онлайн-курс

# СПЕЦИАЛЬНЫЕ АРХИТЕКТУРЫ НЕЙРОННЫХ СЕТЕЙ

# 2. Свёрточные нейронные сети (CNC)

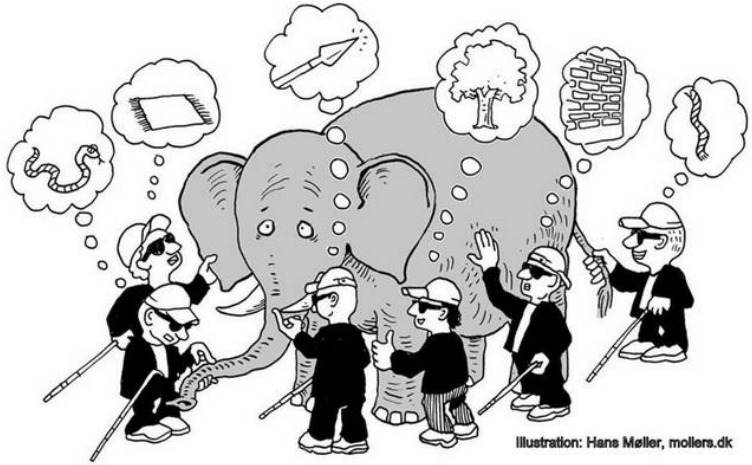
# Фильтры
# в нейронных сетях

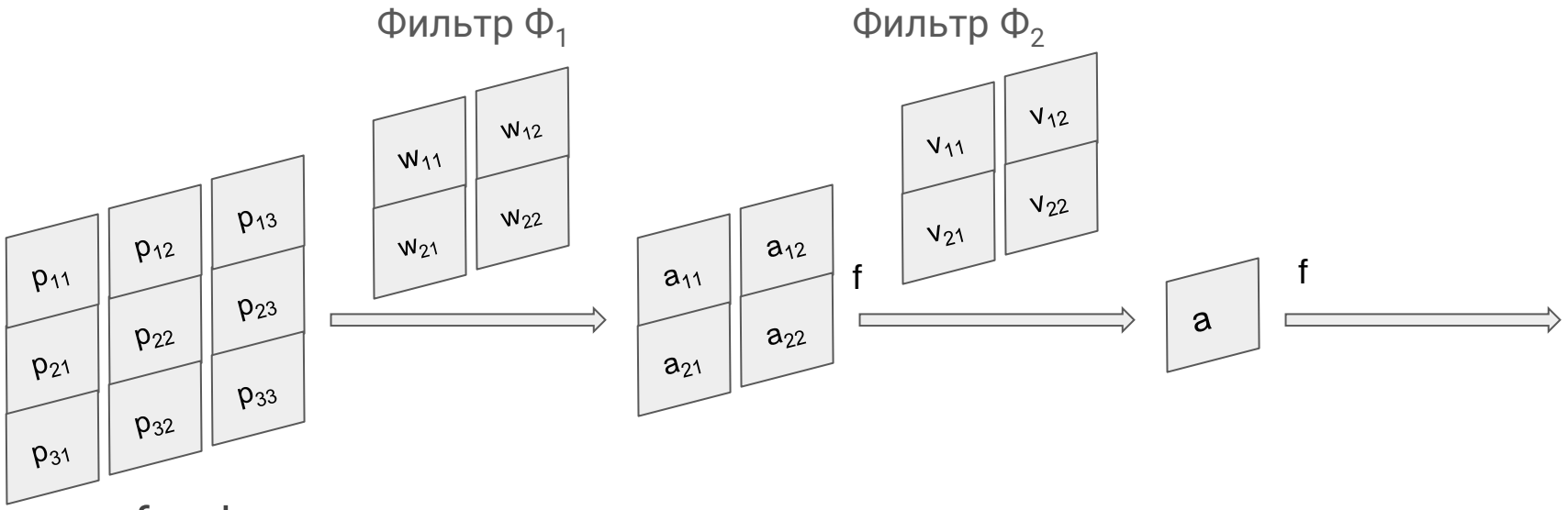
# Основная идея

СНС использует фильтры для свёртки изображений.

Но **коэффициенты фильтра определяет сама СНС**.

Грубо говоря: ИИ сам подбирает слепых для ощупывания слона.
Это не просто: для ощупывания разных частей слона нужен разный жизненный опыт.






f — функции активации.

Выражения $a_{ij}$ являются результатом свертки, можно их выписать:

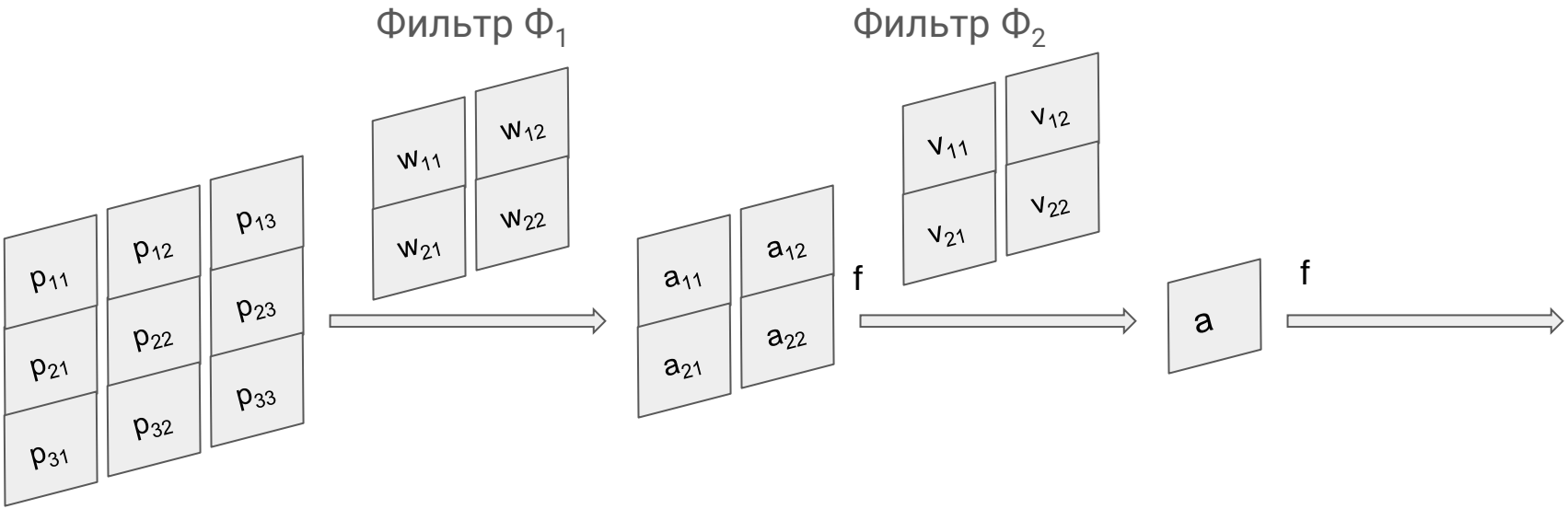
$a_{11}=p_{11}w_{11}+ p_{12}w_{12}+p_{21}w_{21}+p_{22}w_{22}$

$a_{12}=p_{12}w_{11}+ p_{13}w_{12}+p_{22}w_{21}+p_{23}w_{22}$

$a_{21}=p_{21}w_{11}+ p_{22}w_{12}+p_{31}w_{21}+p_{32}w_{22}$

$a_{22}=p_{22}w_{11}+ p_{23}w_{12}+p_{32}w_{21}+p_{33}w_{22}$

Кроме того: $a=f(a_{11})v_{11}+ f(a_{12})v_{12}+f(a_{21})v_{21}+f(a_{22})v_{22}$




**Тогда СНС вычисляет выражение:**

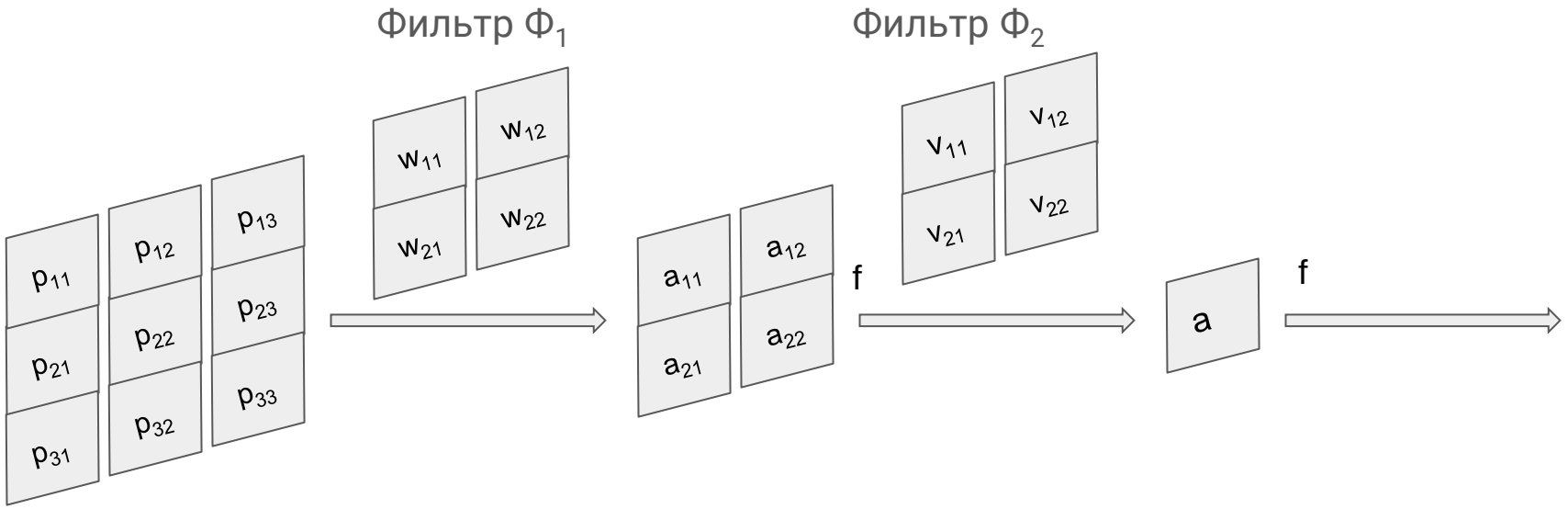
$$F_{NN}(P)=f(a)=f(f(a_{11})v_{11}+ f(a_{12})v_{12}+f(a_{21})v_{21}+f(a_{22})v_{22})=$$

$$=f(f(p_{11}w_{11}+ p_{12}w_{12}+p_{21}w_{21}+p_{22}w_{22})v_{11}+$$

$$+f(p_{12}w_{11}+ p_{13}w_{12}+p_{23}w_{21}+p_{23}w_{22})\,v_{12}+$$

$$+f(p_{21}w_{11}+ p_{22}w_{12}+p_{31}w_{21}+p_{32}w_{22})v_{21}+$$

$$+f(p_{22}w_{11}+ p_{23}w_{12}+p_{32}w_{21}+p_{33}w_{22})v_{22})$$




У нас есть функция $F_{NN}(P)=f(f(p_{11}w_{11}+ p_{12}w_{12}+p_{21}w_{21}+p_{22}w_{22})v_{11}+f(p_{12}w_{11}+ p_{13}w_{12}+p_{23}w_{21}+p_{23}w_{22})\ v_{12}+$
$+f(p_{21}w_{11}+ p_{22}w_{12}+p_{31}w_{21}+p_{32}w_{22})v_{21}+ f(p_{22}w_{11}+ p_{23}w_{12}+p_{32}w_{21}+p_{33}w_{22})v_{22})$.

А далее по тренировочной выборке мы выписываем функцию потерь и минимизируем её относительно весов НС (это стандартные этапы в тренировке НС).

На следующих слайдах мы повторим эти этапы для более простой СНС.

# Тренировочная выборка (ТВ)

Пусть дана тренировочная выборка из изображений и дано значение целевого признака для каждого из объектов ТВ.

**Признак Y количественный, то есть решается задача регрессии.**

| Изображение | Y |
|---|---|
| **P1:**<br>011<br>001 | 1 |
| **P2:**<br>110<br>011 | 2 |
| **P3:**<br>101<br>010 | 0 |

На самом деле мы загадали признак Y как «количество фрагментов вида 11»

# Тренировочная выборка (ТВ)

**В задаче регрессии функция потерь** — это сумма квадратов разности между функцией сети $F_{NN}(P)$ и истинным значением признака Y.

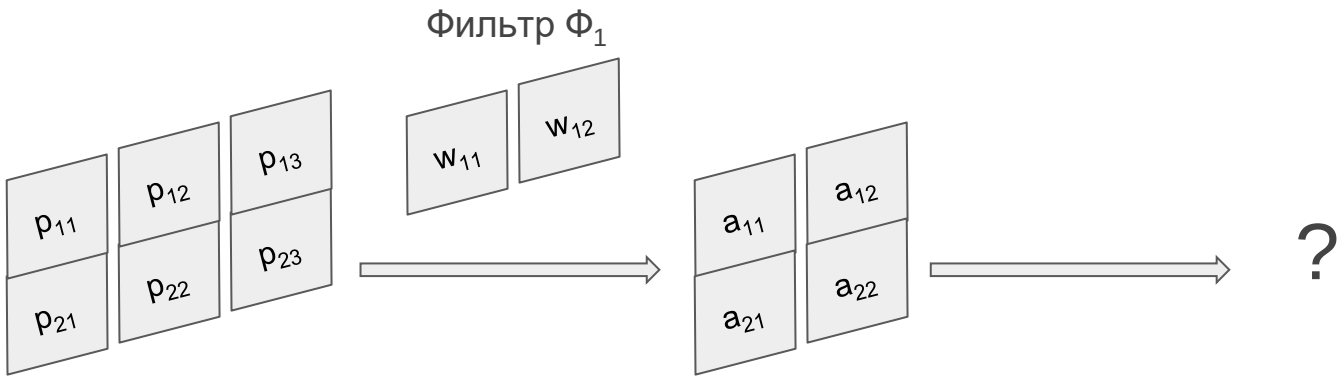
**Получаем функцию потерь:**

$$L(w)=(F_{NN}(P1)-1)^2+(F_{NN}(P2)-2)^2+(F_{NN}(P3)-0)^2$$

её и надо минимизировать.

| Изображение | Y |
|---|---|
| **P1:**<br>011<br>001 | 1 |
| **P2:**<br>110<br>011 | 2 |
| **P3:**<br>101<br>010 | 0 |

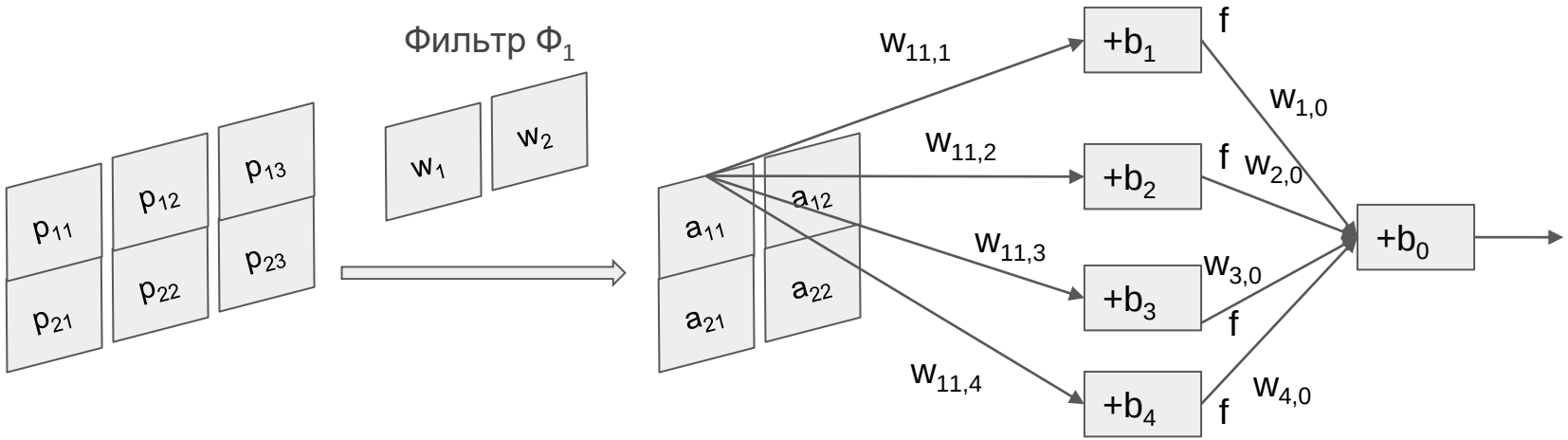
Сейчас выберем подходящую архитектуру сети.




Начнём строить СНС. Первый слой изображён выше.
А какие слои добавить дальше?

**Важная мысль:** **в любой СНС свёрточные слои** (там, где применяется фильтр) **рано или поздно заменяются на** (обычные) **полносвязные**.

Поскольку у нас задача простая, то мы добавим первый полносвязный слой сразу после первого свёрточного слоя...




**Здесь функция сети такая:**

$a_{11}=p_{11}w_1+p_{12}w_2$

$a_{12}=p_{12}w_1+p_{13}w_2$

$a_{21}=p_{21}w_1+p_{22}w_2$

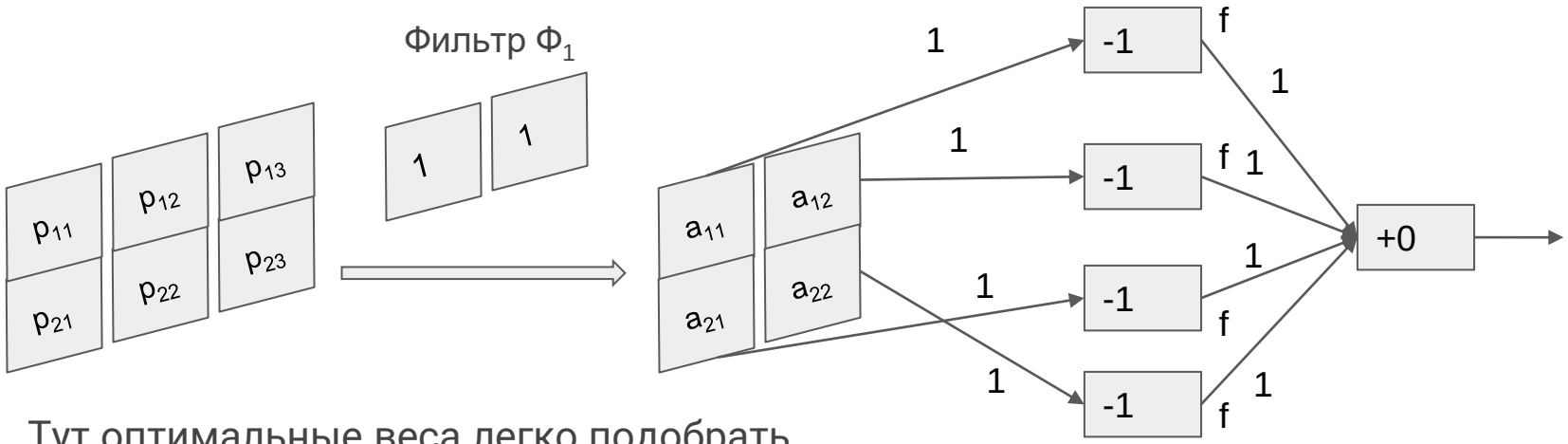
$a_{22}=p_{22}w_1+p_{23}w_2$

$$F_{NN}(P)=b_0+w_{1,0}f(a_{11}w_{11,1}+a_{12}w_{12,1}+a_{21}w_{21,1}+a_{22}w_{22,1}+b_1)+$$
$$+w_{2,0}f(a_{11}w_{11,2}+a_{12}w_{12,2}+a_{21}w_{21,2}+a_{22}w_{22,2}+b_2)+$$
$$+w_{3,0}f(a_{11}w_{11,3}+a_{12}w_{12,3}+a_{21}w_{21,3}+a_{22}w_{22,3}+b_3)+$$
$$+w_{4,0}f(a_{11}w_{11,4}+a_{12}w_{12,4}+a_{21}w_{21,4}+a_{22}w_{22,4}+b_4)$$

$f(x)=Relu(x)$

**Минимизируется функция потерь:**

$L(w)=(F_{NN}(P1)-1)^2+(F_{NN}(P2)-2)^2+(F_{NN}(P3)-0)^2$




Тут оптимальные веса легко подобрать
без использования градиентного спуска.
Веса-связи, равные 0, не показаны.

**Функция сети при этом равна:**

$a_{11}=p_{11}+p_{12}$

$a_{12}=p_{12}+p_{13}$

$a_{21}=p_{21}+p_{22}$

$a_{22}=p_{22}+p_{23}$

$F_{NN}(P)=f(a_{11}-1)+f(a_{12}-1)+f(a_{21}-1)+f(a_{22}-1)=$

$=f(p_{11}+p_{12}-1)+f(p_{12}+p_{13}-1)+f(p_{21}+p_{22}-1)+f(p_{22}+p_{23}-1)$

Проверим правильность выбранных весов
на конкретных примерах.

$F_{NN}(P)=f(p_{11}+p_{12}-1)+f(p_{12}+p_{13}-1)+f(p_{21}+p_{22}-1)+f(p_{22}+p_{23}-1)$

**Например, для изображения**
000
000
**получаем**
$F_{NN}(P)=f(-1)+f(-1)+f(-1)+f(-1)=0+0+0+0=0$
(для функции Relu выполнено Relu(-1)=0)

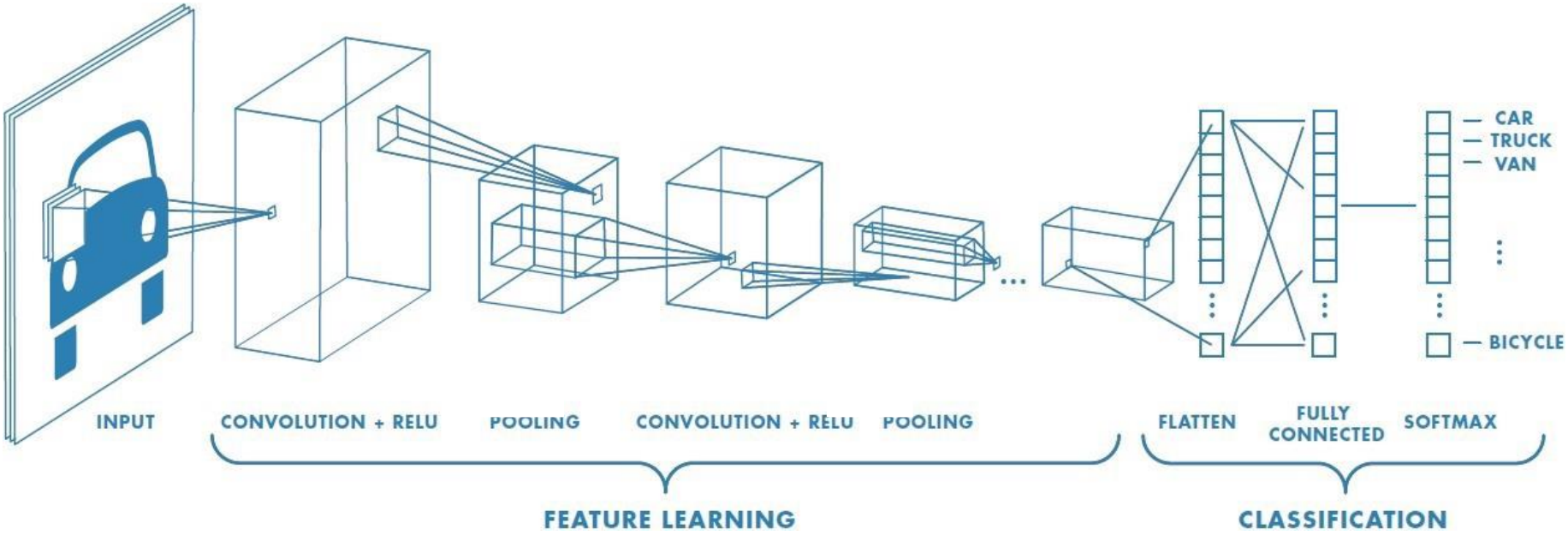
**А для изображения**
111
111
**получаем**
$F_{NN}(P)=f(2-1)+f(2-1)+f(2-1)+f(2-1)=1+1+1+1=4$

Это верные ответы, поскольку мы загадали признак
Y=«количество вхождений фрагмента 11»

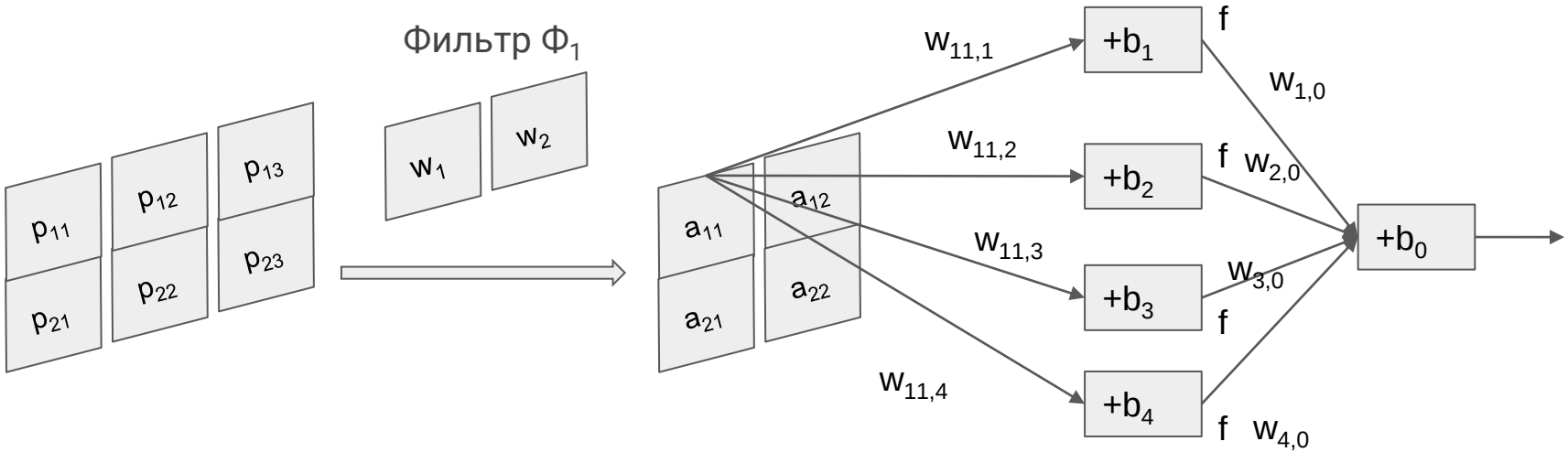
# Самая общая архитектура CNC:

Сначала идут фильтры (свёрточные слои). Это приводит к уменьшению размеров каждого последующего слоя.

В конце CNC стоит несколько полносвязных слоёв.



# В нашем примере архитектура была аналогичной:

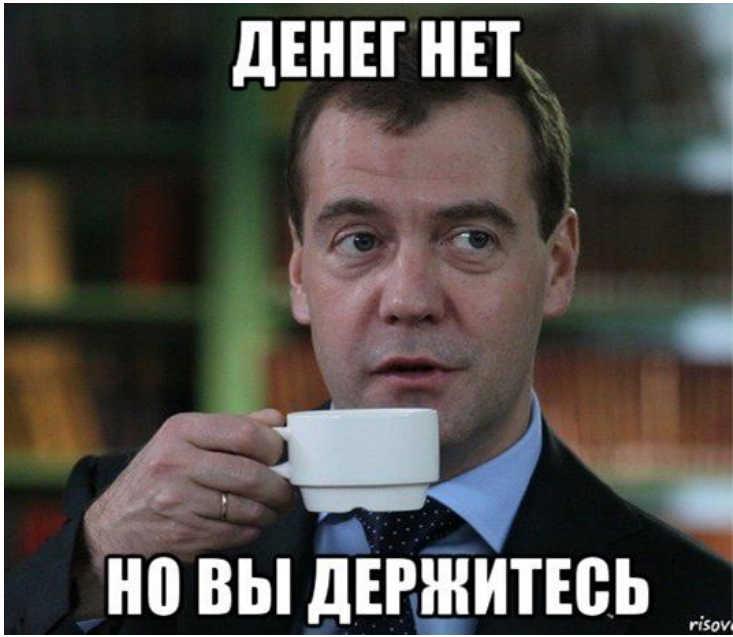


# Пулинг

# Экономия продолжается...

Фильтры, как правило, уменьшают размер изображения (если после каждого слоя не добавлять padding). Но не намного.

Для получения более компактной архитектуры CHC мы должны найти более радикальный способ уменьшить размер слоя.



# Идея!

Во многих практических задачах требуется найти важный фрагмент изображения (например, кошачий нос), который и решит проблему распознавания.

В то время как количество таких фрагментов не сильно важно (например, не важно, сколько кошачьих носов нашла СНС).

**Вывод**: если фильтр нашёл несколько значимых фрагментов в изображении, то часть этих фрагментов мы можем удалить, и таким образом уменьшить размер слоя СНС.

# Пулинг

Пулинг **разбивает изображение**, полученное очередным слоем СНС, на небольшие прямоугольники и из каждого прямоугольника **оставляет ровно один пиксел**.

У пулинга есть параметры n,m — размеры прямоугольничков (естественно, размеры всего изображения должны делиться на n,m).

Почти всегда в реальных СНС n=m (т.е. изображение разбивается на квадратики, и само изображение квадратное).

Здесь n=m=2.

Что дальше?

| | | | |
|---|---|---|---|
| 4 | 3 | 1 | 5 |
| 1 | 3 | 4 | 8 |
| 4 | 5 | 4 | 3 |
| 6 | 5 | 9 | 4 |

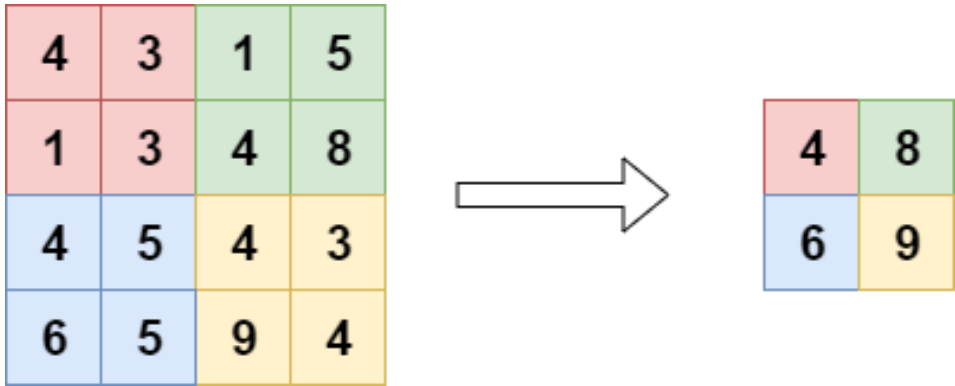
# Пулинг

**Есть несколько разновидностей пулинга:**

- **max-pooling** (наиболее употребителен);
- **min-pooling**
- **average-pooling**

В этих разновидностях каждый прямоугольничек заменяется на один пиксел, содержащий:

- максимальное значение
- минимальное значение
- среднее значение

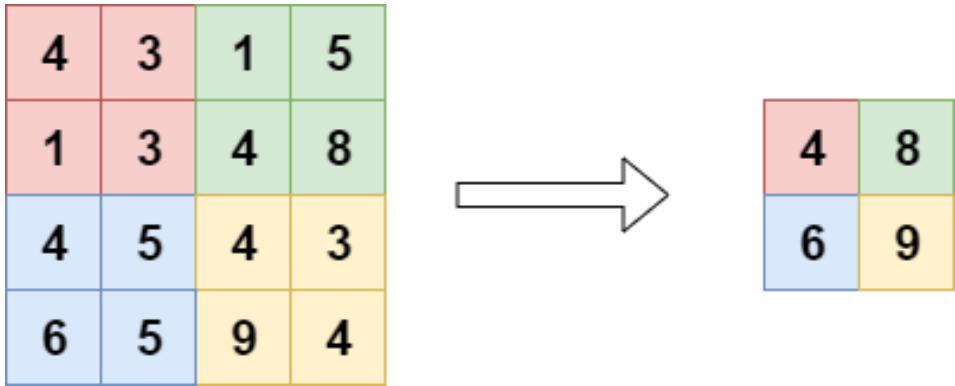


# Пулинг

Пулинг с параметрами n,m будем для краткости называть **(n,m)-пулингом**.

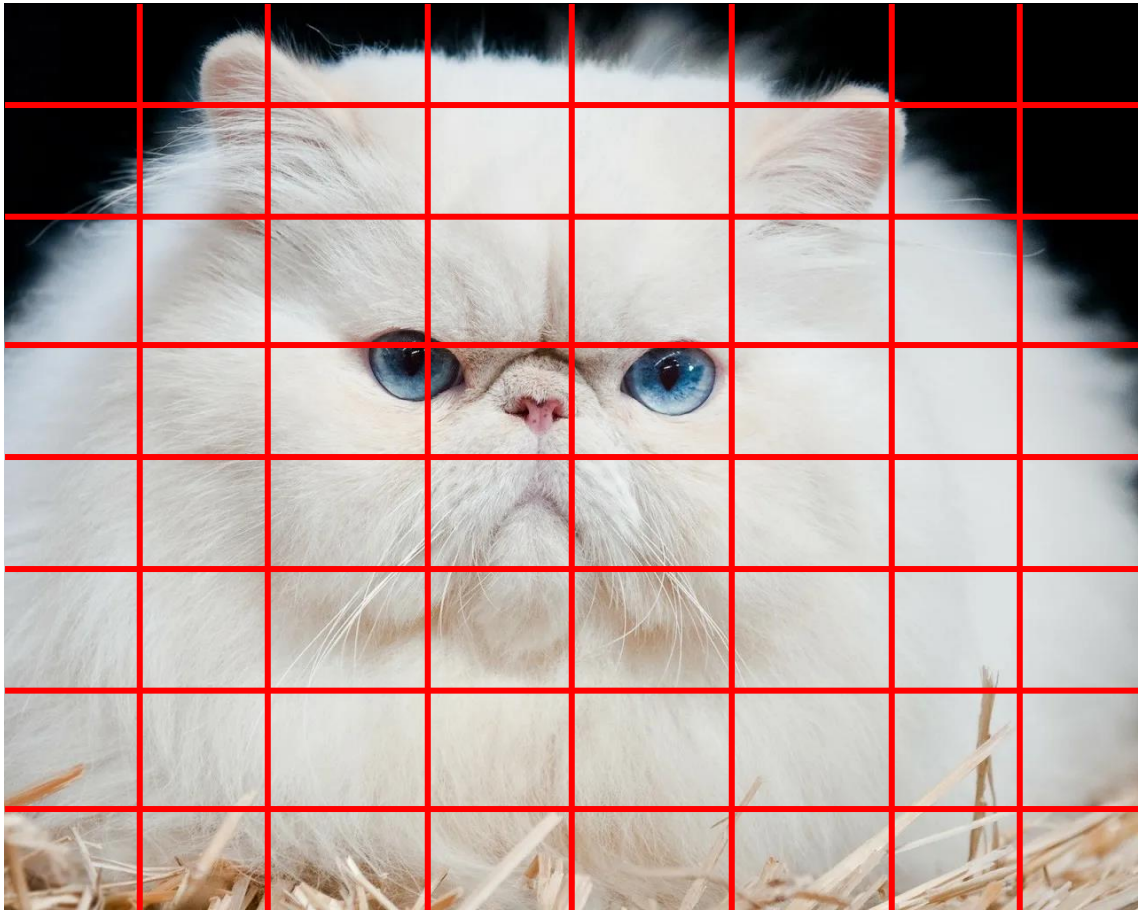
А если эти параметры равны, то будем писать ещё экономнее: **n-пулинг**.

В частности, в нашем примере применяется 2-пулинг.



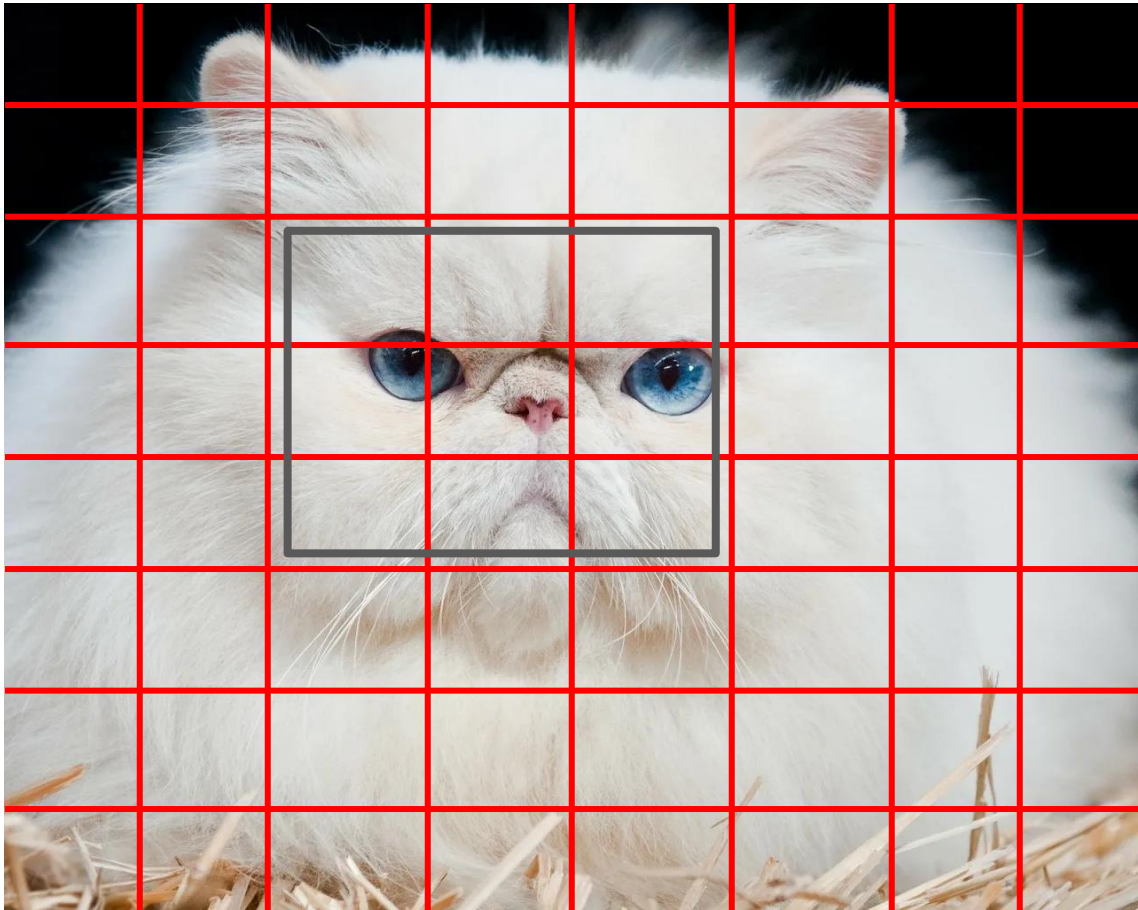
# Ещё раз об идее пулинга

Пусть изображение
разбито на зоны
(в частности: на пиксели).



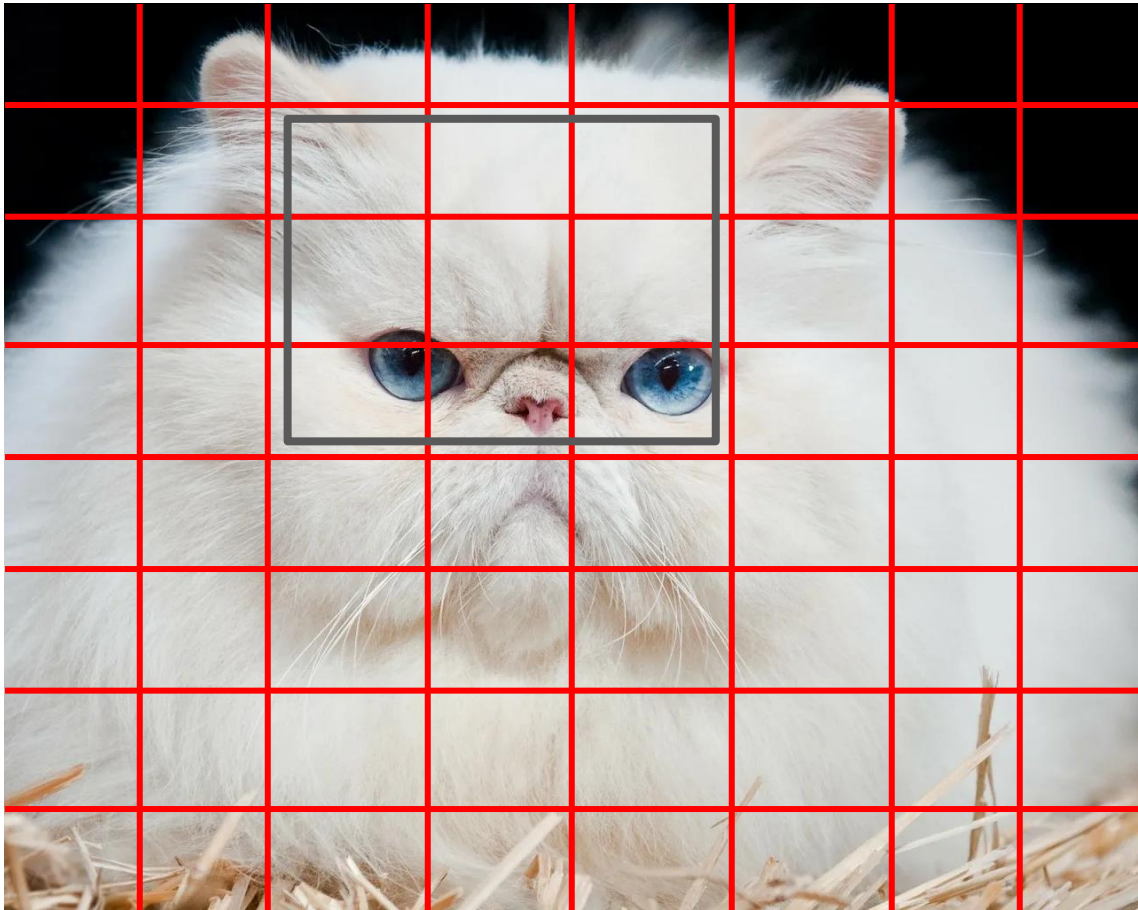
# Ещё раз об идее пулинга

Наличие кошачьей морды можно детектировать по следующим областям изображения размера 3x3:



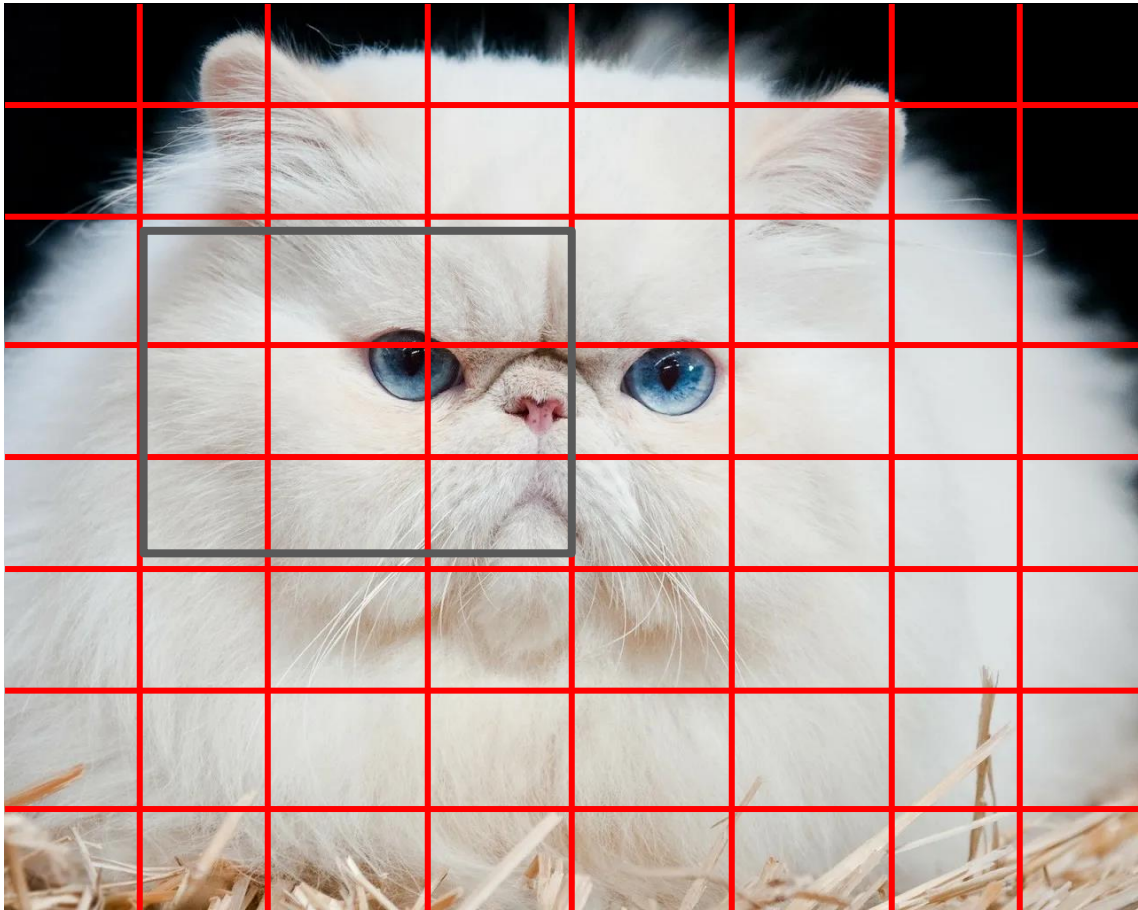
# Ещё раз об идее пулинга

Наличие кошачьей морды можно детектировать по следующим областям изображения размера 3x3:



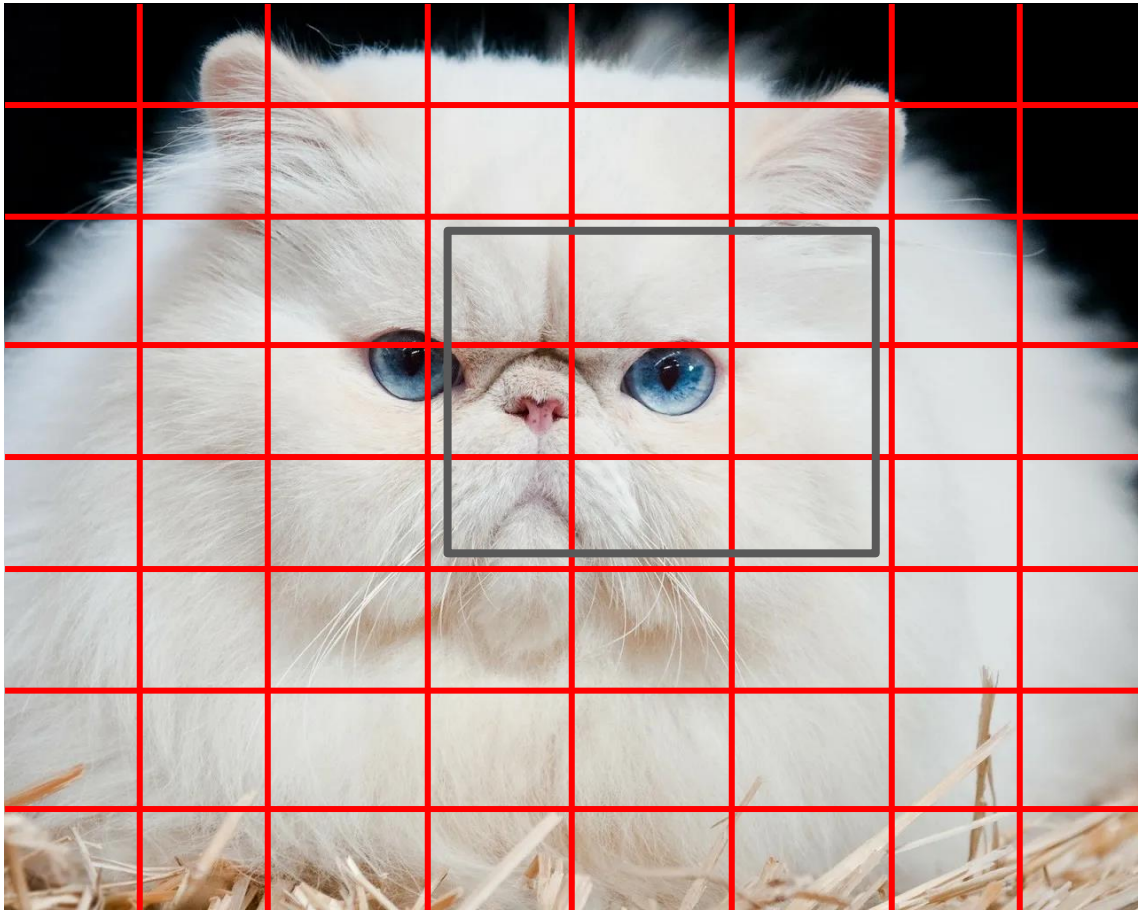
# Ещё раз об идее пулинга

Наличие кошачьей морды можно детектировать по следующим областям изображения размера 3x3:



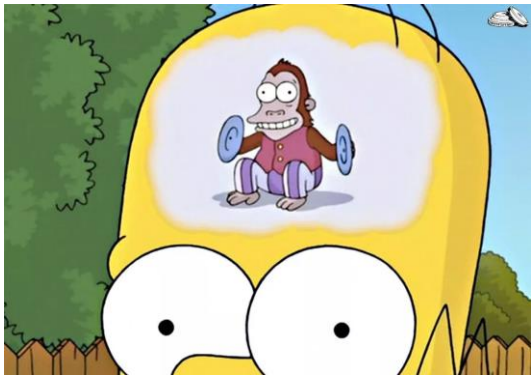
# Ещё раз об идее пулинга

Наличие кошачьей морды можно детектировать по следующим областям изображения размера 3x3:

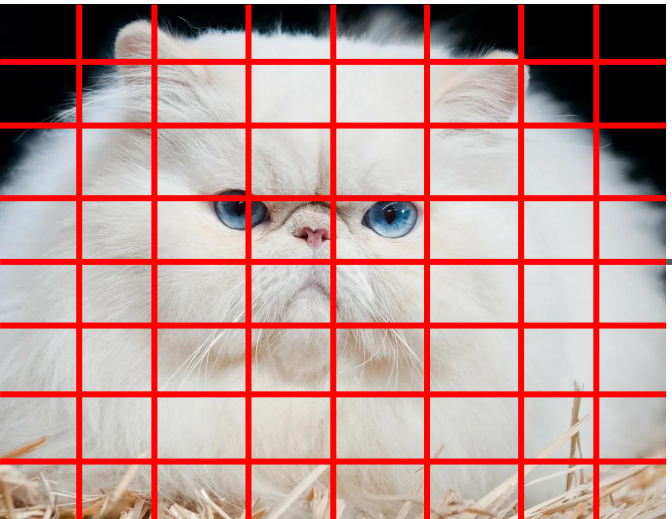


# Что же произойдёт где-то в глубине CHC?

По картинке будет ездить **фильтр, который натренировался на распознавании кошачьей морды**.



По основному свойству фильтра зоны изображения с кошачьей мордой дадут огромные числа при операции свёртки:



**Фильтр 3x3:**

***

***

***

**Результат:**

000000

006530

007650

005640

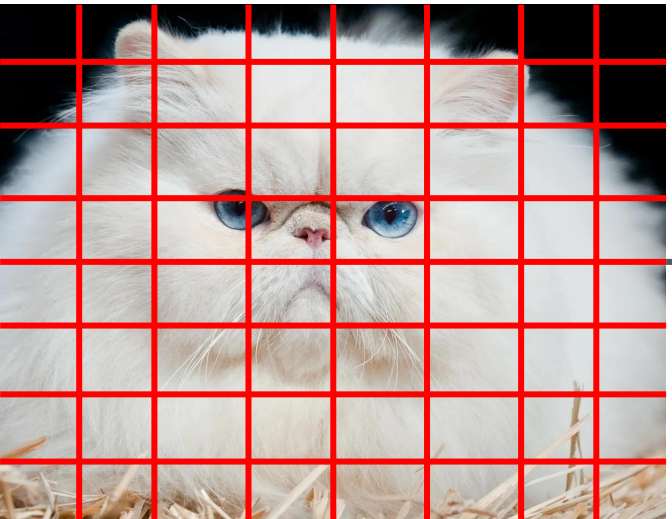
002200

000000

# Что же произойдёт где-то в глубине СНС?

Мы видим, что информация в свёртке избыточна.
СНС уверенно находит кошачью морду в нескольких пикселях.

Избыточную информацию удаляем с помощью **пулинга**.



**Фильтр 3x3:**

***
***
***

**Результат:**

000000
006530
007650
005640
002200
000000

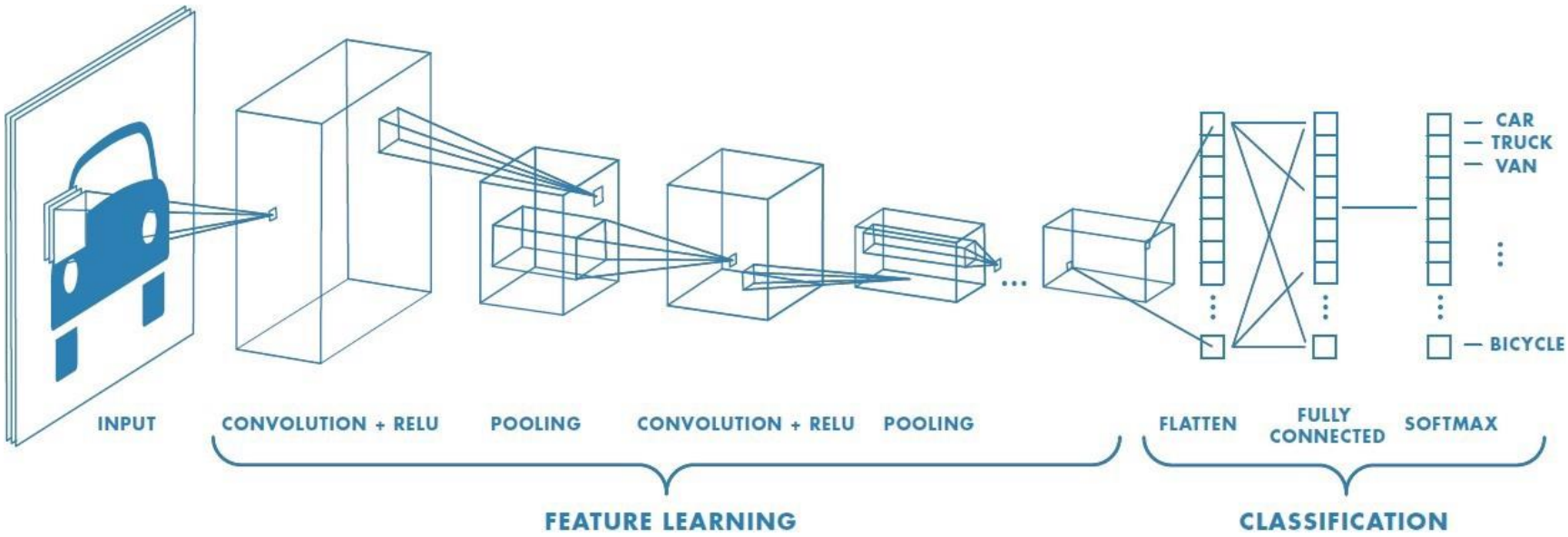
# Что же произойдёт где-то в глубине СНС?

Применим 2-пулинг. Получаем:


**Это позволяет уменьшить объём данных (и количество весов СНС) и ускорить тренировку СНС.**

# Фильтры и пулинг приводят к уменьшению размера изображения в глубоких слоях CHC

Это и обуславливает **общий вид архитектуры CHC**: **воронкообразный**.

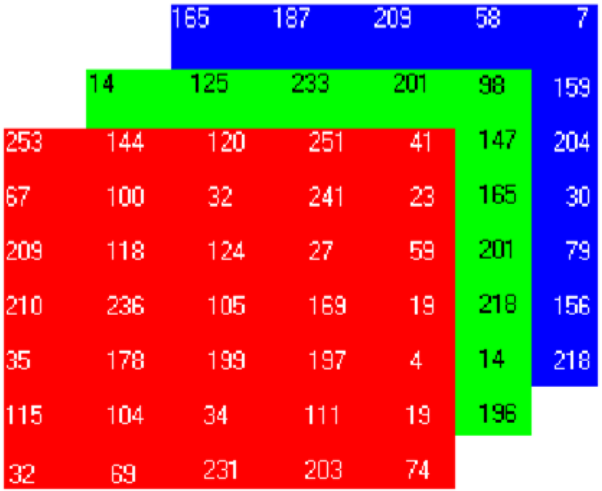


# Многоканальные фильтры

# А если одновременно применить два фильтра?

То получится два результата. То есть изображение расщепится на **два канала**. Это и есть **многоканальное изображение**. Ничего удивительного!

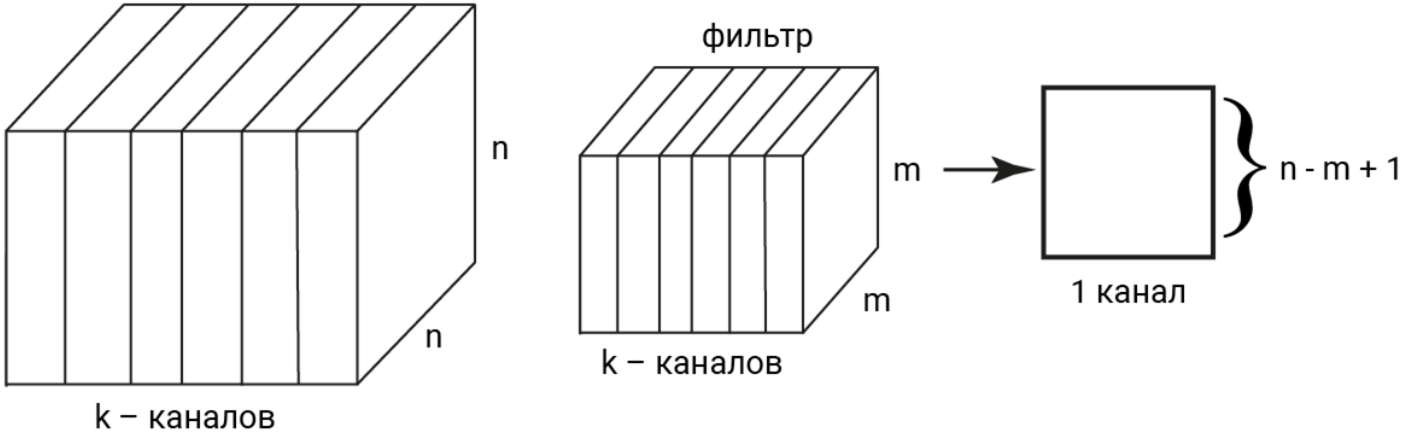
Цветные картинки **изначально** многоканальные (формат RGB).



# Фильтры для многоканальных изображений

**Фильтр для r-канального изображения сам должен быть r-канальным** (например, фильтр для изображения в формате RGB сам должен быть 3-канальным).

**Свёртка** (то есть результат применения r-канального фильтра к r-канальному изображению) — это **одноканальное** изображение.
Ибо формула свёртки многоканальных изображений заключается в применении свёртки по каждому каналу, и поэлементном суммировании свёрток.



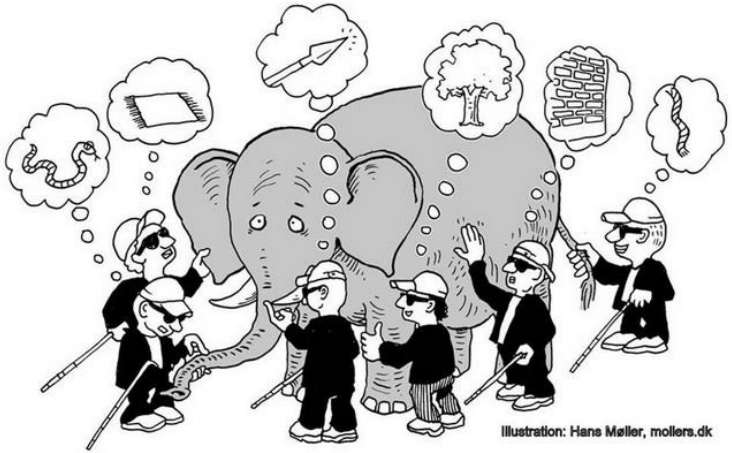
# Снова пример из жизни

Выше мы поняли, чтобы слепым распознать слона, им нужен ещё один «сверх-мудрец», который обработает информацию от «простых мудрецов».

То есть сверх-мудрец должен **работать с многоканальным изображением**, состоящим из ответов простых мудрецов.

Грубо говоря, сверх-мудрец — это второй слой СНС, получающий для работы 6 каналов с изображениями

- змеи
- ковра
- копья
- дерева
- стены
- верёвки




3 канала изображения
3 канала фильтра
Свёртка в каждом канале
Итоговая свёртка
1 0 1 0 2
1 1 3 2 1
1 1 0 1 1
2 3 2 1 3
0 2 0 1 0
1 0 0 1 0
2 0 1 2 0
3 1 1 3 0
0 3 0 3 2
1 0 3 2 1
2 0 1 2 1
3 3 1 3 2
2 1 1 1 0
3 1 3 2 0
1 1 2 1 1
0 1 0
0 0 2
0 1 0
2 1 0
0 0 0
0 3 0
1 0 0
1 0 0
0 0 2
7 5 3
4 7 5
7 2 8
5 3 10
13 1 13
7 12 11
7 5 2
11 8 2
9 4 6
Output
19 13 15
28 16 20
23 18 25

# Ещё пример

Хотим в этой картинке найти некий фрагмент. Применим одновременно к ней три фильтра, получим 3-канальное изображение.

| Картинка | | | | |
|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 |

**Фильтры:**

| | | |
|---|---|---|
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 1 |

| | | |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 1 | 1 |

| | | |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |

**Получим три канала изображения:**

| | | |
|---|---|---|
| 0 | 3 | 1 |
| 2 | 1 | 3 |
| 3 | 3 | 1 |

| | | |
|---|---|---|
| 4 | 5 | 4 |
| 6 | 6 | 6 |
| 7 | 8 | 7 |

| | | |
|---|---|---|
| 1 | 3 | 0 |
| 3 | 1 | 2 |
| 1 | 3 | 3 |

# Ещё пример

**Получим три канала изображения:**

| | | |
|---|---|---|
| 0 | 3 | 1 |
| 2 | 1 | 3 |
| 3 | 3 | 1 |

| | | |
|---|---|---|
| 4 | 5 | 4 |
| 6 | 6 | 6 |
| 7 | 8 | 7 |

| | | |
|---|---|---|
| 1 | 3 | 0 |
| 3 | 1 | 2 |
| 1 | 3 | 3 |

| Картинка | | | | |
|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 |

**Применим к ним трёхканальный фильтр размера 1x1:**

| | | | | |
|---|---|---|---|---|
| 1 | | -1 | | 1 |

**Получим одноканальное изображение:**

| | | |
|---|---|---|
| -3 | 1 | -3 |
| -1 | -4 | -1 |
| -3 | -2 | -3 |

А смысл? Композиция фильтров на самом деле находит:

| | | |
|---|---|---|
| 1 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |

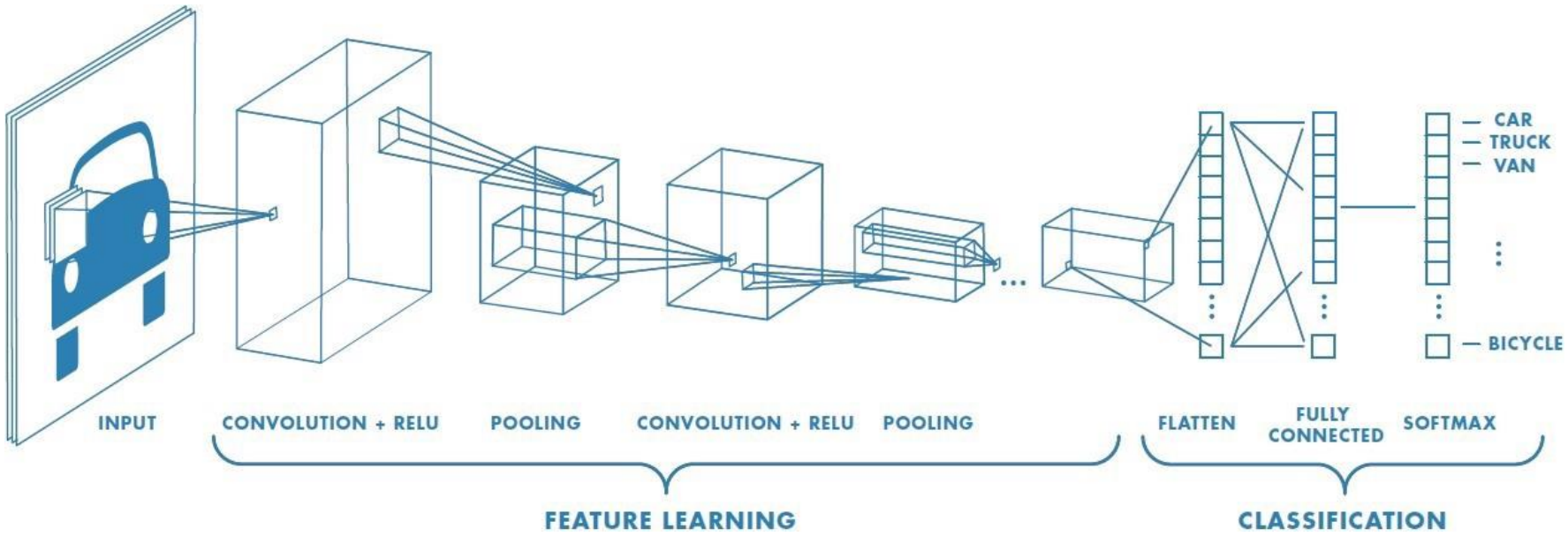
# Мораль

**Идея многоканальности**: мы можем выделять множество признаков, а потом смотреть, как они взаимодействую друг с другом.

Фильтр №1
Фильтр №2

Результат

# Толщина квадратиков — это количество каналов



# Настало время вас наказать и…

… решить задачу по определению количества весов в СНС.

Итак, будет описана архитектура СНС, нужно будет **посчитать количество  её весов**.

— А зачем это нужно?

— А затем, что от количества весов зависит: время тренировки СНС, необходимый объём тренировочной выборки и т.п.

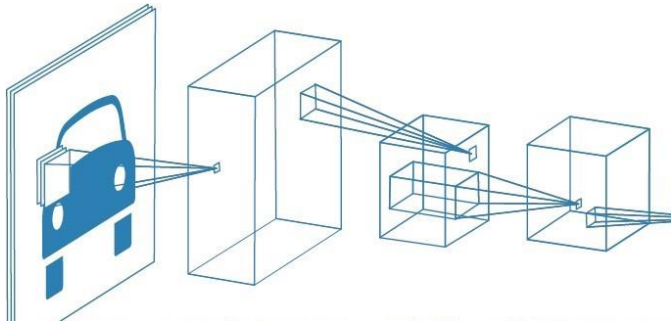
# Задача

Было трёхканальное (привет, RGB) изображение размера 10x10.
К нему применили 4 фильтра (они, естественно, 3-канальные) размера 5x5.
Потом ко всем каналам применили 2-пулинг.
К полученному 4-канальному изображению применили 2 фильтра 3x3.
А потом применили один 2-канальный фильтр 1x1.

**Изображение какого размера (и сколько там каналов) получилось в результате?**

**Сколько весов во всех используемых фильтрах?**

Всюду у фильтров stride_x=stride_y=1, padding=0.



# Анализируем каждый слой CHC:

**1. Было трёхканальное (привет, RGB) изображение размера 10x10.**
**К нему применили 4 фильтра (они, естественно, 3-канальные) размера 5x5.**
Поскольку применили 4 фильтра, то в итоге будет **4 канала.**

**Какова размерность каждого канала?**
Если фильтр 5x5 скользит по изображению 10x10, то в результате **будет изображение 6x6.**

**Сколько тут весов для тренировки?**
Смотрим на размерность **фильтров**: четыре 3-канальных фильтра при размерности одного канала 5x5 дают 4*3*5*5=**300 весов**.

**Ответ**: на 1-м слое возникнет 4-канальное изображение 6x6 и будет 300 весов для тренировки.

# Анализируем каждый слой CHC:

**2. Потом ко всем каналам применили пулинг 2x2.**

С предыдущего слоя пришло 4-канальное изображение 6x6.
2-пулинг сделает из него его 4-канальное изображение 3x3.
Операция пулинга не имеет весов для тренировки.

**Ответ**: на 2-м слое возникнет 4-канальное изображение 3x3
и будет 0 весов для тренировки.

# Анализируем каждый слой CHC:

**3. К полученному 4-канальному изображению применили 2 фильтра 3x3**. Поскольку применили 2 фильтра, то в итоге будет **2 канала.**

**Какова размерность каждого канала?**
Вспоминаем, что с предыдущего слоя пришло 4-канальное изображение 3x3! Если фильтр 3x3 скользит по изображению 3x3, то в результате **будет изображение 1x1.**

**Сколько тут весов для тренировки?**
Смотрим на размерность **фильтров**: два 4-канальных фильтра при размерности одного канала 3x3 дают 2*4*3*3= **72 веса**.

**Ответ**: на этом слое возникнет 2-канальное изображение размера 1x1, 72 веса для тренировки.

# Анализируем каждый слой CHC:

**4. А потом применили один 2-канальный фильтр 1x1.**
Поскольку применили 1 фильтр, то в итоге будет **1 канал.**

**Какова размерность каждого канала?**
Вспоминаем, что с предыдущего слоя пришло 2-канальное изображение 1x1! Если фильтр 1x1 скользит по изображению 1x1, то в результате **будет изображение 1x1.**

**Сколько тут весов для тренировки?**
Смотрим на размерность **фильтров**: один 2-канальный фильтр при размерности одного канала 1x1 дают 1*2*1*1= **2 веса**.

**Ответ**: на этом слое возникнет 1-канальное изображение размера 1x1, 2 веса для тренировки.

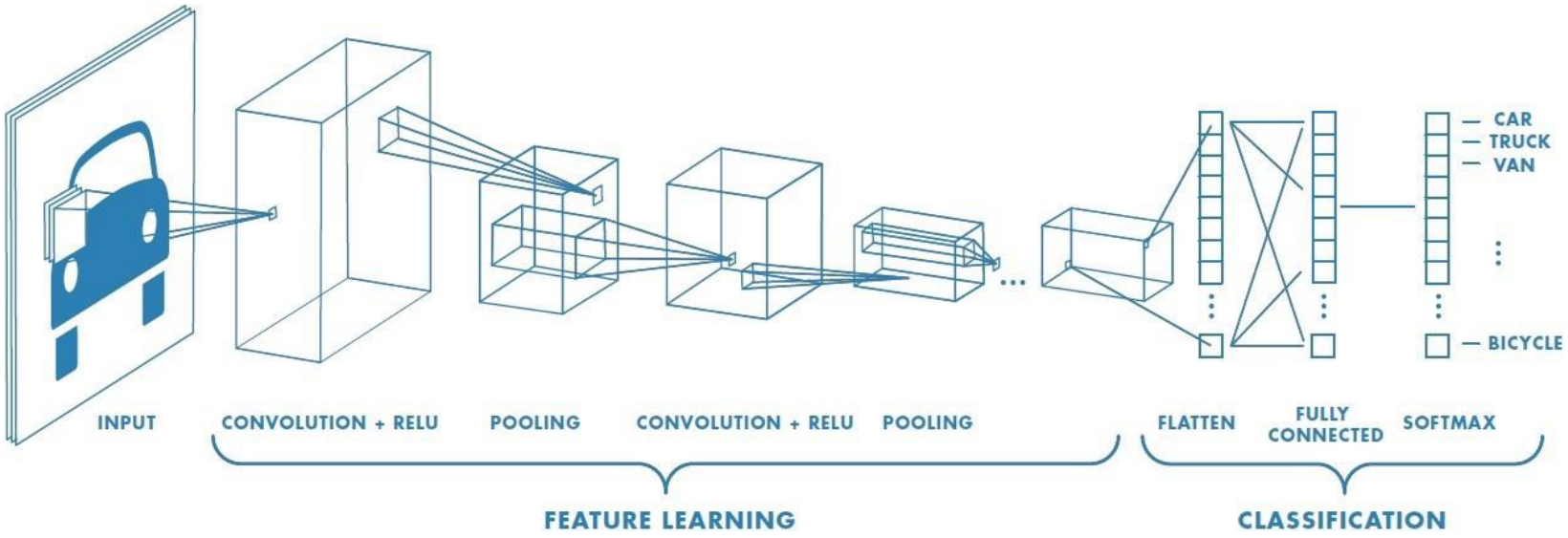
# Итог:

В рассмотренных слоях СНС оказалось **300+72+2=374** весов для тренировки.

# Уточнение к архитектуре CNC:

Мы видели, что **уменьшается размер изображения, но увеличивается количество каналов**.

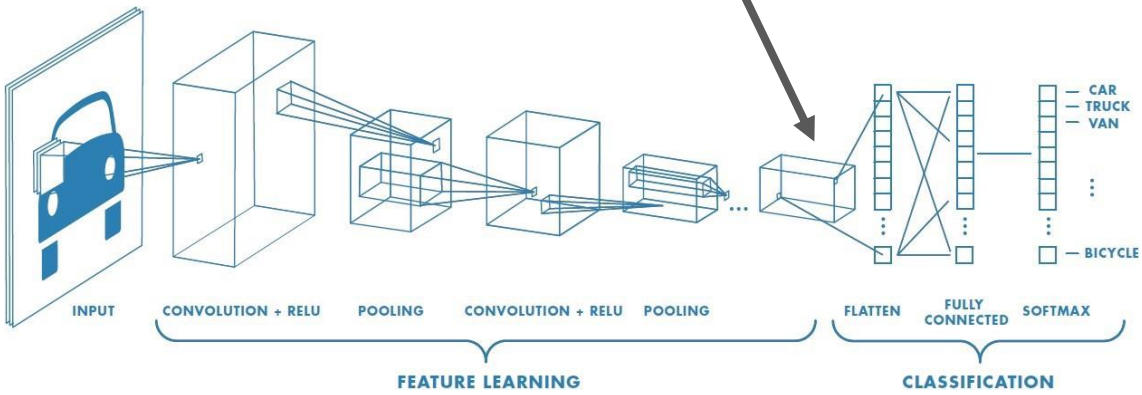
**А зачем?**



# Более-менее рациональное объяснение архитектуры CHC

**Мы увеличиваем количество каналов и уменьшаем размерность изображения.** В пределе мы преобразуем изображение в вектор длины r (r — это количество каналов в последнем свёрточном слое).
Т.е. мы представили картинку (сложный объект)
в виде набора **независимых** числовых признаков.
Эти признаки подаются на вход полносвязным слоям CHC.

(полная аналогия с колбасой и ДНК)

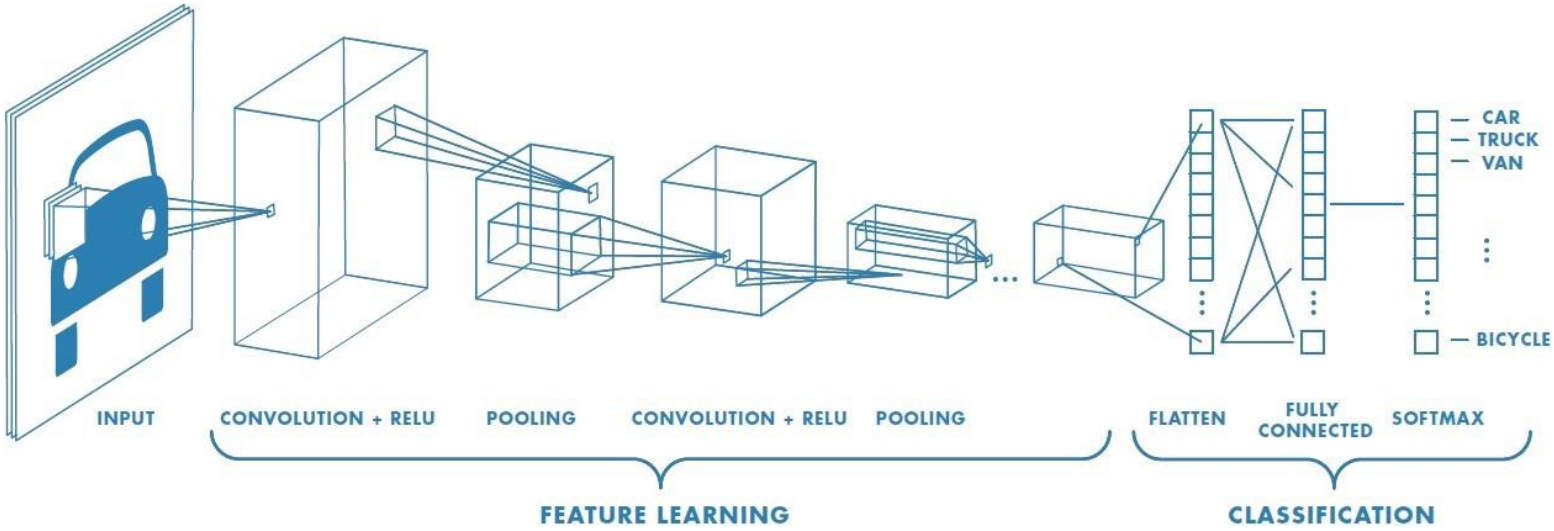


# Картинка превращается в вектор

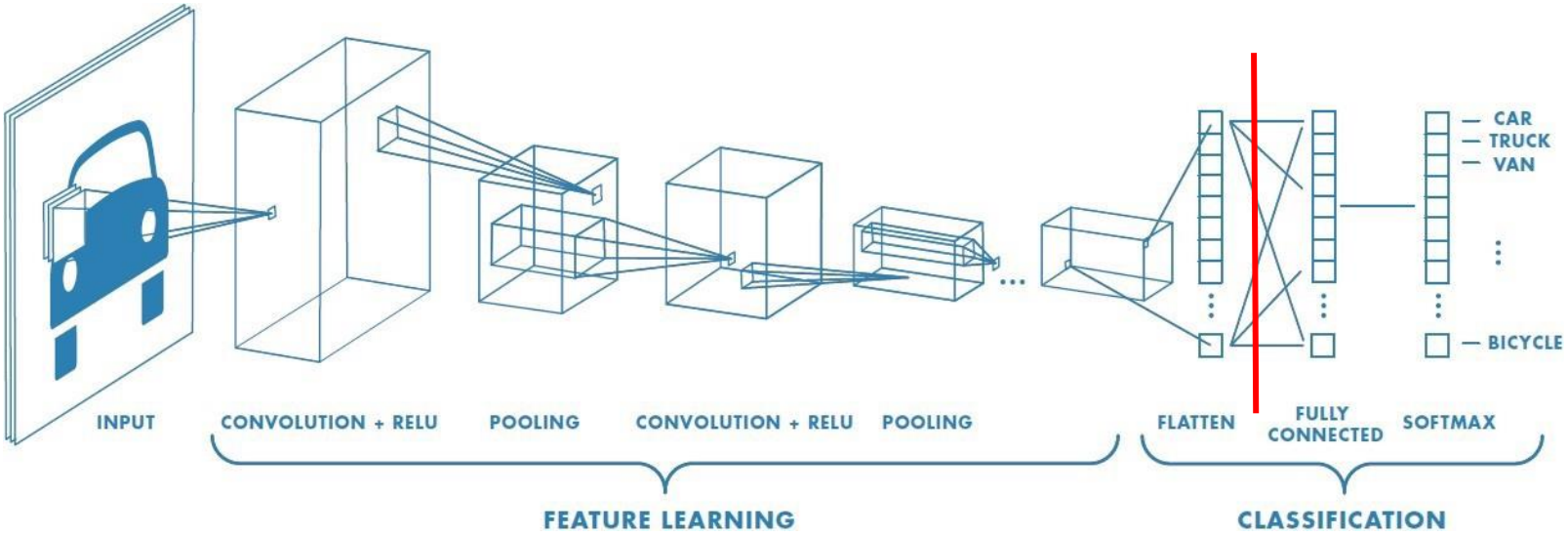
# Более-менее рациональное объяснение архитектуры CHC

Мы представили картинку (сложный объект) в виде набора числовых признаков. Эти признаки подаются на вход полносвязным слоям CHC.

Тут лежит очень мощная идея: **получить embedding** (распределённое представление) объекта.



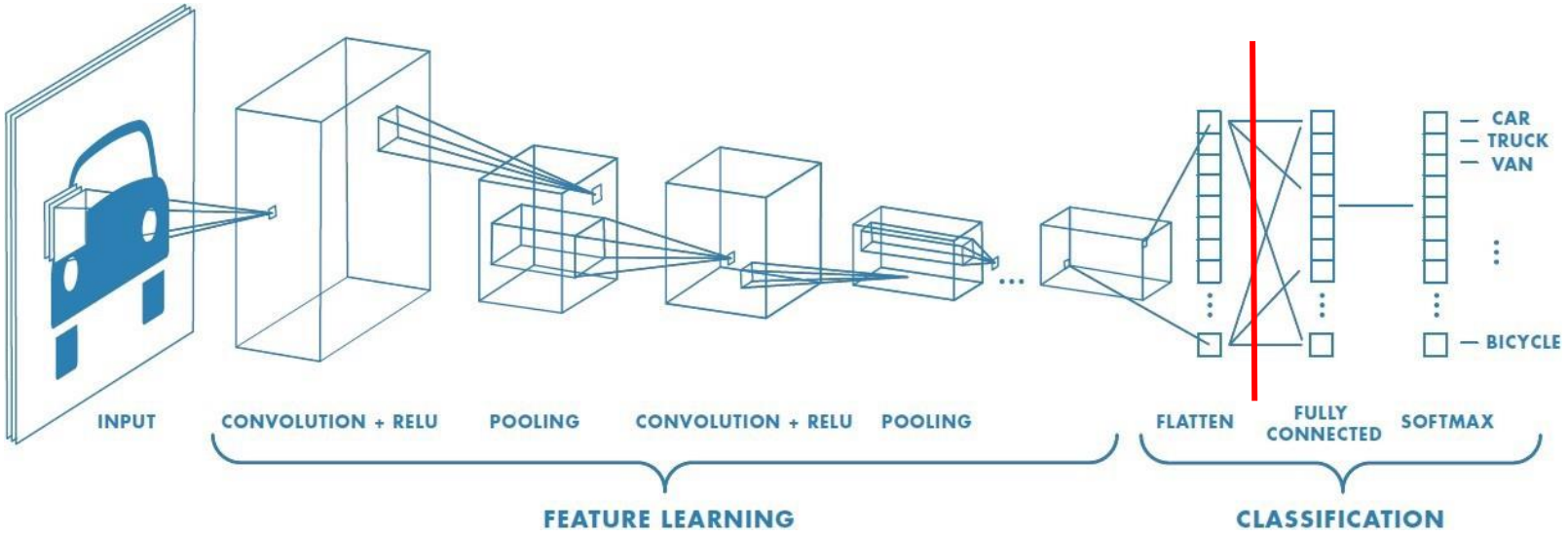
# Что такое embedding?




Пусть полносвязный слой слева от красной черты состоит из N нейронов. При обработке картинки они выдадут в следующий слой N чисел.

**И вот мы получили перевод картинки в N-мерный вектор, то есть сделали embedding картинки.**

# Зачем нужен embedding?




Когда картинки превращаются в вектора, то можно (например) считать расстояния между ними.

Дело в том, что embedding переводит похожие картинки в **близкие вектора**.

# А что такое «близкий вектор»?

Тут надо вспомнить высшую математику и формулы, по которым считаются расстояния между векторами $A=(a_1,\ldots,a_n)$, $B=(b_1,\ldots,b_n)$:

- **классическое, евклидово расстояние**

$$d(A,B) = \sqrt{(a_1 - b_1)^2 + \ldots + (a_n - b_n)^2}$$

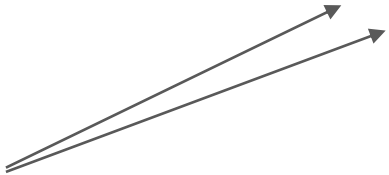
- **max-метрика:**

$$d(A,B) = \max\{|a_1 - b_1|, \ldots, |a_n - b_n|\}$$

# А можно считать не расстояние, а меру сходства

**Косинусное сходство векторов $A=(a_1,\ldots,a_n)$, $B=(b_1,\ldots,b_n)$:**

$$\cos\phi = \frac{a_1 b_1 + \ldots + a_n b_n}{\sqrt{a_1^2 + \ldots + a_n^2}\sqrt{b_1^2 + \ldots + b_n^2}}$$



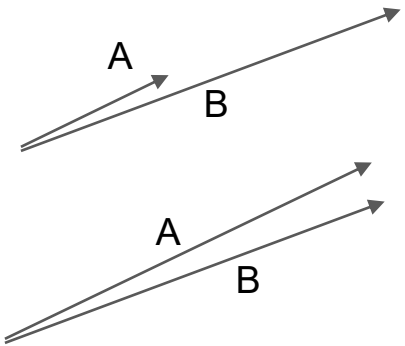
Большое сходство

Малое сходство

# Важное свойство: инвариантность относительно длины

Косинусное сходство векторов A=($a_1$,...,$a_n$), B=($b_1$,...,$b_n$) :

$$\cos\phi = \frac{a_1 b_1 + \ldots + a_n b_n}{\sqrt{a_1^2 + \ldots + a_n^2}\sqrt{b_1^2 + \ldots + b_n^2}}$$




Мера сходства будет одинакова

**Почему это важно?**

# Инвариантность относительно перекрашивания поверхности

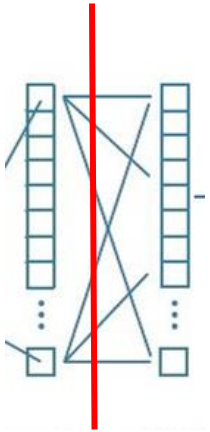
Справа — кот.
А слева тоже кот?

И вообще: кошку можно покрасить в любой цвет и она всё равно останется кошкой.

# А в чём разница между чёрным и белым котом?

**При проходе этих картинок через СНС на слое возникнут вектора**

$e(БК)=(e_1,e_2,...,e_n)$

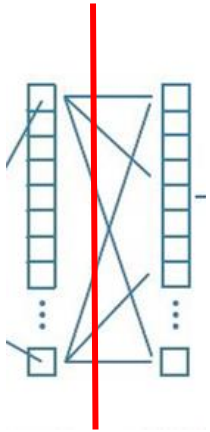


$e(ЧК)=(f_1,f_2,...,f_n)$

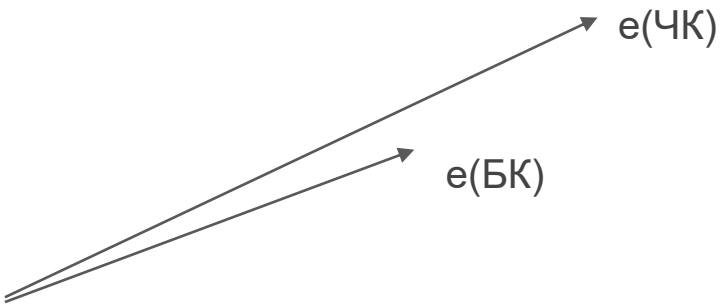
Как связаны между собой числа $e_1,e_2,...,e_n$ и $f_1,f_2,...,f_n$ ?

# А в чём разница между чёрным и белым котом?

**Как связаны между собой числа $e_1,e_2,...,e_n$ и $f_1,f_2,...,f_n$ ?**

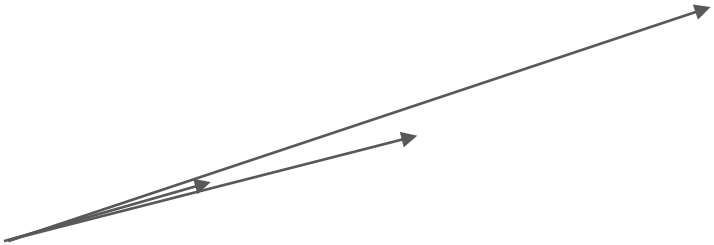


Поскольку коты отличаются лишь интенсивностью цвета
(а все остальные зависимости между их пикселами одинаковые),
то вектора $e(БК)=(e_1,e_2,...,e_n)$ и $e(ЧК)=(f_1,f_2,...,f_n)$ будут
**почти сонаправленными.**




Разница в длине векторов — за счёт изменения интенсивности цвета.

# Кошки разных цветов



**Это embedding белой, чёрной и рыжей кошки.**

Евклидово расстояние между ними большое,
но косинусное сходство велико!

# А в чём разница между чёрным и белым котом?

По этой причине для вычисления степени похожести картинок **важны не длины векторов, а угол между ними**.

По этой причине в качестве метрики для картинок применяют (чаще всего) **косинусное сходство**.

$$\cos\phi = \frac{a_1 b_1 + \ldots + a_n b_n}{\sqrt{a_1^2 + \ldots + a_n^2}\sqrt{b_1^2 + \ldots + b_n^2}}$$

# Выводы

- Были изучены многоканальные фильтры и многоканальные изображения.

- Была приведена схема тренировки СНС на небольшой тренировочной выборке.

- Была объяснена чрезвычайно плодотворная идея embedding-а (которая будет использоваться на последующих занятиях).