 14, 64)	0
max_pooling2d_2 (MaxPooling2D)	(None, 7, 7, 64)	0
flatten_1 (Flatten)	(None, 3136)	0
dense_1 (Dense)	(None, 1024)	3212288
activation_3 (Activation)	(None, 1024)	0
dropout_1 (Dropout)	(None, 1024)	0
dense_2 (Dense)	(None, 10)	10250
activation_4 (Activation)	(None, 10)	0
<hr/>		
Total params: 3,274,634.0		
Trainable params: 3,274,634.0		
Non-trainable params: 0.0		

```
In [13]: # Extracts the outputs of the top 4 layers:
```

```
layer_outputs = [layer.output for layer in model.layers[:4]]  
# Creates a model that will return these outputs, given the model input:  
activation_model = models.Model(inputs=model.input, outputs=layer_outputs)
```

```
In [14]: viz_fig_v = viz_fig.copy()  
viz_fig_v = np.expand_dims(viz_fig_v, axis=0)  
viz_fig_v = viz_fig_v.astype("float32")  
viz_fig_v /= 255  
  
viz_fig_v.shape, input_shape
```

```
Out[14]: ((1, 28, 28, 1), (28, 28, 1))
```

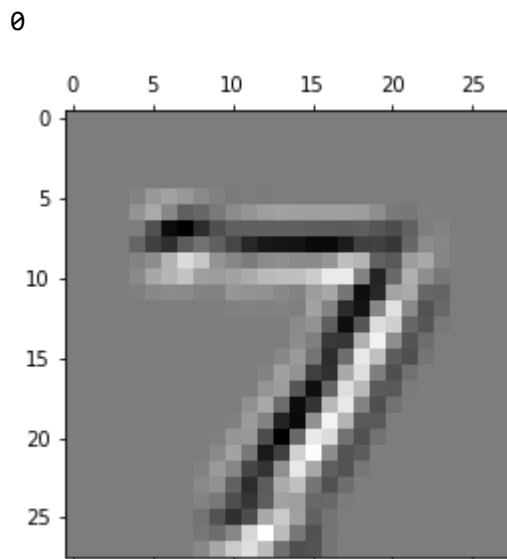
```
In [15]: # This will return a list of 5 Numpy arrays:
```

```
# one array per layer activation  
activations = activation_model.predict(viz_fig_v)
```

```
In [19]: first_layer_activation = activations[0]  
print(first_layer_activation.shape)
```

```
(1, 28, 28, 32)
```

```
In [20]: for i in range(32):
    print(i)
    plt.matshow(first_layer_activation[0, :, :, i], cmap='gray')
    plt.show()
```



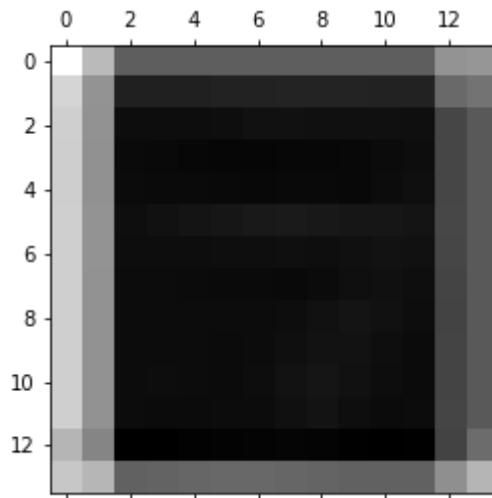
1

```
In [16]: second_layer_activation = activations[3]
print(second_layer_activation.shape)
```

(1, 14, 14, 64)

```
In [17]: for i in range(64):
    print(i)
    plt.matshow(second_layer_activation[0, :, :, i], cmap='gray')
    plt.show()
```

0



1

```
In [ ]:
```

