EURECOM
Sophia Antipolis

MALIS : Final Project Report

Eugène Berta, Patrick Iversenc

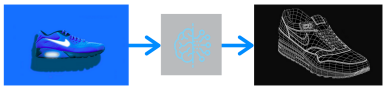
January 15, 2021

Git repository for the project : `https://github.com/eugeneberta/MALIS_Project.git`

Introduction

Motivation

This project deals with 3D pose estimation. It is a widespread problem in the fields of image processing and deep learning. We decided to limit our work to the specific case of sneaker's photographs. From a picture containing a sneaker, our goal is to extract the 3D position of the shoe in the repository induced by the camera.



Input to output, motivation schema

Applications

Pose estimation has a variety of applications. In the case of sneakers, estimating the 3D pose of a shoe in a picture can be used for image manipulation, to change its colors or shape for example. Such applications can be used in advertising, or to help designers and creators in the fashion industry. The french startup "Smartpixels", specialized in digital technologies for retail, is working on this technology. We have been working with this startup to develop our model, using their technologies to implement an innovative method for pose estimation.

Method

The method we decided to implement works with two separate steps :

1. The first step is "instance segmentation". It consists in extracting the silhouette of the sneaker in the photograph. This first step will give us the "shape" of the shoe in the image.

2. The second step is "pose estimation". We use this silhouette to infer the 3D pose of the shoe.

To implement the first step, we used existing deep learning technologies, widely used for image segmentation. We will present you this work in the first part of our report. For the second step, we developed an original model, predicting the pose of the shoe by comparing it's silhouette in the original picture with views from a sneaker's 3D model. We will present you this work in the second part of the report.

1 Sneakers segmentation and Deep Learning

To implement image segmentation, we used "detectron2", a software system based on PyTorch, edited by Facebook AI Research. We first used Detectron2 pre-trained models to segment pictures, but then we decided to train our own model with a self-made data set.

This first part has been coded on a Google colab file, on which you will find all our code, some examples and detailed explanations for each step of the process.

1.1 Building our dataset

To train Detectron2's neural networks with our own data, we needed to create an adequate data set for sneakers segmentation. The disadvantage of image segmentation is that it requires the pictures in the data set to be segmented. Such segmented images are hard to find on the internet, so we decided to build our own data set by hand. The first step was to gather a important number of sneaker image. Then, we cleaned the data set and kept only the most relevant pictures. Finally, we labeled and segmented the sneakers in each image. For this last task, we used a Python-based software named "labelme".

Labelme provided us with json files containing the coordinates of the points used for segmentation. We were able to convert our inputs (photographs) and outputs (json files) into a COCO data set, that we used to train our model.
Using this method, we labeled more than one hundred sneaker pictures, creating a training set for our deep neural network.

1.2 Results

Using our first training set, composed of eighty images only (we kept the rest for testing purposes), we trained our deep neural network with one hundred iterations and obtained pretty decent results.

With more complicated images, we observed that detectron2 tended to segment some elements in the background. We were able to solve this issue by setting a very high testing threshold before segmentation. Basically, if detectron2 estimates that an element is a sneaker with a score smaller that 97/100, we don't segment it. This way, our neural network provides us with very good results with simple enough images.

We can think of simple ways to improve the performances of our deep neural network :

- First, our training set only contains 80 images. It is widely known that deep neural network need a lot of training data to perform well. Using the method described in [1] for generating data, we could create a much larger data set to train our model.
- Then, with this extended data set, we could exploit the accuracy results to train the network with good parameters, which we didn't optimize for the moment.

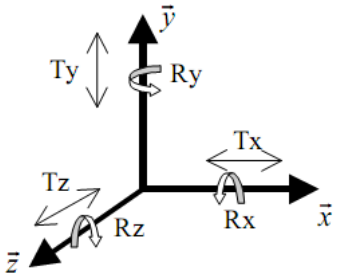
As explained, our segmentation algorithm can be improved easily. However, we decided to keep it this way and to focus on the second part of our project. When a sneaker appears entirely in the picture, our model is able to detect it and segment it precisely, the results are satisfying enough for our needs.

1.3 From image to binary mask

The final step was to extract a binary mask from the output of the neural network, for each sneaker present in the picture. By "binary mask", we mean a binary picture, where the ones corresponds to the pixels where the shoe is present. With this binary mask in hand, we have everything we need to implement the second part of our method.

2 Pose estimation

In this second part of our report, we will expose you our method to infer the 3D pose of the sneaker.



Let's consider a repository like the one above, with the camera fixed at the origin. Then, the 3D position of an object with respect to the camera can be described with six parameters : three translations Tx, Ty, Tz and three rotations Rx, Ry, Rz.

Note that it is equivalent to fix the shoe at the center of the repository and to determine the six parameters for the position of the camera. We will implement this solution as the 3D model provided by Smartpixels was built that way.

2.1 Silhouette Generator

As we introduced before, we had the chance to work with the teams of Smartpixels, a french startup specialized in virtual imaging and retail. Smartpixels is interested in the problem of pose estimation for commercial use.

The company develops photo-realistic 3D models of their clients products (sneakers, for example), and they offered us to use one these 3D models for our project.

We had to decide how to use a sneaker's 3D model to best help us with our problem. Using our deep learning algorithm, we can extract the silhouette of the shoe in the picture, under the form of a binary mask.

We had the idea that the silhouette of a sneaker in a picture contains information about it's 3D pose. The intuition is that if I show you the binary mask of a sneaker, you can estimate well it's 3D pose (orientation and location in space).

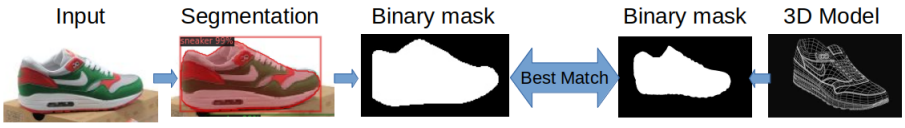
We decided to use the 3D model of the shoe to generate binary masks. Given a set of six parameters defining a 3D pose, the 3D model would return us the silhouette of the shoe as a binary image. Then, we would try to find the right set of parameters for the silhouette of the shoe to match the output of our neural network.

The teams at Smartpixels provided us with this exact tool : a "silhouette generator". Using the 3D model of a Nike Air Max 1 sneaker, the silhouette generator takes in input our six parameters (or "3D coordinates"), it sets the shoe at these coordinates, it takes a snapshot of the shoe at this location, and finally returns a binary mask containing the silhouette of the shoe in the snapshot.

This silhouette generator is a Unity web server, running on a computer that Smartpixels lent us. We can send an HTTP request containing our six parameters to this server, and the request returns a binary mask.

To recap the different steps of our process :

1. First, we use our deep neural network to extract the silhouette of a sneaker in a picture.
2. Then, we use Smartpixels' software to generate silhouette of a sneaker from different perspectives.
3. Using machine learning techniques from the MALIS course, we find the pose of the sneaker that gives us the best match between the two silhouettes.
4. When we reach this "best match", we have an estimation of the 3D pose.



2.2 Finding best match between silhouettes, distance metric

In order to find the best match between our two silhouettes, we needed to define a distance metric. We need to be able to compute the distance between two binary images : our binary mask extracted from the input image and the binary mask given by the 3D model of the shoe. This distance function will be the first step into optimizing the six parameters to estimate the position of the sneaker.

However, it is not obvious how to compute the distance between two binary image containing random shapes. The first solution we tried to implement is using the "Hausdorff distance", a mathematical tool defined specially for measuring a distance between geometrical shapes (in our case, silhouette of shoes). However, the python implementation of the Hausdorff distance proposed by the scipy library is really not adapted to binary masks. It would have required a great deal of work to implement it by ourselves so we decided to use another method.

Let's recall that the data we have in input consists in binary masks, that is to say arrays containing only the values 0 and 255 (or 1). This recalled us of the hamming distance, used to measure the distance between two 1D sequences. We extended the hamming distance to our 2D problem by converting our binary mask into a 1D sequence of bits, with each line following the precedent. We weren't sure that this new distance would provide us with good results, however, in turned out that experimentally, our algorithm behaved well with this distance metric.

The intuitive idea is that if the masks are the same ones, all bits in the sequence are in the same position and the hamming distance will be 0. On the other hand, the more different the binary masks are from each other, the more bits will be different in the sequence and the hamming distance will get bigger.

Given this distance metric, we could compute the distance between two binary masks. We now needed an algorithm to minimize this distance.

2.3 Gradient descent algorithm

Our distance metric enables us to define a cost function :

$$L(\theta) = H(B_{input}, B(\theta))$$

With :

- $H(B_1, B_2)$ the distance between two binary masks as defined in the previous section.
- B_{input} the binary mask, given by our neural network, from which we try to get as close as possible.
- $\theta = (Tx, Ty, Tz, Rx, Ry, Rz)$ the set of parameters defining a 3D pose of the sneaker.
- $B(x)$ the binary mask returned by the silhouette generator when given the parameters θ.

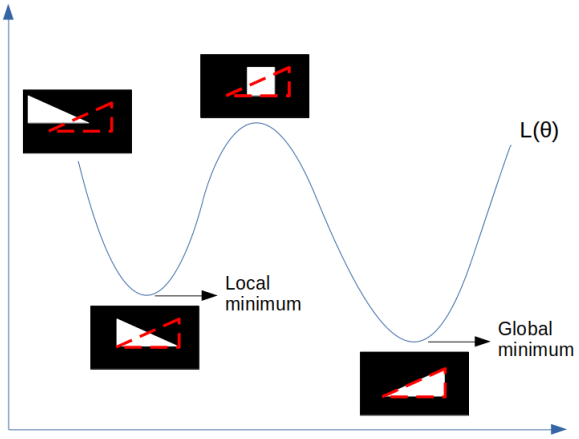
Given this cost function, we decided to implement gradient descent algorithm to find $\theta^* = \min_{\theta}(L(\theta))$, that is the optimal set of parameters to describe the 3D pose of the shoe.

However, we met three main issues with this method that forced us to adapt the algorithm.

1. The first problem we met is the following one : the usual gradient descent algorithm solves the matter of the initialization by picking a random starting point for the algorithm. In our case, that was not possible. Indeed, if we pick θ_0 randomly, it is very likely that the silhouette generator will return an empty image. Indeed, in most cases, when we randomly pick the pose of the sneaker in space, the sneaker doesn't seats in front of the camera. To solve this first problem, we decided to fix the starting point of the algorithm ourselves, with a set of parameters for which the sneaker is in the middle of the picture.

Binary mask for the initial set of parameters

2. The second problem we faced is that the cost function $L(\theta)$ is not convex. It is likely that the gradient algorithm won't reach the global minimum of the cost function. A simple example of this issue can be given with the orientation of the shoe : On the following figure, a white triangle symbolizes the silhouette of a shoe, so the orientation of the triangle gives us the orientation of the shoe. The red-dotted silhouette shows us the output of our Neural Network, it is the best match to reach. If, as in the schema, the initial set of parameters corresponds to the opposite orientation of the shoe, our Loss function will meet a deep local minimum before reaching the best match. It is very unlikely that our algorithm succeeds in this case. Reaching the global minimum would require a 180° rotation on the vertical axis, which include passing through a local maximum.



The solution we proposed to tackle this problem is to run our gradient descent multiple times with different starting parameters, and to keep the result that minimize better the loss function.

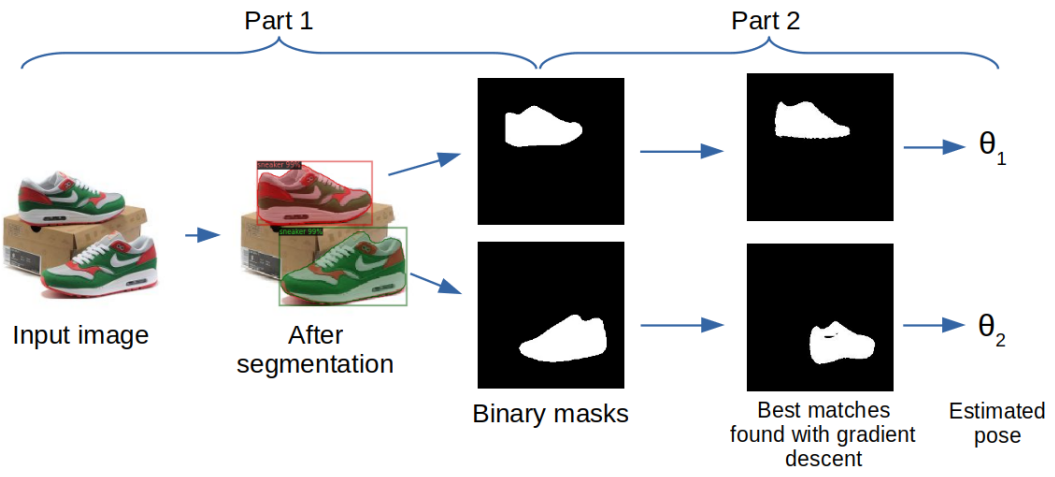
3. The last issue we faced is the choice of the parameters and stop criterion for our gradient descent. We decided to implement a backtracking line search algorithm to choose the learning rate at each iteration. With this implementation, our gradient descent behaves well and reaches a minimum in around 15 iterations. At one point, the learning rate fixed by backtracking line search would become very low. This gave us a good stop criterion, when the learning rate reaches a negligible value, we stop our gradient descent.

Conclusion, Results and contributions

Results

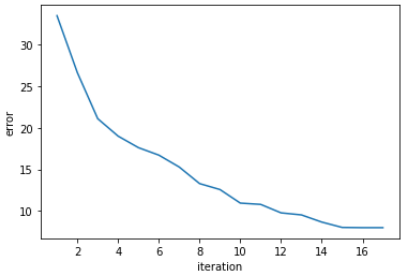
We managed to implement our gradient descent, and to make it work in most cases.

From this position, our gradient descent is working well when the orientation of the silhouette to match is the same. As expected, if the shoe is oriented the other way, our algorithm dives in the first local minimum, without rotating the shoe. This is illustrated in the following schema :



As you see, the best match estimated by our gradient descent in the second case correspond to a wrong orientation of the shoe.

We display here the error rate with the number of iteration for the estimation of θ_1, where the result appears to be pretty good.



As we said before, a good way improve our method would be to run our gradient descent from multiple starting points and to keep the result with the lowest value of error.

Conclusion

We are very happy to conclude that our method for pose estimation provides results. We didn't have the chance to test these results in a proper way so it is hard to conclude on the use of this method. It appears that in certain cases our results are promising, and we could generalize our method to get these results in any case by running our gradient descent multiple times. However, the graphical gradient descent that we implemented presents major drawbacks. First of all, when compared to the traditional deep learning methods used for pose estimation, our method takes a tremendous amount of time, and can hardly be used with videos. Then, finding a proper and easily calculable distance metric for our graphical gradient descent is a real challenge. Finally, the use of the silhouette, even if it simplifies the problem, leaves us with imprecise results.

Let's not forget that there is much more information in a sneaker picture that the silhouette of the shoe. The colors, light, logos and structure of the shoe contain a lot of information that could help pose estimation. This project is a first step towards a machine learning approach to pose estimation (opposed to the classical deep learning approaches). A fully functional and efficient algorithm would require much more time and work, but working on this project convinced us that it is doable.

Contributions

We have been working together for most of the project as we live in the same house next to the school. Each one of us has participated in every step of the project.

References

[1] Javier Martínez-Cesteros, Gonzalo López-Nicolás, "Automatic Image Dataset Generation for Footwear Detection", Grupo de robótica, percepción y tiempo real (RoPeRT)Instituto de Investigación en Ingeniería de Aragón (I3A) Universidad de Zaragoza, Mariano Esquillor s/n, 50018, Zaragoza, Spain. https://www.readcube.com/articles/10.26754%2Fjji-i3a.003547

[2] Gilbert Tanner, (2020). Detectron2 Train a Instance Segmentation Model. https://gilberttanner.com/blog/detectron2-train-a-instance-segmentation-model

[3] Chengwei Zhang, (2019). How to train Detectron2 with Custom COCO Datasets . https://www.dlology.com/blog/how-to-train-detectron2-with-custom-coco-datasets/

[4] Official Git Repository for Detectron2. Facebook Research. https://github.com/facebookresearch/detectron2

[5] Pascale Monasse, Kimia Nadjahi. Classez et Segmentez des Données Visuelles. https://openclassrooms.com/fr/courses/4470531-classez-et-segmentez-des-donnees-visuelles

[6] Derrick Mwiti (2020). Image Segmentation in 2020: Architectures, Losses, Datasets, and Frameworks. https://neptune.ai/blog/image-segmentation-in-2020

[7] S. Song and T. Mei, "When Multimedia Meets Fashion," in IEEE MultiMedia, vol. 25, no. 3, pp. 102-108, July-Sept. 2018, doi: 10.1109/MMUL.2018.2875860. https://ieeexplore.ieee.org/document/8589035