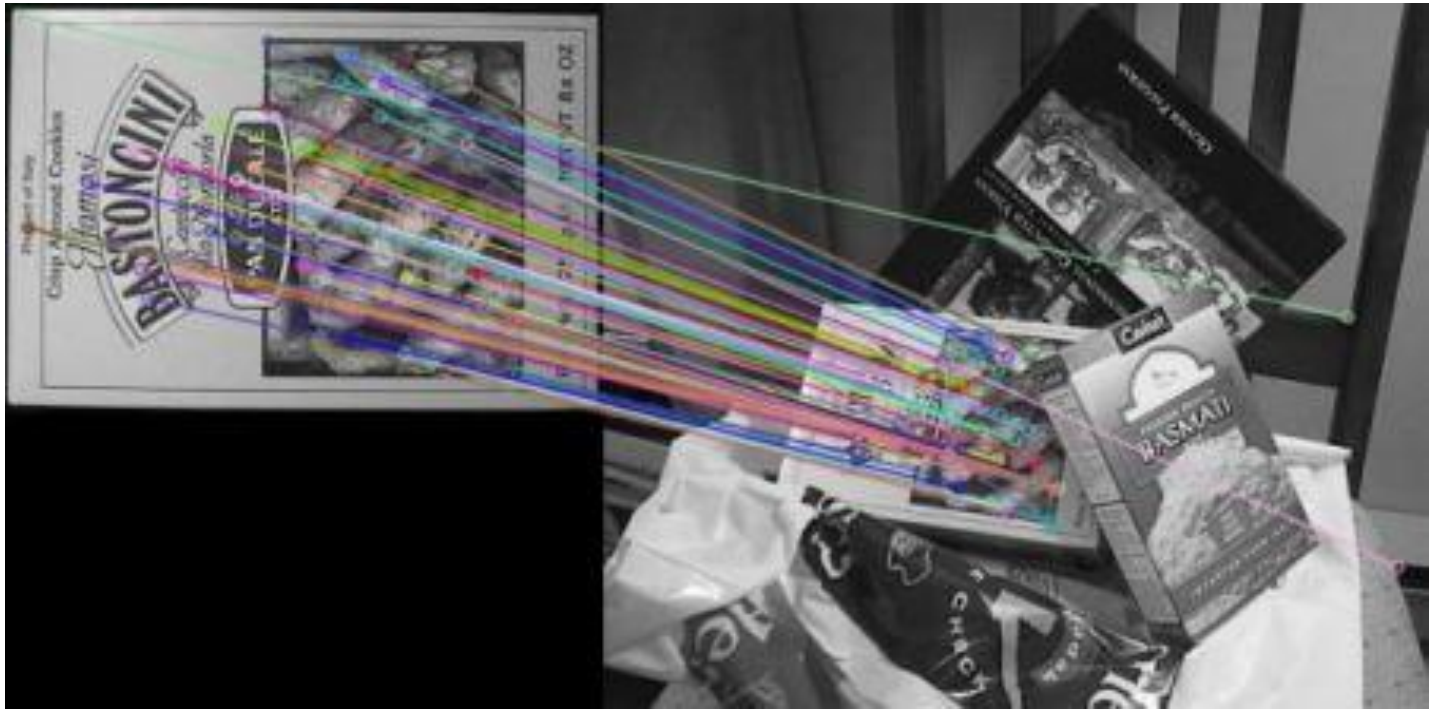
# Ключевые точки, сопоставление изображений

# Ключевые точки




https://docs.opencv.org/4.x/dc/dc3/tutorial_py_matcher.html

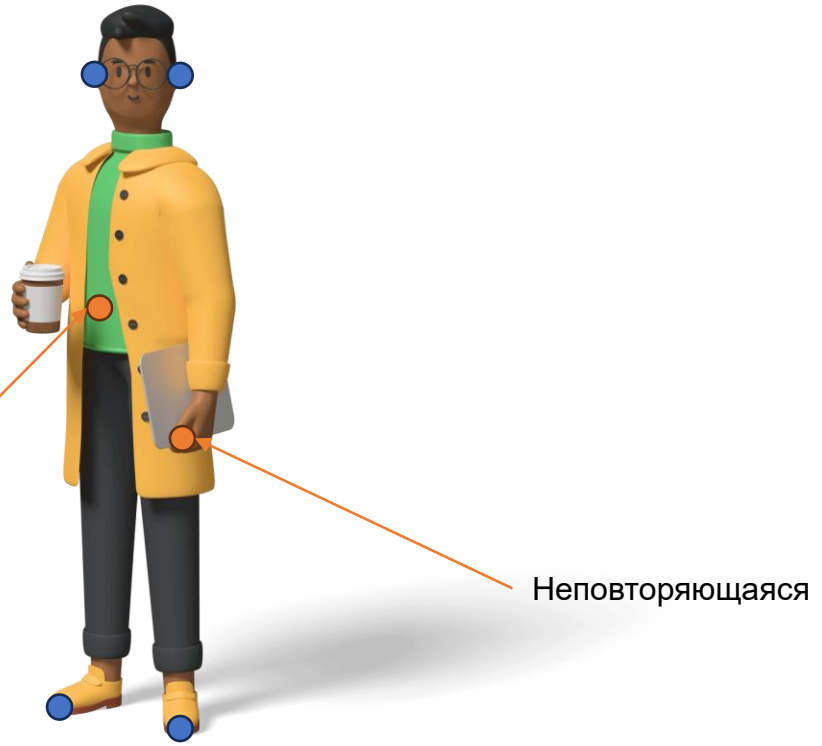
# Ключевые точки

Хотим наложить один объект на другой

Главные свойства для нас:

**Repeatability**
Точка будет в одном и том же месте на разных изображениях

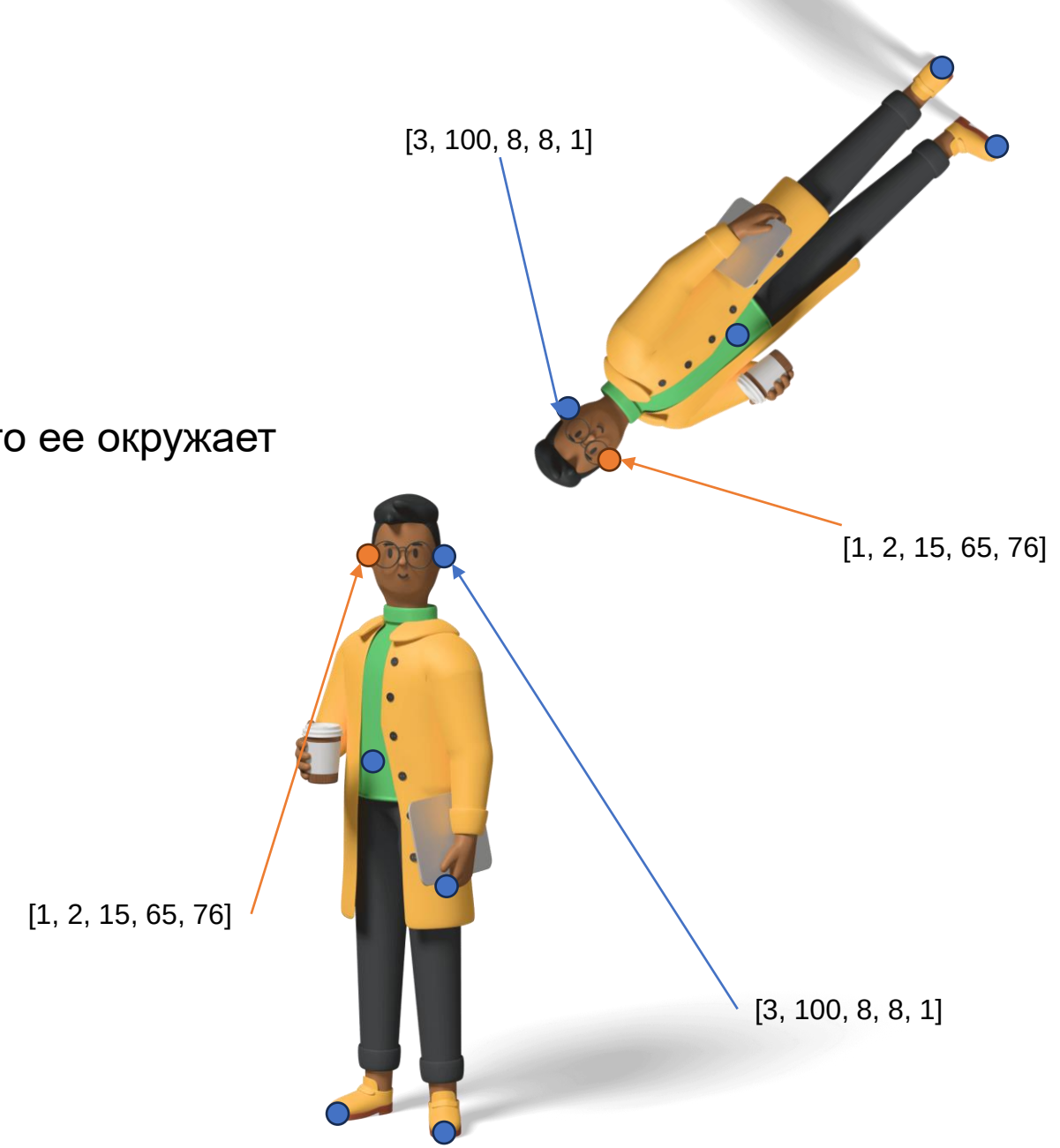
**Reliability**
Точка будет стоять в каком-то важном и характерном месте



# Ключевые точки

**Дескриптор**
Вектор, который будет хорошо описывать точку и что ее окружает

**Хороший дескриптор**
У разных точек непохожие дескрипторы
У одинаковых – похожие дескрипторы



# Ключевые точки

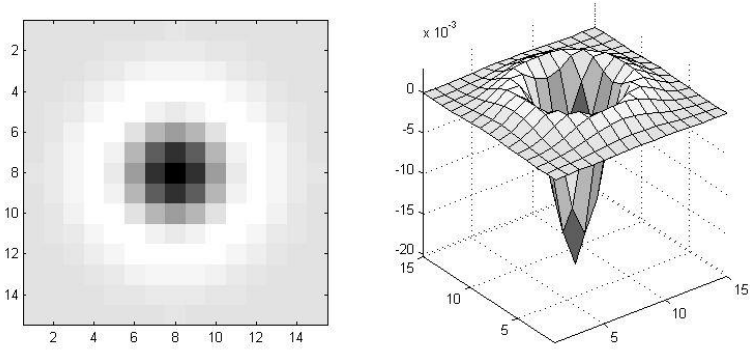
- Harris
- Shi-Tomasi
- BRIEF
- FAST
- SURF
- ORB
- **SIFT**

https://docs.opencv.org/4.x/db/d27/tutorial_py_table_of_contents_feature2d.html

# Scale Invariant Feature Transform

## Локализация

В теории сворачиваем изображение со второй производной гуассианы (Laplacian)
Ищем экстремум по изображению и по размеру этой гауссианы




Blob detection

# Scale Invariant Feature Transform

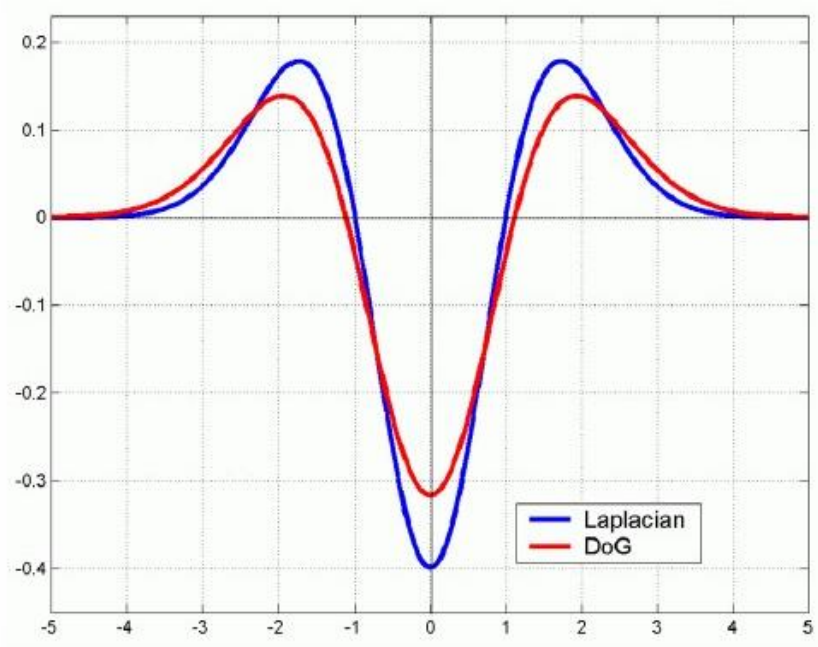
Локализация

$$L = \sigma^2 \left( G_{xx}(x, y, \sigma) + G_{yy}(x, y, \sigma) \right)$$

**(Laplacian)**

$$DoG = G(x, y, k\sigma) - G(x, y, \sigma)$$

**(Difference of Gaussians)**



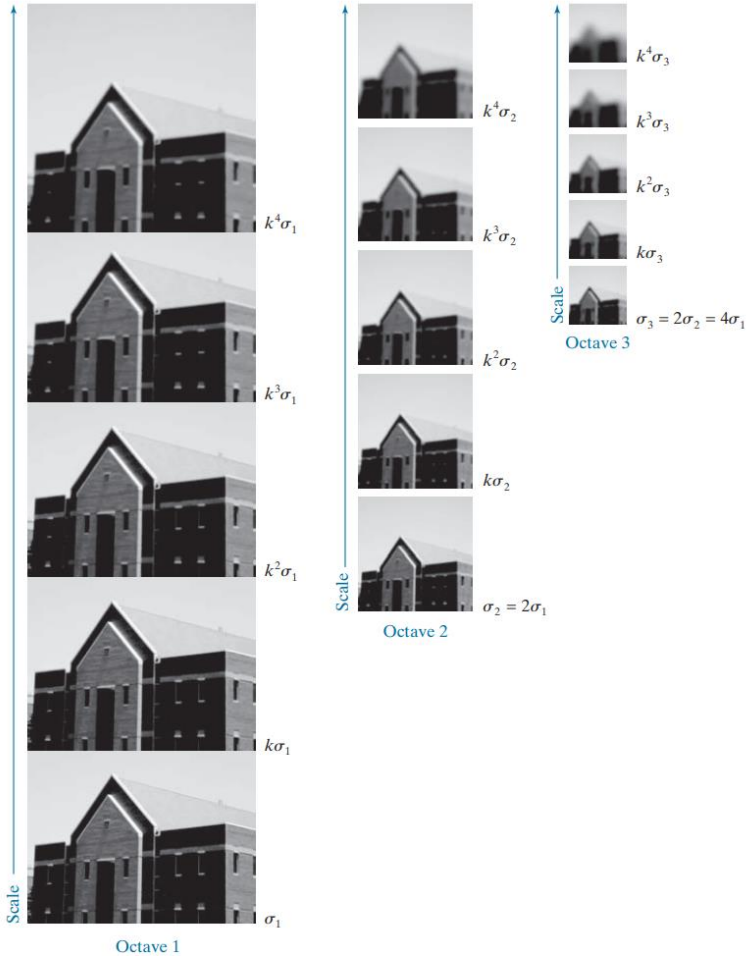
# Scale Invariant Feature Transform

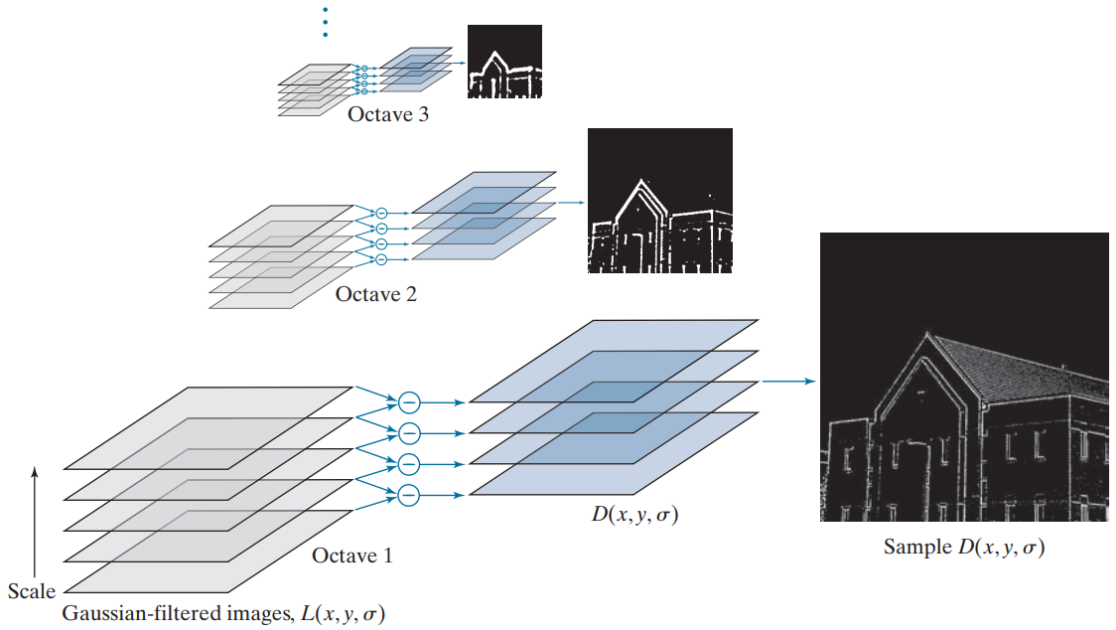
## Локализация

На практике блюрят изображения

Вычитают одно из другого

Получают то же самое



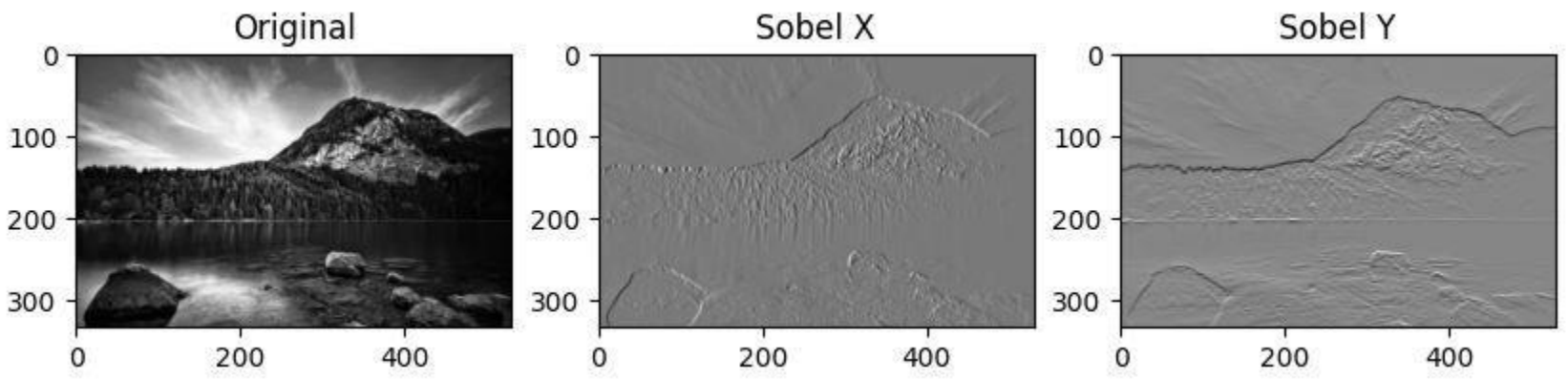



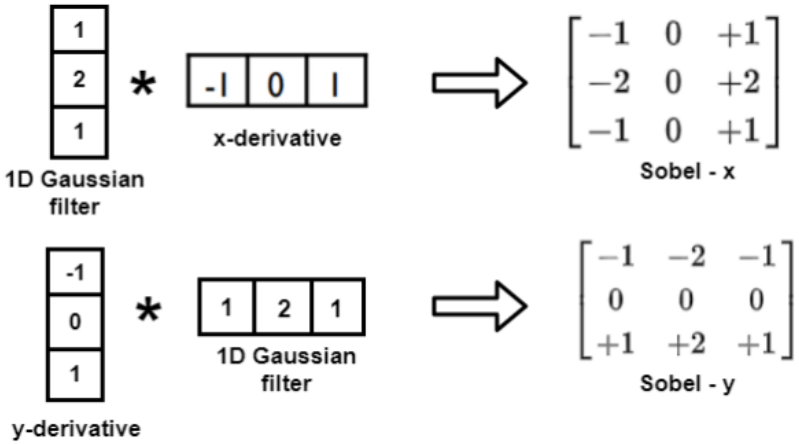
# Scale Invariant Feature Transform

## Извлечение дескриптора

Изображение – это функция *F(x, y)*

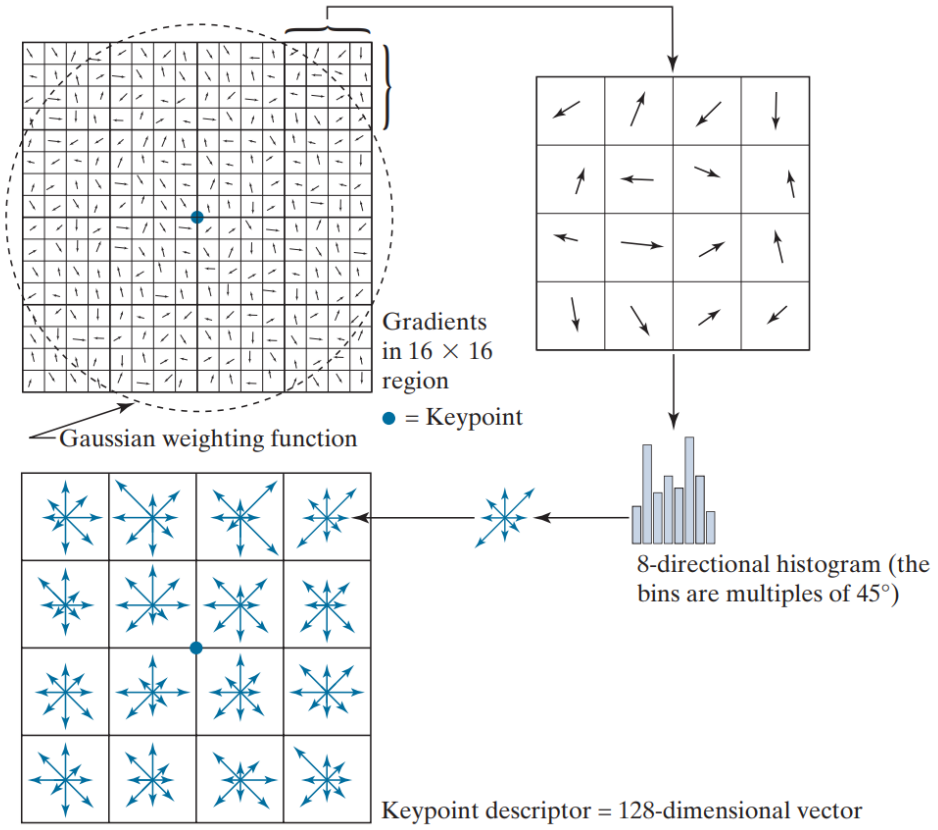
Можем посчитать производную



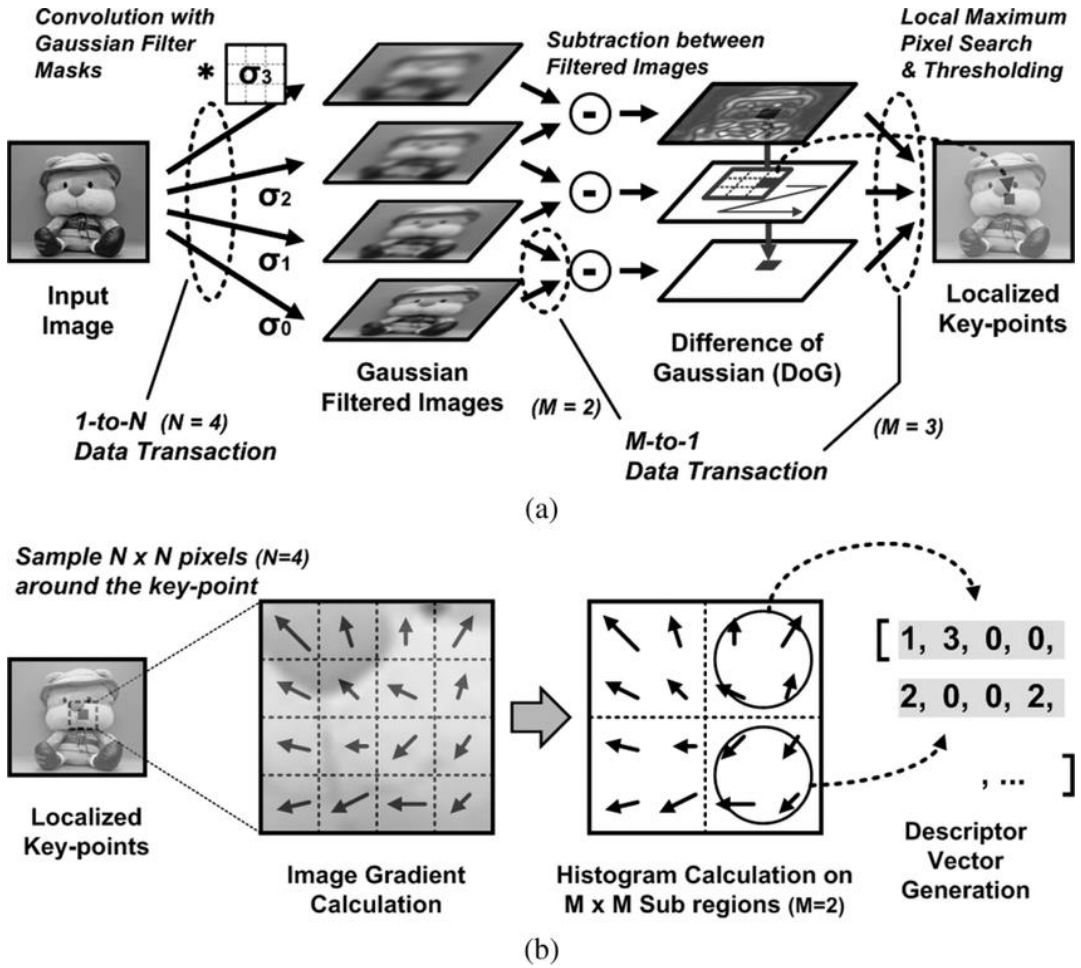



# Scale Invariant Feature Transform

## Извлечение дескриптора



# Scale Invariant Feature Transform



# Scale Invariant Feature Transform

Сравнение дескрипторов

Как правило, берут вот такую метрику L2 Distance

$$d(H_1, H_2) = \sqrt{\sum_k (H_1(k) - H_2(k))^2}$$

Но могут быть усложнения Normalized Correlation

$$d(H_1, H_2) = \frac{\sum_k [(H_1(k) - \bar{H}_1)(H_2(k) - \bar{H}_2)]}{\sqrt{\sum_k (H_1(k) - \bar{H}_1)^2} \sqrt{\sum_k (H_2(k) - \bar{H}_2)^2}}$$

$$\bar{H}_i = \frac{1}{N} \sum_{k=1}^{N} H_i(k)$$

# Scale Invariant Feature Transform

Попарно сравниваем дескрипторы и сопоставляем точки

**Cross Check**

Можно смотреть, что для точки A – ближайшая B,
и для B – ближайшая A

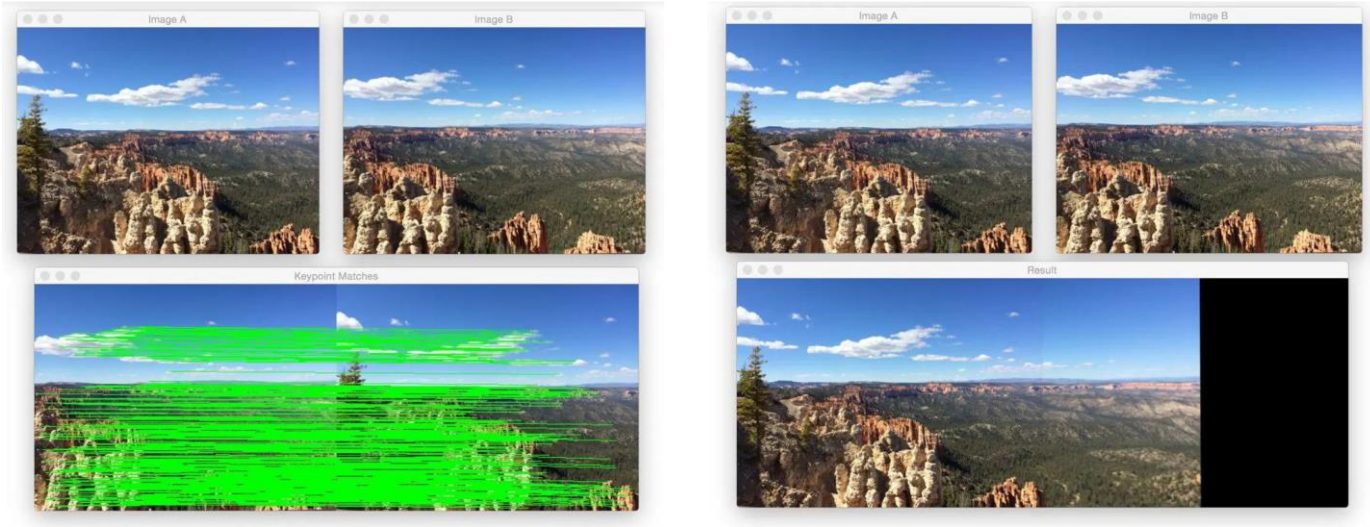
**Lowe’s ratio test**

Можно смотреть, что первый ближайший сосед стоит дальше второго

```
# ratio test as per Lowe's paper
for i,(m,n) in enumerate(matches):
    if m.distance < 0.1*n.distance:
        matchesMask[i]=[1,0]
```

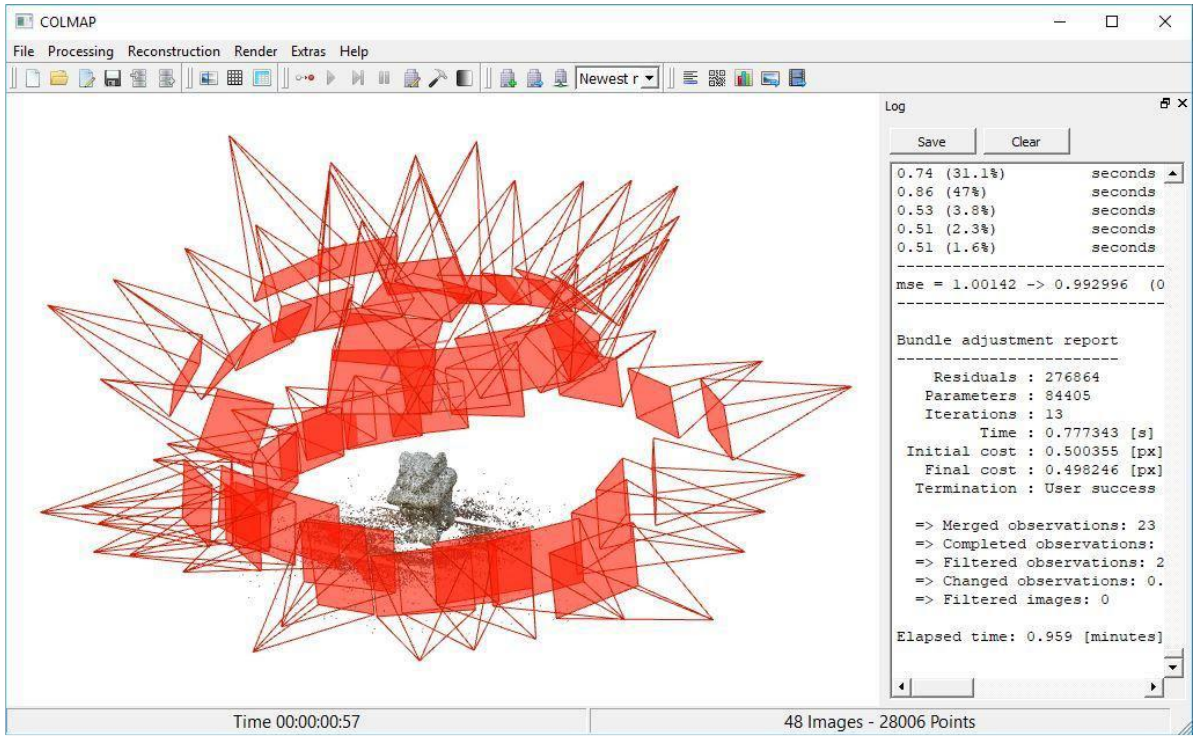
# Scale Invariant Feature Transform

Сматчив изображения между собой, можно, например, сделать панораму



# Scale Invariant Feature Transform

Или восстановить положение камер



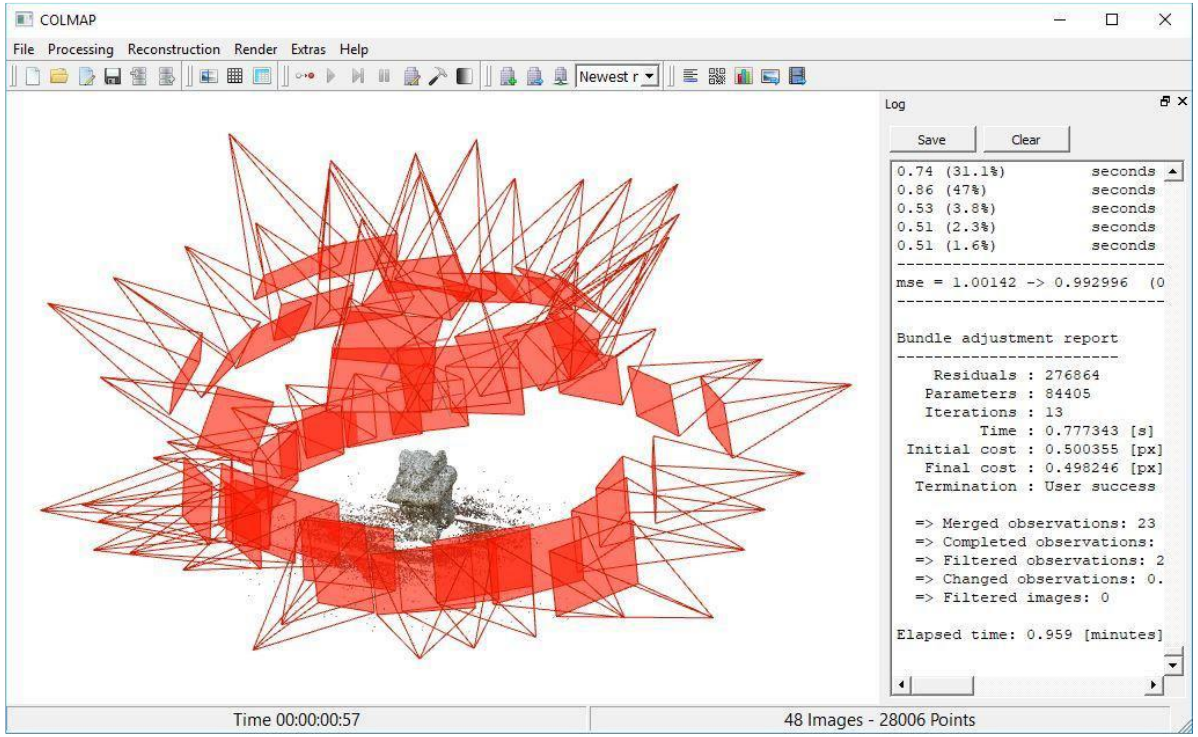
# Зачем Deep Learning?

**Проблемы SIFT**

- Изменение угла обзора
- Похожие паттерны

# Зачем Deep Learning?

Нужно очень плотное покрытие кадрами



# Deep Learning – SuperPoint

Самостоятельное формулирование качественных правил расчета признаков – нетривиальная задача

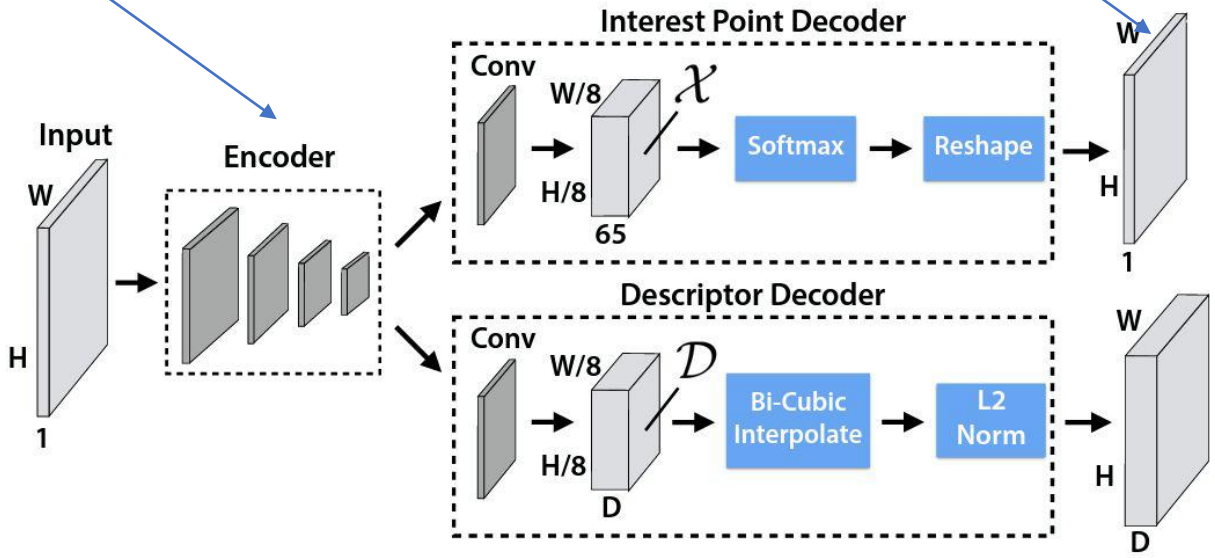
Давайте попросим нейронную сеть искать нам точки и дескрипторы, которые нам нужны

**Надежные и воспроизводимые**

# Deep Learning – SuperPoint

Классификатор – есть точка
или нет в данном пикселе

Самый привычный



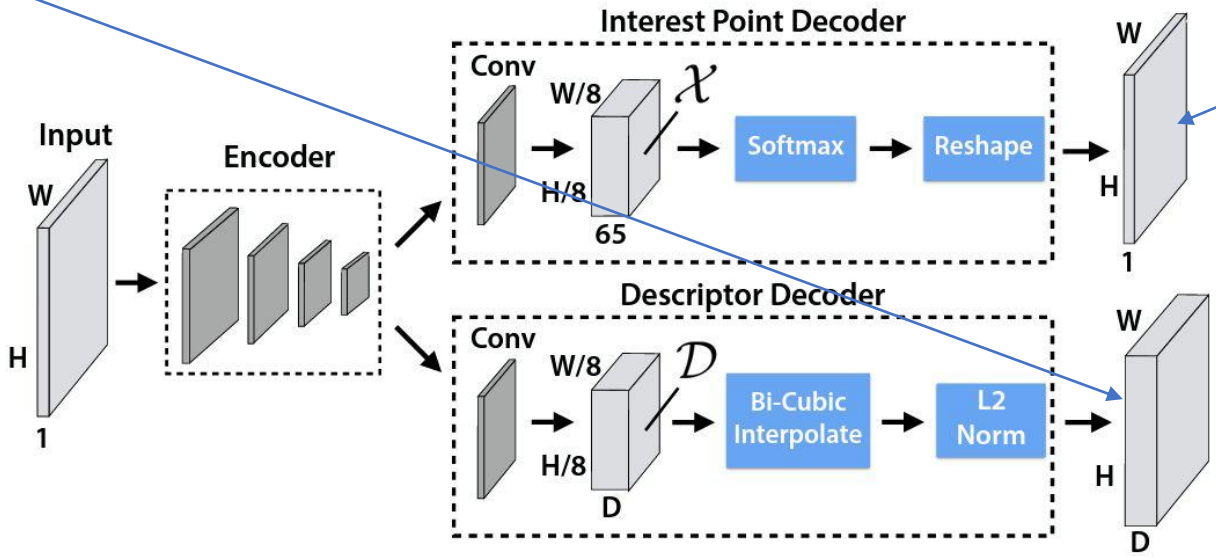

Дескриптор
для каждой точки

# Deep Learning – SuperPoint

Делаем векторы одинаковых точек близкими,
разных – далекими

$$l_d(d, s'; s) = \lambda_d * s * \max\left(0, m_p - d^T d'\right) + (1 - s) * \max(0, d^T d' - m_n)$$

Тут Hinge Loss



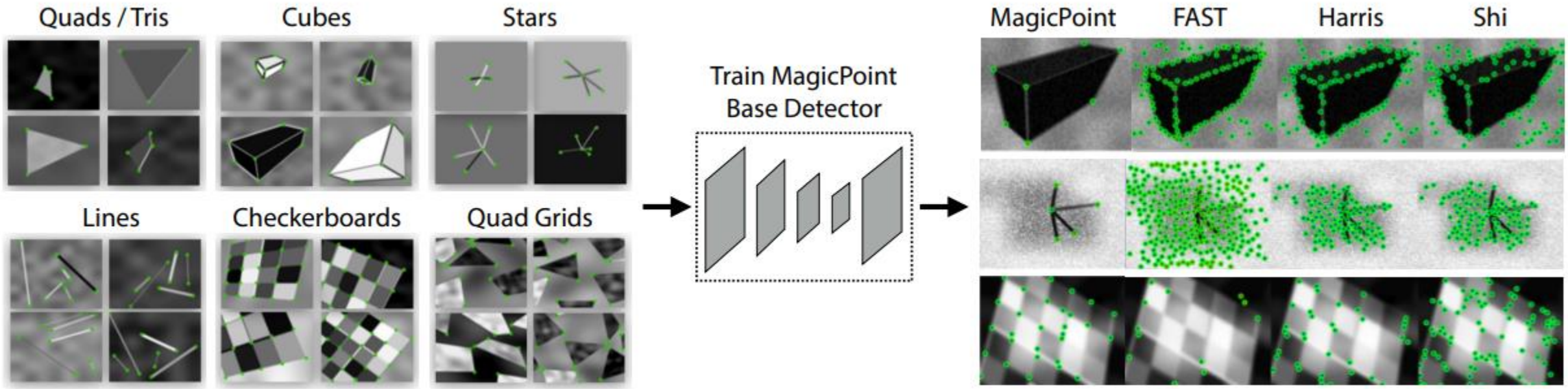

Тут кросс-энтропия

$$Loss = -\sum_{i=1}^{output\ size} y_i \cdot log\hat{y}_i$$

Если есть ключевая точка,
тяни к 1

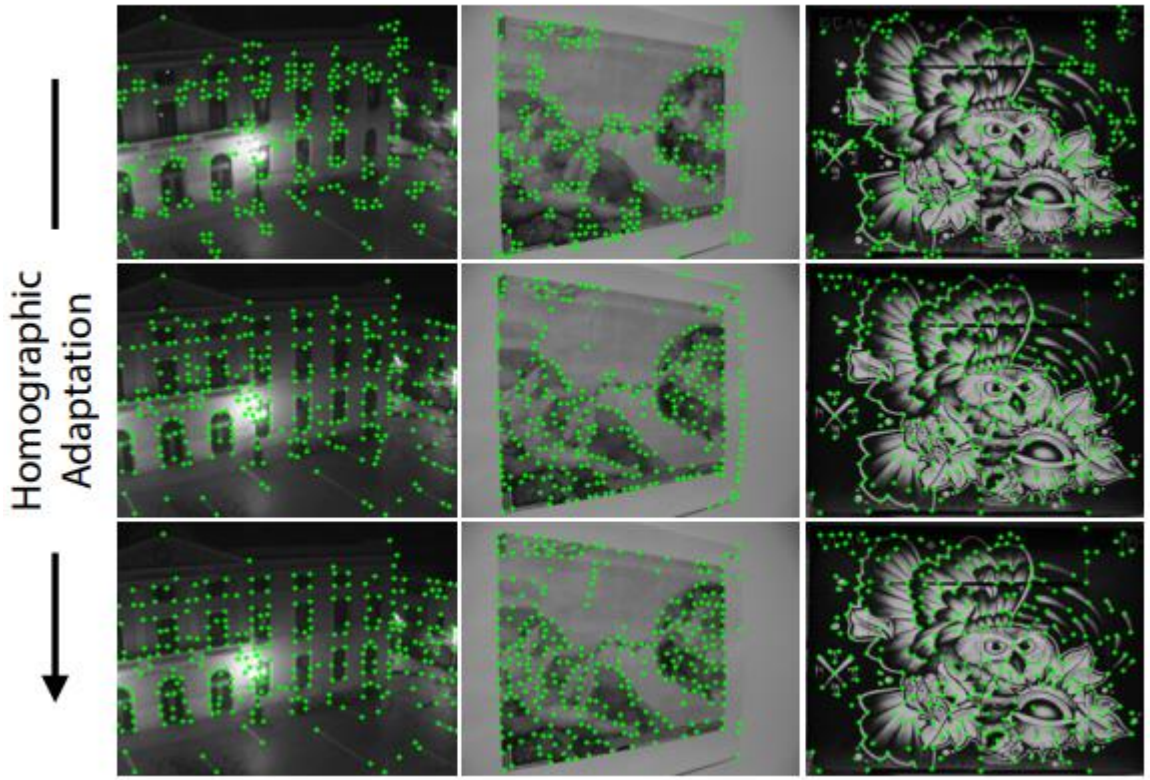
# Deep Learning – SuperPoint

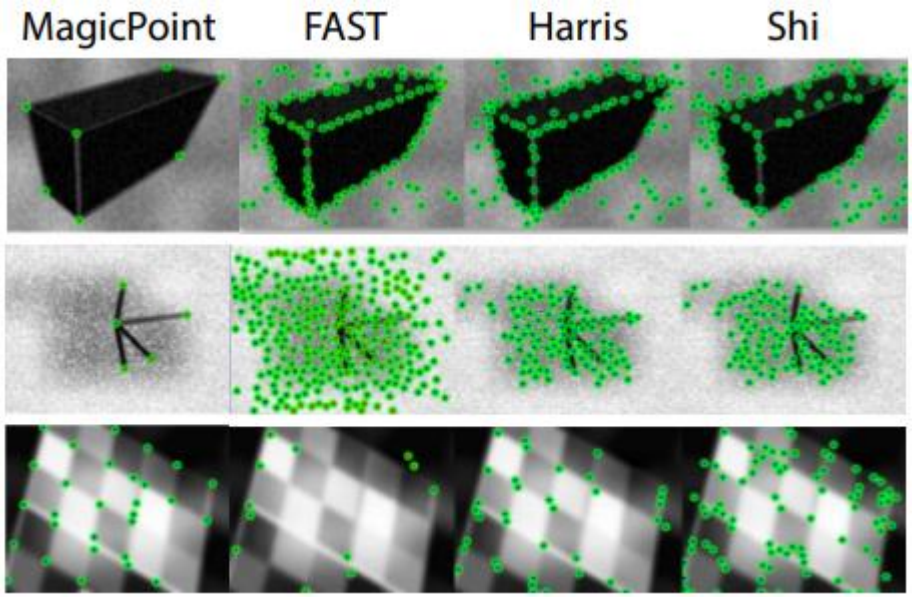
Шаг 1: синтетика



# Deep Learning – SuperPoint

Шаг 2: что-то выдает на реальных данных



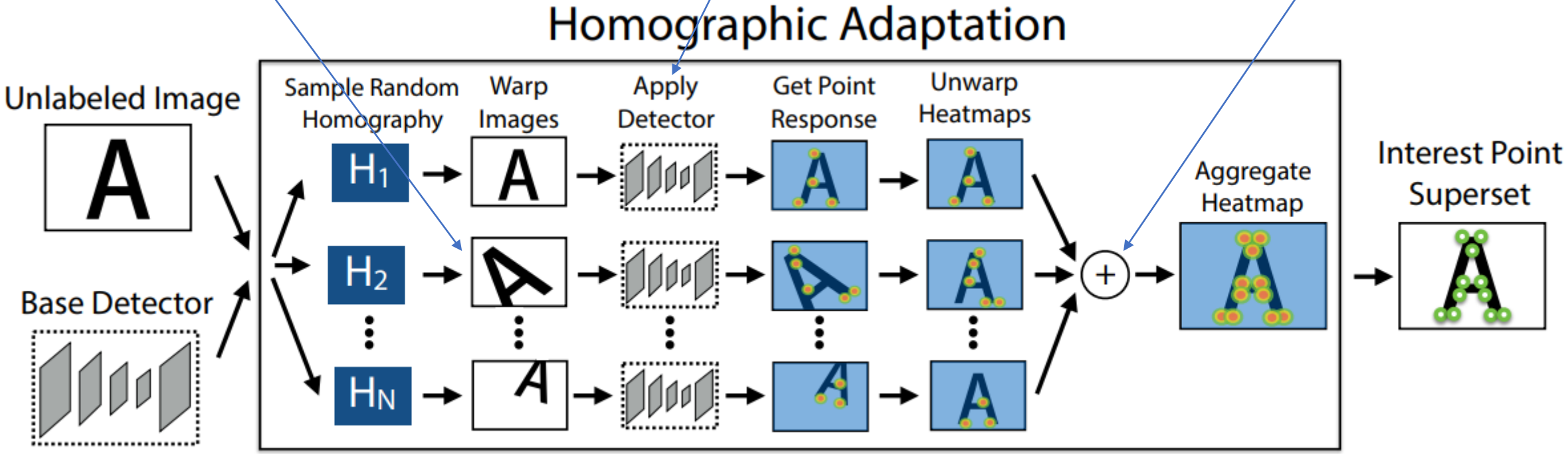



# Deep Learning – SuperPoint

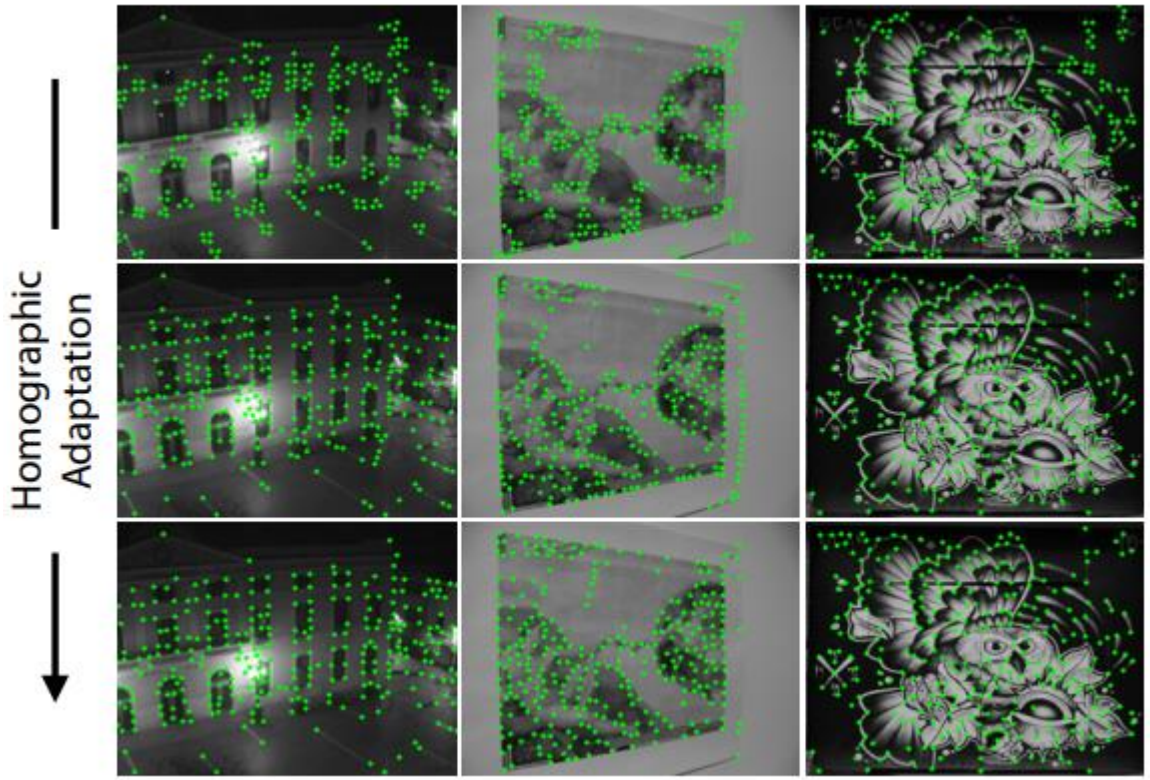
Крутим изображение

Прогоняем через сетку

Собираем все результаты вместе



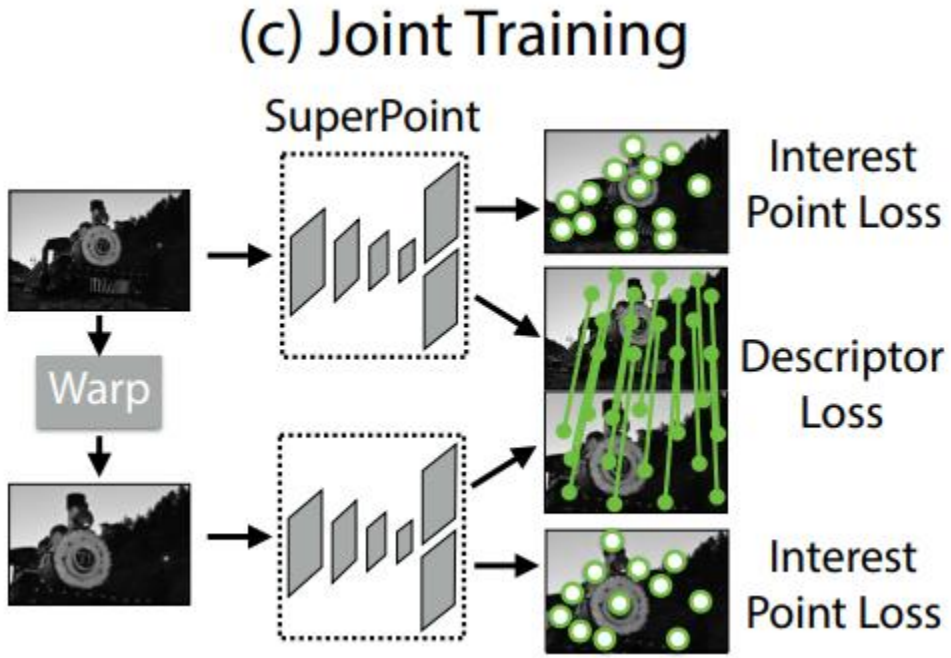
# Deep Learning – SuperPoint

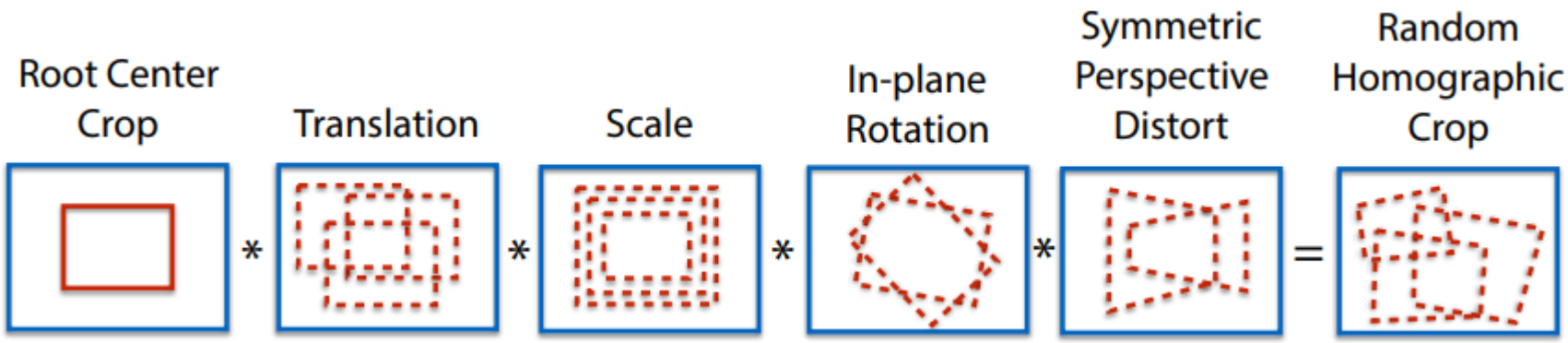



Теперь воспроизводится

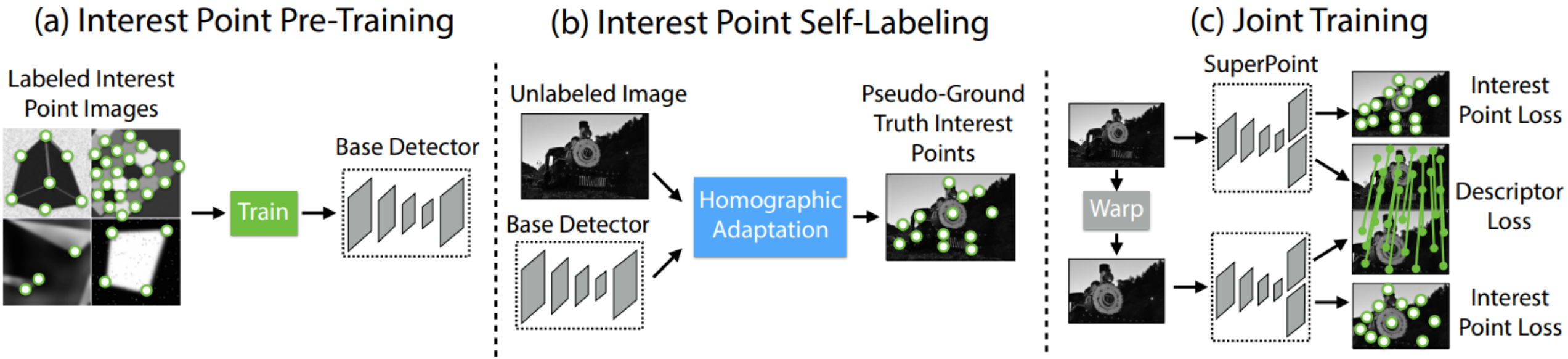
# Deep Learning – SuperPoint

Знаем, как варпилась картинка
Знаем, какая точка, куда перешла
Знаем, куда тянуть по loss



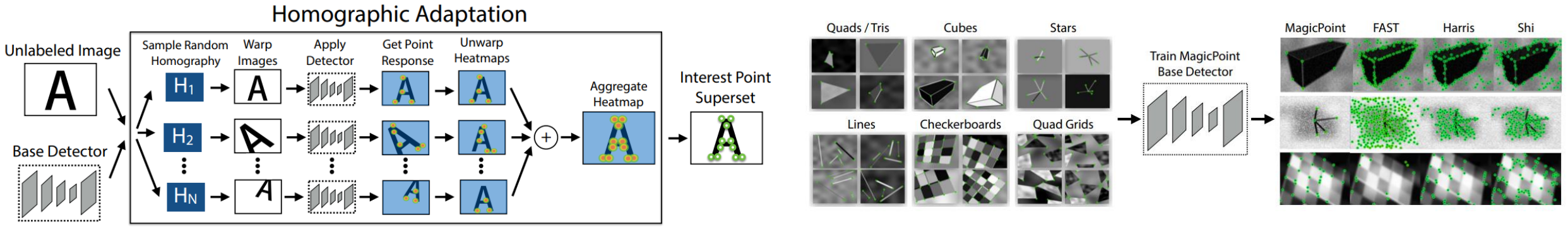
# Deep Learning – SuperPoint



# Deep Learning – SiLK

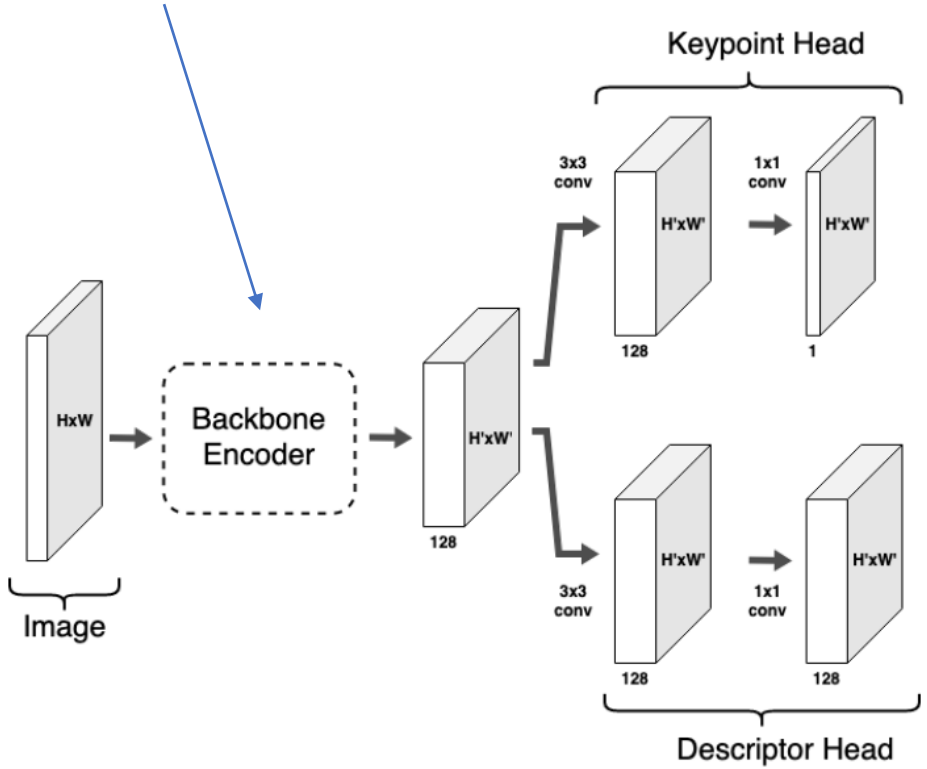
Генерация синтетики и потом homography adaptation – довольно неоднозначные этапы

Синтетика косвенно определяет, что мы будем называть надежной точкой



# Deep Learning – SiLK

Мини VGG без maxpool




$$L_{key} = BCE(q, y, c) + BCE(q', y, c')$$

$$L_{desc} = NLL(s, c, c') =$$

$$= -\frac{1}{N}\sum_{i=0}^{N-1} logP_{c_i \leftrightarrow c_i'} =$$

$$= -\frac{1}{N}\sum_{i=0}^{N-1} \left[logP_{c_i \rightarrow c_i'} + logP_{c_i' \leftarrow c_i}\right]$$

# Deep Learning – SiLK

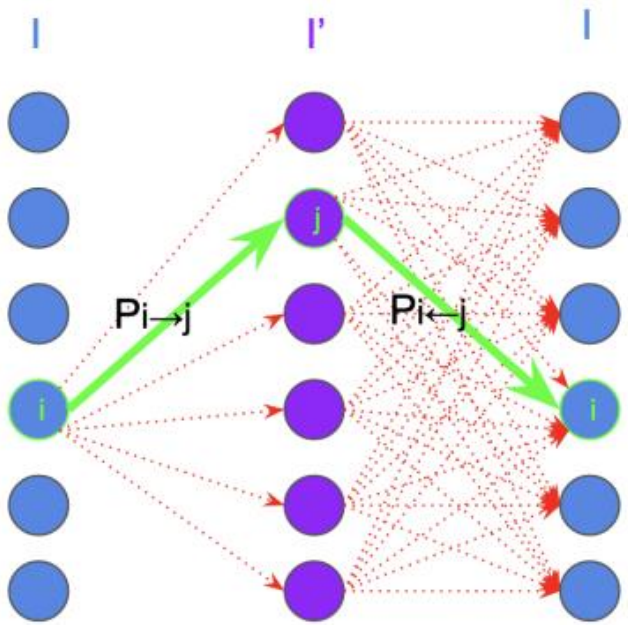
```
def loss(image_0, model):
    # get warped image and pixel correspondences
    image_1, corr_0, corr_1 = rand_homo(image_0)

    # apply image augmentations
    image_0 = augment(image_0)
    image_1 = augment(image_1)

    # extract dense descriptors and keypoints
    desc_0, kpts_0 = model(image_0)
    desc_1, kpts_1 = model(image_1)

    # compute similarity matrix
    sim_mat = cosim(desc_0, desc_1)

    # compute the descriptor loss
    # using ground truth correspondences
    loss_desc = nll(sim_mat, corr_0, corr_1)

    # measure matching success
    y = is_match_success(sim_mat, corr_0, corr_1)

    # compute keypoint loss
    # using current matching success
    loss_kpts = bce(kpts_0, y, corr_0)
    loss_kpts += bce(kpts_1, y, corr_1)

    return loss_desc + loss_kpts
```

Figure 2: Pseudo-code: learning keypoints from a single image.

# Deep Learning – SiLK

Первым идет Descriptor Loss

У нас есть дескрипторы
на оригинальной картинке и варпленой




$$L_{desc} = NLL(s, c, c') =$$

$$= -\frac{1}{N}\sum_{i=0}^{N-1} logP_{c_i \leftrightarrow c_i'} =$$

$$= -\frac{1}{N}\sum_{i=0}^{N-1} \left[logP_{c_i \to c_i'} + logP_{c_i' \leftarrow c_i}\right]$$

Мы стягиваем нужный дескриптор

с «A» ⟶ к «B»

с «B» ⟶ к «A»

Это называется cycle-consistency

# Deep Learning – SiLK

Получается, мы учим дескрипторы в Dense манере

$$P_{i\leftrightarrow j} = P_{i\rightarrow j}\, P_{i\leftarrow j} \longrightarrow s_{ij} = \mathrm{cosim}\left(d_i, d'_j\right) = \frac{\langle d_i, d'_j \rangle}{\sqrt{\langle d_i, d_i, \rangle \left\langle d'_j, d'_j \right\rangle}}$$

# Deep Learning – SiLK

Софтмакс по столбцу и Софтмакс по строке

$$s_{ij} = \text{cosim}(d_i, d'_j) = \frac{\langle d_i, d'_j \rangle}{\sqrt{\langle d_i, d_i, \rangle \langle d'_j, d'_j \rangle}}$$

$$P_{i \to j} = \frac{e^{\frac{s_{ij}}{\tau}}}{\sum_k e^{\frac{s_{ik}}{\tau}}}$$ directional probability of matching $d_i$ to $d'_j$

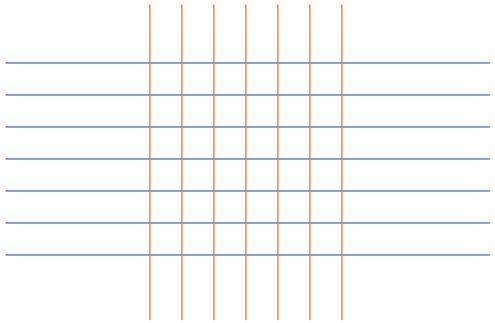
$$P_{j \leftarrow i} = \frac{e^{\frac{s_{ij}}{\tau}}}{\sum_k e^{\frac{s_{kj}}{\tau}}}$$ directional probability of matching $d'_j$ to $d_i$

$$L_{desc} = NLL(s, c, c') =$$

$$= -\frac{1}{N} \sum_{i=0}^{N-1} logP_{c_i \leftrightarrow c'_i} =$$

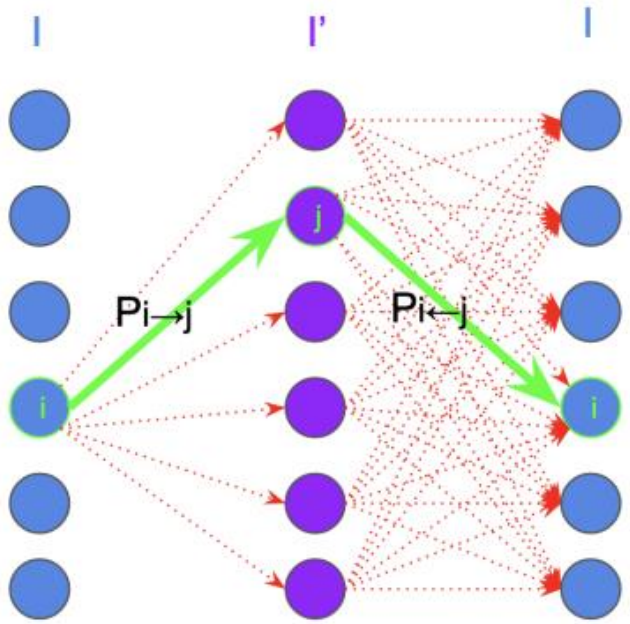
$$= -\frac{1}{N} \sum_{i=0}^{N-1} \left[ logP_{c_i \to c'_i} + logP_{c'_i \leftarrow c_i} \right]$$

$$P_{i \leftrightarrow j} = P_{i \to j} \, P_{i \leftarrow j}$$



# Deep Learning – SiLK

Если правильная пара дескрипторов сматчилась,
то будем там ставить единицу.
Он у того, и у этого – самый близкий по расстоянию дескриптор




$$y_i = 1\left[s_{c_i c_i'} \geq \max_k\{s_{c_i k}\}\right] 1\left[s_{c_i c_i'} \geq \max_k\left\{s_{k c_i'}\right\}\right]$$

# Deep Learning – SiLK

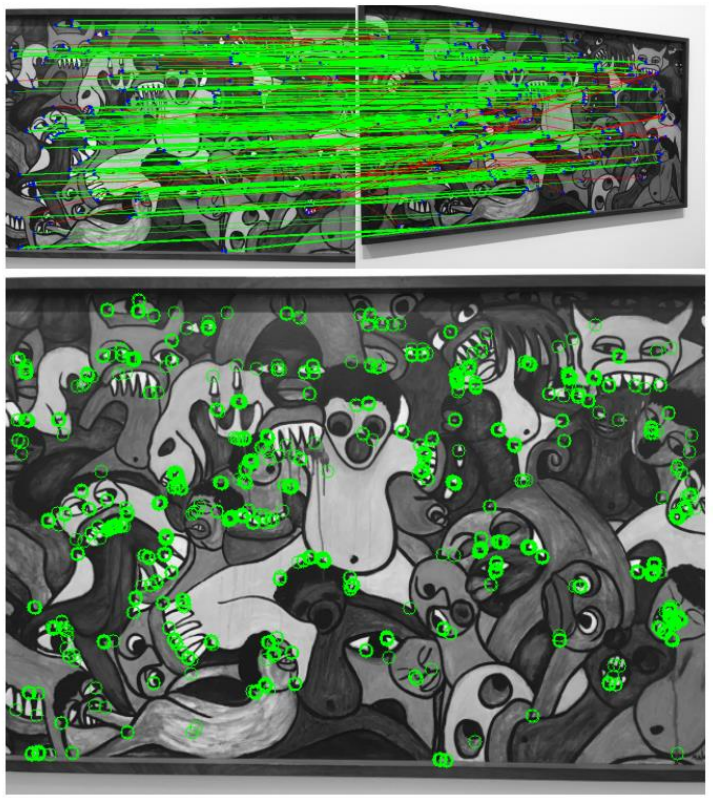
Голова для классификации будет учиться предсказывать эти единицы.
То есть, те пиксели, где эмбеддинги должны быть хорошие

$$y_i = 1\left[s_{c_i c_i'} \geq \max_k\{s_{c_i k}\}\right] 1\left[s_{c_i c_i'} \geq \max_k\left\{s_{k c_i'}\right\}\right]$$

$$BCE(q, y, c) = -\frac{1}{N}\sum_{i=0}^{N-1}\left[y_i \log\sigma\left(+q_{c_i}\right) + (1 - y_i)\log\sigma(-q_{c_i})\right]$$

# Deep Learning – SiLK

Теперь надежность и повторяемость косвенно следуют из обучения.
Не нужно самим на первом этапе выдумывать критерий надежности



# Выводы

**Что такое ключевые точки?**

Повторяемые и надежные точки на изображениях, которые мы можем сравнить между собой и таким образом «позиционно» сравнить изображения

Нулевой этап при переходе от 2D в 3D

# Выводы

**SIFT**

Базовый подход, с него следует начать

- Работает в случаях полного покрытия кадров,
  в очень разноообразной сцене

Не работает, если

- Большая однородность сцены = одинаковые дескрипторы в разных точках
- Малая плотность снимков = сильно отличаются углы обзоров объектов

# Выводы

**SuperPoint и SiLK**

Нейросетевой аналог SIFT

- Работает в очень разнообразной сцене. Легко учить. Все учится в self-supervised

Не работает, если

- Большая однородность сцены = одинаковые дескрипторы в разных точках

# Выводы

**LoFTR и SuperGlue**

Продвинутые подходы на трансформерах

# Использованные материалы

- Материалы курса 3DCV от DeepSchool
- Документация OpenCV

# Примеры работ

- A comparative analysis of pairwise image stitching techniques for microscopy images
- A monocular SLAM system based on SIFT features for gastroscope tracking
- Real time SLAM in Endoscopy Applications
- KDD 2019: SLAM Endoscopy Enhanced by Adversarial Depth Prediction - Reconstruction of Colon Phantom
- EndoSLAM dataset and an unsupervised monocular visual odometry and depth estimation approach for endoscopic videos
- CudaSIFT-SLAM: multiple-map visual SLAM for full procedure mapping in real human endoscopy