LightGlue: High Performance Feature Mapping

AI Tinkerers HK and GBA:
Deep Dive @ AUKI Labs

Marcus Leiwe, Leiwe & Partners

Re-identifying Sea Turtles

Partner: Sea Turtle Conservation Bonaire and FruitPunch.ai

Old Way: Physical metal tags (invasive, prone to damage)

Pain point: Tag loss = hours of manual database searching, high chance of human error,

Opportunity: Sea Turtles have unique faces, especially scutes (scales) which are unique, asymmetrical and permanent.

Fig. 1. Turtles have a unique pattern of facial scutes. L & R sides are asymmetrical. Area used for ID based on Jean et al. (2010)

The Technical Challenge

Environment: Lighting is different, lots of background noise, extraneous detail.

Non-rigid motion: Boat and Turtles are moving.

Variable Operators: Cameras and operators are different.

Goal: Design a contactless ID system that can work “on-edge”

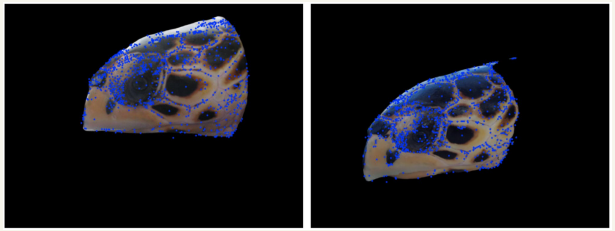
The Search for a Solution

Method	Logic	Verdict
Global Similarity (e.g. SSIM)	Compare every single pixel	**Orientation Hypersensitive**. Fails if the orientations are off.
FaceID (Active 3D) aka “the iPhone method”	30k IR dots and a Depth Map	**Not Scalable / Hardware dependent** Can’t do long-range, unrealistic to face scan every turtle.
Metric Learning	Global Image Embeddings	**Black Box and Large Datasets** Hard to debug why, and requires a lot of training data.
Feature Matching	Find and match specific keypoints in the images	**Winner:** Geometrically explainable and robust.

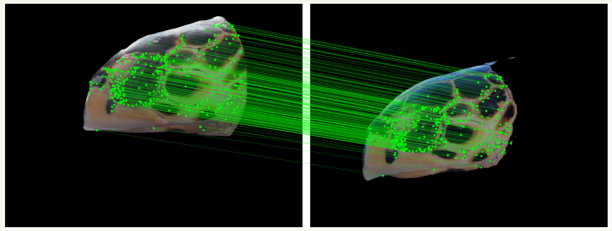
The Sparse Matching Pipeline

NB This is done post image processing with YOLO and SAM

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



What are keypoints?
Classical approaches
Deep Learning



DL-based Models
Identify which key points match

Donatello
64%

Raphael
16%

Michelangelo
11%

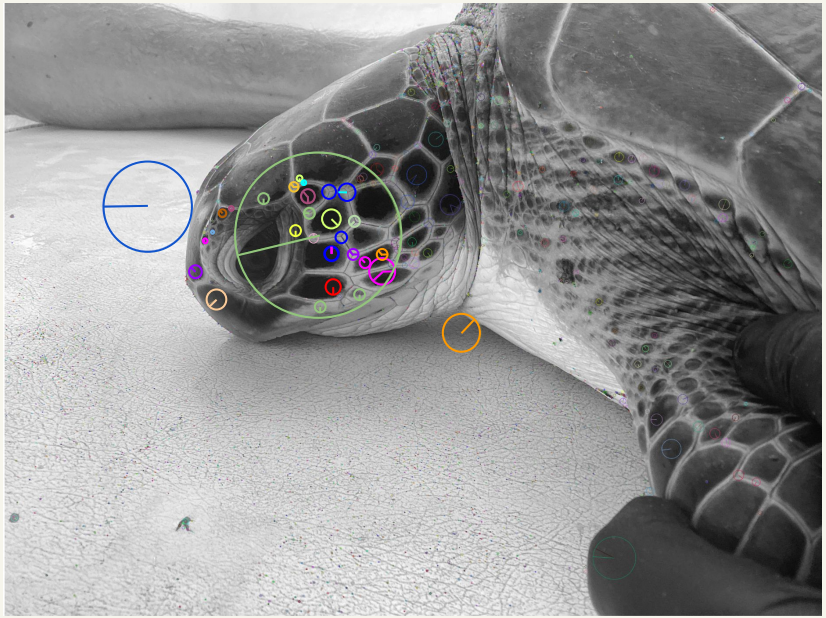
others...

So what exactly is a keypoint?

A keypoint… ***“converts a key part of the image into a vector”***

But…

1. How do you determine *which area* should be a keypoint?
2. *What information* should we store in the vector (aka descriptor)?



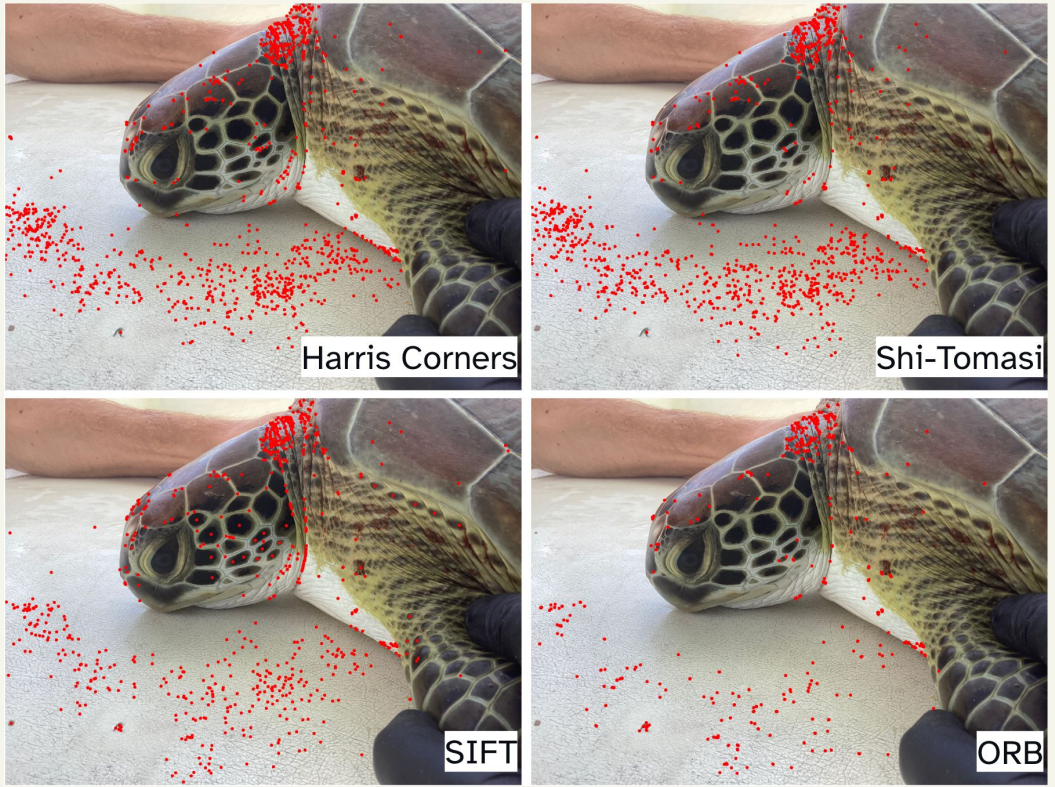
NB Example keypoints for demonstration only

Classical Keypoint Extraction

Rely on hard rules to find keypoints.

SIFT (Scale Invariant Feature Transformation) is the gold-standard for classical techniques

Multiple options available in Python libraries such as OpenCV and Kornia



Possible fixed keypoint extractors

	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

SIFT - Identifying Features

Process

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

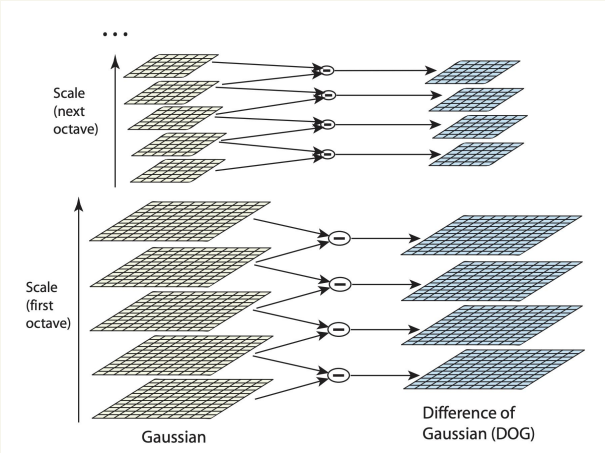
Take home message:
“SIFT works by identifying areas of high contrast that are insensitive to scaling”

generated with Dream by WOMBO

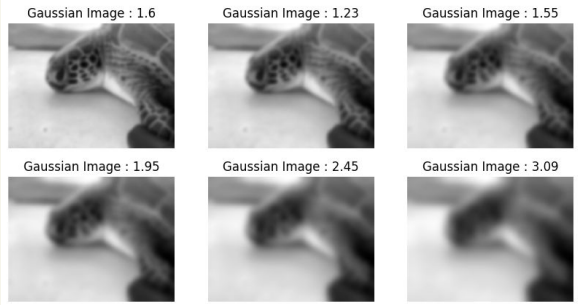
Check out [OpenCV’s description] for a more detailed description, or the [original paper]

SIFT - Scale-space Extrema Detection

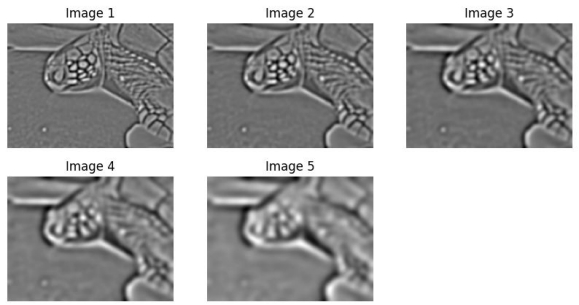
Difference of Gaussians highlight edges of the image at different **Octaves** (scales)



Example Octave



Example Differences of Gaussians

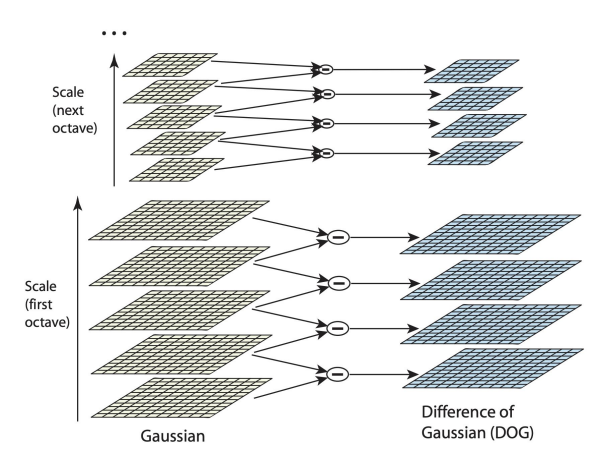


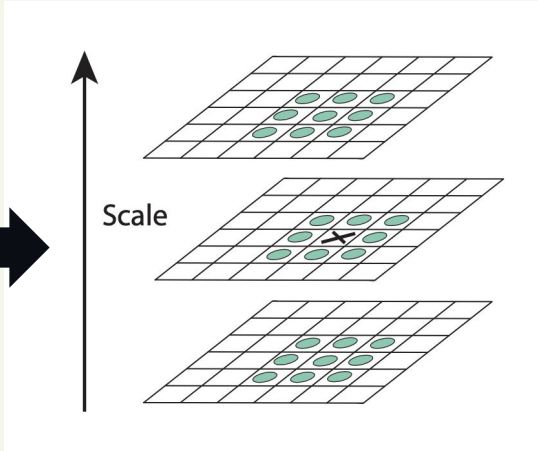
Octaves = Recommended number is 4 but this is tunable

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...

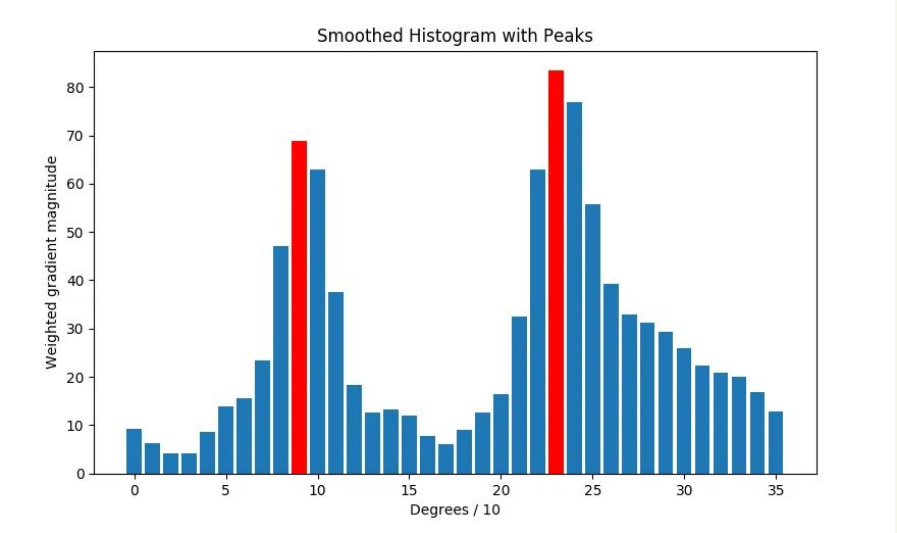
1. Taylor series expansion to locate the extrema in it's octave.
2. Intensity thresholding (<0.03 = no keypoint)

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

SIFT - Orientation Assignment

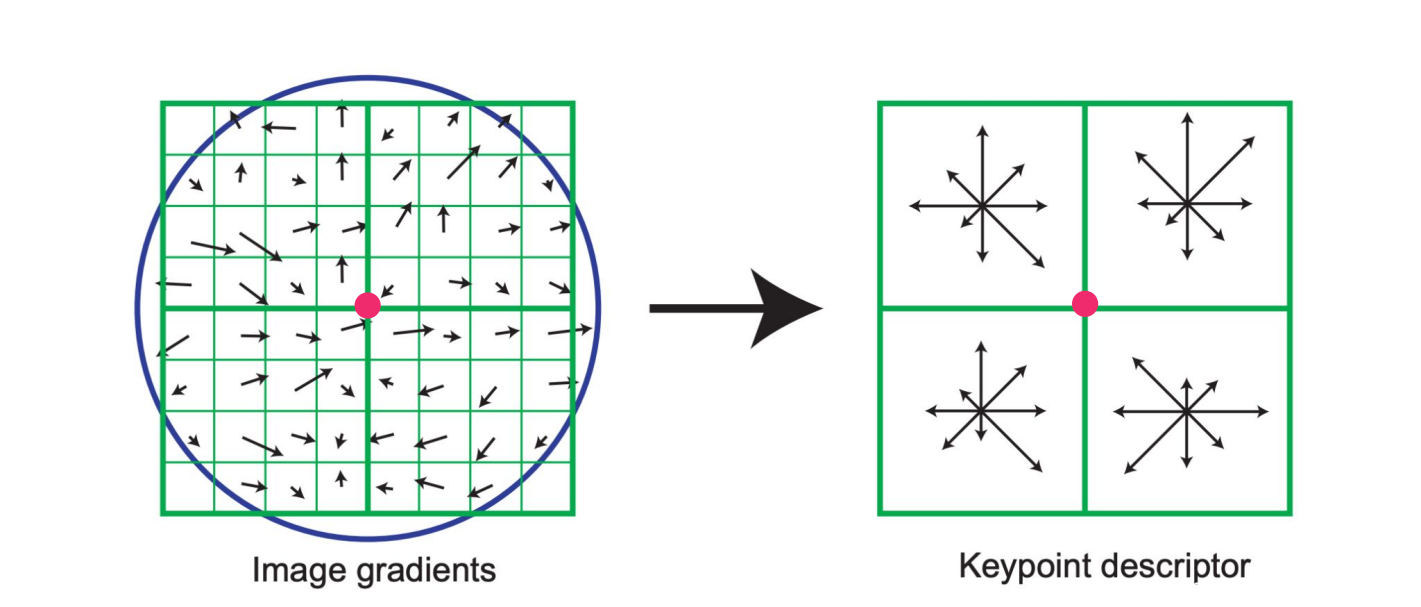
An **orientation** also is assigned to each key-point. This is done by extracting the neighborhood around the key-point and creating a **orientation histogram**, the peak of the histogram is used as the orientation.

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 - Building the Descriptor



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

Deep Learning Based approaches to Keypoint Extraction

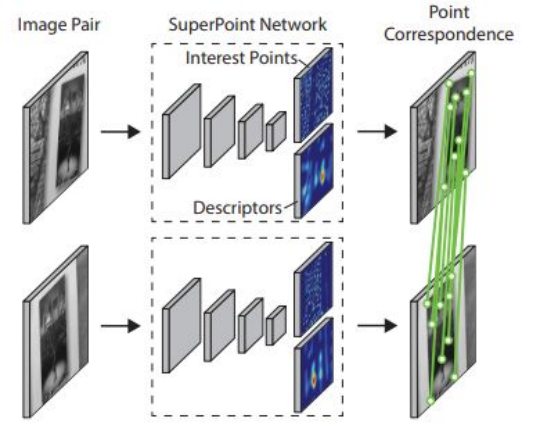
- Classical approaches take a very “local” approach
- Deep Learning by expanding the kernel size can integrate over the whole image
- DL can develop more complex methods for identifying keypoints
- But they are a black box

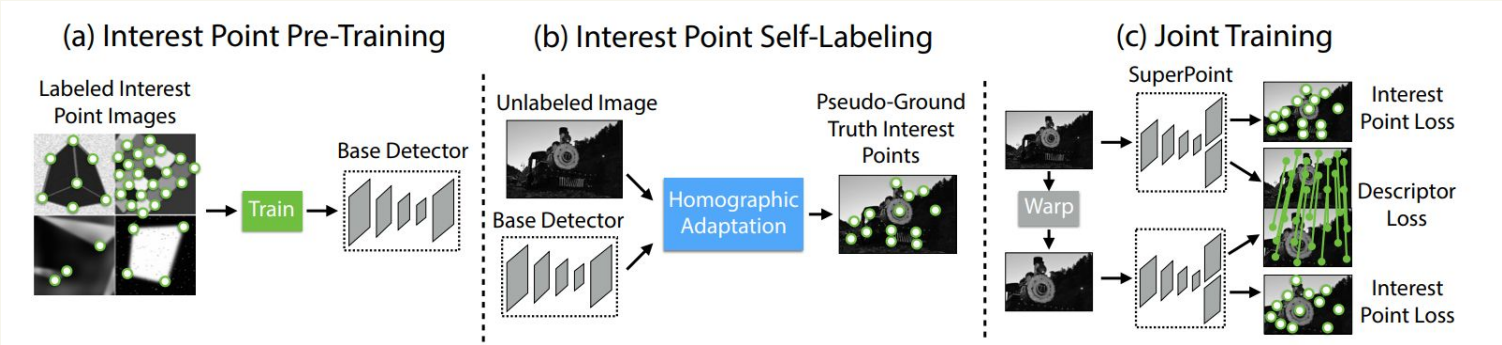
Differences in Features

- SIFT (128-D vector of grid based histograms), DL’s typically 256-D latent features
  - Each model “decides” which visual clues are the most stable

DL-based: e.g. Superpoint

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





Check out the paper here

Other DL-based Keypoint Extractors

Name	Paper	Open Source?	Advantages	Disadvantages
SuperPoint	CVPR (2018)	**Weights Only** (License restricted)	The "Industry Standard." Extremely stable, widely supported in almost all libraries (TensorRT, ONNX).	Technically proprietary (Magic Leap). Struggles with high-res images due to its 8x8 grid bottleneck.
XFeat	CVPR (2024)	**Yes** (Apache 2.0)	**The Edge King.** Designed for CPUs. Up to 5x faster than SuperPoint with similar accuracy. Hardware agnostic.	Newer; fewer specialized hardware kernels (like TensorRT) compared to SuperPoint.
ALIKED	TPAMI (2023)	**Yes**	Very high repeatability. Great at handling different scales. More "modern" architecture than SuperPoint.	Higher memory footprint than XFeat. Can be slower on low-end ARM CPUs.
DISK	NeurIPS (2020)	**Yes**	Learned via Reinforcement Learning. Excellent at finding "matchable" points in natural/chaotic scenes.	Heavier than the "lightweight" models. Performance can be inconsistent in indoor/man-made environments.
DeDoDe	CVPR (2024)	**Yes**	Decouples detection and description. Extremely robust to massive viewpoint changes (e.g., 180° turns).	Complex pipeline. "Heavy" to run in real-time on edge devices.

Classic vs. Deep Learning Approaches

In my opinion there is no “best” approach

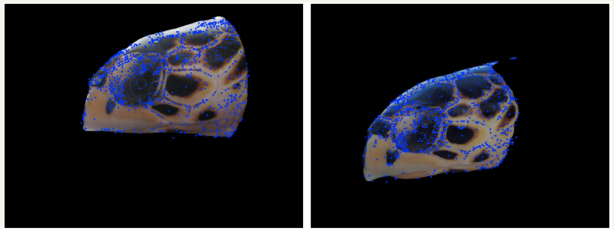
- For the Sea Turtles, SIFT performed best
- But it may vary depending on the problem

Some things to consider

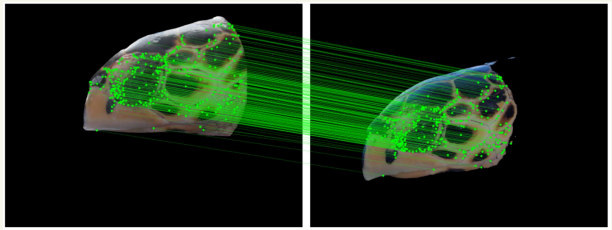
- Classical approaches may be faster, and more consistent
- Generally always use SIFT as a baseline when exploring
- DL approaches can find points that survive extreme noise, low light, and blur
- Evaluation may need to be done with respect to the full pipeline

The Sparse Matching Pipeline

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



Evaluate Classical and Deep Learning approaches ✅



DL-based Models
Which models should we use?

Donatello
64%

Raphael
16%

Michelangelo
11%

others...

Slide courtesy of Davide Coppola
and 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

LightGlue: SOTA (2023) Feature matching

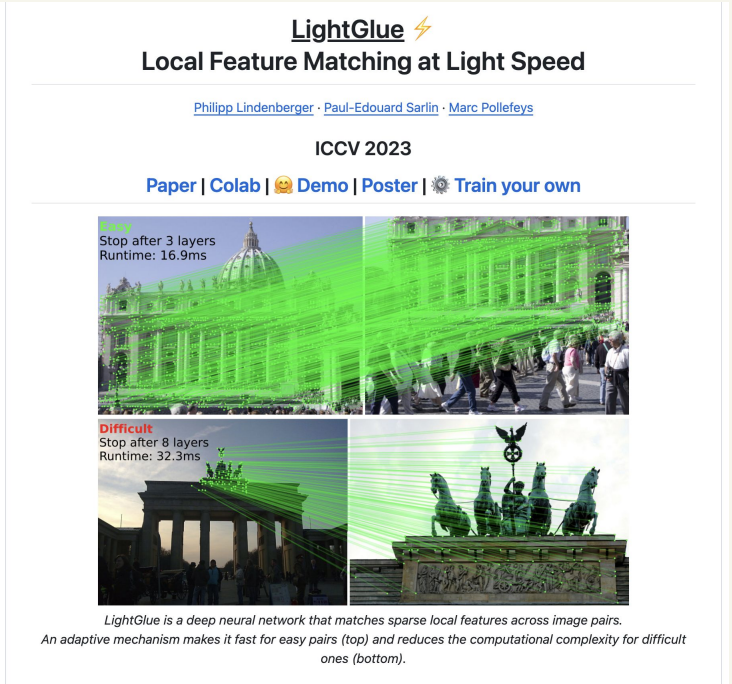
LightGlue is a sparse feature matcher comparing two keypoint/descriptor sets

Basically a reworked version of SuperGlue[1]

But with improved

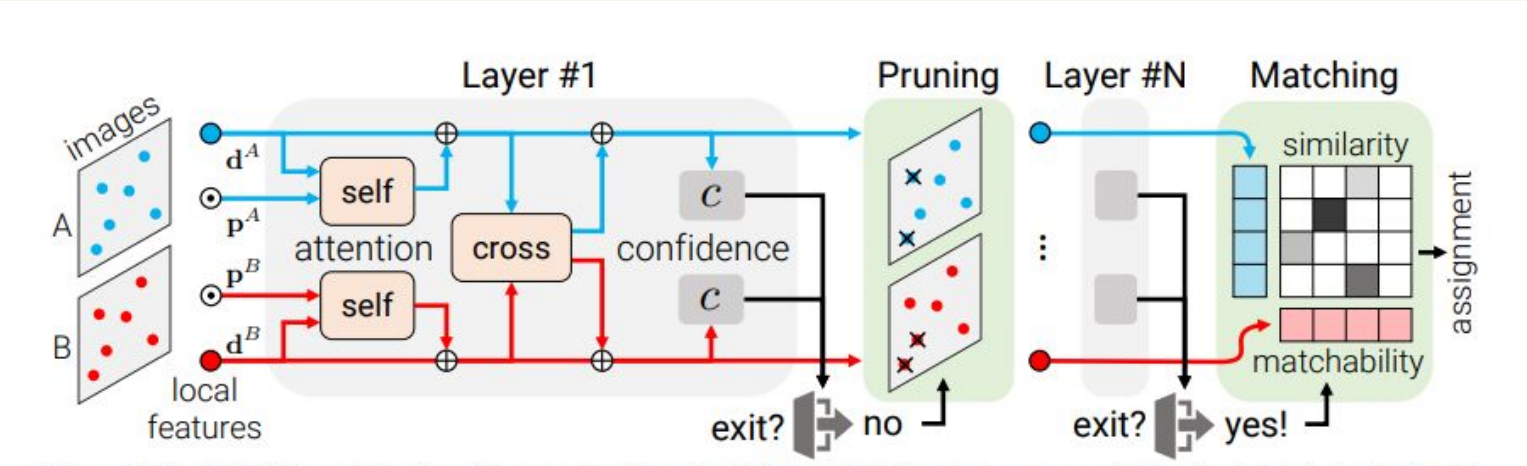
- Efficiency: Premature stopping at varying depths/layers
- Accuracy: Improved positional encoding yielding more stable results
- Training: Re-worked architecture led to easier training

https://github.com/cvg/LightGlue

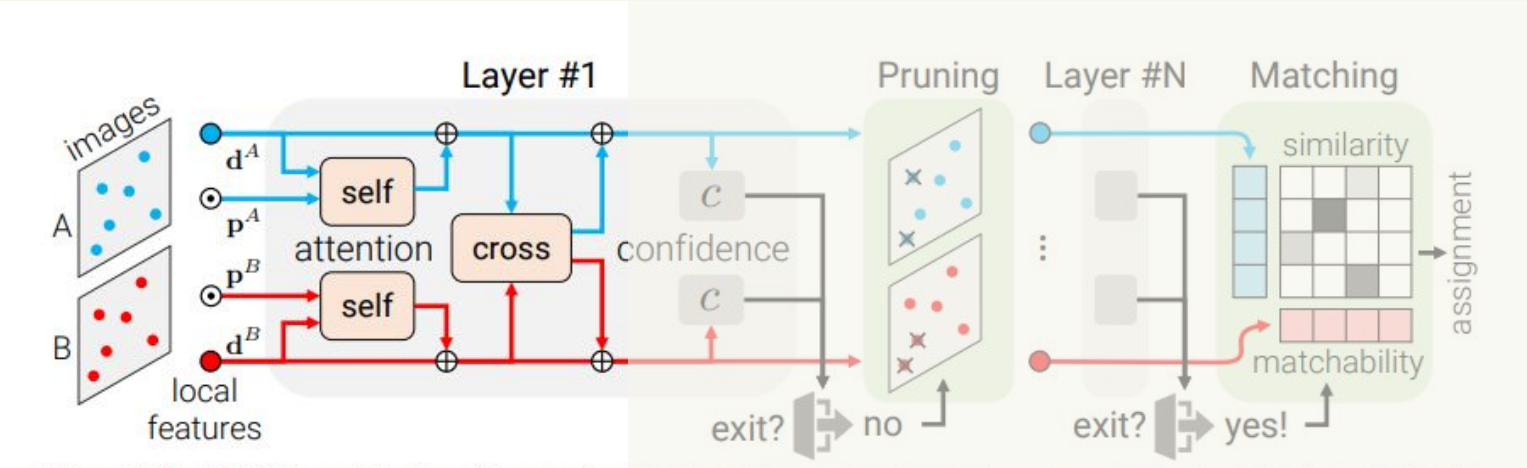


[1] Sarlin, Paul-Edouard, et al. "Superglue: Learning feature matching with graph neural networks." *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020.

Model Architecture: General Overview



Model Architecture: Updating Descriptors

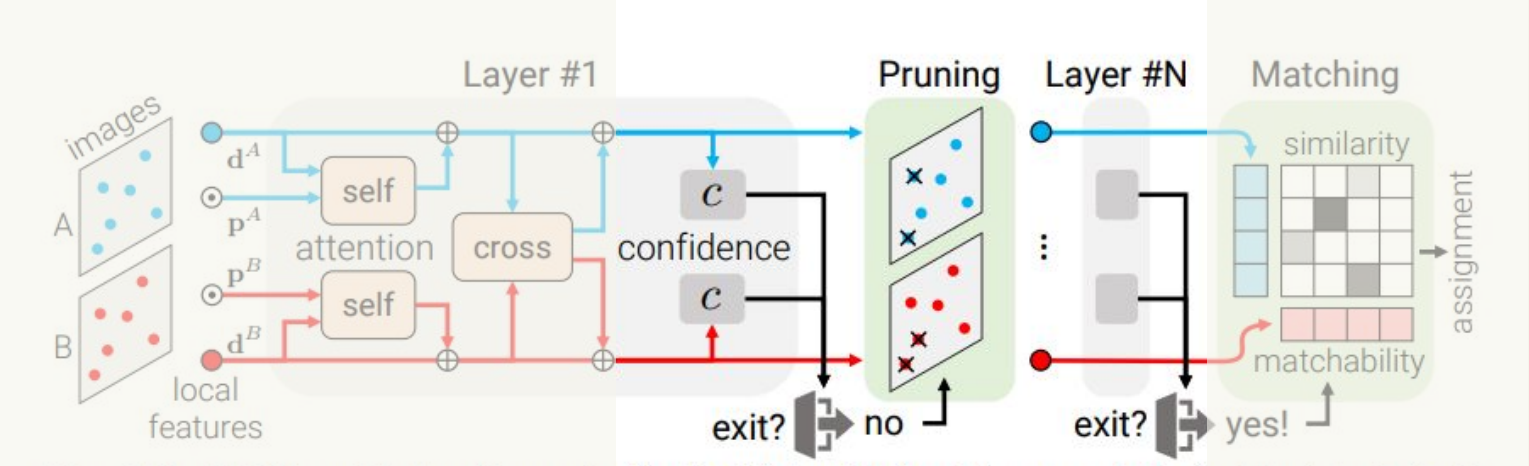


Self-Attention: every point attends to all points extracted from one image + rotary (relative) positional encoding

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

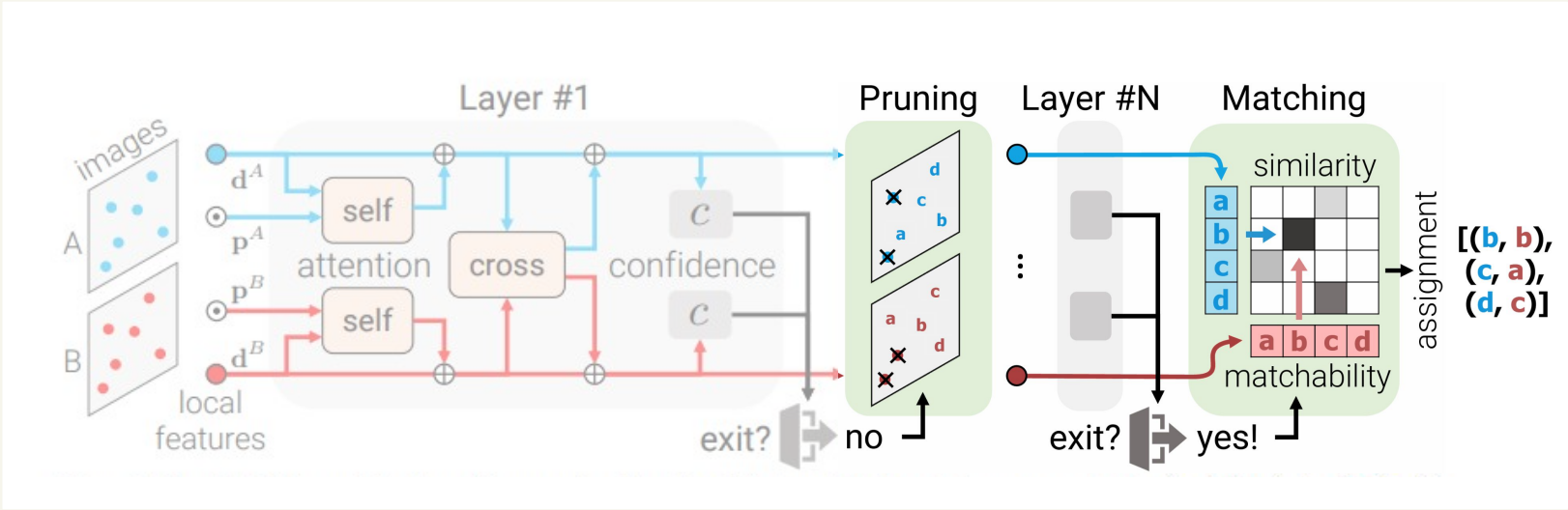
Model Architecture: Exit Criterion



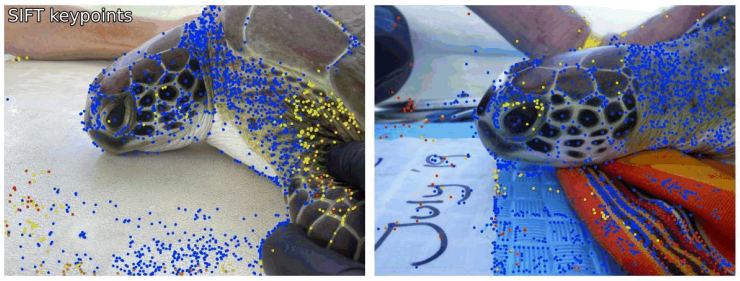
Confidence classifier: trained to predict whether current feature representations are 'final', exit if sufficient ratio of points is confident enough

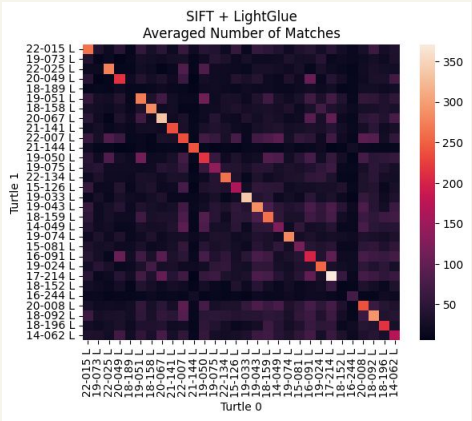
Point Pruning: points predicted confident and unmatchable are removed

Model Architecture: Output



Evaluation of Initial Output





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.

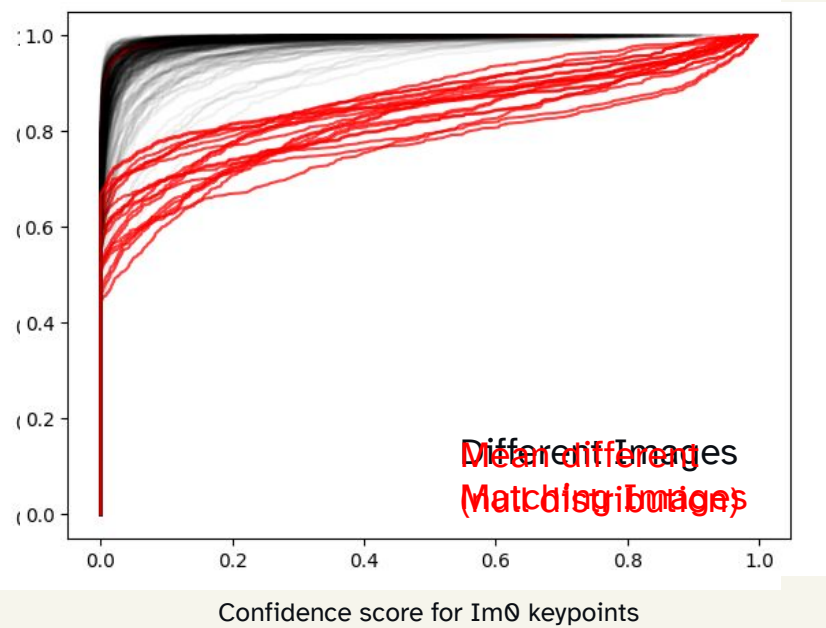
Possible reasons for suboptimal performance...

- # 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.

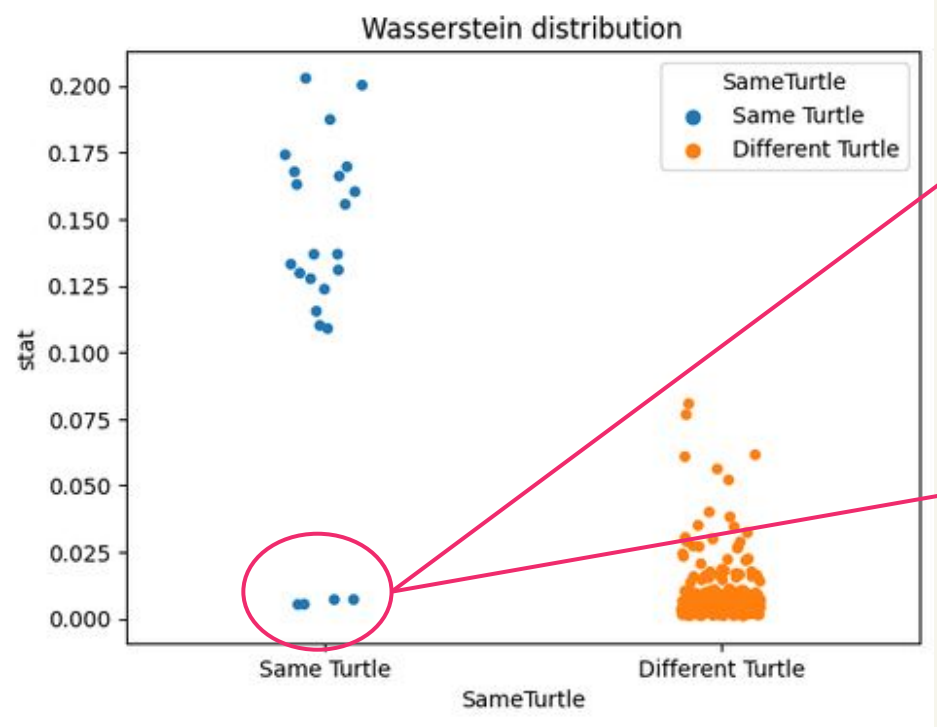
Solution: Look at the **distributions of scores** instead

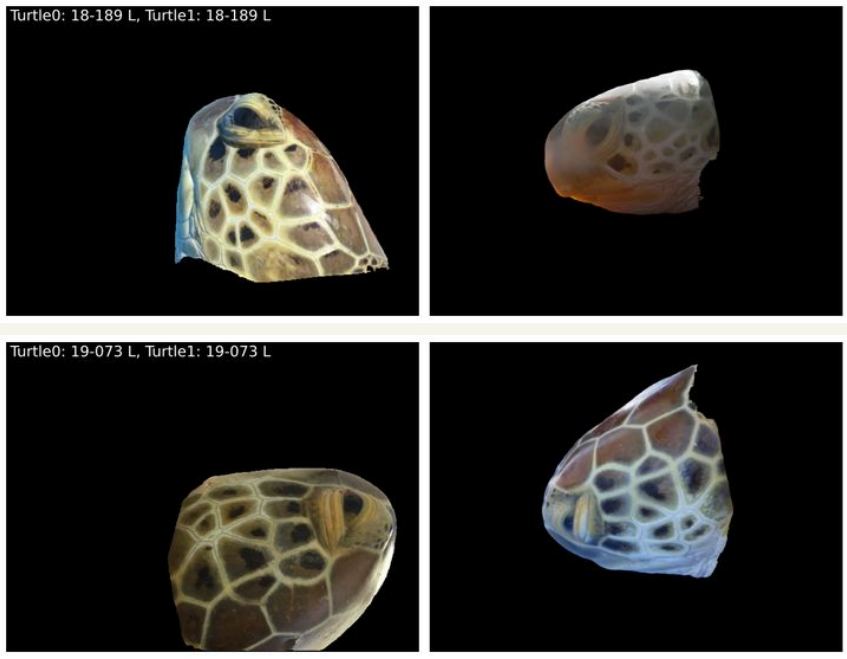
Matching Images have different confidence distributions



- 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

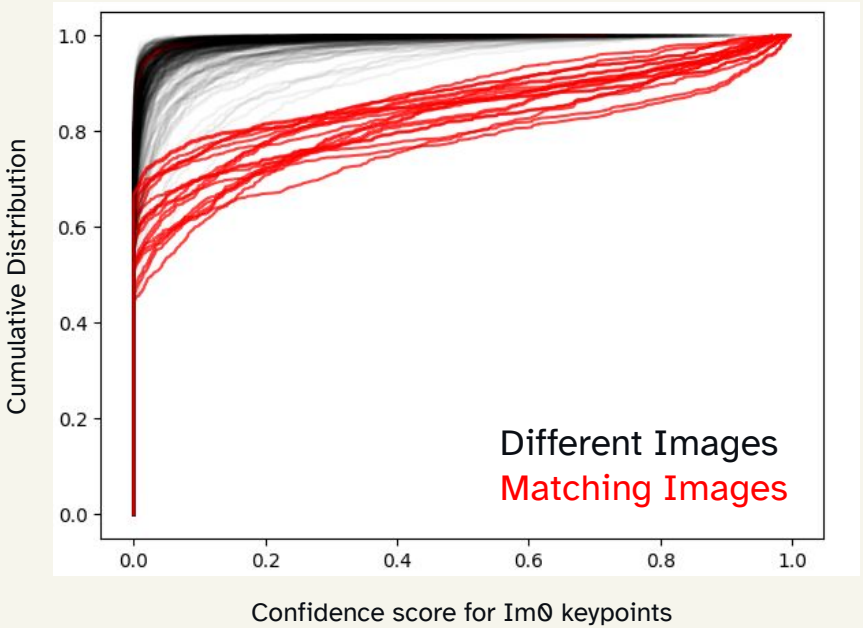
Wasserstein Scores accurately split the data





These four images are wrongly paired/rotated

But there’s an even simpler evaluation



- Still perform the cumulative frequency plots
- But just take an AUC
  - Smaller the AUC the better the fit

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	90	~ 30s [T4 on Colab]	Fast, accurate with current reference set.	Requires a GPU for quick evaluations.
LoFTR	64/80/84	80	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

So what’s the current SOTA?

Take Homes

- Understand your use case
- Try to understand the outputs carefully, and don’t throw away additional information

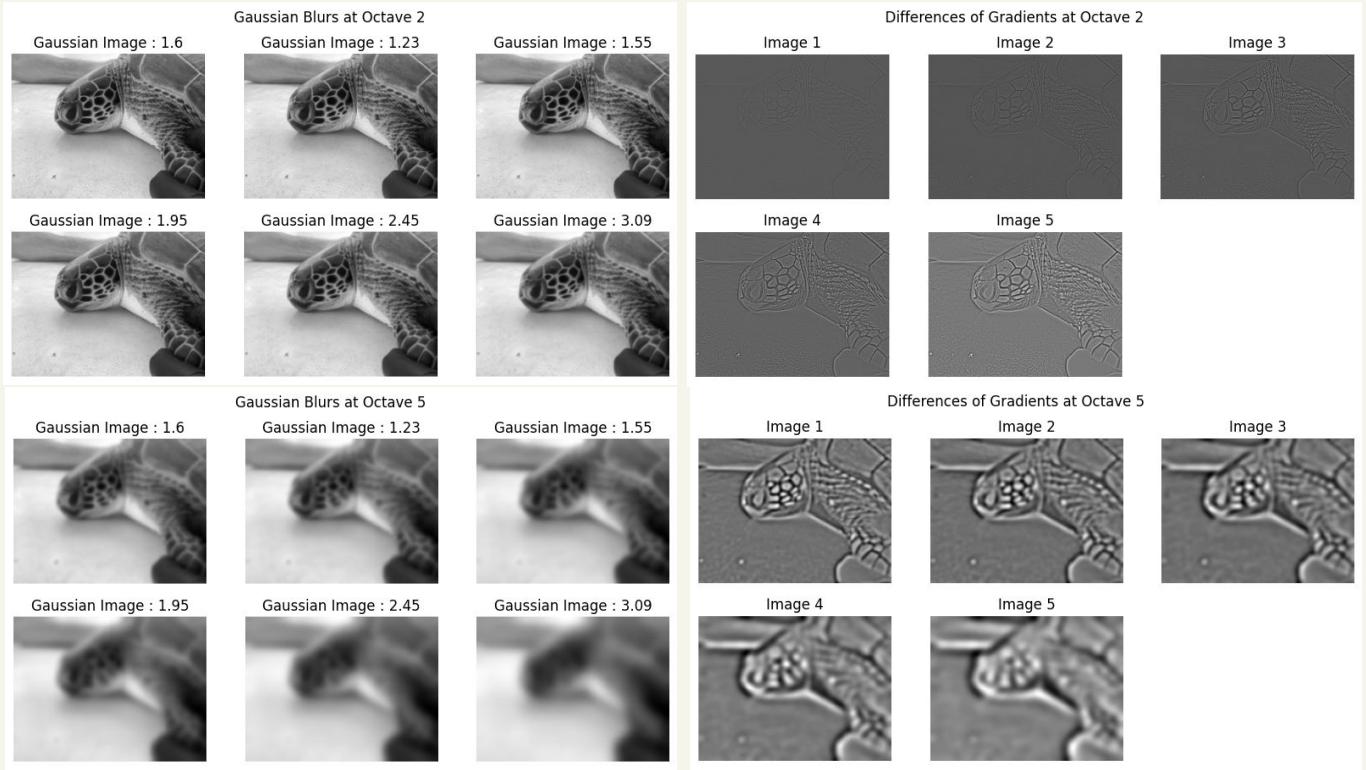
Jupyter Notebook

- Demonstration:
- A dating app PoC: [Combination with GenAI Features]

Dating App PoC

Extra Slides

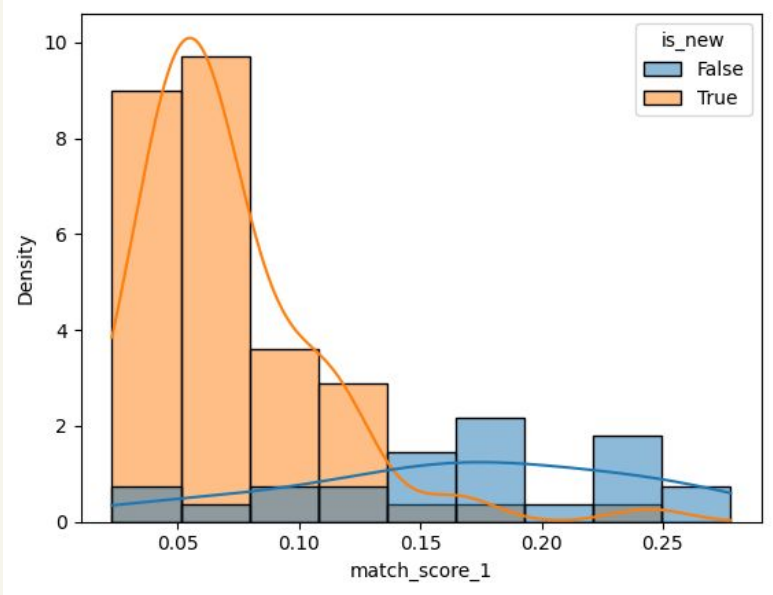
SIFT: The effect of DoG at different octaves



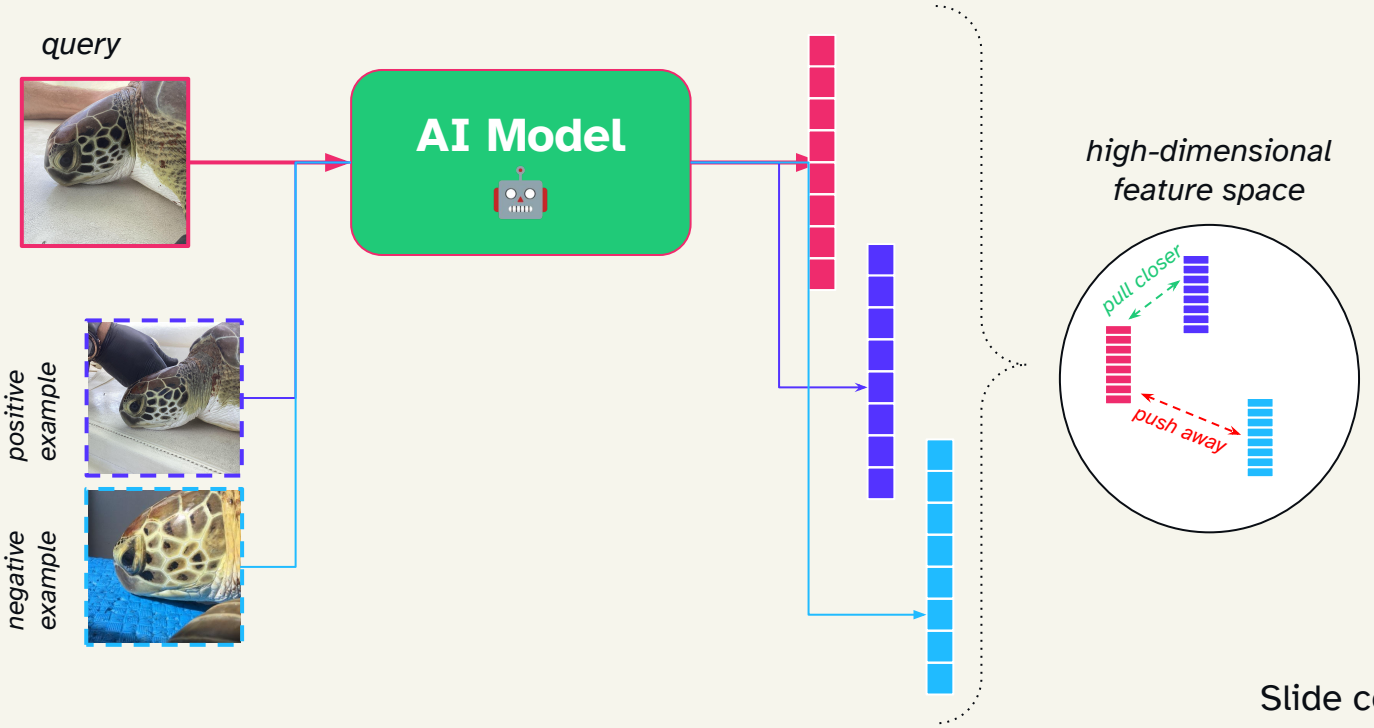
Identifying Novel Turtles

- 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



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

Initial Plan: Sea Turtles as a Demo

- Initial Challenge parameters
- Old school: SSIM, etc. → Actually we want to use keypoints
- Other keypoint creators
- SIFT: Notebook 1 Explain how it works
  - Then explain that there are other methods that use deep learning (NB Apple use >30,000 points from an infrared camera, not feasible for most hacky projects)
- Once we get the keypoints then we can go to the next stage: Matching
  - LoFTR, SuperGlue, and now LightGlue
  - Current SOTA: OmniGlue, JamMa, LoFTR+, X-Feat, LighterGlue
- Statistical improvements
  - Wasserstein, but actually can just used AUC, much easier
- NB Metric Learning is significantly faster but data hungry, but you can distill giant foundational models like Dino_v2
- Notebook 2: Sea Turtle matching -> NB This is simplified, as I will use a smaller dataset and didn't do all the background filtering

- How I used it to solve a different problem