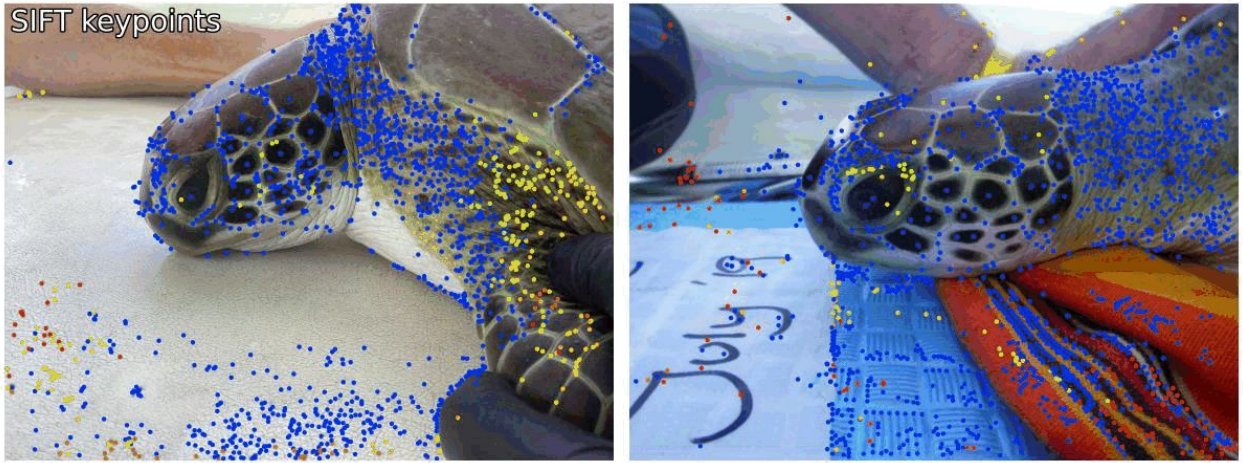
Feature Matching with LightGlue



FruitPunch AI

Why we're here

Metric Learning

- Learn to identify turtles in feature space
- Convolutional Neural Networks (CNN)
- Lightweight
- 16% accuracy

sub-team members

Davide	Rodrigo
Ahmed	Thor

LightGlue

- Learn to match keypoints in images
- Two-step algorithm
- CNN + Transformers
- 100% accuracy

Eelke	Miruna
Laurenz	Marcus

LoFTR

- State-of-the-art keypoint matching
- Transformer-based
- Big and heavy
- 84% accuracy

Enrico	Deb
Lennart	Sonny

Slide courtesy of Davide Coppola

Who we are

Marcus Leiwe
Consultant Data Scientist,
Metacell

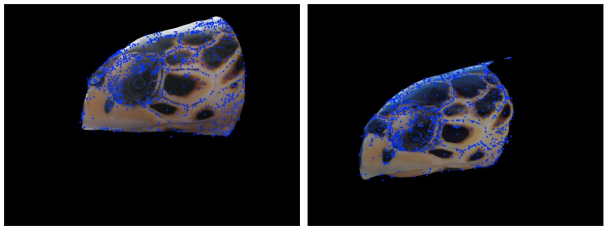
METACELL

Laurenz Schindler
Student,
Radboud University

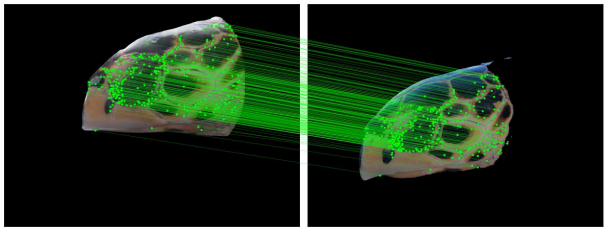
Radboud Universiteit

NB With help from Eelke Folmer and Miruna Morarasu

Keypoint Feature Extraction → **LightGlue Matcher** → **Identifying the best matches**



What are keypoints?
Classical approaches
Deep Learning



Output
LightGlue architecture
Augmenting matching

Donatello
64%

Raphael
16%

Michelangelo
11%

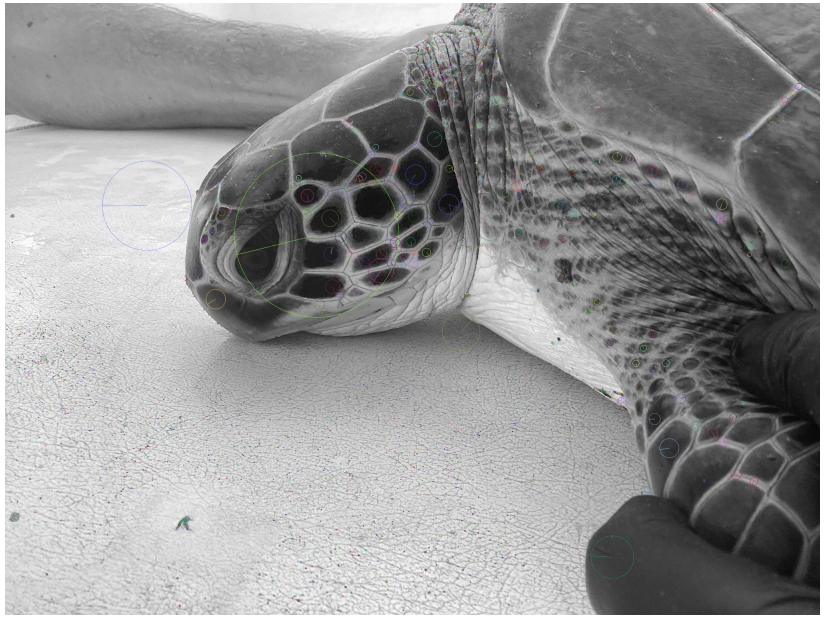
others...

Feature Extraction is ...

“Converting key parts of the image into vectors.”

But...

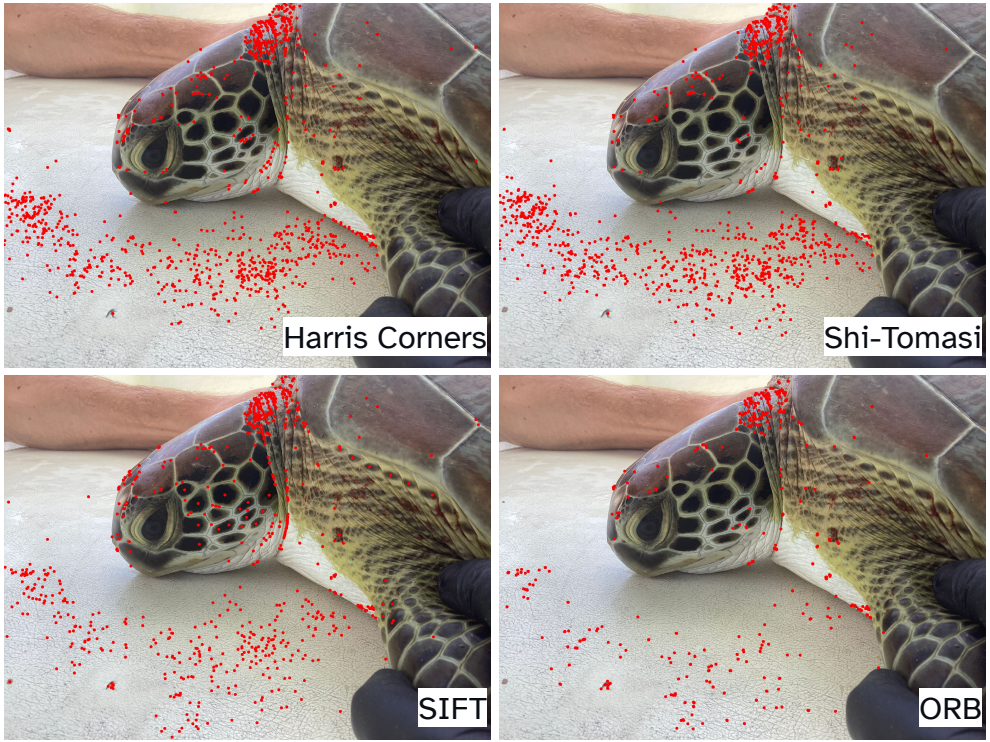
1. How do you find what is a key part (aka “keypoint”)?
2. What information do you store in the vector (“descriptors”)?



Classical Feature Extraction

Relies on hard rules to find keypoints.

SIFT is the gold-standard for classical techniques



Tabular Summary for Reference

	Strengths	Weaknesses
Harris Corners	• Good at finding corners in an image • Less sensitive to noise and different scales	• Computationally more expensive • Sensitive to image rotation
Shi-Tomasi	• Improvement over Harris Corner Detector by changing the scoring	• Computationally more expensive • Sensitive to image rotation
SIFT	• Scale-invariant feature detection • Robust to changes in illumination and viewpoint	• Computationally expensive • Requires a lot of memory (we used batching)
SURF (Patented $)	• Faster than SIFT • Robust to changes in scale and rotation	• Computationally more expensive than FAST and BRIEF • Sensitive to changes in illumination
FAST	• Fast corner detection method • Good for real-time applications	• Sensitive to scale and rotation changes • Less robust than SIFT and SURF
BRIEF	• Computationally efficient • Good for real-time applications	• Sensitive to scale and rotation • Less robust than SIFT and SURF
ORB (Orientated fast, Rotated Brief)	• Fast and efficient • Combines features of FAST and BRIEF	• Not as distinctive as SIFT and SURF • Less robust than SIFT and SURF

Good introduction is here : https://towardsdatascience.com/image-feature-extraction-traditional-and-deep-learning-techniques-ccc059195d04

SIFT - Identifying Features

Process

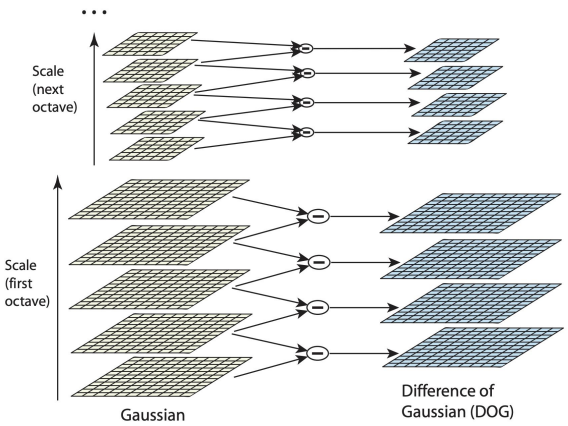
1. Grayscale
2. Scale-space Extrema Detection
3. Keypoint Refinement
4. Orientation Assignment and Refinement
5. Descriptor Computation

Take home message:
“SIFT works by identifying areas of high contrast that are insensitive to scaling”

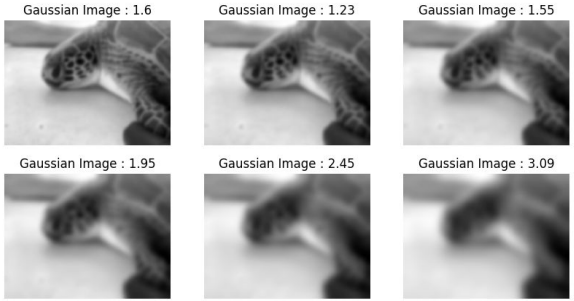
Check out OpenCV’s description for a more detailed description, or the original paper

SIFT - Scale-space Extrema Detection

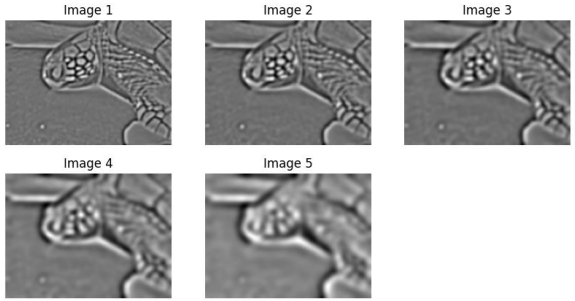
Difference of Gaussians highlight edges of the image at different **Octaves** (scales)



Example Octave



Example Differences of Gaussians

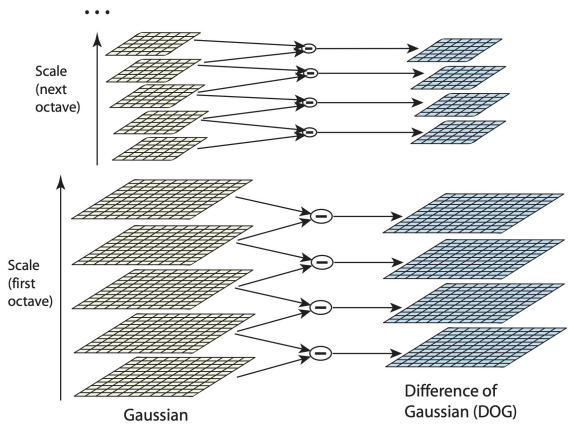


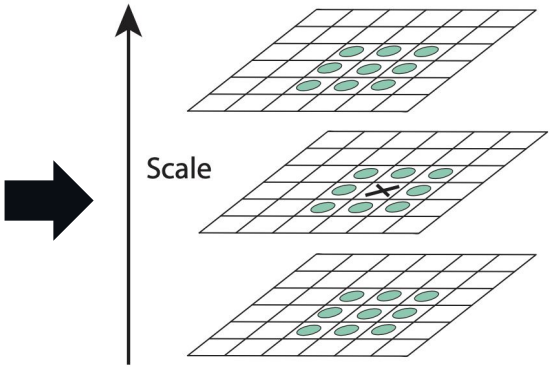
Octaves = Recommended number is 4 but this is tunable

Russ Islam, https://medium.com/@russmislam/implementing-sift-in-python-a-complete-guide-part-1-306a99b50aa5

SIFT - Scale-space Extrema Detection

Finding all **minima and maxima (extrema)** to generate keypoints





Further refined by...

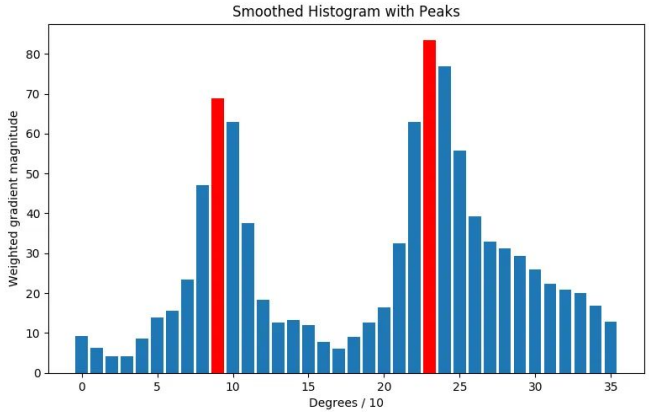
1. Taylor series expansion to locate the extrema in it's octave.
2. Intensity thresholding (<0.03 = no keypoint)

X must be the lowest or highest value compared to its neighbours (n=26) to be a provisional keypoint

SIFT - Orientation Assignment

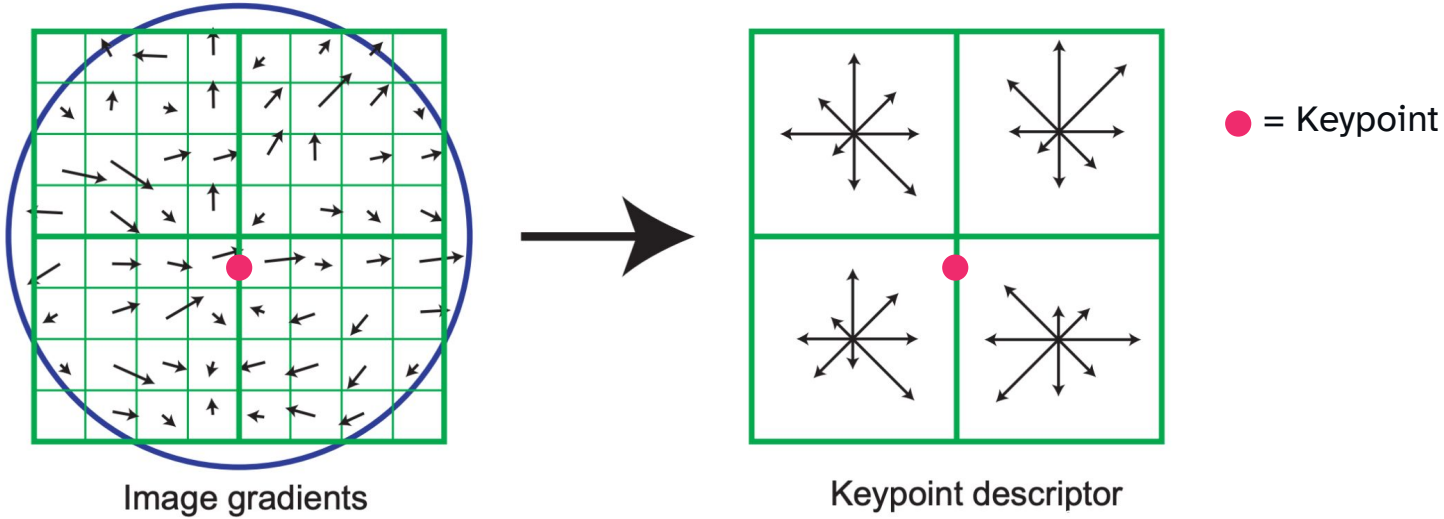
An **orientation** also is assigned to each key-point. This is done by extracting the neighborhood around the key-point and creating a **orientation histogram**, the peak of the histogram is used as the orientation.

No clear peak = no keypoint
2 peaks = 2 keypoints



Credit: Russ Islam, https://medium.com/@russmislam/implementing-sift-in-python-a-complete-guide-part-2-c4350274be2b

SIFT - Descriptor



128 point tensor from an orientation-corrected 8x8 grid that **describes the direction and magnitude of gradients**

Deep Learning Feature Extraction

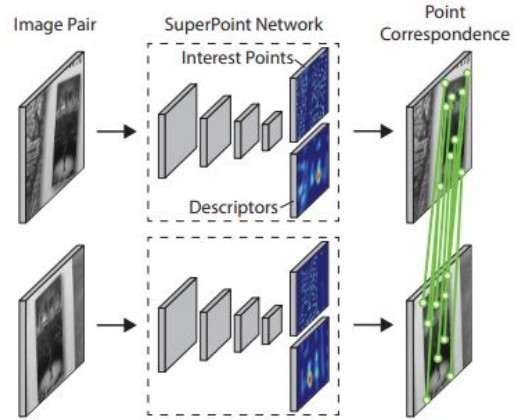
- Hand-crafted methods rely heavily on assumptions and heuristics
  - No guarantee for efficiency or robustness

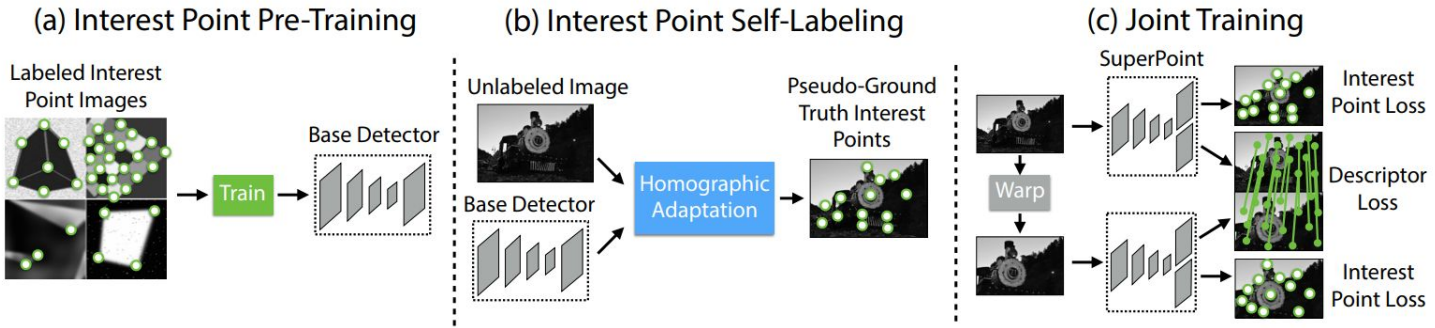
➡ **Data-driven** approach to keypoint extraction

- Goal: Improve Keypoint location and Descriptor embedding
- 3 supported implementations for LightGlue:
  - SuperPoint
  - DISK
  - ALIKED
  - (DoG HardNet)

SuperPoint

CNN for computing keypoint-locations and descriptors in single forward pass





Check out the paper here

DISK

- Reinforcement Learning (RL) for keypoint extraction
  - Probabilistic framework in training procedure
- U-Net (CNN) based architecture
  - output channels for both keypoints and descriptors -> computed in one forward pass

Training Procedure:

1. obtain feature maps $\mathbf{K}$ (keypoints) and $\mathbf{D}$ (descriptors) for images $\mathbf{A}$, $\mathbf{B}$
   - find most probable feature locations in $\mathbf{K}$
   - combine with corresponding descriptors in $\mathbf{D}$, obtaining $\mathbf{F}$ (feature set)
2. match feature-sets $\mathbf{F}_{\mathbf{A}}$, $\mathbf{F}_{\mathbf{B}}$ based on Euclidean distance
3. Reward correct matches, penalize incorrect matches
   - based on ground-truth pose and depth-maps

Check out the paper here

ALIKED

- Improve geometric invariance using Deformable Convolution Networks (DCN)
- Separation of keypoint and descriptor extraction through Sparse Deformable Descriptor Head (SDDH)

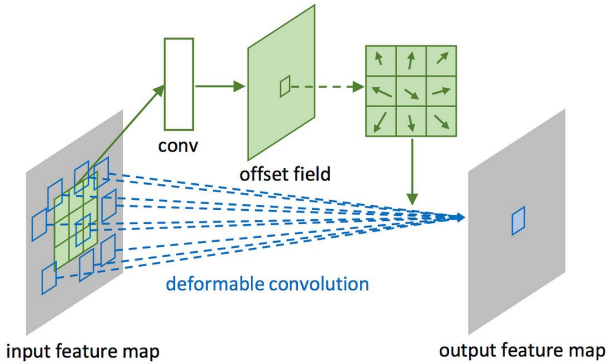
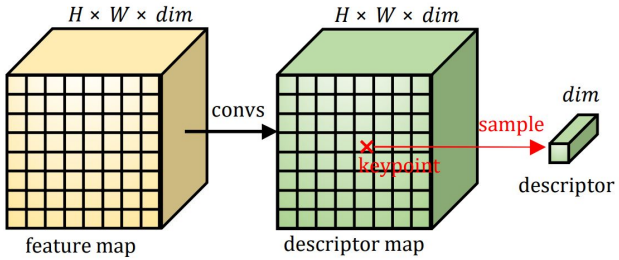


Figure 2: Illustration of 3×3 deformable convolution.

Check out the paper here

ALIKED

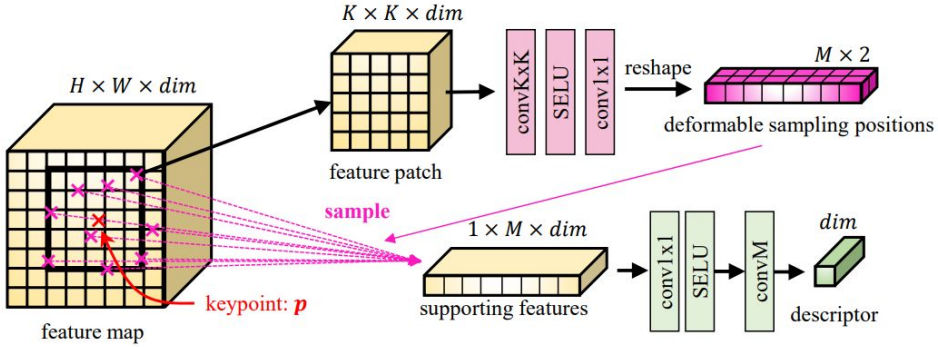
Dense Descriptor Map



expensive, often downsampled -> lower descriptor quality

vs.

SDDH



lower computational efforts, more powerful descriptors

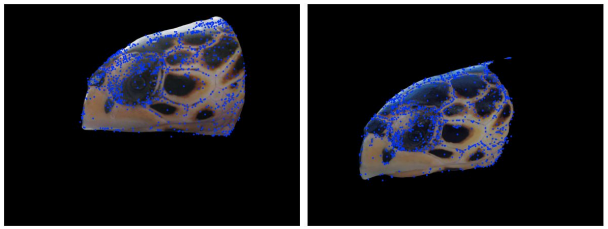
Check out the paper here

Deep Learning Feature Extraction in practice

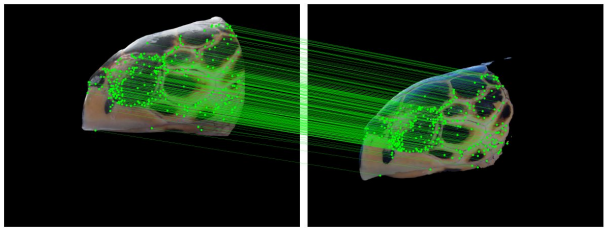
- There is no 'best' approach
  - (SIFT worked best for us)
- Trained/Evaluated on landmark datasets
  - Performance comparisons limited to this domain
  - Not necessarily translatable to your problem

Feel free to experiment!

Keypoint Feature Extraction → LightGlue Matcher → Identifying the best matches



Four Algorithms
Superpoint, DISK, aliked, SIFT



Output
Matching points and confidence scores

Donatello
64%

Raphael
16%

Michelangelo
11%

others...

LightGlue: State-of-the-art feature matching

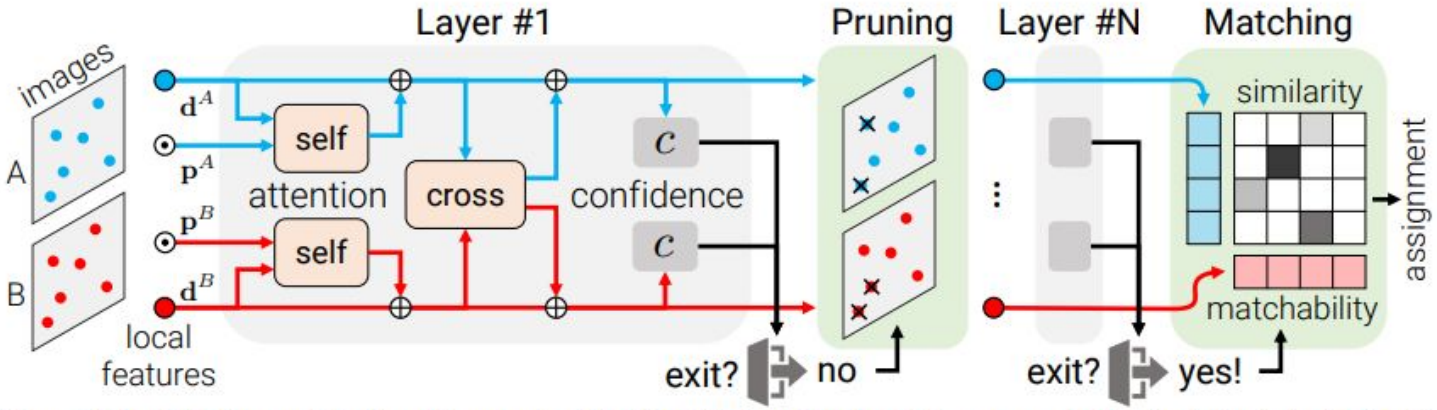
- Reworked version of SuperGlue[1]
- Sparse feature matcher:
  - Computes matchability given two Keypoint/Descriptor-Sets

Improvements:

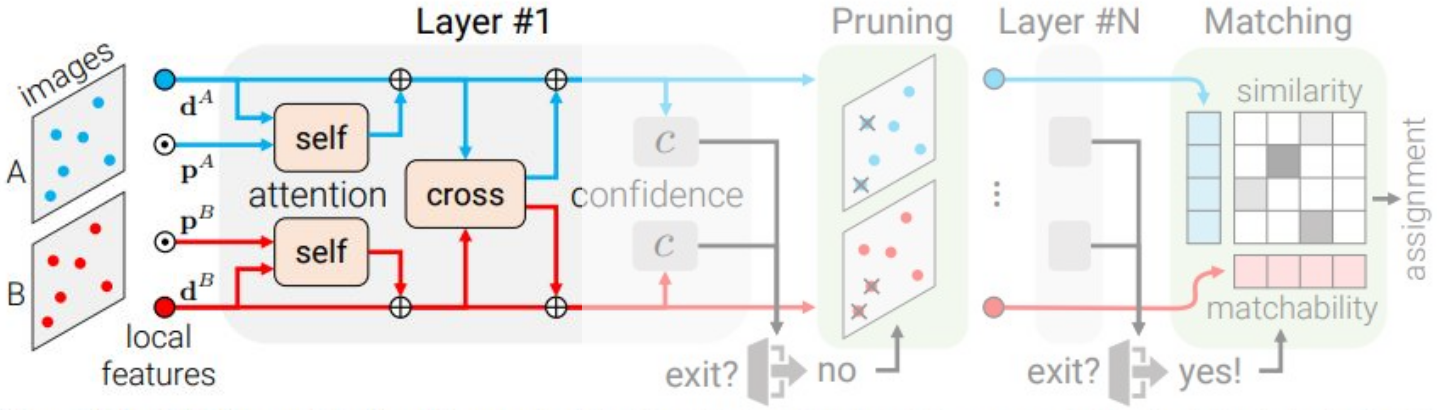
- **Efficiency** - Premature stopping at varying depth into the network
- **Accuracy** - Change in positional encoding of keypoints yields a more stable result
- **Training** - Reworked architecture leads to easier training

[1] Sarlin, Paul-Edouard, et al. "Superglue: Learning feature matching with graph neural networks." *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020.

Model Architecture: Overview



Model Architecture: Updating descriptors

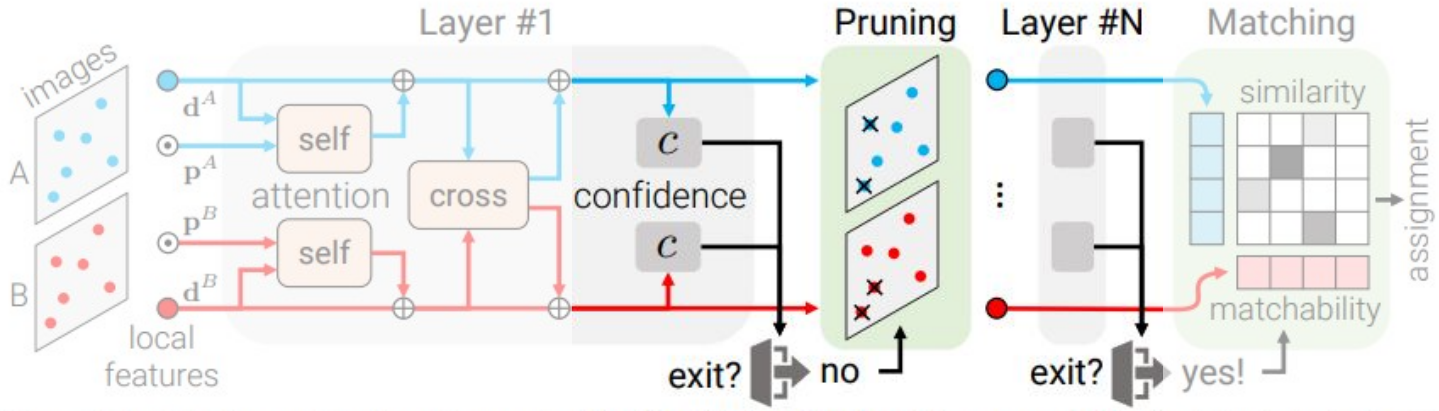


Self-Attention: every point attends to all points extracted from one image + rotary (relative) positional encoding

Cross-Attention: every point attends to all points extracted from the other image, positional information not added

Correspondence matrix between the local features, composed of matchability and similarity scores

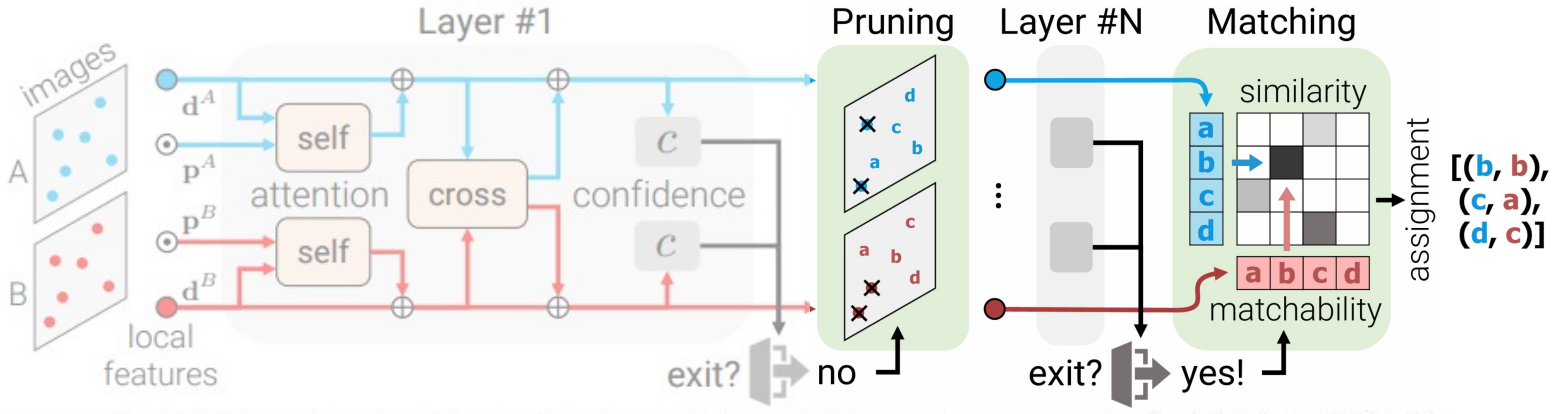
Model Architecture: Exit criterion



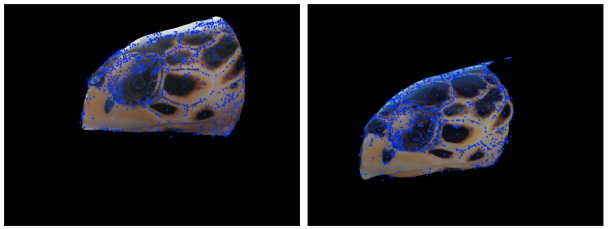
Confidence classifier: trained to predict whether current feature representations are 'final', exit if sufficient ratio of points is confident enough

Point Pruning: points predicted confident and unmatchable are removed

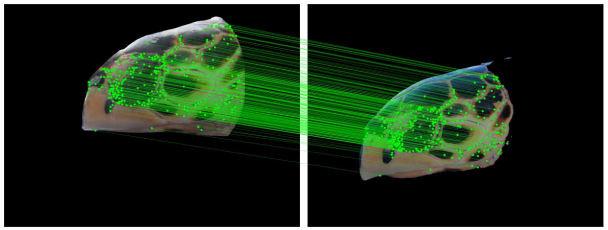
Model Architecture - Output



Keypoint Feature Extraction → LightGlue Matcher → Identifying the best matches



Four Algorithms
Superpoint, DISK, aliked, SIFT



Output
Matching points and confidence scores

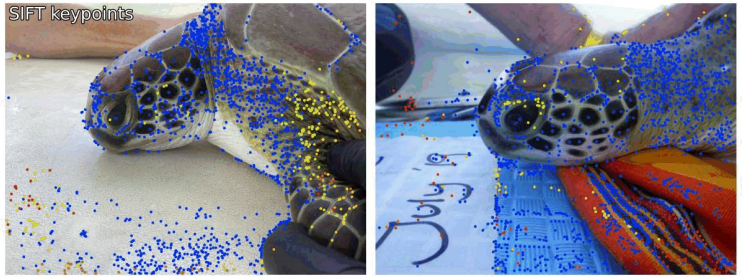
Donatello
64%

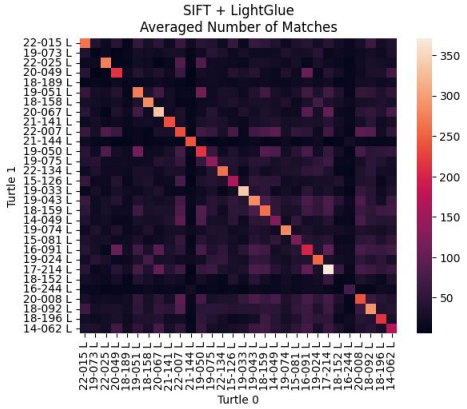
Raphael
16%

Michelangelo
11%

others...

Initial output: # of matching keypoints





The base method for comparison is finding the image with the most matching points.

This gave us a 77% accuracy score on a trial dataset.

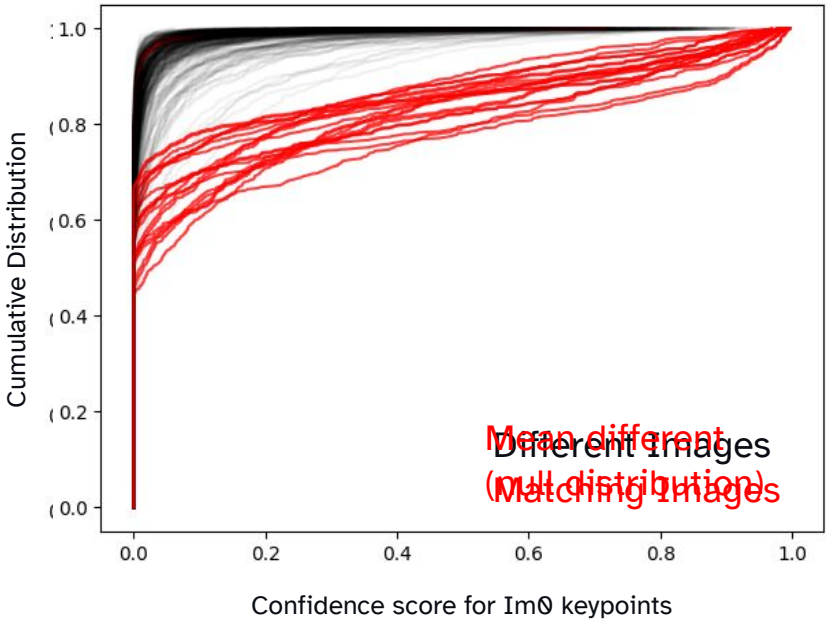
Possible reasons for suboptimal performance...

- # keypoints is not consistent across images.
- # thresholding scores also throws away a lot of information.

In other words we are throwing a lot of information away.

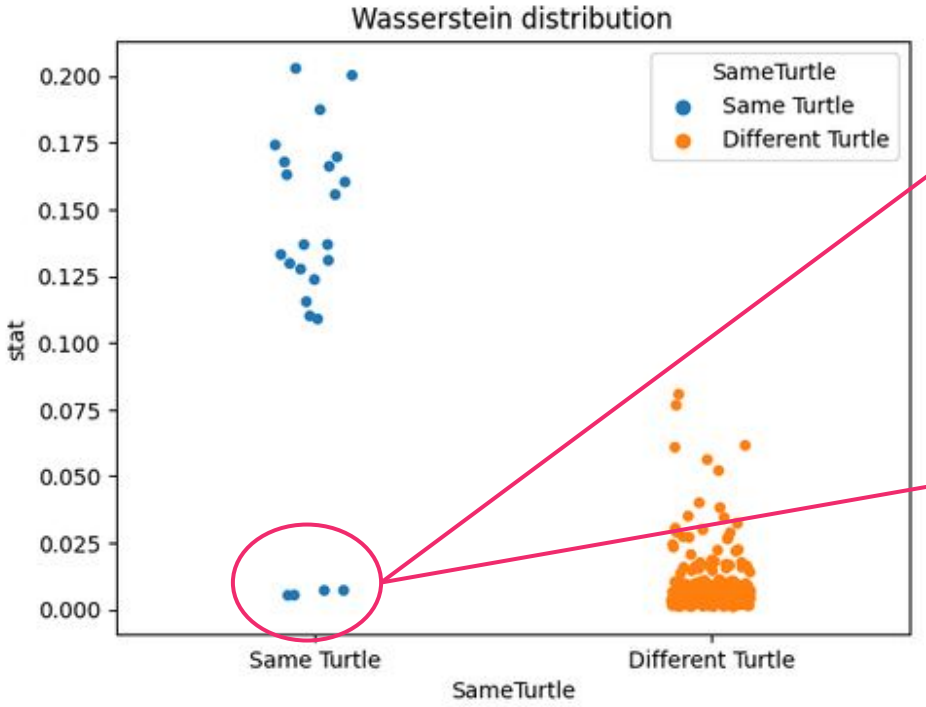
Solution: Look at the **distributions of scores** instead

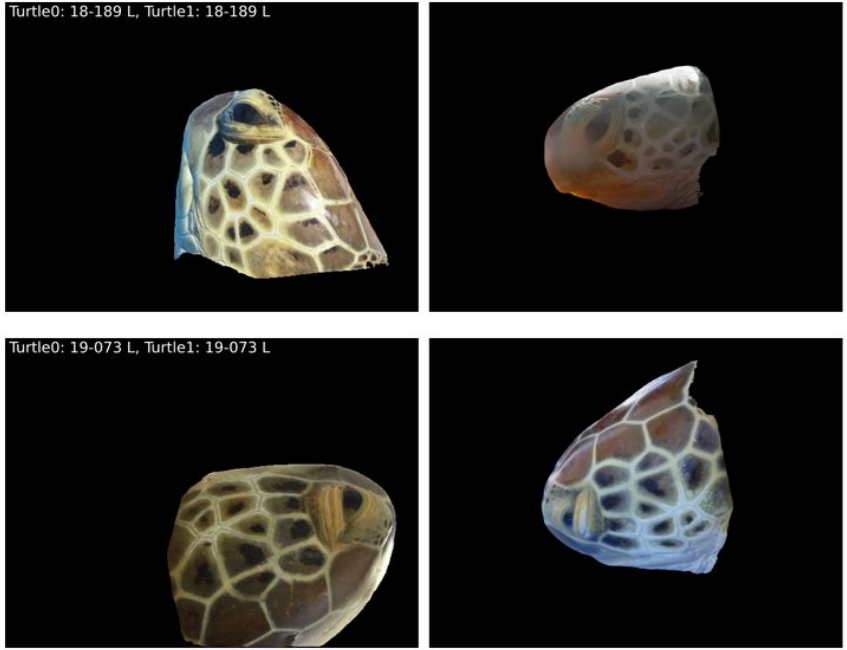
Matching images (Turtles) have different distributions



- Cumulative frequency plots
- Keypoint scores from matching images are much higher (curve shifts right/down)
- So what we are really doing is comparing two distributions.
- With the Wasserstein distances we can calculate how much “energy” is required to match the null distribution
  - Large number = Matching image
  - Small number = Non-matching

Wasserstein distribution scores accurately split the data





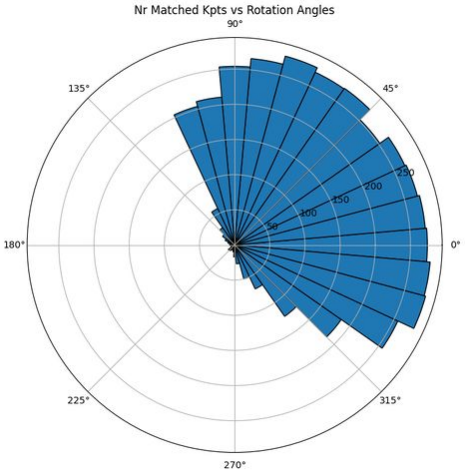
These four images are wrongly paired/rotated

Demo Notebook

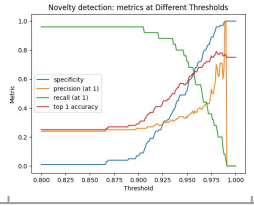
https://colab.research.google.com/drive/1094zv0A5CaCNDs7z8Ae8fN6Og3YcZ0iY?usp=sharing

Additional factors to consider

- Segmentation
- Image quality
- Rotational angle, we found large angle differences (~>90 degrees) tricky
- Processing time
- Novelty detection is a separate question

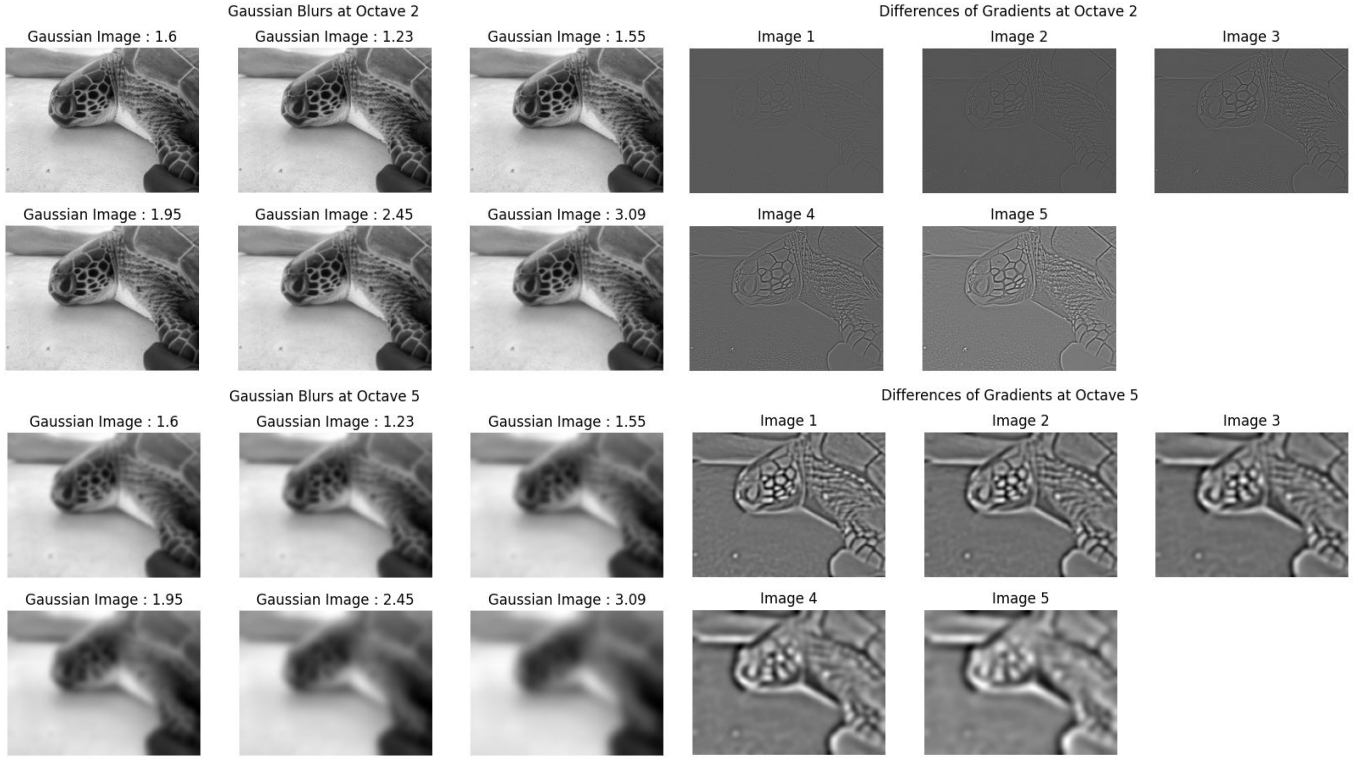


Model comparisons

Model / Algorithm	Accuracy at 1/3/5 (%)	Novelty detection (%)	Inference Time on 1 query (s)	👍Advantages	👎 Disadvantages
Metric Learning (ConvNeXt-Small +/Circle Loss)	8/12/16	76	<10s [Colab]	Fast inference with current size of reference set	Hard to train with too few hard pairs
LightGlue	100/100/100	0.9 accuracy	~ 30s [T4 on Colab]	Fast, accurate with current reference set.	Requires a GPU for quick evaluations.
LoFTR	64/80/84		1,5 hour [locally on macbook (mps)]	Fine grained/Low contrast feature extraction (not needed for turtles)	Inference time is very long and therefore not suitable for turtle reID

EXTRA SLIDES

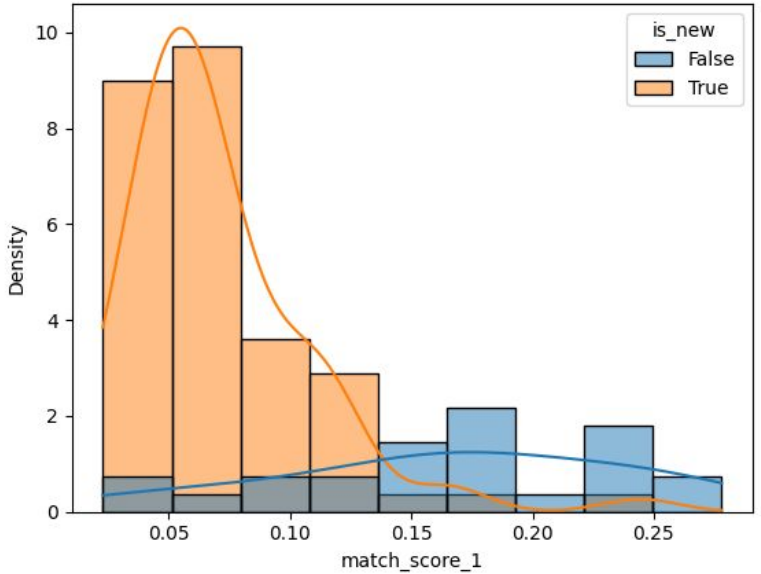
SIFT: The effect of DoG at different octaves



Identifying Novel turtles

- Novel turtles should generate low wasserstein distance scores for every comparison.
- If we model the probability distribution of matching scores we can come up a prediction/confidence on whether this is a new turtle or not.
- NB 90% accurate on identifying new turtles, so not perfect

Largest Wasserstein score per image



Comparison with other approaches

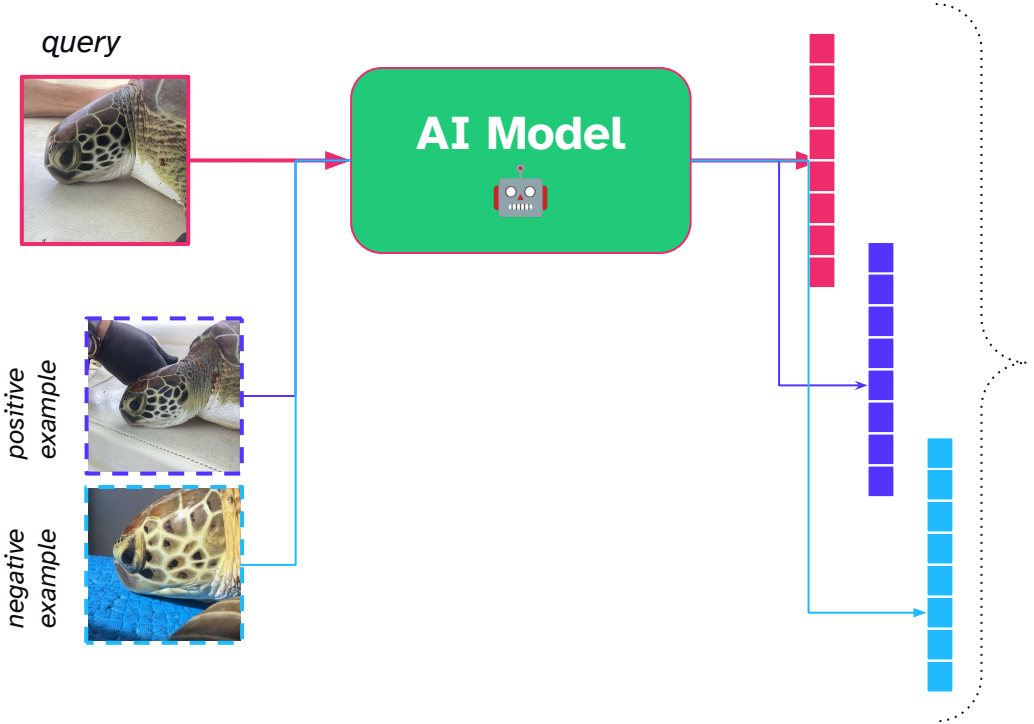
Metric Learning

- Utilize Convolutional Neural Networks (CNN) to find discriminative feature (embedding) space
- One possibility: *Triplet Loss*
  - For every training image, two additional images are provided:
    - positive example (same instance, different image)
    - negative example (different instance)
  - requires lots of (reoccurring) data
- **Fast, but difficult to train**

LoFTR (dense matcher)

- Extraction and matching of keypoints in one model
- Keypoints are extracted densely -> many more keypoints to be matched
- State-of-the-art performance
- **Long inference time**

Metric Learning: learning similarity in the feature space



many metric learning approaches in literature

experiments with various combinations of methods to find the best

Slide courtesy of Davide Coppola