Research and Innovation Project Report

Keypoint detection and application: from reviewing classic and deep learning approaches to creating of a siamese model.

Author: EL IDRISSI Mehdi

Polytech Tours - IT Department

Supervisor: M. Ghulam-Sakhi Shokouh

This report is submitted as part of the PRI project requirements.

Contents

1 Abstract **2**

2 Introduction **3**
2.1 What is a key point ? . . . 3
2.2 What is the point ? . . . 6

3 Research Part **8**
3.1 Introduction to algorithms . . . 8
3.1.1 Classic Algorithms VS Deep Learning . . . 8
3.1.2 Dealing with algorithms . . . 8
3.2 Classic methods . . . 10
3.2.1 Harris . . . 10
3.2.2 Fast . . . 10
3.2.3 Orb . . . 11
3.3 Deep Learning methods . . . 11
3.3.1 Human-Pose Estimation . . . 11
3.3.2 SuperPoint . . . 12
3.4 Comparing algorithms . . . 14

4 Innovation Part **16**
4.1 Personal Siamese network . . . 16
4.1.1 How to use the program . . . 16
4.1.2 Architecture of the network . . . 19
4.2 Optimizing Superpoint . . . 21
4.2.1 Superpoint architecture . . . 21
4.2.2 Superpoint Limits . . . 22

5 Conclusion **24**

1 Abstract

Key points are everywhere today with a lot of applications. In fact, they represent information taken from pictures, and dealing with them can explain how facial recognition or tracking works. They are detected, and then described. The key points in this article are mainly corners, and while there are many methods to detect them (**Harris**, **Fast**, **Orb**...), they give the same results with the same set of parameters.

This is why deep learning approaches came : training a model on a dataset can increase the accuracy and performance of an algorithm. The best of them is **SuperPoint**, and is extremely powerful in detecting and tracking key points.
Thanks to this knowledge, I created my own **Siamese network**, a deep learning approach to train a model in corner detection. We will see in more detail how it works.

2 Introduction

2.1 What is a key point ?

In general, a key point is a point of interest that we define in an object.
It is inspired by the ability of our brain to recognize shapes and differentiate an object from another.

In fact, a key point has to be detected in a picture or a video, manually or thanks to a software :

Figure 1: Key points differentiate objects

In the example above, we want to focus on the corners of the shapes, which represent the most important information to take from both objects.

It is possible to use this technology on 3D real objects, and detect their corners.

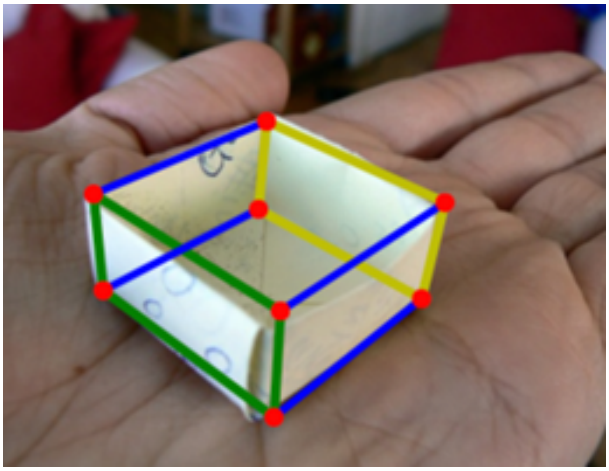


Figure 2: 3-Dimensional key point analysis
(Source)

But, if we want to work on faces for the recognition of people, we will focus instead on facial key points :

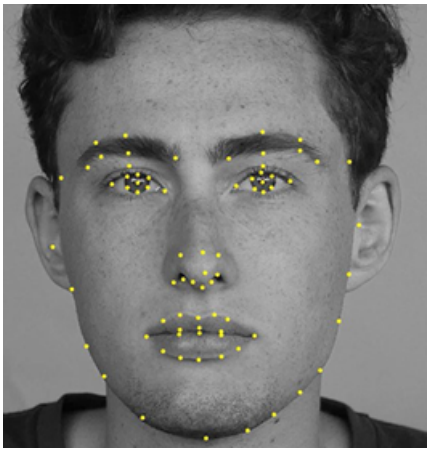


Figure 3: Example of facial landmarks.*(Source)*

These points define the identity of the people scanned, by the distance between each facial landmark. For instance : the distance between the **bottom of the nose and the upper point of the mouth**, or between the **far right corner of the right eye and the far left corner of the left eye**.

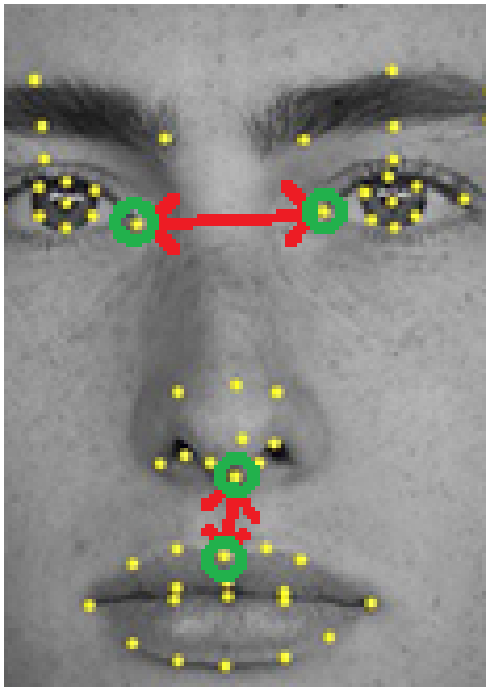


Figure 4: Example of distance between facial landmarks

If needed, key points can also define articulations of a body, to determine his pose :

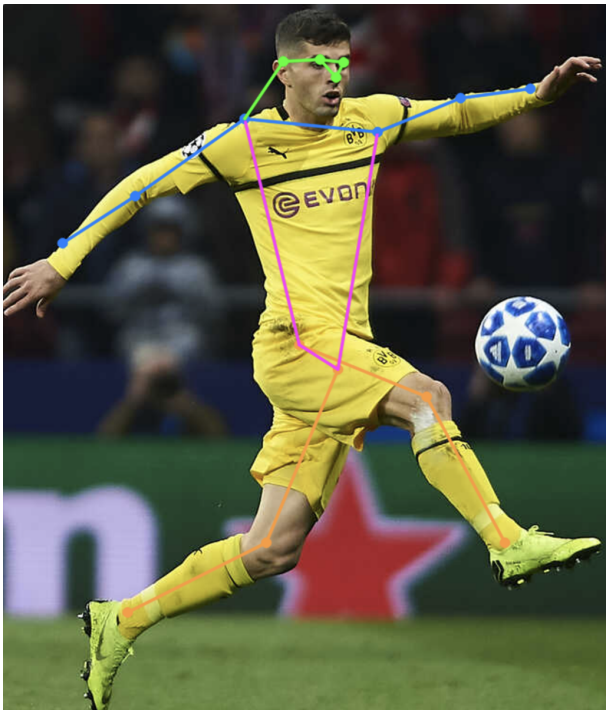


Figure 5: Example of pose estimation - A jumping soccer player *(Source)*

So, a key point is an information to extract from an object, and a set of key points is the ID Card of an object ! Our brain recognizes objects, faces, or poses with the placement of these points. So, software have to transcribe the role of the brain in two steps : detecting and describing interest points on pictures.

Indeed, an interest point must be detected (*we will focus on the technologies behind this process later*), and described : this last part consists of understanding the context surrounding a point, with a vector of information unique to each one.

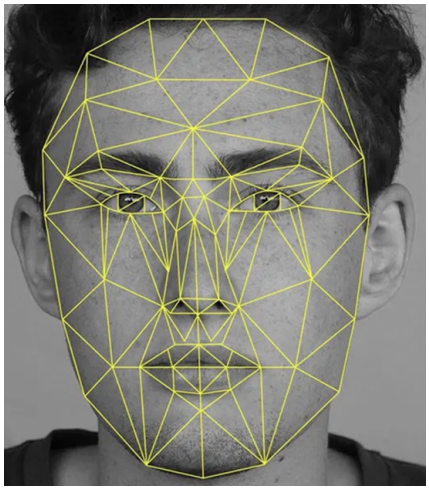


Figure 6: Example of key points description on a face *(Source)*

However, this report will not focus on description, but **only on detection of interest point**.

2.2 What is the point ?

The detection (and description) of key points is used nowadays for many purposes, as well in science as in sports or research in history.
Indeed, cameras and phones are tracking and recognizing people with this technology, for control or security purpose.

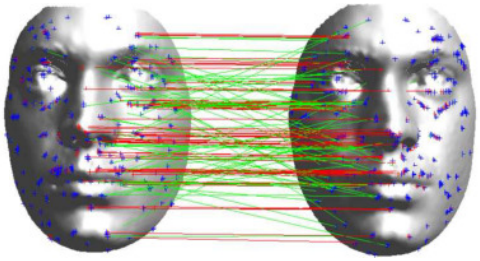


Figure 7: Matching two faces with key points *(Source)*

Recent phones can be unlocked with face ID technology, to identify its owner :

Figure 8: Security of phones by Face ID *(Source)*

Sportive events use also key points and pose estimation to improve the accuracy of their measure :

Figure 9: Example of pose estimation during skying events *(Source)*

Another concrete application is the analysis of old hand-written documents to identify its author, check the authenticity of the text or understand the reading order of the elements in a paper.

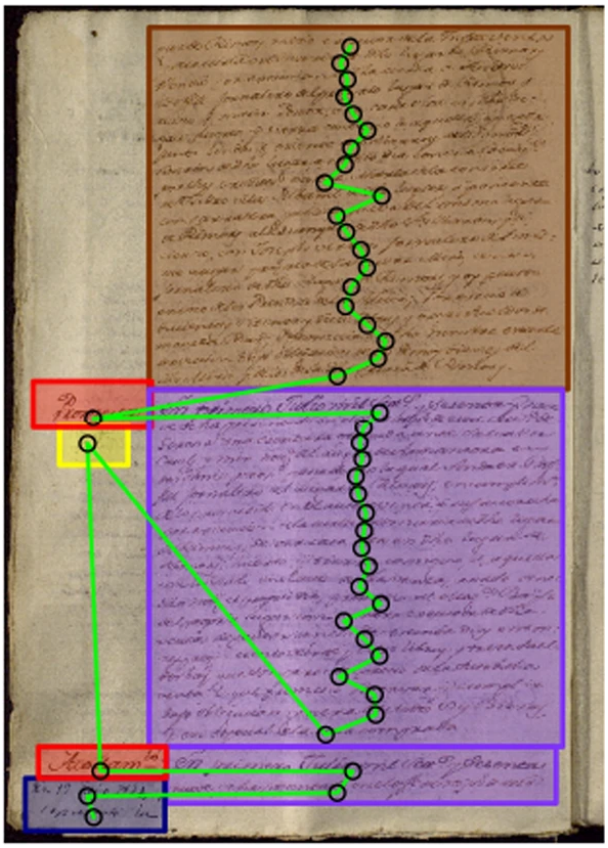


Figure 10: Reading order detection on handwritten documents *(Source)*

So the key points are omnipresent in our society, from a wide range of technologies in research, private or even public domain. Improving their detection therefore seems necessary to keep pace with the latest technological advances : we will now look at the different methods available.

3 Research Part

This part will focus on the research I have done across numerous algorithms, able to detect key points. Each one is specialized on a certain type of key point, classic (deterministic) algorithms and those using Deep Learning. I will briefly explain the difference between both and then go into more detail about how they work.

3.1 Introduction to algorithms

The state of the art of these technologies shows that the principal evolution came from training on datasets.

3.1.1 Classic Algorithms VS Deep Learning

In fact, we can make a clear distinction between determinist algorithms - Classical - and those training on their datasets - Deep Learning ones.

With classical methods, the result will depend on its parameters, whereas the deep learning approach is like a black box depending on the datasets used during the training.

Their use differs from the accessible resources and the context of the application. We will choose a classical algorithm if we lack of resources, don't have too much dataset to train with, or need a quick and transparent solution. The main algorithms are : Harris, Fast and Orb.

Otherwise, for more precise work, with a huge amount of data and a plenty of computing performance available, we will prefer to work with Deep Learning techniques. The most commonly used are : Lift, Siamese - Triplet networks and Superpoint.

3.1.2 Dealing with algorithms

To work with algorithms that detect key points, we need to test their accuracy while choosing the right parameters, as well as supervising the model train process. This is possible thanks to "***ground truth***" : these are pictures where all expected key points to be detected are placed by hand, at the exact place where we want the algorithm to detect them.

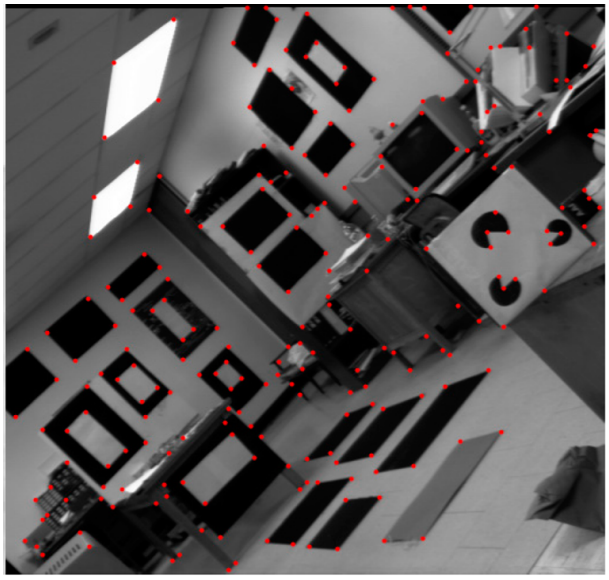


Figure 11: Ground truth picture for corners detection *(Source)*

Comparing accuracy :

The accuracy is the error rate between the result given by an algorithm, and the ground truth that we expected the result to be the closest possible. This step allows to compare the error rate between algorithms on the same dataset, and to adjust the parameters in order to adapt the algorithm to a precise context close to its ground truth.
For instance, we want to detect corners on a dataset of noisy and dimly lit rooms : we will tune the algorithm to a similar ground truth, so the process of detection will suit the dataset.

Training models :

As explained above, ground truth is the tool to check if algorithms are fine-tuned. But for Deep Learning methods, the training is made directly on ground truth, to teach the model the exact result we want. And with a large set of these, the model can learn to detect key points despite noise in a room, rotation of objects, and scale variability : this is the homographic adaptation step. We will see later the "***MagicPoint***", exactly based on that with a self-supervised aspect : the program generates its own ground truth (pseudo-ground truth) and learns with it.

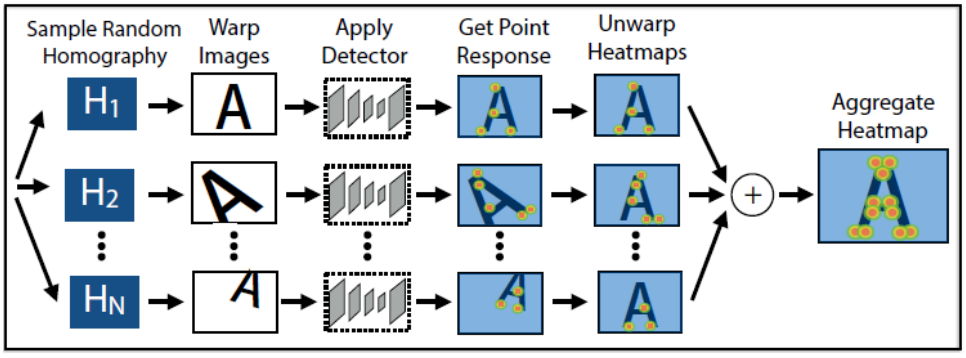


Figure 12: Homographic adaptation on training datasets *(Source)*

3.2 Classic methods

The following algorithms are only ***corners detection*** *focused.*

3.2.1 Harris

The Harris corner detector is a popular algorithm that is used to detect corners in an image. It works by analyzing the intensity variations in all directions around a point.

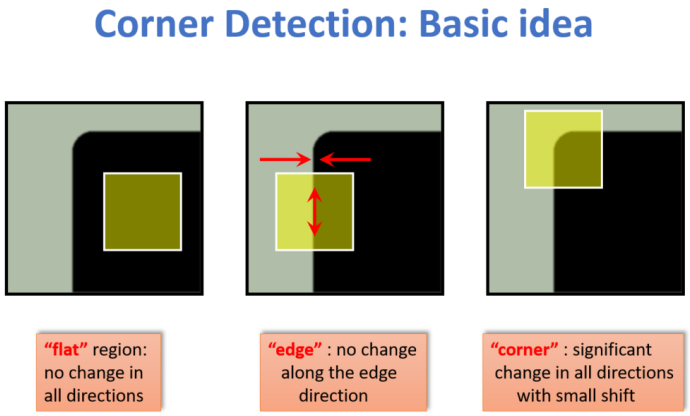


Figure 13: Harris detection of intensity variation *(Source)*

Specifically, it computes a matrix that captures gradient information in the x and y directions, thanks to a Sobel filter.
By examining the eigenvalues of this matrix, the algorithm determines if a point is a corner, an edge, or a flat region. A corner is identified when both eigenvalues are large, indicating significant intensity changes in multiple directions.

This method is included in many libraries in Python, such as **OpenCV** and **SciKit**.

3.2.2 Fast

The Fast (Features from Accelerated Segment Test) algorithm is also a method to detect corners in a picture. It works by examining a circular neighborhood of 16 pixels around a candidate point. If an adjacent set of pixels (about 9) in this circle is significantly brighter or darker than the center pixel, the point is considered a corner.

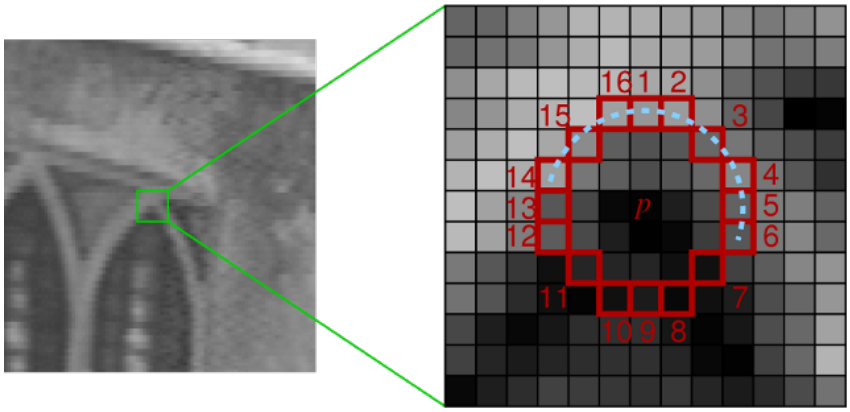


Figure 14: Fast Corner detection neighboring process *(Source)*

This test is simple and fast to execute for a computer : that makes Fast ideal for real-time applications. It is also available in the Python library **Open CV** .

3.2.3 Orb

Orb (Oriented Fast and Rotated Brief) is a feature detection and description algorithm designed for efficiency and performance. It combines the Fast corner detector (explained above) with the Brief (Binary Robust Independent Elementary Features) descriptor. First, Fast detects key corner points in the picture with an improvement estimating the rotation, and then Brief is applied to create binary vectors describing the point. *We will not explain in further detail the Orb description process, but for the detection part, this is an application of improved Fast.* Orb is widely used in real-time applications like object recognition.

3.3 Deep Learning methods

The two following methods are very different : in fact, H-Pose Estimation focuses on recognizing articulations, while SuperPoint detects mainly corners and patterns. However, the H-Pose Estimation is an interesting example because it offers a practical perspective of key points detection, and introduces the "heat map" notion for the next part.

3.3.1 Human-Pose Estimation

Human Pose Estimation is a computer vision technique used to detect and map the positions of a person's body joints, such as elbows, knees, and shoulders, in an image or video. It works by analyzing the body and recognizing members that are similar to its training.

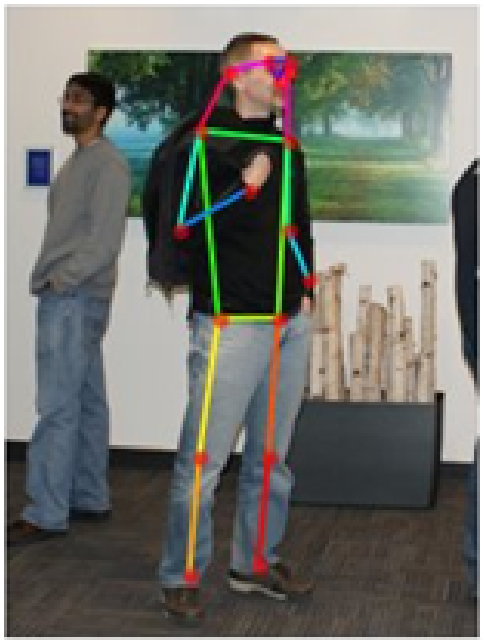


Figure 15: Example of a pose estimation *(Source)*

Modern methods often use deep learning models (like the one sourced on the figure above), such as Convolutional Neural Networks (CNNs), trained on large datasets of annotated human poses. These models predict heat maps indicating the probability of presence of a key point in each pixel, enabling applications such as motion tracking, fitness analysis, and augmented reality. The process involves detecting key points, connecting them to form a skeleton, and refining the pose for accuracy.

Figure 16: A heat map (at the left) represents the probability of presence of each articulation to create the skeleton (at the right) *(Source)*

3.3.2 SuperPoint

The SuperPoint model is a deep learning-based algorithm with fully convolutional neural network, designed for interest point detection and description, and trained through a process called "synthetic to real adaptation." Indeed, it is trained on a simple geometric dataset to detect basic interest points like corners, edges and patterns : (MagicPoint training).

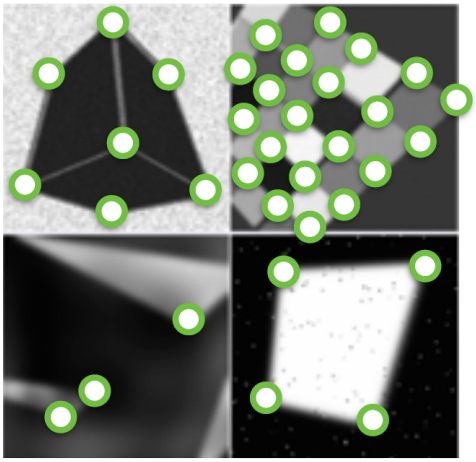


Figure 17: Magic Point training on simple shapes database *(Source)*

The model then undergoes homographic adaptation, where random homographies (scale, rotation, translation, or cropping) are applied to real images, and the detected keypoints are reversed to the position on the original image, to create a pseudo-ground truth.

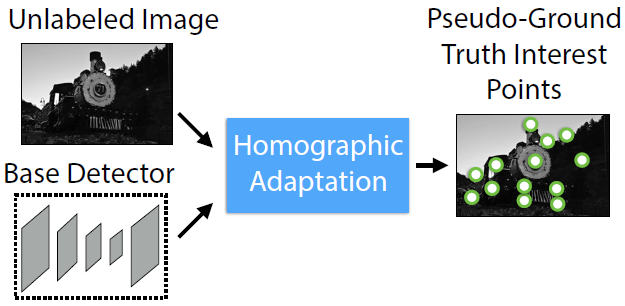


Figure 18: Homographic adaptation induces pseudo-ground truth *(Source)*

This process enhances the model's ability to generalize to real-world images (see Figure 12).

The third stage is the "joint training", to refine the detector and descriptor of homographic adaptation, and use this result as a new dataset for training again the model : this is the step of "self-supervised" training.

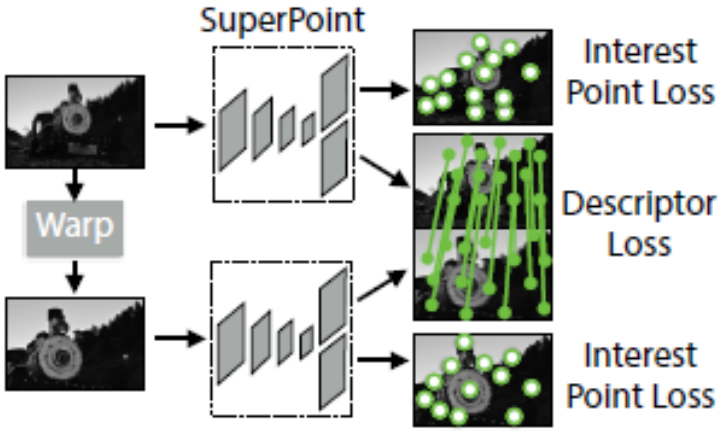


Figure 19: Joint training joins the result of homographic adaptation detection and description, to train again the model*(Source)*

This approach enables SuperPoint to efficiently detect and describe keypoints in a single forward pass, making it suitable for image matching or real-time applications such as tracking.

3.4 Comparing algorithms

Classic and deep learning algorithms have their own speed of calculation or weight of deployment (use of libraries, length of the code, etc.), but must be as accurate as possible.

Here is a comparison of *mAP* (mean Average Precision) made on these algorithms. This is a research made by Daniel DeTone, Tomasz Malisiewicz, Andrew Rabinovich for their SuperPoint scientific paper (Source). Because the comparison is only made between the detection of corners and patterns, with noise and not, the "SuperPoint" does not have its description part : so it refers to its "MagicPoint" part. In the same way, ORB does not have a large change from Fast to its detection part : so it will share the same result as it.

Algorithms :	**Harris**	**Fast** (and **Orb**)	**MagicPoint**
Without Noise	0.678	0.405	0.979
With Noise	0.213	0.061	0.971

Table 1: Performance comparison of algorithms on corners detection

The realization of this table was made thanks to the dataset "Synthetic Shapes", and compares the distance between the point detected and the expected one. Despite its simple composition, Harris seems more accurate than Fast (and Orb), but MagicPoint overpasses them by far : this is due to its training on homographic adaptation. With this example, we can clearly see the difference between classic and deep learning algorithms : the balance has to be made between efficiency and the weight of processing.

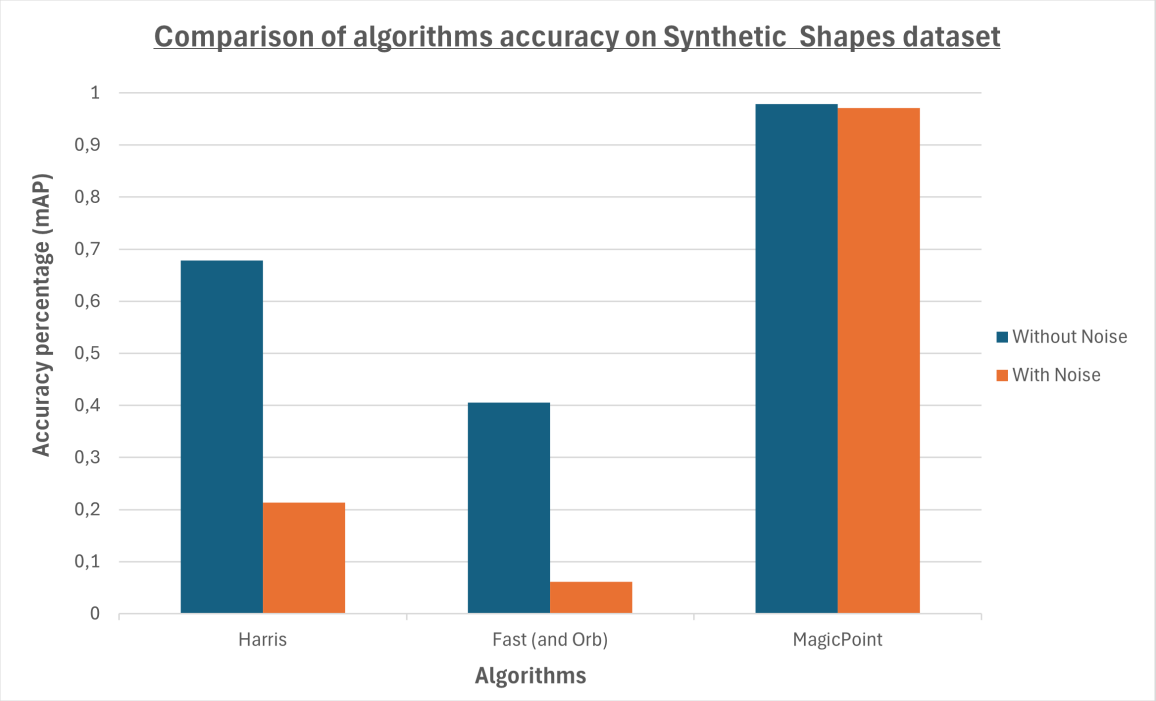


Figure 20: Graphic version of the comparing algorithms table *(Source)*

4 Innovation Part

In this part, I will focus on the deep learning method I have made and explore more the limits and perspectives of Superpoint.

4.1 Personal Siamese network

Before creating my own Siamese network to detect corners, I searched about these types of networks and found the "Triplet" and "Siamese", which operate in much the same way : where Siamese learns with "positive" and "negative" pairs of patches, the Triplet one knows the difference between the "source", "similar" and "different" patches. So, I prefered to focus on the Siamese, because of the better number of examples I could find on the Internet to test it.
The following network I will present you is based on the work of "sohaib023" : Siamese Network in PyTorch.

4.1.1 How to use the program

This program was made in Python, using the PyTorch library. You need to provide the network two libraries, "*dataset/generated/clean*" and "*dataset/generated/noisy*", that are generated randomly : each image in the **clean** folder is named "image-gen-(*n°image*).png" and has its noisy equivalent inside the **noisy** folder, with its name similar but quite different, "image-gen-n-(*n°image*).png".
If you want to use your own dataset, just clear everything in the "clean" folder and put yours, respecting the exact name provided, but either you give the equivalent noisy in the noise or you clear all the files in the "noisy" folder : the program is able to pass through non available noise files.

So, firstly execute the program - "**python siamese corner-detection.py**". You arrive in the main menu :

```
No model found. Creating a new model.

=== Main Menu ===
0. Generate a dataset
1. Train the model
2. Test an image
3. Quit

Choose an option:
```

Figure 21: The main menu

If you don't have a model file, "*siamese model-harris-orb-fast.pth*", the program will create one for you. But if you already have one in the root folder, the program will load it :

```
Model loaded from 'siamese_model-harris-orb-fast.pth'
```

Figure 22: Message attesting the model loading

From here, you have 4 choices :

- 0. Generate a brand new dataset
- 1. Training the model on the dataset
- 2. Test the model on the picture of your choice
- 3. Exiting the program

If you choose to execute the first one, you will have to enter how many images to generate in the dataset :

```
Choose an option: 0
How many images do you want to generate ?
1000
```

Figure 23: Generating the dataset

Everything will be stored in the "dataset/generated" folder. If this path does not exist, the program will create it. Once done, the same number of pictures will be created both in the "clean" and "noisy" folders. Because every clean image created has its noisy equivalent created in the other folder :

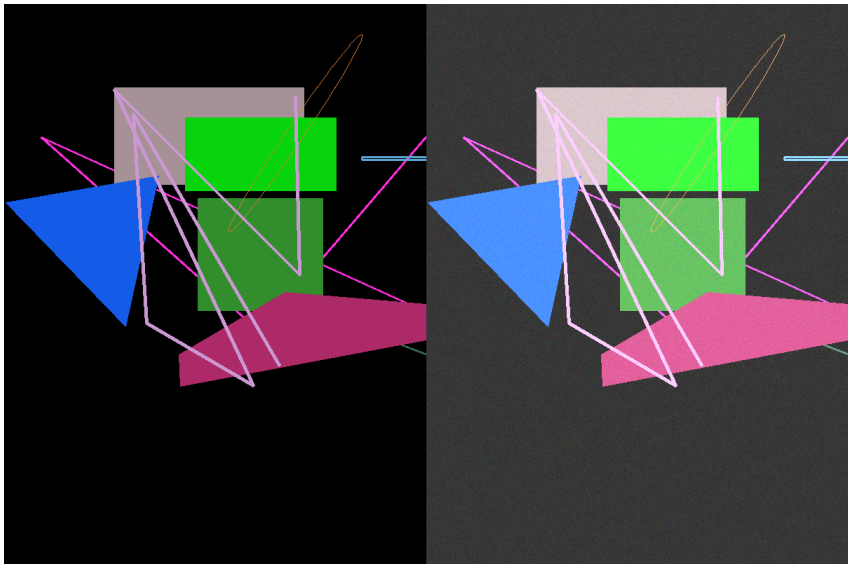


Figure 24: Example of a **clean** picture *(left)* and its **noisy** equivalent *(right)*

This is primordial for training, because the architecture need to learn from clean and noise pictures (see the next part for further explanation).
Once done, you can use this dataset to train the model :

```
Choose an option: 1
On how many images do you want to train? (Max = 100)
100
```

Figure 25: Training the siamese model

The training will be done over the number of pictures you want to analyze - the more is the better. Just enter the number you want, but be careful to not exceeding the maximum (Max =...) of picture available in the dataset ! You will be then invited to enter the number of epochs you want to train with :

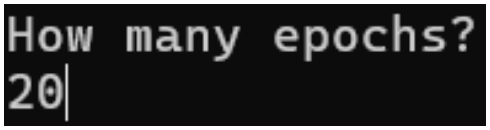


Figure 26: Enter the number of epochs

For each picture trained with, you can see in real time how many corners are detected by the following algorithms : Harris, ORB and FAST.

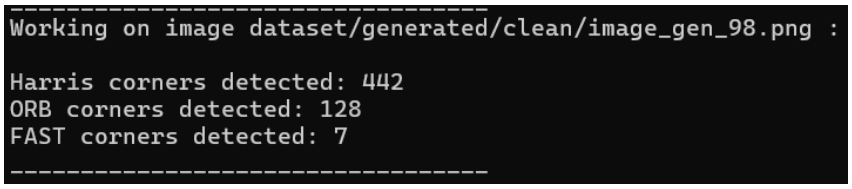


Figure 27: Harris, ORB and FAST result for each dataset

You can now execute the model on a picture. Just tap "2" and enter the name of the picture you want to work with - the picture must be in the same folder root.

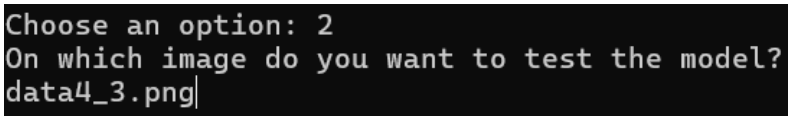
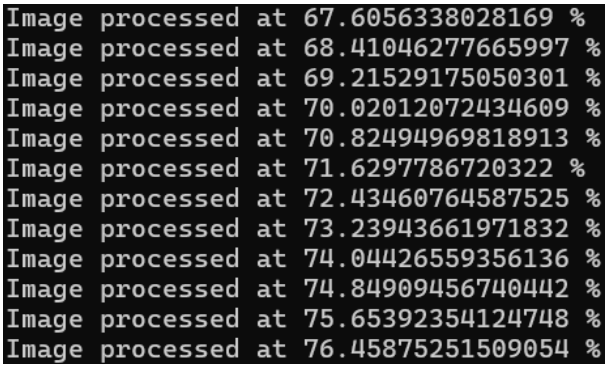


Figure 28: Testing the program on a picture

The percentage of process is given in real time :



After processing on each patch, you can see the distribution of similarity score between every one of them :

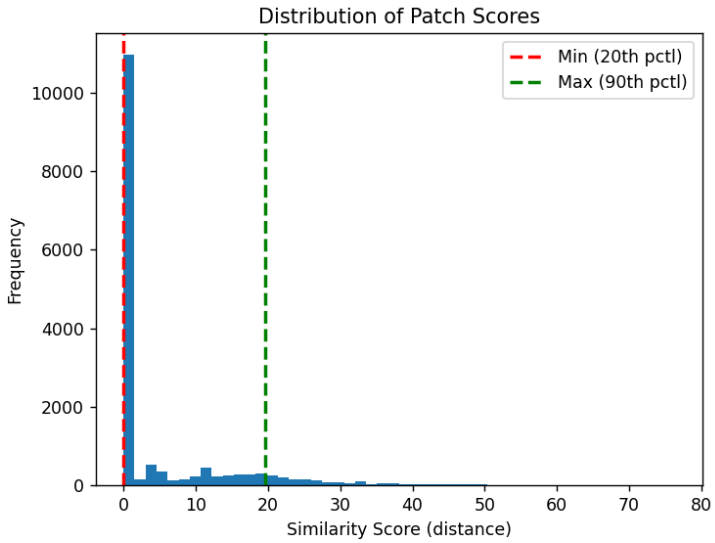


Figure 29: Distribution of patch matching score in the picture

This score allows us to see the minimum and maximum threshold between which a point is classified as a valid corner. This repartition of similarity depends of the training quality of the model as well as the picture itself : it is better to adjust it by taking a range between 20% (min) and 90% (max) of the global distribution :

```
Adjusted thresholds: min=0.0000, max=19.6130
```

Figure 30: Minimum and maximum threshold adjusted thanks to the repartition of patch scores

You can now view the result of the Siamese model, as well as the result given by each algorithm on the same picture, and compare them. In the Siamese window, each point is colored red, and the clusters (group of points) are colored blue. There is also the total number of points detected by the Siamese model and the number of cluster induced :

```
Number of corners detected by the Siamese model (before grouping): 3389
Number of clusters detected by the Siamese model: 0
```

Figure 31: Number of points and clusters detected by the Siamese model

To end the program, just enter the last option :

```
Choose an option: 3
Program finished.
```

4.1.2 Architecture of the network

The program works with 3 classes :

1. **SiameseNetwork** : it defines a CNN to compare each patch of a picture.

2. **ConstrastiveLoss** : it implements a function of loss to train the model, by measuring the euclidean distance between each output and separating similar from different pairs.

3. **CornerDataset** : this class manages the dataset. It loads clean and noisy pictures to generate positive and negative patches (with or without corner).

The other functions call these classes to run the program :

- **detect_corners** : it detects keypoints in a picture thanks to *Harris*, *ORB* and *FAST*.
- **group_nearby_points** : this function gathers detected keypoints, using *DBSCAN*.
- **train_siamese_network** : it trains the network using the previous class "CornerDataset".
- **evaluate_with_siamese** : it runs the model on a chosen picture, shows the result compared to *Harris ORB* and *FAST*.
- **load_siamese_model** : loads the model saved previously. This function is called before training or evaluating the model.
- **test_model** : takes the picture to work with and checks if the model is existing and loaded before evaluating it.
- **generate_dataset_call** : it calls the Python script *generate_dataset.py*, to generate a set of pictures to train with.
- **main** : the main menu that guides us through the program.

The **patch size** is put at *16*, and the model trains on *16* **batches**. Moreover, the network takes 16x16 pictures in gray scale (for a patch). It is made of **12 layers** (*9* of CNN and *3* of FC), with a number of filters from *64* to *256*. The CNN layers detect key patterns, and the FC layers convert the information in the picture into a vector of size 128 : this vector is used to compare each patch.

Globally, this architecture consists of training a siamese model with patches and predicting if a pattern is a corner or not. It works in 3 times : firstly, the training will take a picture in the dataset folder "clean", and apply *Harris-ORB-FAST* to detect its keypoints. Once the program knows the coordinates of the detected keypoints, the same coordinates are applied to the noisy equivalent of the picture : at this point, we know where the corners are in both clean and noisy pictures, despite the quality of the images.

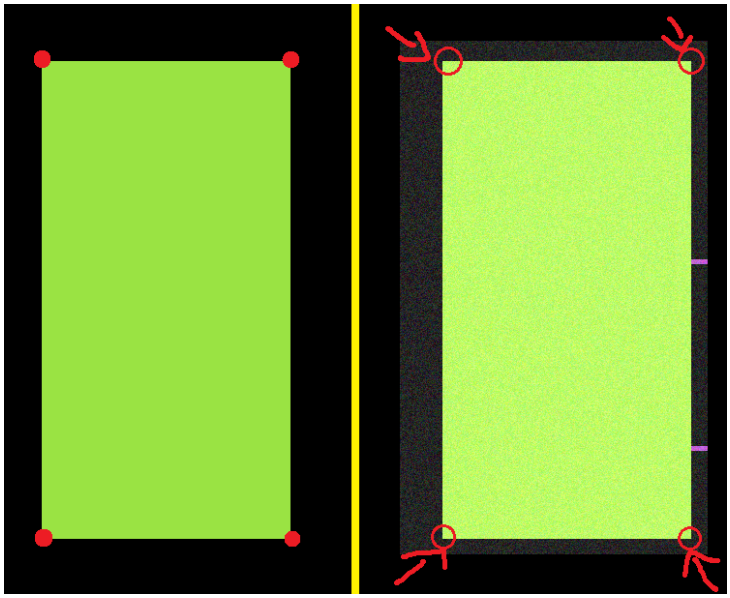


Figure 32: The algorithms detect corners on the clean image *(left)*, then we copy them on the noisy image *(right)*.

Then, the pictures are divised in patches, with positive ones (that contain keypoints) and negatives ones (without keypoints). These patches are given by pairs to the siamese network that will learn differences and similarities between patches, to at the end try to "understand" what is a key point and a corner.

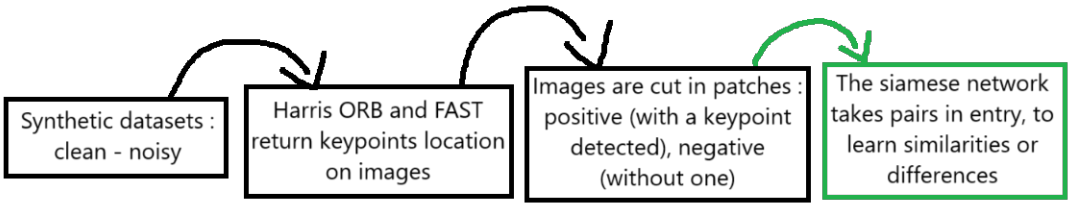


Figure 33: The architecture of the network

The main idea of this work was to combine the detecting skills of *Harris, FAST and ORB*, that can better detect a coin in a polygon or a circle, and make them work with noisy images to increase the robustness of the model on real images.

4.2 Optimizing Superpoint

Here is a deeper explanation of the Superpoint architecture, to search just after some of its limits and imagine perspectives of optimization.

4.2.1 Superpoint architecture

Here is the architecture of the code, with its classes and their role :

- **SuperPointNet** : the main network, that encodes and shares detecting and descipting heads.

- **SuperPointFrontend** : this class initialize and loads the model, then applies *run* that returns detected keypoints, their descriptors and a heat map. Thus, the function *nms_fast* takes keypoints to only keep the important ones.
- **PointTracker** : this class use the description of keypoints to track them between images.
- **VideoStreamer** : the manager of entry sources (camera, movies or pictures folder).

The network architecture (in **SuperPointNet**) consists of an encoder with **8 CNN layers** (64, 128, and 256 filters) and **three max-pooling** layers, which reduce the image size by a **factor of 8**. The decoder resizes the heat map to the original image size using a **softmax** and **pixel shuffle** methods, while a "*dustbin*" channel filters out noninteresting pixels.

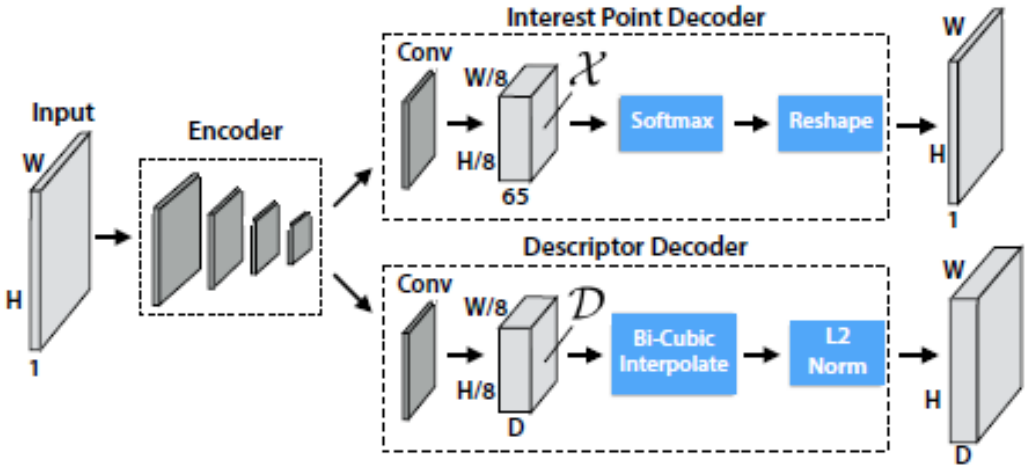


Figure 34: Encoder and Decoder of Superpoint *(Source)*

This allows a very fast and precise analyze of pictures, as well as description and tracking of keypoints.

4.2.2 Superpoint Limits

During its training, the *MagicPoint* does a lot of homographies on pictures. But I found that the only rotations done are in the Z and Y axis : it could be better to get the training dataset, and add a **rotation around the X** axis.

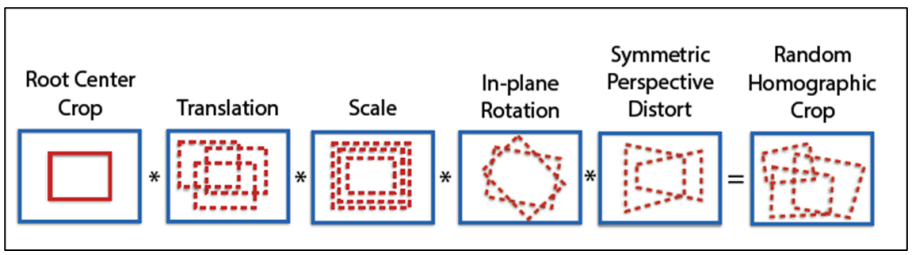


Figure 35: Superpoint training with homographies.

So, adding a rotation around the X axis could improve the tracking of objects that rotate in real-time like this.

Figure 36: Adding an homography around the **X axis**, for a better training

The Superpoint algorithm has an issue to recognize keypoints after intense light variations. In fact, this article explains the inaccuracy of their results during a tracking that is subject to intense light variations, from lighting to darkening.

We can imagine other new homographies for the training : a brightness one, and a noisy - that adds noise to the picture like the previous Siamese networks (figure 32).

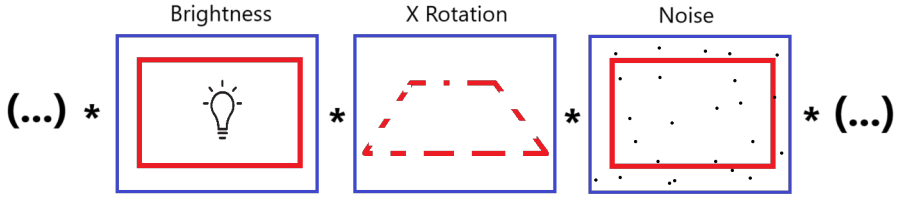


Figure 37: New homographies for the training : **Brightness**, **X rotation** and **Noise**.

Moreover, thanks to the previous explanation (Part 4.2.1), the code has many classes and functions, and could be scinded in different files, each of which corresponds to a class : it could increase the performance of the algorithm and had a better readability to the code.

5 Conclusion

In conclusion, the journey to create my own Siamese network was made thanks to the previous state-of-the-art, with classic and deep learning approaches. Each one of them has its balance between accuracy and weight to deploy, but mixing their qualities was the idea of the network.
Indeed, the classification between "corner" or "non-corner" patches is allowed by the combination of *Harris, FAST and ORB* analyze, that are not perfect or as accurate as a ground truth handmade.

This was not the idea, because I wanted the program to be low in weight and mobile, and able to create its own ground truth in the dataset, so it could automatically increase its accuracy as long as it creates its own data to train with. This is a low resource code to deploy, and a mimic of MagicPoint homographies but applied to the generation of datasets.

The main network could be improved by modifying the architecture : changing the patch size, adding new CNN, or training on better datasets - that is to say, improving the generation of pictures with more complex shapes.

I want to thank M. Ghulam-Sakhi Shokouh, the supervisor of this work, for his disponibility, whose guided me with a lot of discussion and explanations.