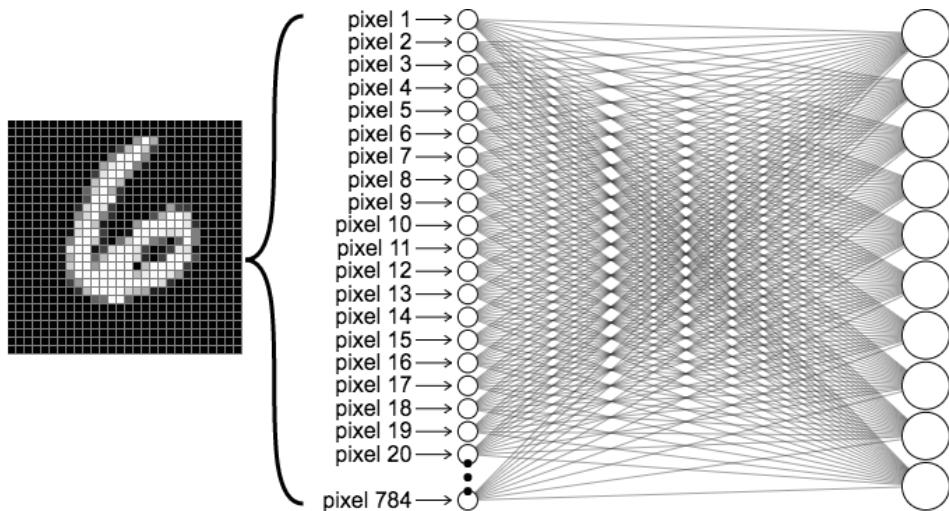


# **Компьютерное зрение**

# **Работа с изображениями в нейронных сетях**

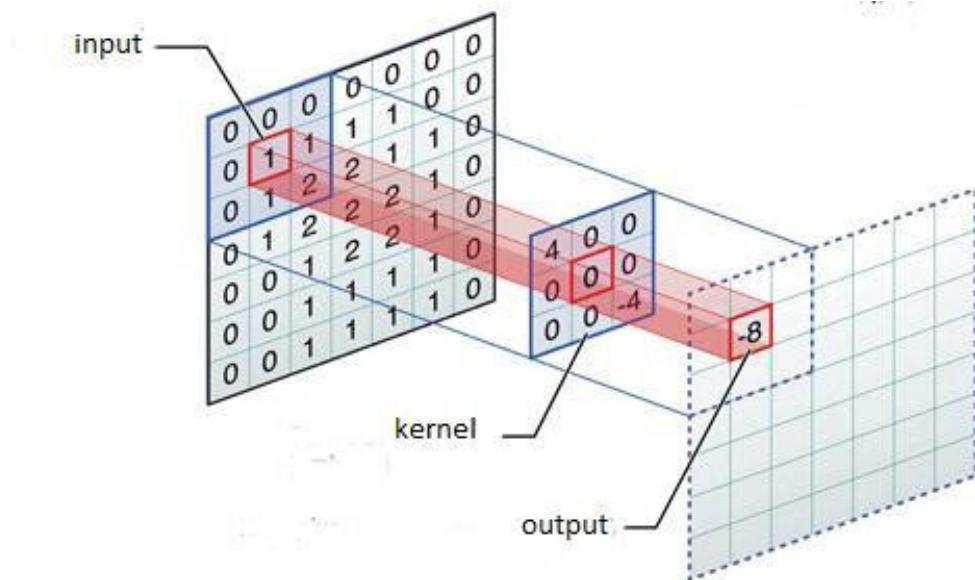
# Полносвязная сеть



Количество весов первого слоя = высота ширина каналы \* количество нейронов скрытого слоя

# Свёртка

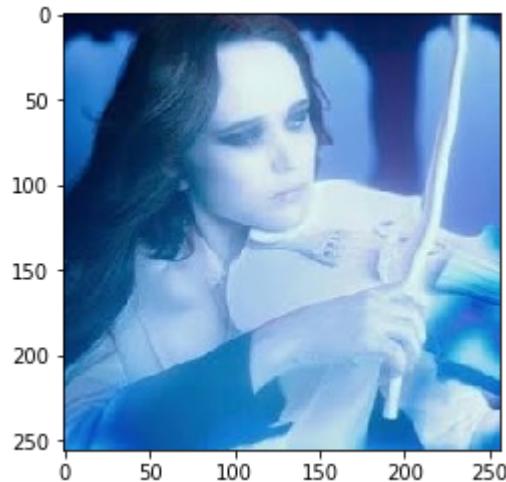
Свертка (англ. convolution) – это операция, показывающая «схожесть» одной функции с отражённой и сдвинутой копией другой. Понятие свёртки обобщается для функций, определённых на группах, а также мер.



In [1]:

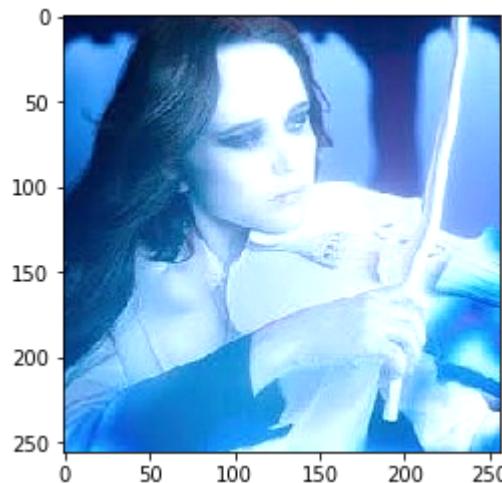
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

image = cv2.imread('data/Vanya.jpg')[..., ::-1]
plt.imshow(image)
plt.show()
```



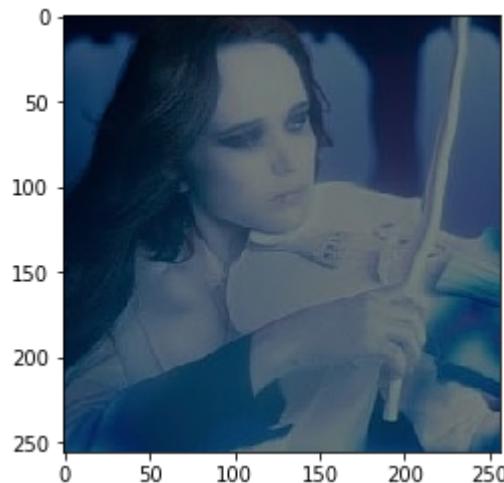
In [2]: # увеличение чёткости

```
kernel = np.array([
    [-0.1, -0.1, -0.1],
    [-0.1, 2, -0.1],
    [-0.1, -0.1, -0.1],
])
img_out = cv2.filter2D(image, -1, kernel)
plt.imshow(img_out)
plt.show()
```



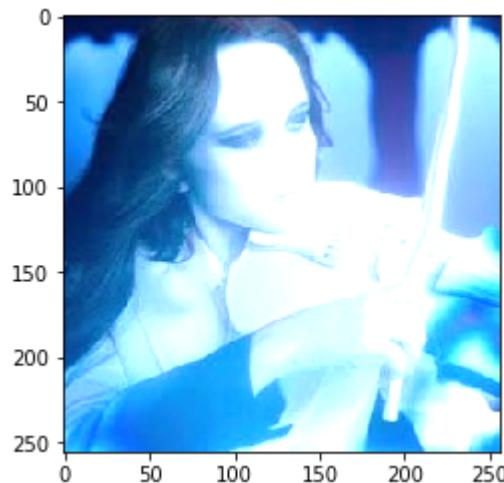
In [3]:

```
# затемнение
kernel = np.array([
    [-0.1,  0.1, -0.1],
    [ 0.1,  0.5,  0.1],
    [-0.1,  0.1, -0.1],
])
img_out = cv2.filter2D(image, -1, kernel)
plt.imshow(img_out)
plt.show()
```

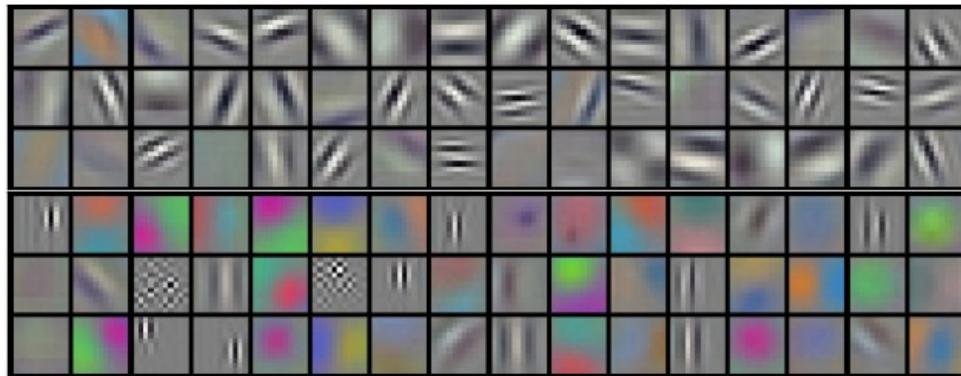


In [4]: # увеличение яркости

```
kernel = np.array([
    [-0.1,  0.2, -0.1],
    [ 0.2,   1,  0.2],
    [-0.1,  0.2, -0.1],
])
img_out = cv2.filter2D(image, -1, kernel)
plt.imshow(img_out)
plt.show()
```

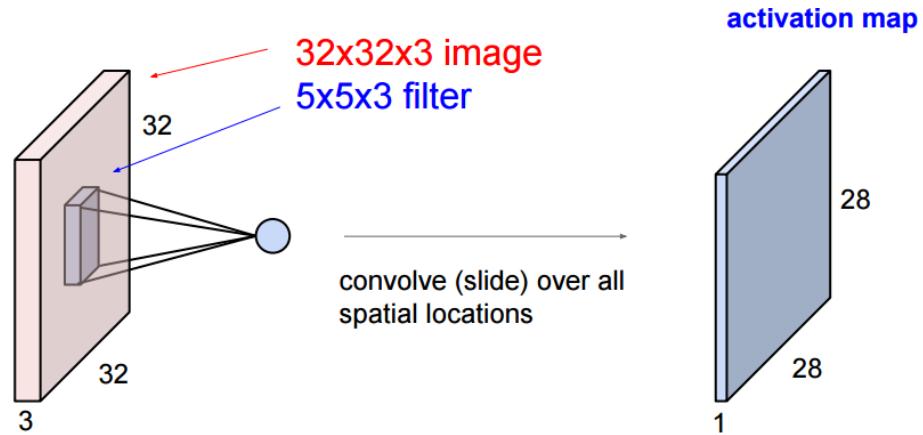


## Ядра нейронной сети

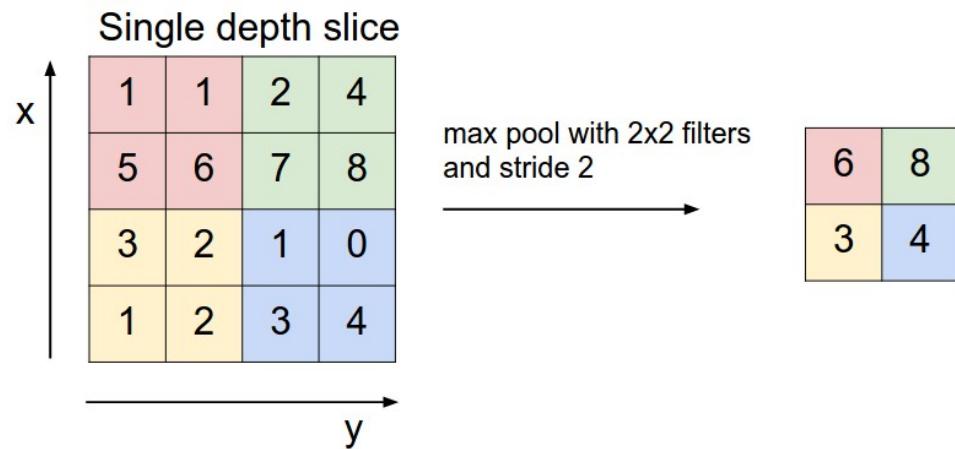


Каждый из 96 фильтров, показанных здесь, имеет размер  $11 \times 11 \times 3$

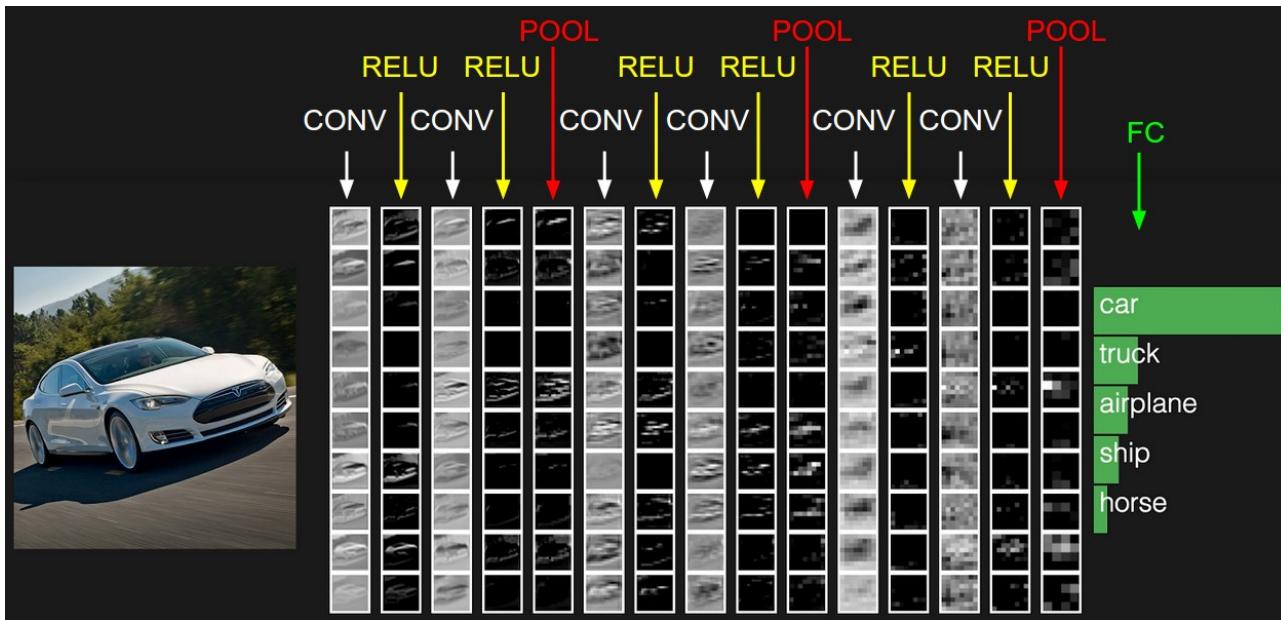
# Свёрточный слой



# Max Pooling



# Свёрточная сеть



Карты активации примера архитектуры ConvNet. Первоначальные данные представляют собой необработанные пиксели изображения (слева), а последний выход хранит оценки классов (справа).

# Padding

Input	Kernel	Output																																													
<table border="1" style="border-collapse: collapse; width: 100%;"><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td><td>1</td><td>2</td><td>0</td></tr><tr><td>0</td><td>3</td><td>4</td><td>5</td><td>0</td></tr><tr><td>0</td><td>6</td><td>7</td><td>8</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>	0	0	0	0	0	0	0	1	2	0	0	3	4	5	0	0	6	7	8	0	0	0	0	0	0	<table border="1" style="border-collapse: collapse; width: 100%;"><tr><td>0</td><td>1</td></tr><tr><td>2</td><td>3</td></tr></table>	0	1	2	3	$\times$ = <table border="1" style="border-collapse: collapse; width: 100%;"><tr><td>0</td><td>3</td><td>8</td><td>4</td></tr><tr><td>9</td><td>19</td><td>25</td><td>10</td></tr><tr><td>21</td><td>37</td><td>43</td><td>16</td></tr><tr><td>6</td><td>7</td><td>8</td><td>0</td></tr></table>	0	3	8	4	9	19	25	10	21	37	43	16	6	7	8	0
0	0	0	0	0																																											
0	0	1	2	0																																											
0	3	4	5	0																																											
0	6	7	8	0																																											
0	0	0	0	0																																											
0	1																																														
2	3																																														
0	3	8	4																																												
9	19	25	10																																												
21	37	43	16																																												
6	7	8	0																																												

## Stride

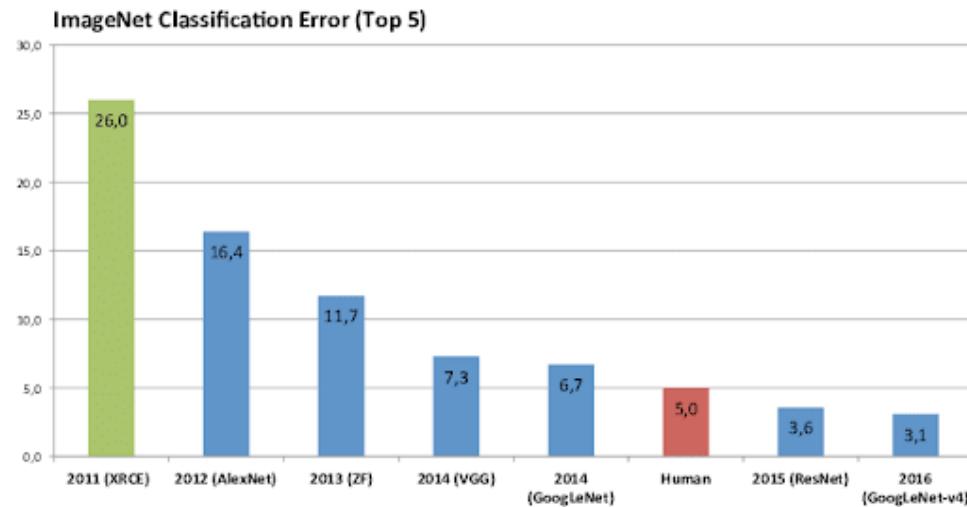
Input	Kernel	Output																													
<table border="1" style="border-collapse: collapse; width: 100%;"><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td><td>1</td><td>2</td><td>0</td></tr><tr><td>0</td><td>3</td><td>4</td><td>5</td><td>0</td></tr><tr><td>0</td><td>6</td><td>7</td><td>8</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>	0	0	0	0	0	0	0	1	2	0	0	3	4	5	0	0	6	7	8	0	0	0	0	0	0	$\ast$	<table border="1" style="border-collapse: collapse; width: 100%;"><tr><td>0</td><td>1</td></tr><tr><td>2</td><td>3</td></tr></table>	0	1	2	3
0	0	0	0	0																											
0	0	1	2	0																											
0	3	4	5	0																											
0	6	7	8	0																											
0	0	0	0	0																											
0	1																														
2	3																														
	=	<table border="1" style="border-collapse: collapse; width: 100%;"><tr><td>0</td><td>8</td></tr><tr><td>6</td><td>8</td></tr></table>	0	8	6	8																									
0	8																														
6	8																														

Шаг (stride) 3 и 2 по высоте и ширине соответственно.

# Image Net



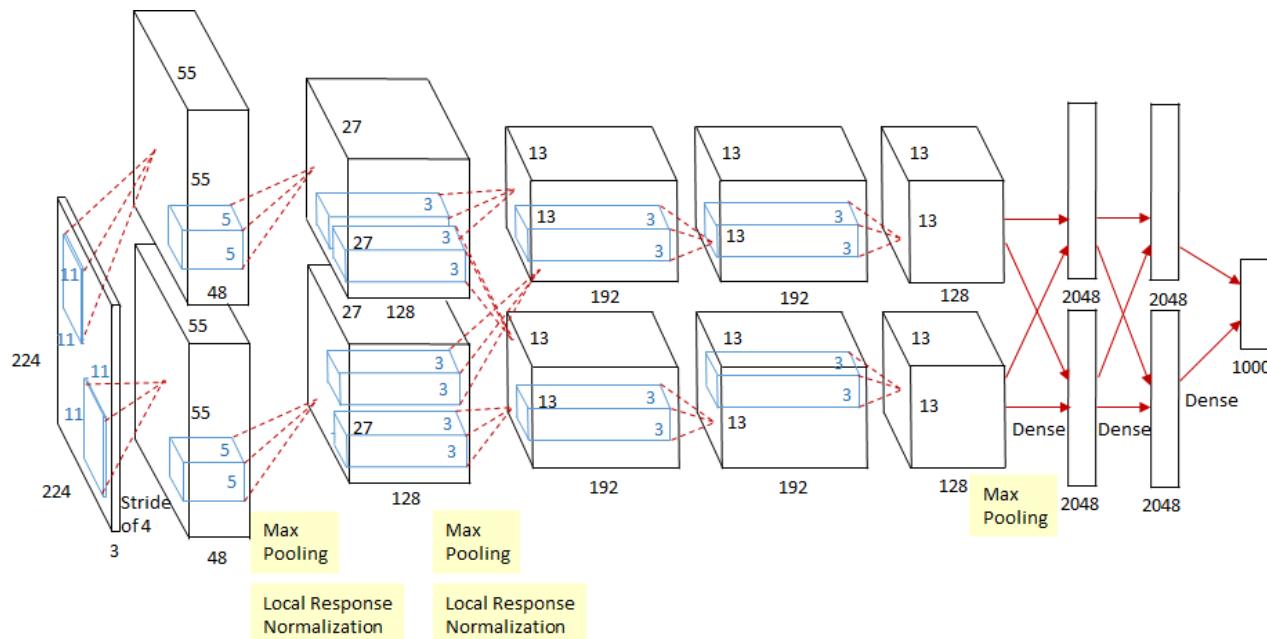
На август 2017 года в ImageNet 14 197 122 изображения, разбитых на 21 841 категорию.



# AlexNet

Архитектура AlexNet состоит из пяти свёрточных слоёв, между которыми располагаются pooling-слои и слои нормализации, а завершают нейросеть три полносвязных слоя.

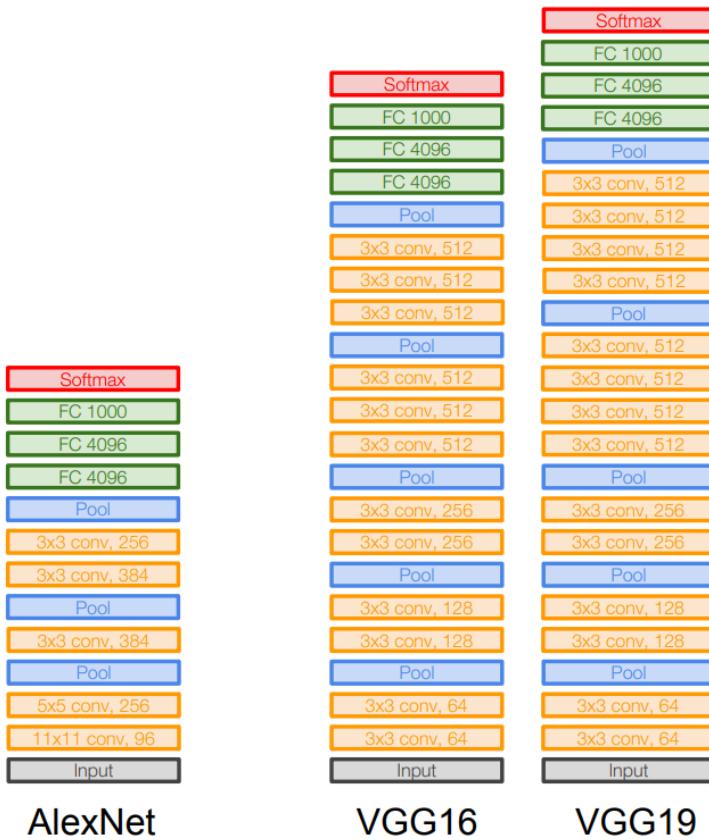
На схеме архитектуры все выходные изображения делятся на два одинаковых участка – это связано с тем, что нейросеть обучалась на старых GPU GTX580, у которых было всего 3 ГБ видеопамяти. Для обработки использовались две видеокарты, чтобы параллельно выполнять операции над двумя частями изображения.



# VGG

Основная идея VGG-архитектур – использование большего числа слоёв с фильтрами меньшего размера. Существуют версии VGG-16 и VGG-19 с 16 и 19 слоями соответственно.

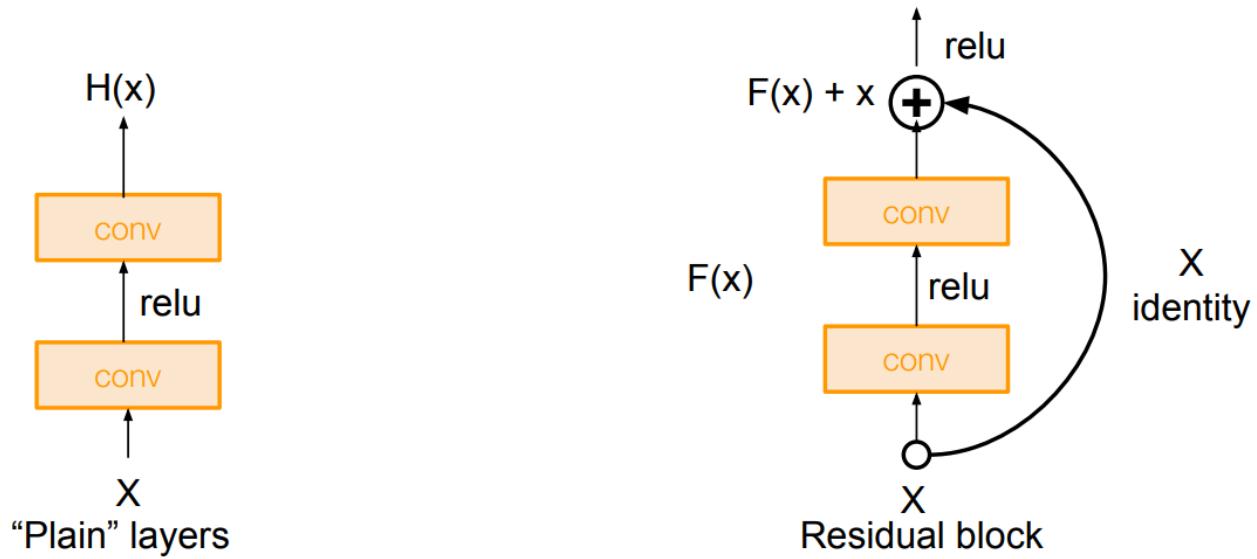
С маленькими фильтрами получается не так много параметров, но при этом мы сможем гораздо эффективнее обрабатывать их.





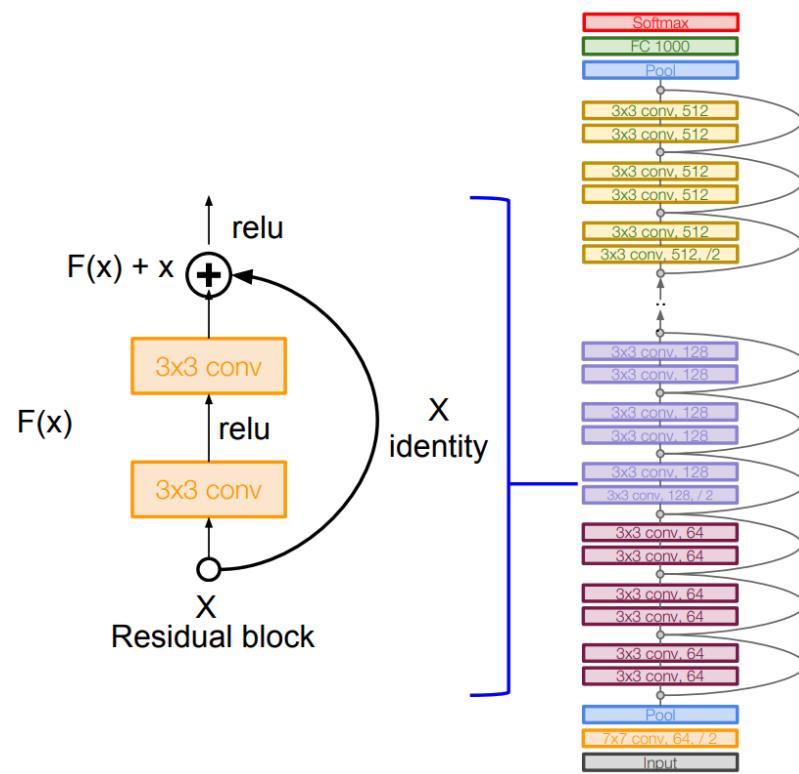
# ResNet

Создатели ResNet решили не складывать слои друг на друга для изучения отображения нужной функции напрямую, а использовать остаточные блоки, которые пытаются «подогнать» это отображение. Так ResNet стала первой остаточной нейронной сетью. Иначе говоря, она «перепрыгивает» через некоторые слои. Они больше не содержат признаков и используются для нахождения остаточной функции  $H(x) = F(x) + x$  вместо того, чтобы искать  $H(x)$  напрямую.



# ResNet

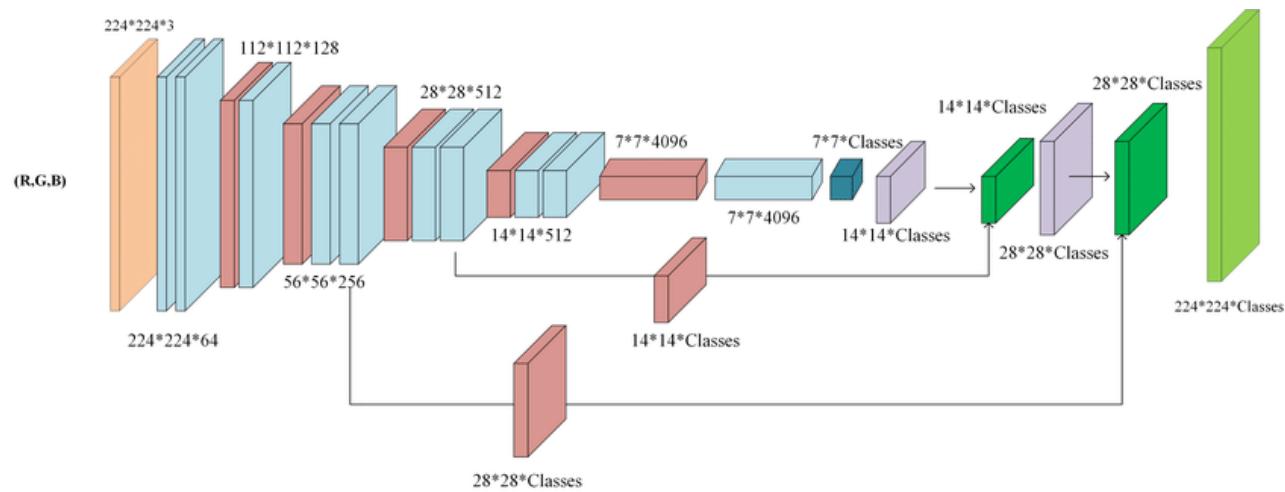
В результате экспериментов с ResNet выяснилось, что очень глубокие сети действительно можно обучить без ухудшения точности. Нейросеть достигла наименьшей ошибки в задачах классификации, которая превзошла даже человеческий результат.



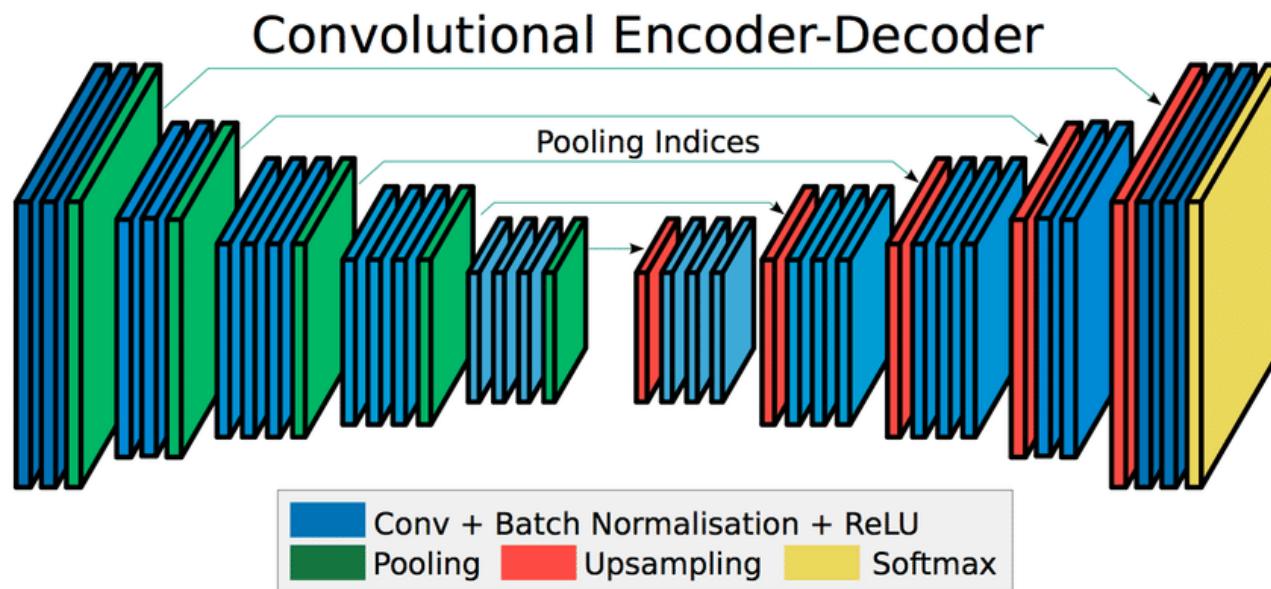
# Сегментация изображений



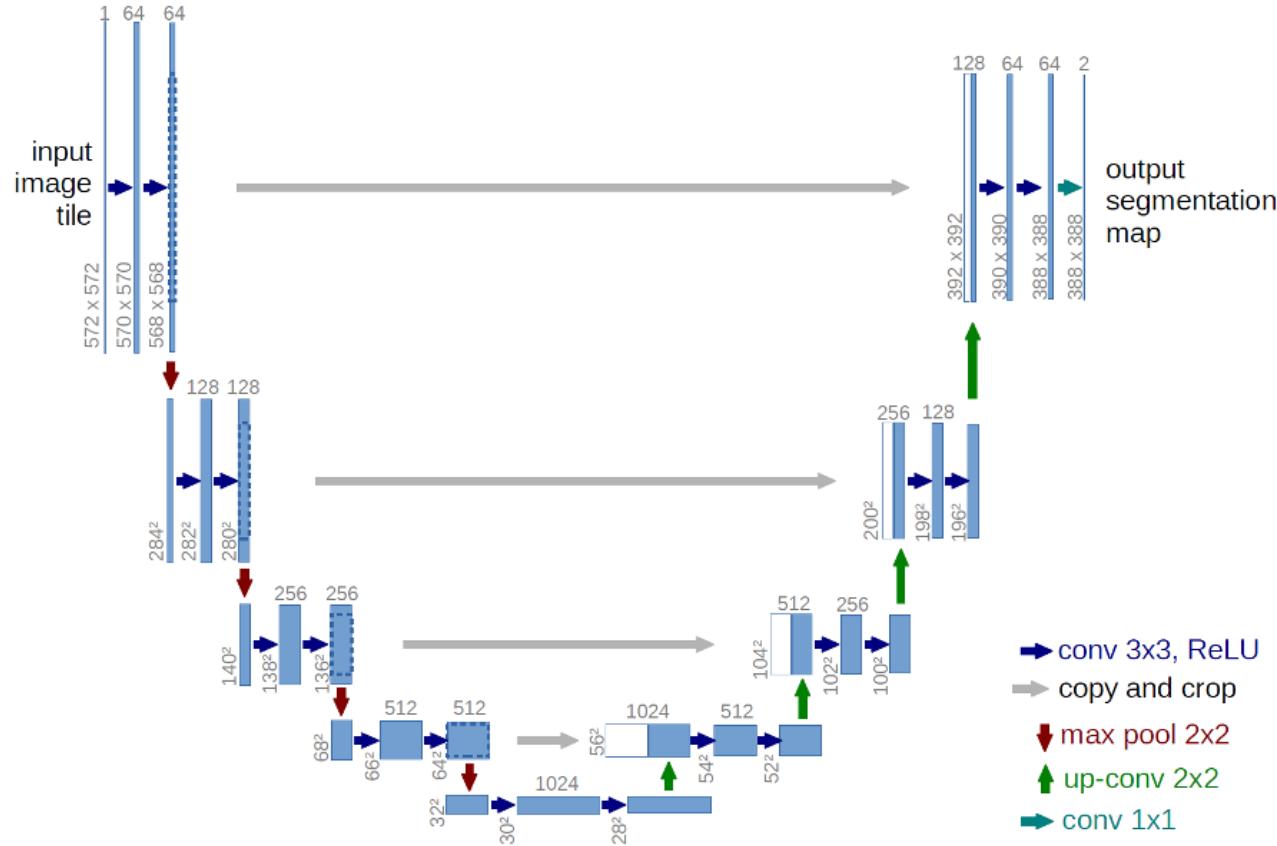
# Fully Convolutional Network



# SegNet



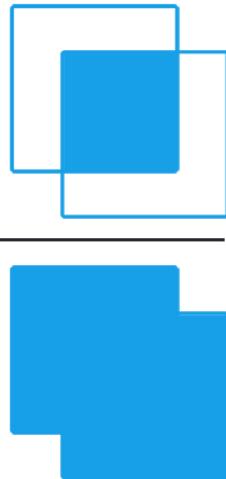
# UNet



# Метрика

Intersection over union (IoU)

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

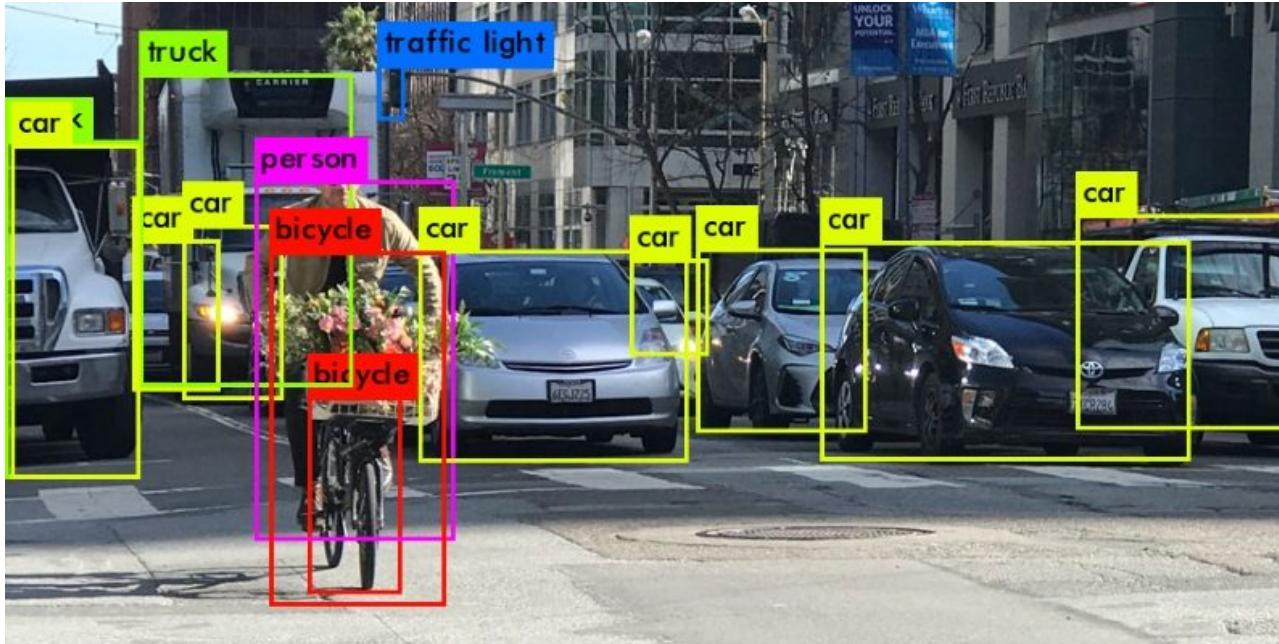


## **Функция потерь**

- Cross entropy
- Weighted cross entropy
- DICE

$$DC = 1 - \frac{2 \sum y_i p_i}{\sum y_i + \sum p_i}$$

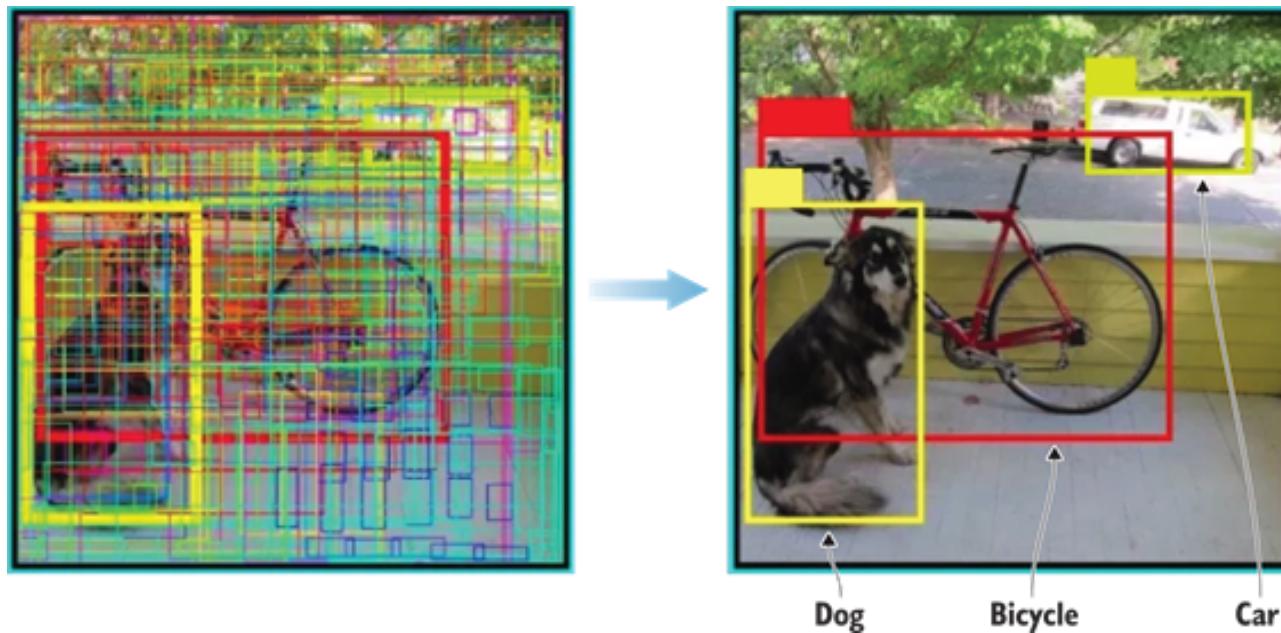
# Детекция



# Non-maximum Suppression (NMS)

Алгоритм:

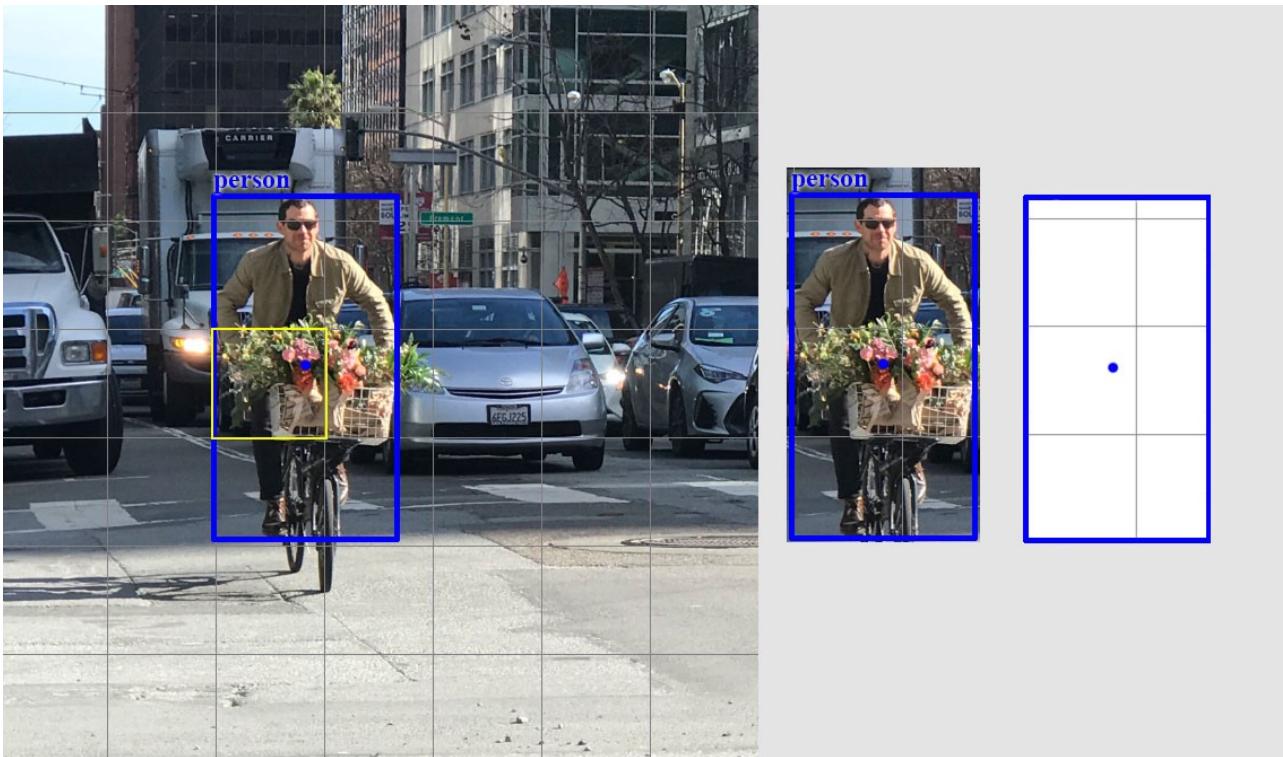
- 1) Выбрать область с наибольшей уверенностью и запомнить его
- 2) Найти IoU с оставшимися областями
- 3) Удалить области с IoU больше заданного порога
- 4) Повторять пп. 1-3, пока все области не отфильтруются



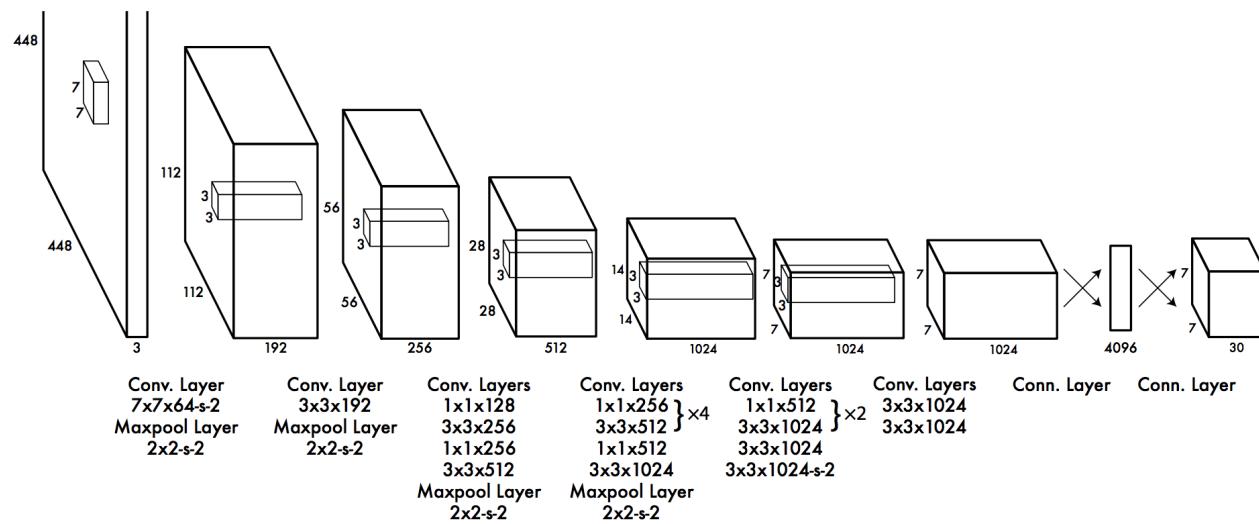
# **Single Shot Detectors**

# Yolo

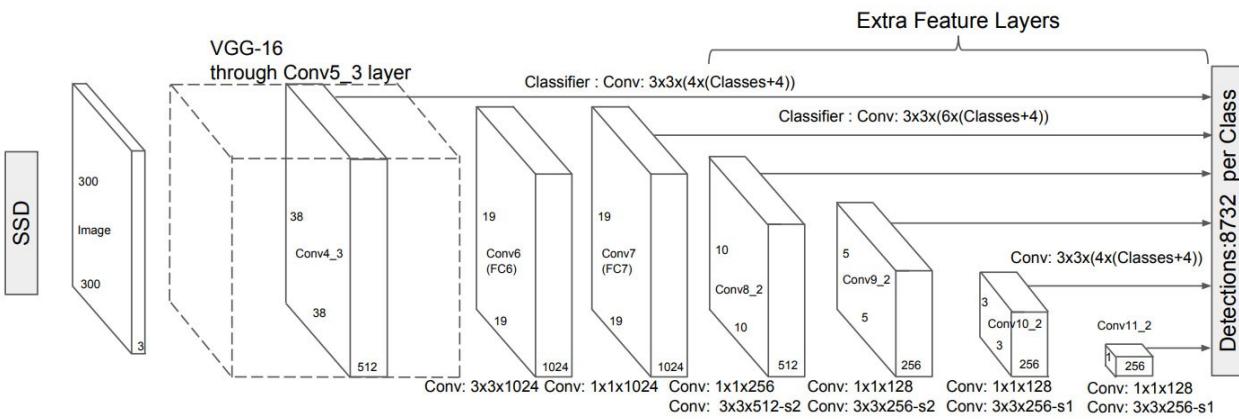
Изображение делится на части и предсказывается отдельно для каждой части



Каждая часть предсказывает координаты, уверенность и вероятность каждого из 20 классов

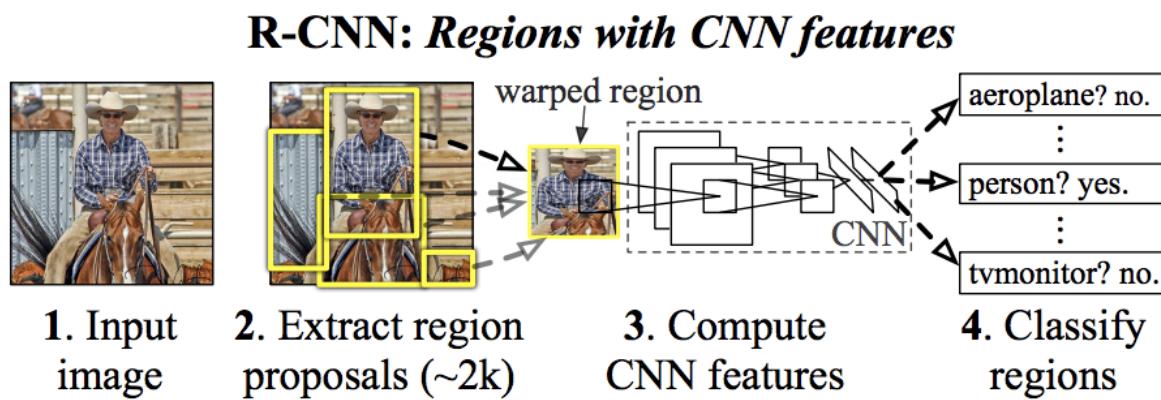


# SSD

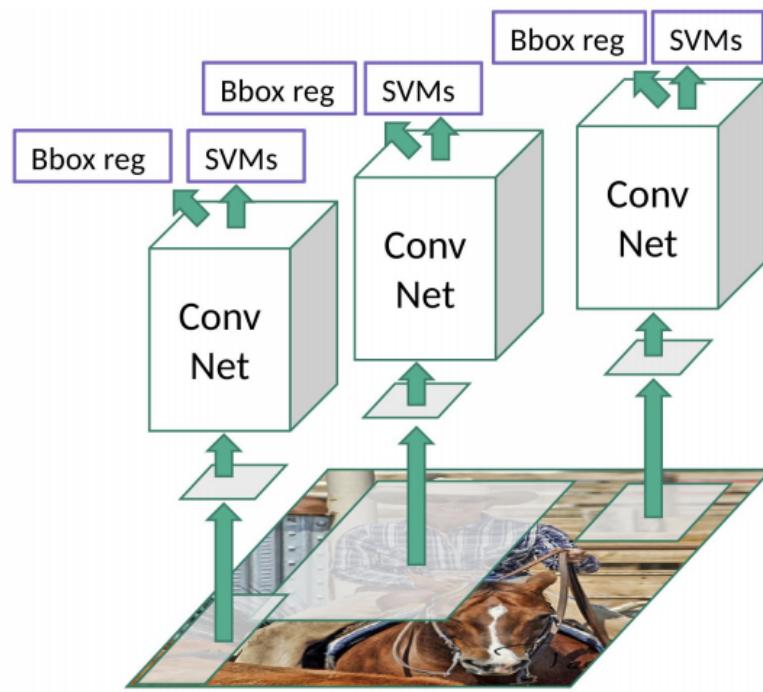


# **Two Shot Detectors**

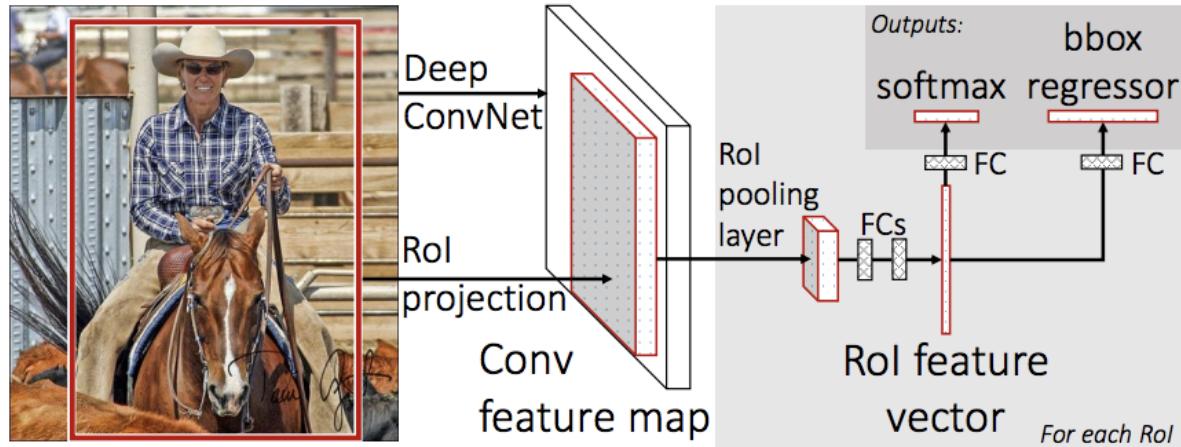
# R-CNN



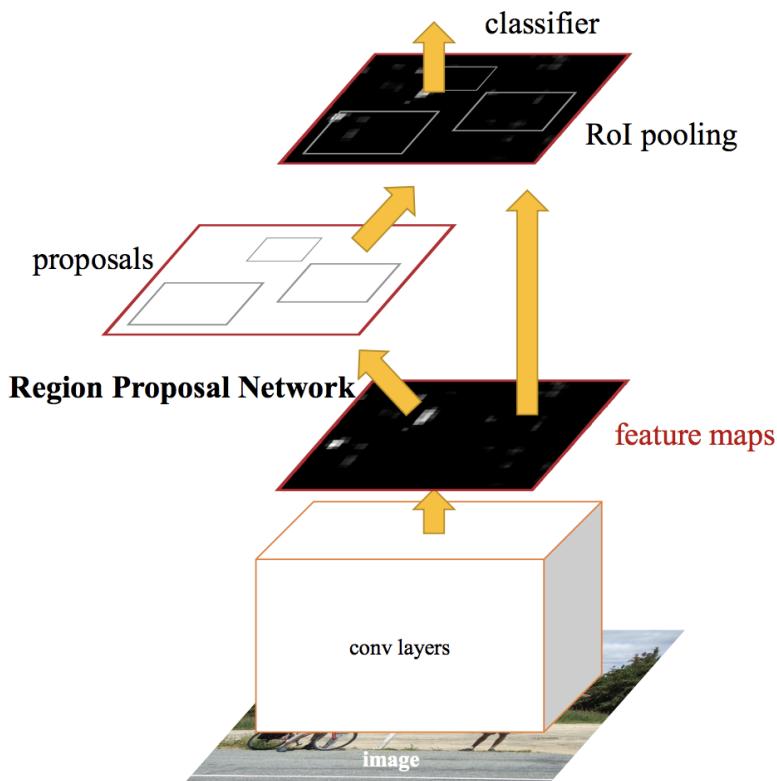
# R-CNN



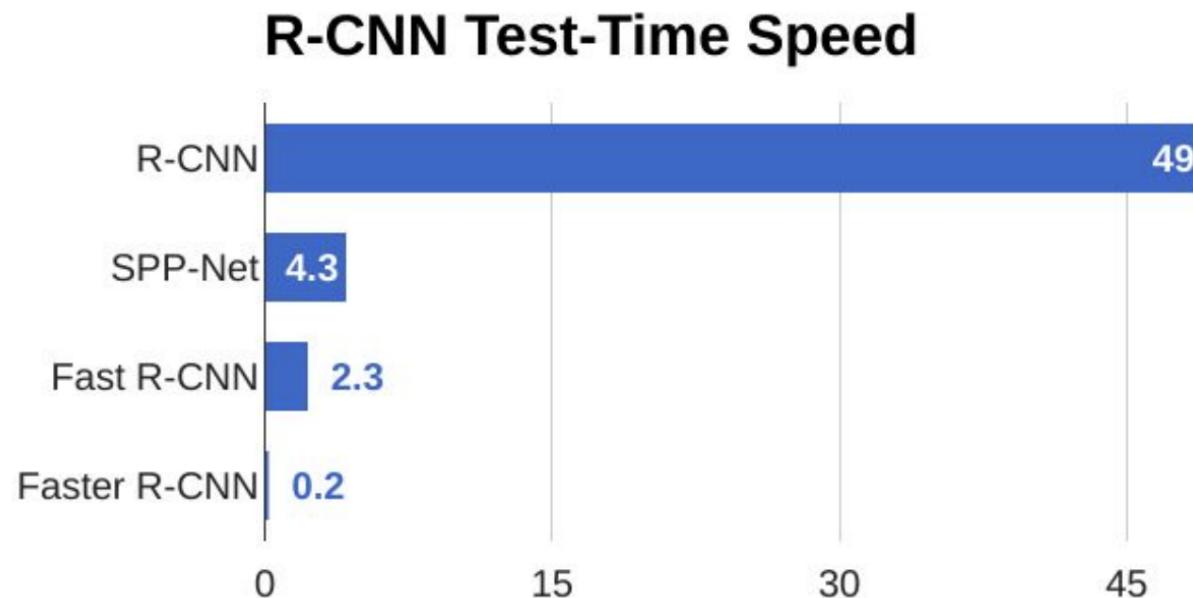
# Fast R-CNN



# Faster R-CNN



## Сравнение скоростей

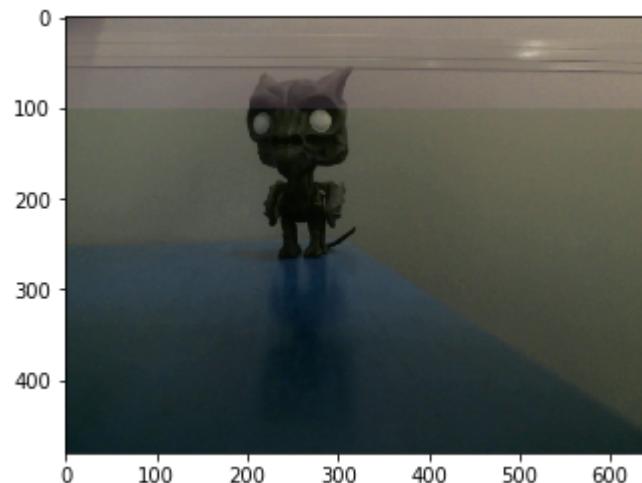


# OpenCV

In [7]:

```
# Получение видео. Объект VideoCapture позволяет работать с видеопотоком с камеры или из файла
# Для работы с файлом указывается путь к нему, для работы с вебкамерой указывается её номер, начиная с 0
cap = cv2.VideoCapture(0)
# Кадр получается с помощью метода read (возвращается флаг успешной операции и кадр)
ret, frame = cap.read()
# После работы с камерой VideoCapture освобождается
cap.release()
```

```
In [10]: # Изображение получается в формате BGR  
plt.imshow(frame)  
plt.show()
```



```
In [11]: # OpenCV позволяет выводить кадры в отдельном окне. Для этого служит функция imshow  
# На вход подаётся изображение в формате BGR  
# Первым аргументом указывается имя окна  
cv2.imshow('Frame', frame)  
# Для задержки показа используется функция waitKey  
# Аргумент указывает задержку в мс. Если 0, то ждёт нажатия любой клавиши  
# Функция waitKey возвращает код нажатой клавиши  
cv2.waitKey(0)  
# После работы с ним окно удаляется  
cv2.destroyAllWindows()
```

## Воспроизведение видео

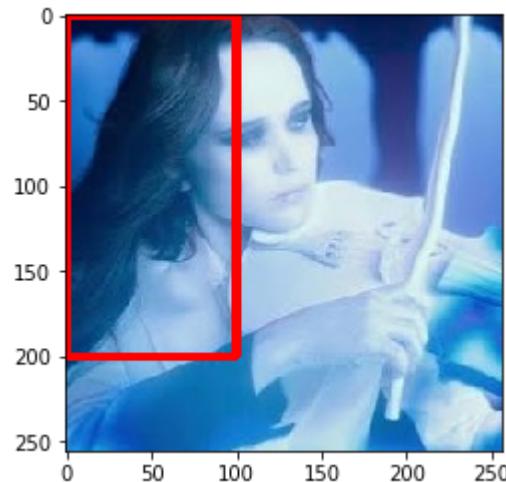
```
In [12]: cap = cv2.VideoCapture(0)
while True:
    ret, frame = cap.read()
    key = cv2.waitKey(20) & 0xff
    cv2.imshow('Frame', frame)
    if key == 27: # Esc
        break

cv2.destroyAllWindows('Frame')
cap.release()
```

# Рисование

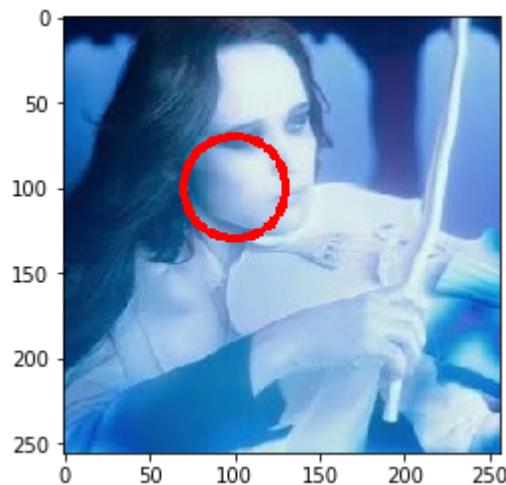
In [13]:

```
# Прямоугольник
image2 = image.copy()
cv2.rectangle(image2, (0, 0), (100, 200), (255, 0, 0), 4)
plt.imshow(image2)
plt.show()
```



In [14]:

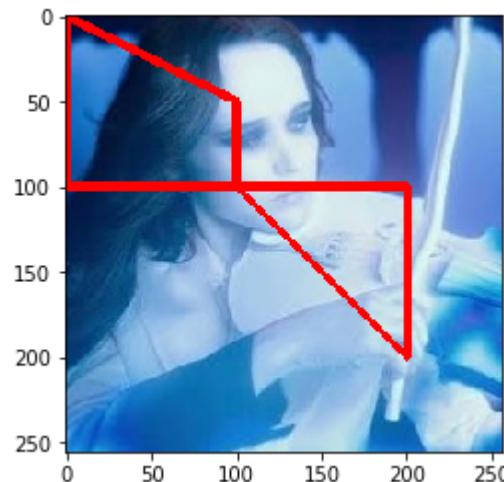
```
# Круг  
image2 = image.copy()  
cv2.circle(image2, (100, 100), 30, (255, 0, 0), 4)  
plt.imshow(image2)  
plt.show()
```



In [44]:

```
# Контур
image2 = image.copy()
contours = [
    np.array([(0, 0), (0, 100), (100, 100), (100, 50)]),
    np.array([(200, 200), (100, 100), (200, 100)]),
]
cv2.drawContours(image2, contours, -1, (255, 0, 0), 4)

plt.imshow(image2)
plt.show()
```



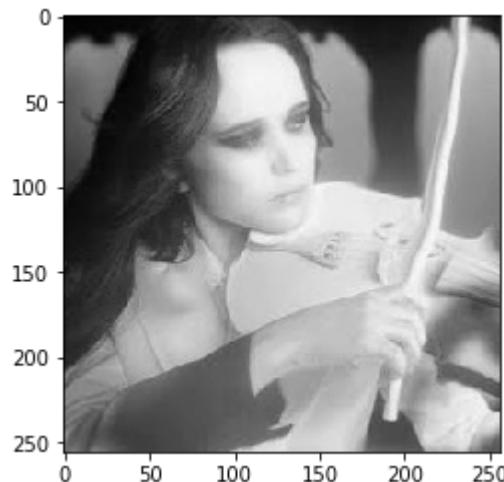
```
In [25]: # Текст  
image2 = image.copy()  
cv2.putText(image2, 'Text It', (10, 100), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255,  
0), 2)  
plt.imshow(image2)  
plt.show()
```



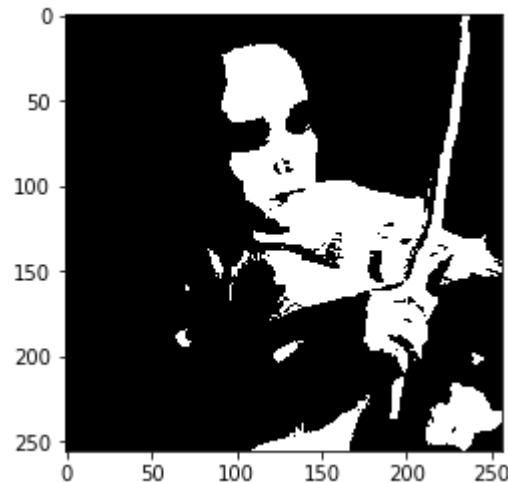
# Работа с изображением

In [29]:

```
# Перевод в градации серого
image_gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
plt.imshow(image_gray, cmap='gray')
plt.show()
```



```
In [36]: # Бинаризация  
_, thresh = cv2.threshold(image_gray, 200, 255, cv2.THRESH_BINARY)  
plt.imshow(thresh, cmap='gray')  
plt.show()
```



```
In [41]: # Поиск контуров
contours, hierarchy = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
print(len(contours), contours[2])

71 [[[133 247]]
 [[134 246]]
 [[135 247]]
 [[134 248]]]
```

# **Задания**

1. Для данных Nails segmentation объедините пары изображение-маска
2. Выведите по очереди пары с помощью OpenCV эти пары (переключение по нажатию клавиши)
3. Выделите контуры на масках и отрисуйте их на изображениях
4. Воспроизведите видеофайл с помощью OpenCV в градациях серого