Master Research Project Report

Carlos Castillo

August 28, 2023

Contents

1 Introduction

1.1 Project Overview

AlphaTracker is a groundbreaking multi-animal tracking and behavioral analysis tool that revolutionizes system neuroscience research. Designed to empower researchers, it incorporates advanced features such as multi-animal tracking, pose estimation, and unsupervised behavioral clustering. AlphaTracker sets new standards in accuracy for multi-animal tracking, providing a strong foundation for rigorous biological studies.

The application's exceptional capabilities stem from cutting-edge computer vision and machine learning techniques. By seamlessly integrating multi-animal tracking and pose estimation, AlphaTracker enables researchers to gain deeper insights into the complex interactions and behaviors of multiple subjects in real-time video streams.

One of the key strengths of AlphaTracker lies in its adaptability to diverse research settings. The application is designed to operate with minimal hardware requirements, making use of regular webcams, and boasts an efficient training procedure. This ease of adoption facilitates widespread usage across various neuroscience laboratories, ensuring researchers can readily harness its potential for their experiments.

The AlphaTracker project is the result of the collective efforts of a highly skilled and collaborative team, comprising computer vision engineers, machine learning experts, and software developers. Each member's expertise plays a crucial role in developing specific components of the application, culminating in an integrated and high-performing solution.

Within this report, the focus centers on my individual contributions to the AlphaTracker project. It presents in-depth insights into the methodologies I employed, the challenges I encountered, and the results achieved during my involvement. By sharing my accomplishments, I aim to highlight the significance of my work within the team and demonstrate how it contributes to the success of AlphaTracker as a groundbreaking tool for advancing system neuroscience research.

1.2 Role and Responsibilities

As a member of the support group for AlphaTracker, I was part of a team of master's degree students working closely with the main developers, who were primarily PhD researchers. Our role in the project was vital in assisting the core team with various essential tasks, contributing to the overall success of the application's development and refinement. Our responsibilities encompassed a diverse range of activities, each tailored to complement the expertise of the main developers and ensure the efficient progression of the project.

1. Data Preparation: One of our key responsibilities was to handle data preparation, including data collection, cleaning, and organization. We ensured that the

datasets used for model training and evaluation were curated meticulously to provide accurate and reliable inputs for the algorithms.

2. Model Training, Correction, and Visualization: Our team actively engaged in model training, fine-tuning, and corrections. We worked collaboratively to optimize the performance of the tracking and pose estimation models, employing cutting-edge computer vision and machine learning techniques. Additionally, we developed visualizations to analyze model outputs and identify potential areas for improvement.

3. Metric Development and Testing: Another crucial aspect of our role was the development and testing of various evaluation metrics. We designed and implemented metrics to assess the tracking accuracy, pose estimation precision, and clustering performance, thereby contributing to the quantitative evaluation of the application's capabilities.

4. Evaluation of Pose Estimation: We dedicated effort to rigorously evaluating the pose estimation functionality within AlphaTracker. By conducting thorough assessments and comparisons with ground truth data, we helped ensure the accuracy and reliability of the pose estimation algorithms.

5. Clustering Algorithm Testing: Our team also played a significant role in testing different clustering algorithms, both for visual and audio data. This involved conducting experiments and analyzing results to determine the most effective approach for behavioral clustering.

6. Documentation and Reporting: Throughout the project, we diligently documented our work, methodologies, and results. We prepared detailed reports and contributed to the project's documentation, providing clear and comprehensive insights into our contributions and findings.

7. Continuous Collaboration and Support: As a support group, we maintained open and effective communication with the main developers. We actively participated in team meetings, discussions, and brainstorming sessions, offering valuable inputs to enhance the application's functionality and addressing challenges collaboratively.

Our contributions as a support group significantly enriched the development process of AlphaTracker, helping to bridge the gap between research and application. By taking on diverse responsibilities and leveraging our skills, we played a crucial role in the success of the project, contributing to the advancement of system neuroscience research and the empowering tools for the scientific community.

1.3 Scope of Work

As a member of a project support group, my role is to assist developers with tasks as needed. Not having belonged to the project since its inception, the main tasks in which I have given support have not been related to software development of the application itself, but have been more related to more technically simple and experimental tasks. In any case, the understanding of the operation of the application and its software structure has been necessary to perform these tasks. Finally, the objective of this project also lies in learning the techniques used in the project and the appropriate methodology for the development of a technological project of this category.

2 AlphaTracker Description

The AlphaTracker pipeline consists of three main stages: tracking (AlphaTracker), behavioral clustering, and result analysis with a customized user interface (UI). AlphaTracker allows for multi-animal tracking on videos recorded via webcam, making it convenient and cost-effective for laboratory settings. The behavioral clustering stage enables unbiased identification of behavioral motifs, with results further reviewed and error-corrected using a customized UI.

The tracking component (AlphaTracker) is adapted from AlphaPose15, a human pose estimation algorithm. It comprises three steps: animal detection using YOLOv316, keypoint estimation using Squeeze-and-Excitation Networks (SENet)17, and identity (ID) tracking across frames. The pipeline proposes a novel target association method to consistently track IDs of nearly identical animals by defining descriptors for animal positions and orientations and calculating similarities between descriptors for tracking.

To address inaccuracies due to tracking errors or occlusion, the pipeline utilizes Kalman filtering21 to predict keypoint locations in consecutive frames. This filtering technique models motion with velocity and acceleration, providing enhanced tracking accuracy when users amend results in the UI.

The pipeline's innovative approach to multi-animal tracking, behavioral clustering, and result analysis using a user-friendly interface demonstrates its potential to advance system neuroscience research and empower researchers in their studies of animal behavior. [1]

2.1 Tracking

AlphaTracker demonstrates reliable performance in tracking multiple unmarked animals. It excels in complex environments like home cages with bedding and metal operant chambers, achieving high accuracy.

The tool handles challenges posed by occlusion due to head implants, maintaining robust performance. It also exhibits good tolerance for low-resolution videos from webcams (576p), enabling continuous monitoring over extended periods.

In social neuroscience research, AlphaTracker efficiently tracks four identical-looking mice during home cage interaction.

The model can be quickly and easily trained, involving two steps: training the animal detector and training the pose estimator. Users can modify training hyperparameters to optimize performance without overfitting risks.

AlphaTracker offers a rapid and accessible solution for neuroscience labs, as testing showed quick training times (0.2 seconds for a batch of 10 images and 30 minutes for 6000 training images) on compatible hardware. [1]

2.2 Clustering

AlphaTracker's behavioral clustering allows unsupervised clustering of individual and social behaviors, minimizing human bias. Features extracted from keypoints and binary masks are input to the hierarchical clustering algorithm. It successfully captures various behaviors and facilitates associative analysis between behavior motifs and experimental factors. The algorithm's performance is validated using the Adjusted Rand Index (ARI), showing improved results with larger datasets. AlphaTracker can be operated on Google Colab without GPUs, offering accessibility and comparable performance to local GPU-enabled setups. Detailed usage guidance can be found in the project's Github repository. [1]

2.3 User Interface

Customized user interfaces (UIs) for inspecting and revising tracking and clustering results were designed to address potential errors in the deep learning framework. The web-based UIs run on systems with a pre-installed Python environment.

The tracking UI enables users to scroll through the video, inspect keypoints, and verify mouse IDs over time. The event timeline facilitates error identification, displaying the confidence score to indicate periods of low confidence. Users can easily correct errors by dragging keypoints or reassigning identities. The "curate" function automatically corrects subsequent frames based on user adjustments.

The clustering UI features an interactive dendrogram to display clustering results. Users can choose to view broader cluster assignments or explore details. The scatterplot in the timeline visualizes clip distributions of specific clusters. Users can rename clusters, modify assignments, merge clusters, and save curated results in JSON format for further analysis. [1]

3 Tasks Description

In this section I will focus on my personal work done on each assignment received. Each task will be composed of 4 sections.

The first one would be the task description, in this section I will explain the instructions we received for each task. Here I will mention each step of the task, including the parts I did not work on.

The next part would be the approach and methodology I used to tackle the task. This part will focus only on the parts of the task that I did work on, including how I tackled the task and the steps I planned to follow.

Next, I will explain what problems I encountered in following the initial plan and how I tried to solve them in order to achieve the results.

Finally, I will write a short paragraph with the conclusions and the results I obtained, including if I did not achieve satisfactory results.

3.1 Task 1: Data Preparation

3.1.1 Description of the task

The first step of this task was to investigate how the *AlphaTracker* application worked. This step was necessary to be able to manipulate the data, since in a first step we had to know how they were structured in order to transform them to the desired format. Also, there were multiple files, so it was crucial for us to identify which files corresponded to which information.

Once we had a more complete picture of the project, our goal was to manipulate the data. We were given several *.csv* files and had to generate *.json* files with a specific format (See in Figure 1).

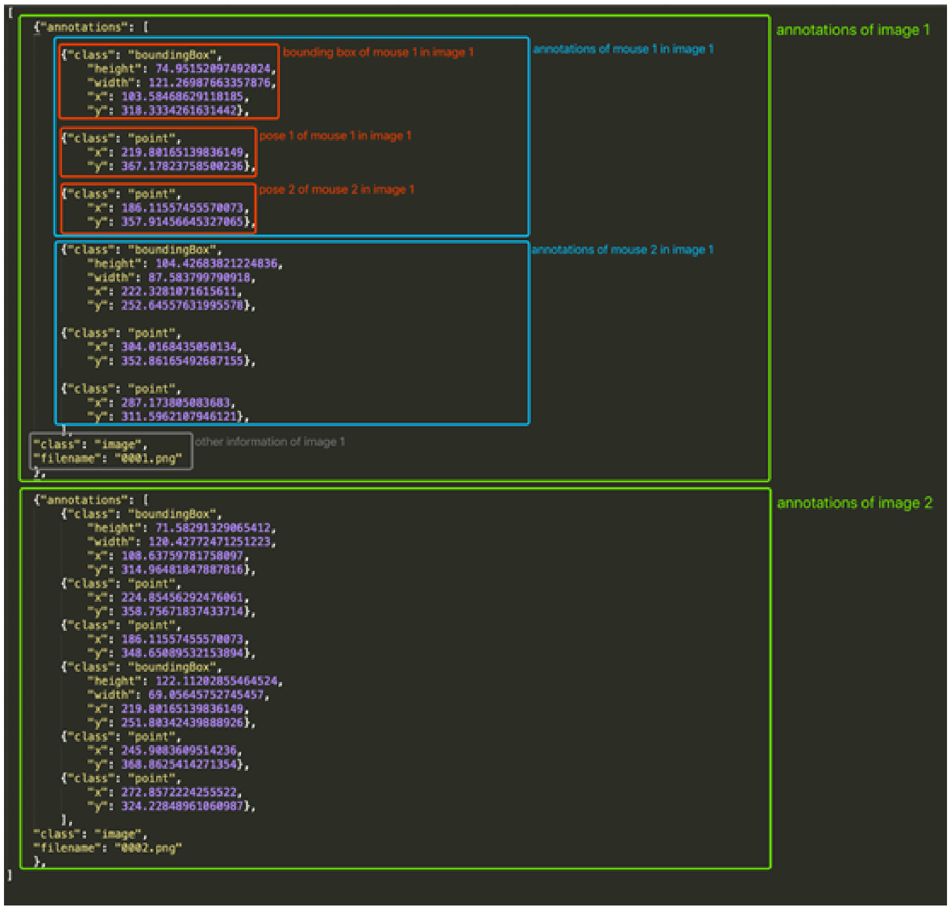
We had to generate a single output file, so we also had to merge all the information from the different *.csv* files we got as input.

We divided the main task into different subtasks, so I went for the cleaning and merging part.

3.1.2 Approach and Methodology Used

Since I had some experience with data manipulation in Python, I decided to use it as the main tool for this project. In this case, I thought it would be useful to generate some scripts that would clean, merge and finally transform the data.

To generate these scripts, I first created a python notebook (more interactive and easier to debug), and when I got the desired code, I converted those notebooks into *.py* files (faster and less expensive).


annotations of image 1
bounding box of mouse 1 in image 1
annotations of mouse 1 in image 1
pose 1 of mouse 1 in image 1
pose 2 of mouse 2 in image 1
annotations of mouse 2 in image 1
other information of image 1
annotations of image 2

```
[
  {"annotations": [
      {"class": "boundingBox",
          "height": 74.95152097492024,
          "width": 121.26987663357876,
          "x": 103.58468629118185,
          "y": 318.3334261631442},

      {"class": "point",
          "x": 219.80165139836149,
          "y": 367.17823758500236},

      {"class": "point",
          "x": 186.11557455570073,
          "y": 357.91456645327065},

      {"class": "boundingBox",
          "height": 104.42683821224836,
          "width": 87.583799790918,
          "x": 222.3281071615611,
          "y": 252.64557631995578},

      {"class": "point",
          "x": 304.0168435050134,
          "y": 352.86165492687155},

      {"class": "point",
          "x": 287.1738050083683,
          "y": 311.5962107946121},
      ],
  "class": "image",
  "filename": "0001.png"
  },
  {"annotations": [
      {"class": "boundingBox",
          "height": 71.58291329065412,
          "width": 120.42772471251223,
          "x": 108.63759781758097,
          "y": 314.96481847887816},
      {"class": "point",
          "x": 224.85456292476061,
          "y": 358.75671837433714},
      {"class": "point",
          "x": 186.11557455570073,
          "y": 348.65089532153894},
      {"class": "boundingBox",
          "height": 122.11202855464524,
          "width": 69.05645752745457,
          "x": 219.80165139836149,
          "y": 251.80342439888926},
      {"class": "point",
          "x": 245.9083609514236,
          "y": 368.8625414271354},
      {"class": "point",
          "x": 272.8572224255522,
          "y": 324.22848961060987},
      ],
  "class": "image",
  "filename": "0002.png"
  },
]
```

Figure 1: JSON template

3.1.3 Challenges Encountered and Solutions Implemented

The first challenge I faced when trying to visualize the data was that some of the files could not be loaded. The cause of this problem was that most of the raw data had been entered manually, so it contained some human errors. To solve this problem, I tried loading each file in python with a loop over the directory and checked the python cell error information to find out which file was causing the problem. Then, I focused on that file to find out which row was causing the problem. Because this problem only affected a small number of files, it was not worth developing an automatic fix, and I could fix all the files in a reasonable amount of time.

Once I had all the files ready, the next step was to clean them up and put them in the same format so that I could merge them later. For this step, I created the *clean_data.ipynb* notebook. This notebook works as follows:

- Rename the columns
- Set the index
- Delete useless columns
- Modify headers
- Deleting rows with header information

Once I checked that all the steps worked correctly, I generated the *clean_data.py* script and ran it on each file with a for loop in the ubuntu terminal.

You can see the code at the end of the document.

This script generated the *raw data* file containing each and every cleaned and structured data file.

The next step was to combine all the files into a single *.csv* file. For this task I created the *csv_generator.ipynb* file. The notebook works as follows:

- Create a list of all csv files.
- Load all data sets into a data frame.
- Sort the data frame by csv index and png filename
- Create new features for mouse 1 and mouse 2
- Set the final columns
- Write the data frame to a csv file

Since this task did not have to be repeated many times, I did not generate the script and just ran the notebook.

You can see the code at the end of the document, in the *Code Snippets* section.

This notebook generated the file *processed_data.csv*. It contains the data from each file, cleaned, structured and sorted by frame.

3.1.4 Results and Future work

Performing these steps resulted in the desired output, a csv file with each frame's information completed, cleaned and sorted. The next step would be to generate the *.json* file, but this specific task was part of the work of other teammates.

3.2 Task 2: Model Training

3.2.1 Description of the task

The main objective of this task was to interact with the AlphaTracker application to better understand how it works. First we needed to install the program, then verify the installation, set up the configurations and train the model. We were encouraged to train the model separately on different data sets depending on the number of mice in the images.

The required presentation included the AlphaTracker demonstration results as an image and the training checkpoints and test results.

3.2.2 Approach and Methodology Used

There were two options to perform the installation, the first and most recommended was to install conda and use a GPU in our computer. This option required using linux and having a dedicated GPU in my computer. The alternative option was to perform a google colab training, much simpler but less powerful.

I followed the steps given by the official instructions of the AlphaTracker application. [2]

1. Preparation: Download the Alphatracker repository by cloning it from github.
2. Install Conda: Install Anaconda3 or miniconda3 and set up a specific environment.
3. NVIDIA driver: Check the version of my nvidia driver and update it if necessary.
4. Install AlphaTracker: Update the install.sh script to reflect the correct hardware for your system.

Once all the steps were completed, or I decided to go ahead with the google colab option, I had to train the model.

Training the model had 3 steps:

1. Data preparation: The data needs to be labeled in order to train the model.

2. Configuration: Load the parameters to the model.
3. Execute the code: Change directory to the alphatracker folder and train the model.

3.2.3 Challenges Encountered and Solutions Implemented

Since I usually use linux and was not sure whether or not I had a dedicated GPU, I tried this option first. I successfully completed the first two steps, but once I tried to install the NVIDIA driver, I got several errors. I later discovered that this was because I did not have a GPU in my system.

This caused many incompatibilities in my system and many important packages were removed. I had to fix the operating system, which took a long time.

Then, I tried to go ahead with the google colab option, but it got stuck while updating the parameters and I was unable to complete the model training this week.

3.2.4 Results and Future work

As I had several problems with the operating system and the google colab option required a lot of files to be examined carefully, I did not get conclusive results this week. On the other hand, this work allowed me to find out more about how the application worked and how to continue the task the following week.

3.3 Task 3: Correction and Visualization

3.3.1 Description of the task

This task consisted of 3 main parts.

1. Data annotation: This section consisted of identifying some problems in previous annotations and trying to fix them. The main problems in the "Mouse_behaviour_video_recording" data set were:

- The annotation is incomplete
- The invisible joint must be changed to meet Alphatracker annotation requirements.

On the other hand, the errors in the "Mouse_pose_dataset" dataset were:

- Rechecking a coordinate problem.
- Removing some points from the dataset, as they were not included in other datasets.

The annotation files should be reordered as follows: For each data set, prepare four json files: annotations for one and two mice from top and side view. In total, there would be eight annotations.

1. Model training: In this part we need to train two models for the top and side view, respectively. We had to use annotations from two datasets to train them together.
2. Visualization: Visualize the ground truth joints for both datasets before training, and run the demo with the trained models after training.

3.3.2 Approach and Methodology Used

Since I was unable to perform the training in the previous task, this time I focused on trying to solve the problems I had previously and complete the training of the model.

This time, I performed the training in google colab. I followed the 9 steps below as indicated in the guide:

1. Upload the data in a folder called *training-data* to the main folder *My drive.*
2. Open the alphatracker notebook in Google Drive.
3. Change the runtime type to GPU.
4. Connect our Python session to Google Drive
5. Download the AlphaTracker into my Google Drive
6. Set the variables that define the training
7. Install conda and pytorch in my Google Drive
8. Train the model with my data
9. Tracking / pose estimation

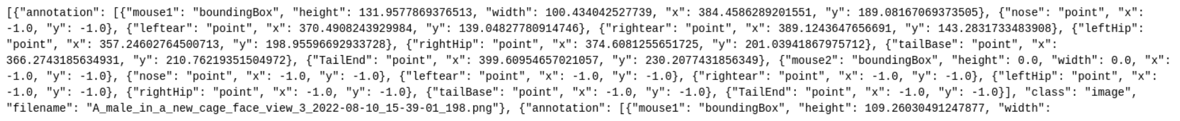
First, I needed to train the model with the demo datasets, and then use the data with our annotations.

3.3.3 Challenges Encountered and Solutions Implemented

The training with the demonstration data sets was successful. I followed all the steps and in the end I obtained a trained model with the given data.

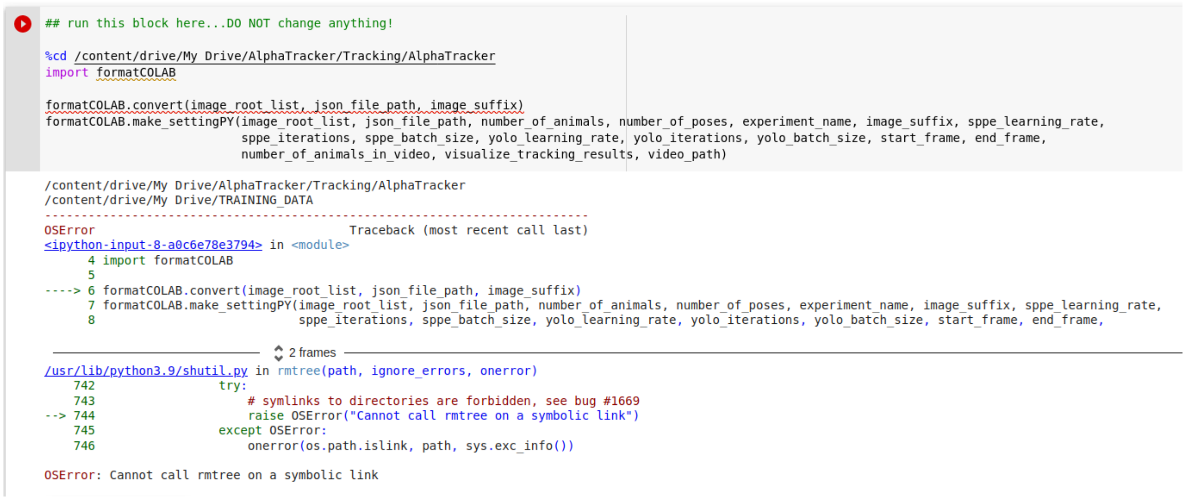
On the other hand, I encountered some problems when trying to train with my annotated data.

The first one was related to modifying parameters and routes. While the parameters were set by default for the demo data, I had to modify them to train the model with our own data. The problem here was that some of the required variables were not very



```
[{"annotation": [{"mouse1": "boundingBox", "height": 131.9577869376513, "width": 100.434042527739, "x": 384.4586289201551, "y": 189.08167069373505}, {"nose": "point", "x":
-1.0, "y": -1.0}, {"leftear": "point", "x": 370.4908243929984, "y": 139.04827780914746}, {"rightear": "point", "x": 389.1243647656691, "y": 143.2831733483908}, {"leftHip":
"point", "x": 357.24602764500713, "y": 198.95596692933728}, {"rightHip": "point", "x": 374.6081255651725, "y": 201.03941867975712}, {"tailBase": "point", "x":
366.2743185634931, "y": 210.76219351504972}, {"TailEnd": "point", "x": 399.60954657021057, "y": 230.2077431856349}, {"mouse2": "boundingBox", "height": 0.0, "width": 0.0, "x":
-1.0, "y": -1.0}, {"nose": "point", "x": -1.0, "y": -1.0}, {"leftear": "point", "x": -1.0, "y": -1.0}, {"rightear": "point", "x": -1.0, "y": -1.0}, {"leftHip": "point", "x":
-1.0, "y": -1.0}, {"rightHip": "point", "x": -1.0, "y": -1.0}, {"tailBase": "point", "x": -1.0, "y": -1.0}, {"TailEnd": "point", "x": -1.0, "y": -1.0}], "class": "image",
"filename": "A_male_in_a_new_cage_face_view_3_2022-08-10_15-39-01_198.png"}, {"annotation": [{"mouse1": "boundingBox", "height": 109.26030491247877, "width":
```

Figure 2: Annotated data


```
## run this block here...DO NOT change anything!

%cd /content/drive/My Drive/AlphaTracker/Tracking/AlphaTracker
import formatCOLAB

formatCOLAB.convert(image_root_list, json_file_path, image_suffix)
formatCOLAB.make_settingPY(image_root_list, json_file_path, number_of_animals, number_of_poses, experiment_name, image_suffix, sppe_learning_rate,
                           sppe_iterations, sppe_batch_size, yolo_learning_rate, yolo_iterations, yolo_batch_size, start_frame, end_frame,
                           number_of_animals_in_video, visualize_tracking_results, video_path)
```

```
/content/drive/My Drive/AlphaTracker/Tracking/AlphaTracker
/content/drive/My Drive/TRAINING_DATA
---------------------------------------------------------------------------
OSError                                   Traceback (most recent call last)
<ipython-input-8-a0c6e78e3794> in <module>
      4 import formatCOLAB
      5
----> 6 formatCOLAB.convert(image_root_list, json_file_path, image_suffix)
      7 formatCOLAB.make_settingPY(image_root_list, json_file_path, number_of_animals, number_of_poses, experiment_name, image_suffix, sppe_learning_rate,
      8                            sppe_iterations, sppe_batch_size, yolo_learning_rate, yolo_iterations, yolo_batch_size, start_frame, end_frame,

                                    2 frames
/usr/lib/python3.9/shutil.py in rmtree(path, ignore_errors, onerror)
    742             try:
    743                 # symlinks to directories are forbidden, see bug #1669
--> 744                 raise OSError("Cannot call rmtree on a symbolic link")
    745             except OSError:
    746                 onerror(os.path.islink, path, sys.exc_info())

OSError: Cannot call rmtree on a symbolic link
```

Figure 3: Unknown error

clear in our annotated data. For example, the data set we were trying to work with was obtained from a video in which sometimes a single mouse appears and sometimes a second mouse appears. Thus, it was not easy to define the correct parameter. To fix this problem I tried to perform the training with different parameters.

The next problem I had to face was about the format of the files. When I tried to train the model, I could not load the data because the json format was not structured in the same way as the test data sets (See in 2).

To fix this problem it was necessary to transform the whole files, so in order to test the model and obtain some results, I created a new file with the first rows of the annotated data written in the correct format.

Finally, when I had fixed the two previous errors, I ran the model and I obtained an unknown error (See in 3).

When I got this error I tried to find out the source by inspecting the library files that caused the problem. However, since I had not participated in the development of the application, it became very complicated to find out the origin of the error, since many functions of different libraries were involved.

3.3.4 Results and Future work

There were different conclusions and results throughout this week:

- Complete the training for the first time
- Understand how to connect the data to the google colab notebook
- Realize the annotated data was not in the correct format

Related to the future work, it was crucial to figure out which was the correct format of the *.json* files in order to train the model. Also, was necessary to figure out what caused the problem and how to solve it.

3.4 Task 4: Evaluation and Visualization

3.4.1 Description of the task

The goal was to evaluate the 7 predictions for each video of a concrete dataset. There were different metrics we could use to evaluate the predictions. We were encouraged to use the following ones:

1. Track-mAP
2. IDF1
3. HOTA
4. MOTA

In relation to the visualization part, we had to display the bounding boxes of mice.

The required submission included the evaluation results, the evaluation code and the visualization for predicted results.

3.4.2 Approach and Methodology Used

Since there were 4 metrics we had to work with, we decided to assign one of them for each of us. In my case, I chose the HOTA metric.

Firstly, I did some research on how the metric worked.

The "Higher Order Tracking Accuracy" is an evaluation metric used to assess the performance of object tracking algorithms in computer vision and video analysis tasks.

HOTA takes into consideration multiple aspects of tracking performance, such as localization accuracy, false positives, and false negatives, while also considering higher-order relationships between objects across frames. Here's how HOTA works:

1. Matching Objects: Before calculating any metrics, the objects predicted by the tracking algorithm are matched with the ground truth objects. This matching

process is typically done using some form of similarity measure between predicted and ground truth bounding boxes, often based on the Intersection over Union (IoU) of the bounding boxes.

2. HOTA Calculation: The HOTA metric is calculated by considering three main components:

1. Localization Accuracy (L): This component evaluates how well the predicted bounding boxes align with the ground truth bounding boxes. It takes into account both the position and scale differences between predicted and ground truth boxes.

2. False Positives (FP): This component measures the number of false positive predictions, i.e., predicted objects that don't correspond to any ground truth object.

3. False Negatives (FN): This component measures the number of false negative predictions, i.e., ground truth objects that aren't correctly predicted by the algorithm.

1. Higher-Order Relationships: In addition to the basic metrics (L, FP, FN), HOTA considers higher-order relationships between objects. These relationships involve the temporal consistency of object identities across frames. For instance, if an object is correctly tracked across multiple frames, it contributes positively to the HOTA score. On the other hand, inconsistencies in tracking contribute negatively.

2. Aggregation: HOTA combines these components into a single metric by considering their interactions. It can be calculated in different ways, but often involves finding a balance between localization accuracy and the ability to maintain correct tracks over time.

3. Comparison and Interpretation: HOTA offers a comprehensive evaluation of tracking performance. It considers both spatial and temporal aspects of tracking accuracy, making it a more nuanced metric compared to traditional metrics. Higher values of HOTA indicate better tracking performance.

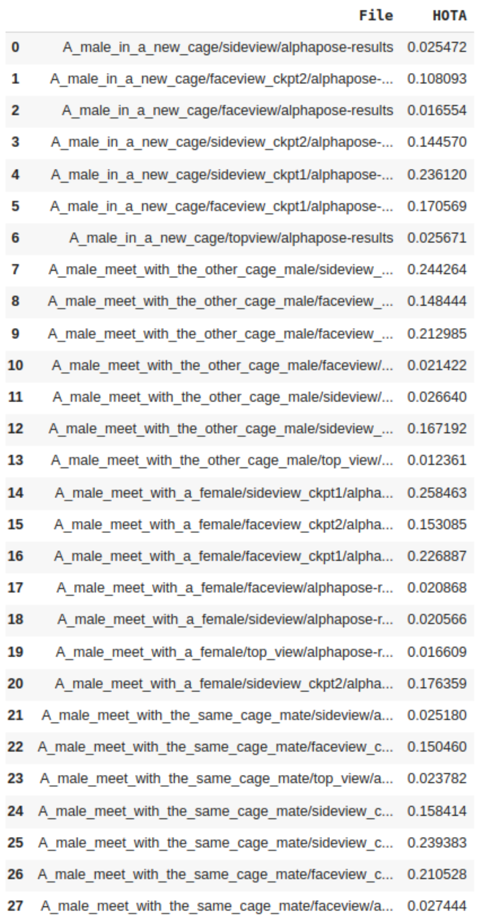
Once I knew how the metric worked, I started working on the code. In order not to repeat the same code, we decided to create some common functions that work for all of them, and other concrete functions for each metric.

3.4.3 Challenges Encountered and Solutions Implemented

When I finished the code (See the code in the next section), I ran it and obtained some results. However, after comparing it to other metrics, they appeared to be too low.

In fact, when I presented those results, I was encouraged to check the code, because the developers were not convinced by the results.

To prove that the code was correct, I compared two identical data sets. In theory, the



	File	HOTA
0	A_male_in_a_new_cage/sideview/alphapose-results	0.025472
1	A_male_in_a_new_cage/faceview_ckpt2/alphapose-...	0.108093
2	A_male_in_a_new_cage/faceview/alphapose-results	0.016554
3	A_male_in_a_new_cage/sideview_ckpt2/alphapose-...	0.144570
4	A_male_in_a_new_cage/sideview_ckpt1/alphapose-...	0.236120
5	A_male_in_a_new_cage/faceview_ckpt1/alphapose-...	0.170569
6	A_male_in_a_new_cage/topview/alphapose-results	0.025671
7	A_male_meet_with_the_other_cage_male/sideview_...	0.244264
8	A_male_meet_with_the_other_cage_male/faceview_...	0.148444
9	A_male_meet_with_the_other_cage_male/faceview_...	0.212985
10	A_male_meet_with_the_other_cage_male/faceview/...	0.021422
11	A_male_meet_with_the_other_cage_male/sideview/...	0.026640
12	A_male_meet_with_the_other_cage_male/sideview_...	0.167192
13	A_male_meet_with_the_other_cage_male/top_view/...	0.012361
14	A_male_meet_with_a_female/sideview_ckpt1/alpha...	0.258463
15	A_male_meet_with_a_female/faceview_ckpt2/alpha...	0.153085
16	A_male_meet_with_a_female/faceview_ckpt1/alpha...	0.226887
17	A_male_meet_with_a_female/faceview/alphapose-r...	0.020868
18	A_male_meet_with_a_female/sideview/alphapose-r...	0.020566
19	A_male_meet_with_a_female/top_view/alphapose-r...	0.016609
20	A_male_meet_with_a_female/sideview_ckpt2/alpha...	0.176359
21	A_male_meet_with_the_same_cage_mate/sideview/a...	0.025180
22	A_male_meet_with_the_same_cage_mate/faceview_c...	0.150460
23	A_male_meet_with_the_same_cage_mate/top_view/a...	0.023782
24	A_male_meet_with_the_same_cage_mate/sideview_c...	0.158414
25	A_male_meet_with_the_same_cage_mate/sideview_c...	0.239383
26	A_male_meet_with_the_same_cage_mate/faceview_c...	0.210528
27	A_male_meet_with_the_same_cage_mate/faceview/a...	0.027444

Figure 4: First HOTA results

evaluation results with two identical data sets should be 100%, however, the results did not reflect that.

To debug this problem, I visualized the outputs of each function, so I could figure out which part of the code was causing the problem.

I identified that the function that was providing the wrong result was the *compute_hota* function, so I debugged it in more detail.

First, I identify that the *calculate_iou* function was returning negative results when there were incomplete annotations. This was clearly an error, since the minimum value that the intersection over the joint can take is 0. Also, when there were missing points in the image, the value was automatically set to -1. This caused a large distortion in the metric value. To correct these two errors, I added two if statements, which check if the input data is correct.

In the following two code cells we can see how the original *compute_hota* code was before and how it is after the modifications.

Before:

```
for i, pred_box in enumerate(pred_boxes):
    for j, gt_box in enumerate(gt_boxes):
        pred_iou = calculate_iou(pred_box,gt_box)
```

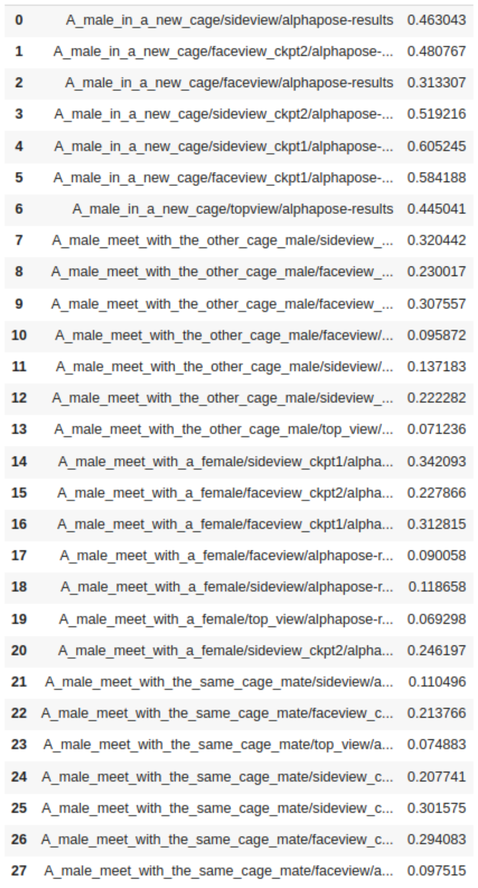
After:

```
for i, pred_box in enumerate(pred_boxes):
    for j, gt_box in enumerate(gt_boxes):
        pred = 0
        if gt_box["x"] != -1:
            pred_iou = calculate_iou(pred_box,gt_box)
            if pred_iou > pred:
              pred = pred_iou
          # Boxes that are x=-1 y=-1 because there is only one mouse
        else:
            pred = 1
        iou_matrix[i, j] = pred
```

3.4.4 Results and Future work

After all the modifications, I finally obtained a code that correctly computes and provides the results of the HOTA metric in every file. In the section *Code Snippets* you can see the whole code of the python notebook, including the code cell with the iterations of some files.

In the following image you can see the final results of the metric, which shows a great improvement when compared to the previous results, which can be seen in the figure 5.



0	A_male_in_a_new_cage/sideview/alphapose-results	0.463043
1	A_male_in_a_new_cage/faceview_ckpt2/alphapose-...	0.480767
2	A_male_in_a_new_cage/faceview/alphapose-results	0.313307
3	A_male_in_a_new_cage/sideview_ckpt2/alphapose-...	0.519216
4	A_male_in_a_new_cage/sideview_ckpt1/alphapose-...	0.605245
5	A_male_in_a_new_cage/faceview_ckpt1/alphapose-...	0.584188
6	A_male_in_a_new_cage/topview/alphapose-results	0.445041
7	A_male_meet_with_the_other_cage_male/sideview_...	0.320442
8	A_male_meet_with_the_other_cage_male/faceview_...	0.230017
9	A_male_meet_with_the_other_cage_male/faceview_...	0.307557
10	A_male_meet_with_the_other_cage_male/faceview/...	0.095872
11	A_male_meet_with_the_other_cage_male/sideview/...	0.137183
12	A_male_meet_with_the_other_cage_male/sideview_...	0.222282
13	A_male_meet_with_the_other_cage_male/top_view/...	0.071236
14	A_male_meet_with_a_female/sideview_ckpt1/alpha...	0.342093
15	A_male_meet_with_a_female/faceview_ckpt2/alpha...	0.227866
16	A_male_meet_with_a_female/faceview_ckpt1/alpha...	0.312815
17	A_male_meet_with_a_female/faceview/alphapose-r...	0.090058
18	A_male_meet_with_a_female/sideview/alphapose-r...	0.118658
19	A_male_meet_with_a_female/top_view/alphapose-r...	0.069298
20	A_male_meet_with_a_female/sideview_ckpt2/alpha...	0.246197
21	A_male_meet_with_the_same_cage_mate/sideview/a...	0.110496
22	A_male_meet_with_the_same_cage_mate/faceview_c...	0.213766
23	A_male_meet_with_the_same_cage_mate/top_view/a...	0.074883
24	A_male_meet_with_the_same_cage_mate/sideview_c...	0.207741
25	A_male_meet_with_the_same_cage_mate/sideview_c...	0.301575
26	A_male_meet_with_the_same_cage_mate/faceview_c...	0.294083
27	A_male_meet_with_the_same_cage_mate/faceview/a...	0.097515

Figure 5: Final HOTA results

In this table we can see how the results show a good performance when we are tracking a single mouse (around 0.5). On the other hand, this number decreases a lot when we add another mouse to the cage (around 0.2). This result is expected, due to the fact that adding a second mouse generates more points to recognize, so it becomes more difficult to correctly predict these points.

The development of this metric opens up a range of possibilities for the continuation of the project. First of all, it will be necessary to establish the appropriate metric for the project. This can be a combination of the four proposed metrics or just one. Once this indicator has been established, it should be used as a reference when establishing improvements and changes in the model. Therefore, I believe that once all the metrics have been correctly implemented, the next step should be to generate a reference metric on which to measure the progress of the project.

3.5 Task 5: Position and Audio Clustering

3.5.1 Description of the task

In this task, our objective was to gain a deeper understanding of mouse behavior patterns through the integration of various data sources, including pose estimation results, audio files, and detection outcomes from the "Mouse_behavior_video_recording" dataset. Our task was divided into several distinct phases, each designed to provide valuable insights into different aspects of mouse behavior analysis.

1. Pose Evaluation (Team size: 1 person) First, we had to check how accurate the computer's guesses about mouse poses are. We had predictions for each video, and we would use a measure called MPJPE to see how close these predictions are to the real positions of the mice.

2. Visual Clustering (Team size: 1 person) Next, we had to group similar mouse behaviors together. We would use math to figure out patterns in how they move and pose. This would help us understand if mice have common activities they do.

3. Audio Processing & Clustering (Team size: 2 people) After, we had to also listen to the sounds the mice make. We would learn how to work with sound files and find ways to group similar sounds together. This can give us more clues about what the mice are up to.

4. Audio + Visual Clustering (Data Unavailable) In the last part, we wanted to put everything together – poses, movements, and sounds – to really understand mouse behavior. Unfortunately, we did not have the right data for this step yet, but it was an exciting step if we get the data later.

In the end, this project let us look at mouse behavior from different angles. We had to discover new things about how they behave by analyzing their poses, sounds, and activities.

3.5.2 Approach and Methodology Used

Since there were different tasks and each task required 1 or 2 people, we decided to divide the work from the beginning, so that each of us could focus on a specific task. I had used clustering on some university assignments, so I decided on visual clustering.

In order to perform the visual clustering, firstly I had to decide which clustering algorithms I was going to use. After selecting the algorithm, it was crucial to employ different methods to measure the performance of the clustering.

I decided to go with the K-Means algorithm for its simplicity and efficiency. Now I am going to briefly explain how the K-Means algorithm works:

1. Initialization: Choose the number of clusters, K, and randomly initialize K cluster centroids within the feature space of your data.
2. Assignment: For each data point, calculate the distance to each centroid and assign the point to the nearest centroid's cluster. The distance can be computed using metrics like Euclidean distance or Manhattan distance.
3. Update Centroids: After all data points are assigned to clusters, calculate the mean of data points within each cluster. Update the centroid of each cluster to this mean value.
4. Repeat: Steps 2 and 3 are repeated iteratively until the centroids no longer significantly change or a predefined number of iterations is reached.
5. Termination: The algorithm terminates when the centroids stabilize or the maximum number of iterations is reached.

In order to evaluate the performance of the algorithm, I used two methods: the elbow method and the silhouette analysis.

The elbow method is a technique used to determine the optimal number of clusters, K, for a dataset in a clustering algorithm like K-Means. The main idea behind the elbow method is to identify the point on a plot where adding more clusters does not significantly reduce the within-cluster variance or distortion.

Here's how the elbow method works:

1. Run Clustering: Apply the clustering algorithm (e.g., K-Means) to your dataset for a range of K values.
2. Calculate Distortion: For each value of K, calculate the sum of squared distances between data points and their assigned cluster centroids. This is often referred to as the "distortion" or "inertia."
3. Plot the Results: Create a line plot where the x-axis represents the number of clusters (K), and the y-axis represents the corresponding distortion values.

4. Identify the Elbow: Look for a point on the plot where the decrease in distortion starts to slow down, forming an "elbow" shape. This point indicates the number of clusters where adding more clusters doesn't provide significant reduction in distortion.

Choosing the K value at the "elbow point" is a heuristic approach to selecting the optimal number of clusters. However, it's important to note that the elbow method is not foolproof. In some cases, the elbow might not be well-defined, or the optimal K might not correspond to an obvious elbow on the plot. Additionally, the shape of the dataset and the clustering algorithm used can impact the effectiveness of this method.

On the other hand, the silhouette analysis is a technique used to evaluate the quality of clusters formed by a clustering algorithm, such as K-Means. It provides a numerical measure of how well each data point is clustered and can help identify the optimal number of clusters. Here's how silhouette analysis works:

1. Calculate a: For each data point, calculate its "a" value, which represents the average distance between the point and all other points in the same cluster. This measures how well the point is clustered with its own cluster mates.
2. Calculate b: For each data point, calculate its "b" value, which represents the smallest average distance to points in a different cluster. This measures how separated the point is from other clusters.
3. Calculate Silhouette Score: For each data point, calculate the silhouette score using the formula: silhouette score = (b - a) / max(a, b). The silhouette score ranges from -1 to 1, where higher values indicate better clustering (points are well matched to their own cluster and poorly matched to neighboring clusters).
4. Average Silhouette Score: Compute the average silhouette score for all data points in the dataset. This provides an overall measure of how well the entire dataset is clustered.
5. Repeat for Different K: Perform the above steps for different values of K (number of clusters) to see how the silhouette scores change as the number of clusters increases.
6. Choose Optimal K: The optimal number of clusters corresponds to the value of K that yields the highest average silhouette score. This indicates the configuration where data points are well-separated into distinct clusters while being closely matched with their own cluster members.

The silhouette analysis method offers a quantitative way to assess the quality of clustering results and to choose the appropriate number of clusters. It's particularly useful when the elbow method doesn't provide a clear indication of the optimal K or when comparing different cluster configurations. Higher silhouette scores suggest more distinct and well-defined clusters, while lower scores might indicate that data points are assigned to the wrong clusters.

Once I had an idea about the methodologies I was going to use in this task, I developed the code and test it using the two methods.

3.5.3 Challenges Encountered and Solutions Implemented

This task took me several weeks, so I will explain the problems and results I had each week and what I did to solve them. In the next chapter I will show the final results and possible ways to improve them.

In the first week I chose to use a data set with only one mouse and perform a simple clustering, so that I could get different perspectives on the model and then I could adjust the parameters and add another mouse, with more complex features.

For the first model, I used the dataset *A_male_in_a_new_cage.json* and I used every single row for the model. For every frame, I used the predicted box of the mouse position as the input. The first problem I recognized was that the json files didn't provide the keypoints of the frames in a readable format. Every row showed the following information: the image id, the file name, the category id and the boxes, which actually was the useful information. In order to obtain just the coordinates of the box, I developed a function called *get_dict* which obtained a dictionary that contains the information of every point for every row. This dictionary was appended to a list and after getting all the data, I used that list to generate a python data frame.

Once I obtained usable data, I performed the k means clustering with different number of clusters and plotted them.

In figure 6 we can see the results of this first clustering.

According to the elbow method, explained in the previous chapter, we can see how the results showed that the optimal number of clusters for this data set was two. This conclusion was not very convincing neither for me nor for the other researchers of the project. The main goal of this clustering was to detect and group different types of mouse behaviors, and, given that the animal was constantly moving and reacting to different stimuli, grouping the behavioral patterns into two clusters was not the goal of the project.

This result could have been produced by different causes. However, I wanted to evaluate the same clustering with another method to get more information about the results.

Therefore, the following week I focused on silhouette analysis. As I mentioned earlier, silhouette analysis provides a numerical measure of how well each data point is clustered. Thus, I performed the analysis on the model above.

In this plot we can see how the optimum number of plots is still two. I needed to modify the input data in order to obtain a larger number of clusters. I tried two different approaches:

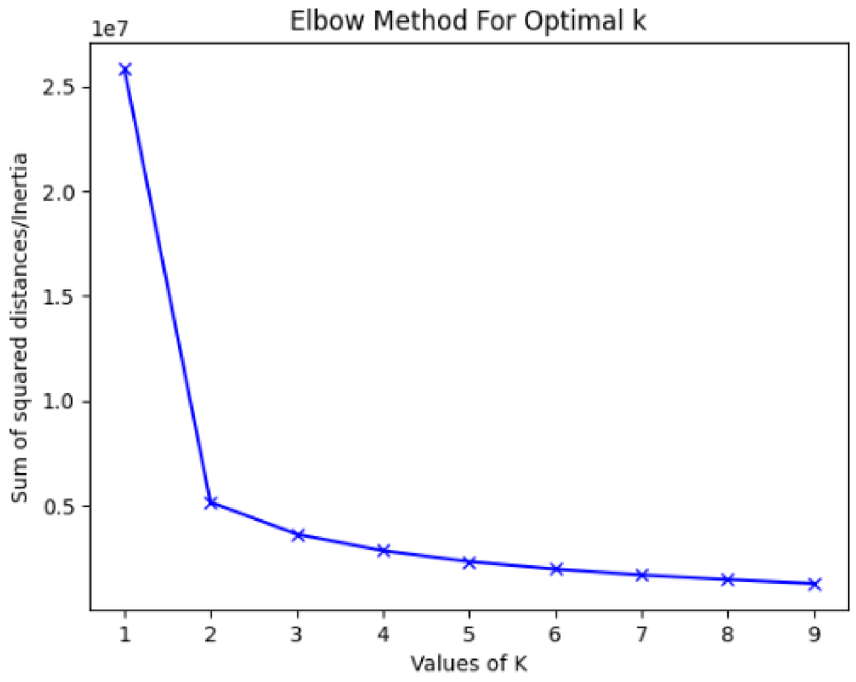


Figure 6: First clustering results

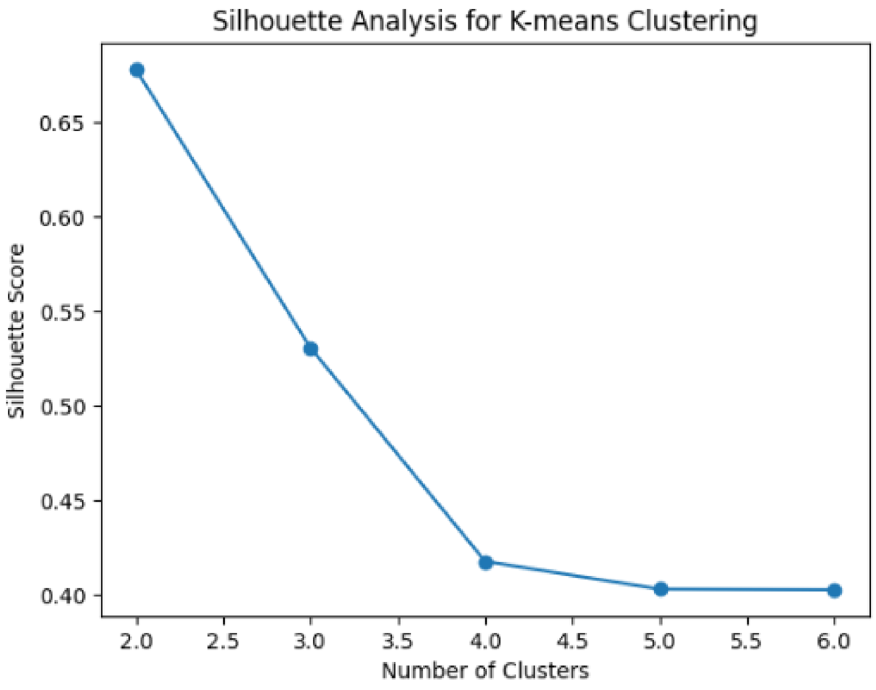


Figure 7: Silhouette analysis on first clustering

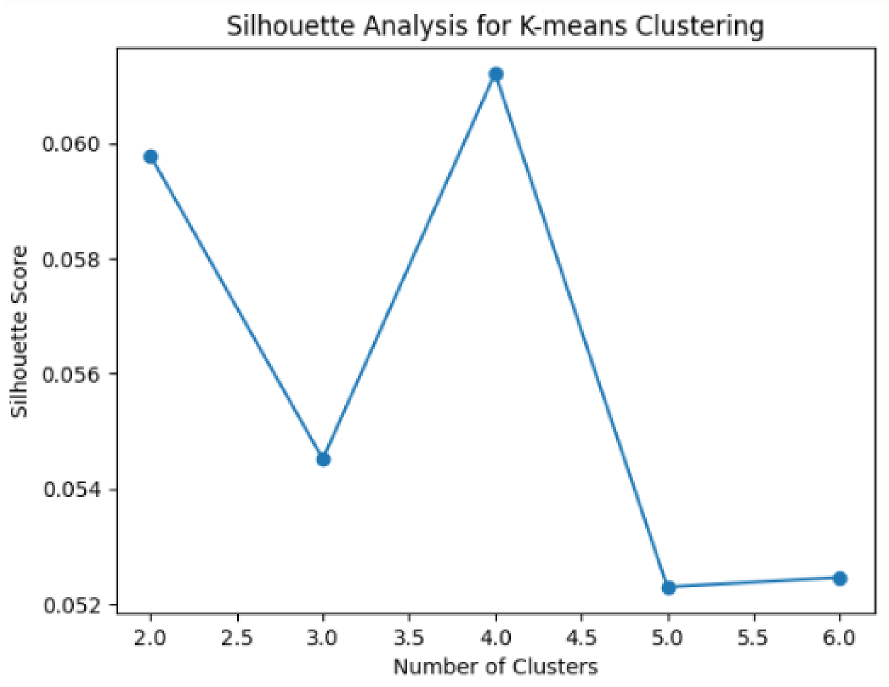


Figure 8: Silhouette analysis on sequenced data

- Adding more points to the data: In the first model I just used the four coordinates of the box as the input features. However, adding more features like the position of the ears, mouth and other keypoints would make the data more complex and more clusters would be identified. That's why I decided to use every point of the mouse, going from four to sixteen features.
- Concatenate consecutive frames: Previously I was using every single frame as the input, but the behavioral patterns are usually identified as concrete movements, which cannot be identified with just one image. That's why I decided to manipulate the data to use sequences as the input instead of single frames. I grouped the data in sequences of 20 frames, so each row contained 320 features (16 feature per frame).

After adding these modifications I repeated the same k-means clustering with different number of clusters, and this was the result of the silhouette analysis:

This result revealed that the optimal number of clusters in this model was four, which was a better result than the two number of clusters that showed the first model. On the other hand, there still were different challenges and possibilities to improve the model.

For example, the use of sequences as input greatly reduced the number of rows in the data set. This caused overfitting in the model. To solve this problem, I realized that I could repeat frames in different sequences. So instead of grouping the data into groups of 20 different frames, I created sequences of 20 consecutive frames every 5 frames. This

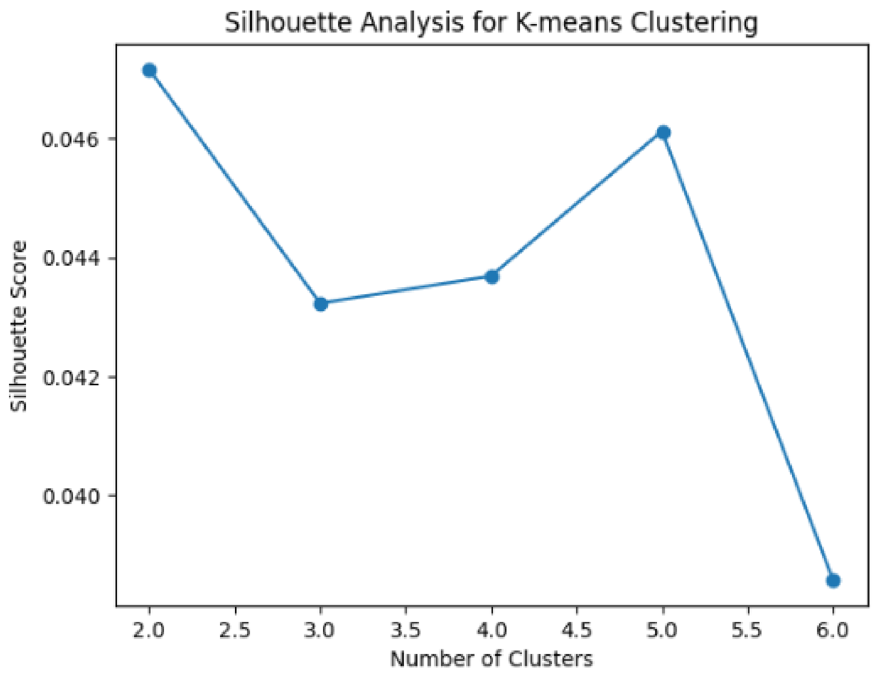


Figure 9: Final silhouette analysis

way I got 147 rows of data, instead of the 36 rows I had previously.

This was the last modification I added to the clustering task, so I will show the final results and the possible future work in the next chapter.

3.5.4 Results and Future work

The final code can be seen in the last section of the report, below *vision_clustering.ipynb*. In the code we can plot the result of every cluster we were making and how the results changed. The last result I obtained can be seen in the following image:

Here we can see that after all the modifications, the optimal numbers of clusters were 2 and 5, which shows a positive result because it means that the patterns of the mouse can be grouped in either 2 big or 5 smaller groups.

From this point on, there are several ways to continue this work. First, for this initial model we have used a single file, which includes the data from a single camera. This means that we can obtain the mouse position from only one side. However, for the experiment we used three different cameras for three sides (top, face and side). A good future approach could be to join these data and use them together to generate a model that includes more features. The difficulty of this step lies in the fact that each camera starts recording at different times, so it is not easy to align the frames. However, with more information about the recordings, it can be done.

Secondly, for this model we have just used the data collected from the recording of one single mouse. However, it is also very interesting to capture the patterns of a mouse when it interacts with other mice. That's why another possible future approach would be to use different datasets that include two or more mice.

Finally, I strongly believe that it is important to manipulate the data to create a good clustering. For example, for this model we have used the sliding window technique, which includes data from 20 consecutive frames. However, these data are exposed in absolute terms, they show the coordinate of each point at each time, when what can explain the pattern of mouse behavior is the change of this position from one frame to another. So I think a good approach might be to manipulate the data and show each point in each frame of the sequence as the difference of the coordinate with respect to the first frame.

These are some ideas about how the project can be continued, however, there can be several other approaches depending of the objectives of the model.

4 Summary of Contributions

4.1 Overview of my Overall Impact on the Project

Throughout the project, I actively participated in various tasks aimed at understanding and analyzing mouse behavior using a combination of visual and audio data. My contributions were focused on tasks related to data preparation, model training, evaluation, and behavioral pattern clustering. I played a significant role in the technical implementation of these tasks, helping to derive insights and valuable results that contributed to the project's overall goals.

4.2 Key Accomplishments

1. Data Preparation and Transformation: In the initial phase of the project, I undertook the task of data preparation. I investigated the structure of the raw data from the AlphaTracker application and developed scripts to clean and merge the data from different .csv files into a unified .json format. This enabled subsequent analysis and model training. My experience with Python scripting facilitated this process, ensuring the data was appropriately formatted and ready for further tasks.

2. Model Training and Understanding: While facing challenges with the hardware requirements for the AlphaTracker model training, I explored alternative options such as Google Colab. Although I encountered issues and was unable to complete the training, this experience deepened my understanding of the model training pipeline, and I documented the challenges and solutions for future reference.

3. Evaluation Metrics Implementation: I took on the responsibility of implementing evaluation metrics for the trained models. I successfully developed a script to calculate the HOTA (Higher Order Tracking Accuracy) metric, a complex metric that considers multiple aspects of tracking performance. This involved understanding the metric's mathematical foundations and translating them into practical code. By providing accurate and comprehensive evaluation results, I helped the team assess the quality of the models' predictions effectively.

4. Behavioral Pattern Clustering: I contributed to the project's goal of analyzing mouse behavior patterns by employing clustering techniques on pose estimation data. I explored the K-Means algorithm and implemented it using Python. I iteratively adjusted the features and input data to improve clustering accuracy. Through silhouette analysis and the elbow method, I identified optimal numbers of clusters, aiding in understanding distinct behavioral patterns among the mice.

4.3 Challenges Overcome

1. Data Loading and Cleaning: One of the initial challenges was handling the diverse data formats and merging them into a consistent structure. I overcame this challenge by writing Python scripts to clean and transform the data into a usable format for downstream tasks. My systematic approach and familiarity with data manipulation in Python facilitated this process.

2. Model Training and Hardware Limitations: The challenge of training the Alpha-Tracker model due to hardware limitations was addressed by exploring Google Colab as an alternative. While I couldn't complete the training, I documented the steps and challenges faced, providing insights for potential future attempts or improvements in hardware resources.

3. Metric Implementation and Interpretation: Implementing the HOTA metric required a deep understanding of its mathematical underpinnings and how it relates to tracking accuracy. I tackled this challenge by thoroughly studying the metric's definition and consulting external resources. By carefully translating mathematical concepts into functional code, I successfully implemented the metric and produced meaningful evaluation results.

4. Behavioral Clustering and Data Transformation: Clustering mouse behavior patterns using pose estimation data presented the challenge of selecting appropriate features and dealing with temporal data. I addressed this challenge by experimenting with different feature selections and transforming the data into sequences for improved clustering. Additionally, I enhanced the clustering results by utilizing silhouette analysis to determine optimal cluster numbers.

5 Lessons Learned

5.1 Skills Developed or Enhanced

1. Data Preparation and Transformation: I enhanced my skills in data manipulation and transformation. I learned how to handle diverse data formats, extract relevant information, and unify data into a consistent structure for analysis.

2. Model Training and Evaluation: While I couldn't complete the model training, I gained valuable insights into the intricacies of training complex models and understanding their evaluation metrics. This experience deepened my understanding of the challenges and considerations in machine learning projects.

3. Metric Implementation: Implementing the HOTA metric improved my mathematical coding skills and taught me the importance of understanding complex metrics to accurately evaluate model performance. I learned how to translate abstract mathematical concepts into practical code.

4. Clustering Techniques: Through the clustering task, I gained a solid understanding of clustering algorithms, feature selection, and the importance of data representation. I also learned how to interpret clustering results using silhouette analysis and the elbow method.

5.2 Knowledge Gained

1. Project Collaboration: Working on a multi-faceted project with different tasks and team members provided insights into effective collaboration and task delegation. I learned how each task contributes to the overall project goals and the importance of seamless information sharing.

2. Complex Metric Evaluation: Implementing the HOTA metric deepened my understanding of the intricacies of tracking evaluation in computer vision projects. I learned how metrics can account for various factors and contribute to a holistic assessment of model performance.

3. Behavioral Analysis: The clustering task enhanced my understanding of extracting meaningful patterns from complex data. I learned how clustering algorithms can uncover insights into behavior and how data preprocessing impacts the results.

5.3 Reflections on the Project Experience

Participating in this project was a valuable experience that exposed me to the challenges and rewards of multidisciplinary research and data-driven analysis. Collaborating with team members with different expertise allowed me to contribute my technical skills while also learning from others. Navigating challenges like data preprocessing, metric implementation, and behavioral clustering broadened my problem-solving capabilities.

Although the project had its share of complexities and uncertainties, it provided a realistic glimpse into the iterative nature of research and the importance of adaptability and continuous learning. I am proud of the contributions I made and the insights I gained throughout the project, and I look forward to applying these experiences to future endeavors.

6 Code Snippets

clean_data.py:

```python
#!/usr/bin/env python3

"""This script must be executed from the raw_data directory. Type:
for FILE in *; do python3 ../clean_data.py \$FILE; done"""

import pandas as pd
import os
import sys

# Get the file name from command line argument
file_name = sys.argv[1]

# Create the path to the file
data_path = os.path.join(os.getcwd(), file_name)

# Load the data into a Pandas DataFrame
df = pd.read_csv(data_path)

# Rename the columns
df = df.rename(columns={
    "Unnamed: 1": "csv_index",
    "Unnamed: 2": "file_name"
})

# Set the csv index and file name as the DataFrame index
df = df.set_index(["csv_index", "file_name"])

# Drop the first column as it doesn't contain useful information
df = df.iloc[:, 1:]

# Get the first three rows to modify the headers
headers = df.iloc[:3, :].copy()

# Concatenate the first three rows to form the new headers
new_headers = []
for i in range(headers.shape[1]):
    new_header = "_".join(headers.iloc[:, i].tolist())
    new_headers.append(new_header)

# Drop the first three rows from the DataFrame
df = df.iloc[3:, :]

# Rename the columns with the new headers
df.columns = new_headers

# Save the cleaned data to a new file
cleaned_data_path = os.path.join(os.pardir, "cleaned_data", f"cleaned_
                                   {file_name}")
```

```
df.to_csv(cleaned_data_path)
```

csv_generator.yipynb:

```
# Import libraries
import pandas as pd
import numpy as np
import os

# List with all csv files
csv_path = os.path.join(os.getcwd(), "cleaned_data")
csv_files = os.listdir(csv_path)
csv_files_path = list(("cleaned_data/" + csv for csv in csv_files))

# Load all the datasets in one data frame
df = pd.concat(map(pd.read_csv, csv_files_path))

# Sort the data frame by the csv index and the png file name
df = df.set_index(["csv_index", "file_name"])
df = df.sort_index()

df["mouse1_topleft_x"] = pd.to_numeric(df["mouse1_topleft_x"], errors=
                                   'coerce')

# Create new features for mouse 1
df["height_1"] = df["mouse1_topleft_y"] - df["mouse1_rightdown_y"]
df["width_1"] = df["mouse1_rightdown_x"] -  df["mouse1_topleft_x"]
df["x_1"] = df["mouse1_topleft_x"]
df["y_1"] = df["mouse1_rightdown_y"]

# Create new features for mouse 2
df["height_2"] = df["mouse2_topleft_y"] - df["mouse2_rightdown_y"]
df["width_2"] = df["mouse2_rightdown_x"] -  df["mouse2_topleft_x"]
df["x_2"] = df["mouse2_topleft_x"]
df["y_2"] = df["mouse2_rightdown_y"]

# Set the final columns
columns = df.columns[4:-8]
columns = (df.columns[-8:]).append(columns)

# Generate the final data frame with the desired columns
df = df[columns]

# Write the data frame into a csv file
df.to_csv("processed_data.csv")
```

evaluation_HOTA.ipynb:

```python
# Import necessary libraries and modules
from google.colab import drive
drive.mount('/content/drive')

import os
import json
import re
import matplotlib.pyplot as plt
from sklearn.metrics import auc
import pandas as pd

# Function to extract ID from ground truth annotation
def extract_id(gt_annotation):
    match = re.search(r"_([0-9]+)\.png\$", gt_annotation["filename"])

    if match:
        gt_id = match.group(1)
    return gt_id

# Function to find the corresponding ground truth annotation for a
                                   predicted image
def find_gt_annotation(gt_annotations, pred_image):
    gt_image = None
    for annotation in gt_annotations:
        if str(extract_id(annotation)) == str(pred_image["image_id"]):
            gt_image = annotation
    return gt_image

# Function to get bounding boxes for predicted images
def get_pred_image_boxes(pred_image):
    boxes = []
    i = 1
    for box in pred_image["boxes"]:
        height = abs(box[1] - box[3])
        width = abs(box[0] - box[2])
        boxes.append({'mouse' + str(i): 'boundingBox',
                      'height': height,
                      'width': width,
                      'x': width / 2 + min(box[0], box[2]),
                      'y': height / 2 + min(box[1], box[3])})
        i += 1
    return boxes

# Function to calculate Intersection over Union (IoU) between two
                                   bounding boxes
def calculate_iou(boxA, boxB):
    # Calculate intersection coordinates
    xA = max(boxA['x'], boxB['x'])
    yA = max(boxA['y'], boxB['y'])
    xB = min(boxA['x'] + boxA['width'], boxB['x'] + boxB['width'])
    yB = min(boxA['y'] + boxA['height'], boxB['y'] + boxB['height'])

    # Calculate intersection area
    interArea = max(0, xB - xA) * max(0, yB - yA)

    # Calculate areas of both bounding boxes
    boxAArea = boxA['width'] * boxA['height']
    boxBArea = boxB['width'] * boxB['height']

    # Calculate IoU
    iou = interArea / float(boxAArea + boxBArea - interArea)

    return iou

# Function to match predictions with ground truth using IoU threshold
def match_predictions_gt(alpha=0.5, results={}):
    TPTr = {k: v for k, v in results.items() if v["iou"] >= alpha}
    PrTrajs = dict(sorted(results.items(), key=lambda x: x[1]["iou"],
                                           reverse=True))
    return TPTr, PrTrajs

# Function to calculate precision and recall
def calc_prec_recall(PrTrajs, results, alpha=0.5):
    n = 1
    TPTrn = 0
    for item in PrTrajs:
        if PrTrajs[item]["iou"] > alpha:
            TPTrn += 1
        PrTrajs[item]["Precision"] = TPTrn / n
        PrTrajs[item]["Recall"] = TPTrn / len(results)
        n += 1
    return PrTrajs

# Import necessary libraries
import numpy as np
from scipy.optimize import linear_sum_assignment

# Function to compute Higher Order Tracking Accuracy (HOTA)
def compute_hota(pred_boxes, gt_boxes, alpha=0.5):
    num_pred = len(pred_boxes)
    num_gt = len(gt_boxes)

    # Initialize the IoU matrix
    iou_matrix = np.zeros((num_pred, num_gt))

    # Fill the IoU matrix with calculated IoU values
    for i, pred_box in enumerate(pred_boxes):
        for j, gt_box in enumerate(gt_boxes):
            pred_iou = calculate_iou(pred_box, gt_box)
            iou_matrix[i, j] = pred_iou

    # Use the Hungarian algorithm to find the optimal assignment
    row_ind, col_ind = linear_sum_assignment(-iou_matrix)

    # Count true positives based on the optimal assignment and IoU
                                   threshold
    tp = sum(iou_matrix[row_ind, col_ind] >= alpha)

    # Calculate false negatives and false positives
    fn = num_gt - tp
    fp = num_pred - tp

    # Calculate HOTA components: association and detection
    association = tp / (tp + 0.5 * (fn + fp)) if tp + 0.5 * (fn + fp)
                                   > 0 else 0
    detection = tp / (tp + fn + fp) if tp + fn + fp > 0 else 0

    # Calculate the HOTA score
    hota = 2 * association * detection / (association + detection) if
                                   association + detection > 0
                                   else 0

    return hota

# Function to calculate HOTA score for a given set of predictions and
                                   ground truth
def calculate_hota(pred_path, gt_path, alpha=0.5):
    with open(pred_path, 'r') as f:
        pred_annotations = json.load(f)

    with open(gt_path, 'r') as f:
        gt_annotations = json.load(f)

    hota_sum = 0
    num_frames = 0

    # Iterate through each predicted image and compute HOTA
    for image in pred_annotations:
        gt_image = find_gt_annotation(gt_annotations, image)
        if gt_image is not None:
            num_frames += 1
            pred_boxes = get_pred_image_boxes(image)
            gt_boxes = [item for item in gt_image["annotations"] if
                                   len(item) > 3]

            hota = compute_hota(pred_boxes, gt_boxes, alpha)
            hota_sum += hota

    hota_avg = hota_sum / num_frames
    print(f"The average HOTA score is: {hota_avg}")
    return hota_avg

# Function to calculate HOTA score for a test scenario
def calculate_hota_test(pred_path, gt_path, alpha=0.5):
    with open(pred_path, 'r') as f:
        pred_annotations = json.load(f)

    with open(gt_path, 'r') as f:
        gt_annotations = json.load(f)

    hota_sum = 0
    num_frames = 0

    # Iterate through each predicted image and compute HOTA test score
    for image in pred_annotations:
        gt_image = find_gt_annotation(gt_annotations, image)
        if gt_image is not None:
            num_frames += 1
            pred_boxes = get_pred_image_boxes(image)
            gt_boxes = [item for item in gt_image["annotations"] if
                                                     len(item) > 3]

            hota = compute_hota(gt_boxes, gt_boxes, alpha)
            hota_sum += hota

    hota_avg = hota_sum / num_frames
    print(f"The average HOTA test score is: {hota_avg}")
    return hota_avg

# Function to extract file name from a path
def get_file(string):
    terms = string.split("/")[-3:]
    joined_terms = "/".join(terms)
    return joined_terms[:-5]

# Set the base path for data
path = '/content/drive/MyDrive/Master_Mouse_Project/Data/track_result_
                                   /'

# Get subdirectories within the base path
directories = [dir for dir in os.listdir(path) if os.path.isdir(os.
                                   path.join(path, dir))]

# Initialize dictionaries to store evaluation results
evaluation = {}

# Iterate through subdirectories and calculate HOTA scores
for file in os.listdir(path + "/" + directories[2]):
    if "top" in file:
        gt_path = "/content/drive/MyDrive/Master_Mouse_Project/
                                          Submission/Task4/
                                          one_mouse_top.json"
        pred_path = path + directories[2] + "/" + file + "/alphapose-
                                          results.json"
        evaluation[get_file(pred_path)] = calculate_hota(pred_path,
                                          gt_path)
    elif "side" in file:
        gt_path = "/content/drive/MyDrive/Master_Mouse_Project/
                                            Submission/Task4/
                                            one_mouse_side.json"
        pred_path = path + directories[2] + "/" + file + "/alphapose-
                                            results.json"
        evaluation[get_file(pred_path)] = calculate_hota(pred_path,
                                            gt_path)
    elif "face" in file:
        gt_path = "/content/drive/MyDrive/Master_Mouse_Project/
                                            Submission/Task4/
                                            one_mouse_face.json"
        pred_path = path + directories[2] + "/" + file + "/alphapose-
                                            results.json"
        evaluation[get_file(pred_path)] = calculate_hota(pred_path,
                                            gt_path)
```

vision_clustering.ipynb

```
from google.colab import drive
drive.mount('/content/drive')


import os
import json
import re
import matplotlib.pyplot as plt
from sklearn.metrics import auc
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score

def get_dict(results):
    keypoints = results[0]["keypoints"]
    box = results[0]["box"]
    height= abs(box[1]-box[3])
    width = abs(box[0]-box[2])
    if (len(box) == 4 and len(keypoints) == 12):
        points = ({'height': height,
                    'width': width,
                    'x': width/2 + min(box[0],box[2]),
                    'y': height/2+min(box[1],box[3]),
                    'p1': keypoints[0],
                    'p2': keypoints[1],
                    'p3': keypoints[2],
                    'p4': keypoints[3],
                    'p5': keypoints[4],
                    'p6': keypoints[5],
                    'p7': keypoints[6],
                    'p8': keypoints[7],
                    'p9': keypoints[8],
                    'p10': keypoints[9],
                    'p11': keypoints[10],
                    'p12': keypoints[11],
        })
    return points

path = '/content/drive/MyDrive/Master_Mouse_Project/Data/track_result_
                                   /'
directories = [dir for dir in os.listdir(path) if os.path.isdir(os.
                                   path.join(path, dir))]
file = os.listdir(path + "/" + directories[2])[0]
pred_path = path + directories[2] + "/" + file + "/alphapose-results-
                                   forvis.json"

with open(pred_path, 'r') as f:
    pred_annotations = json.load(f)
```

```python
images = pred_annotations
all_frames = []
id = []

for frame_id, results in images.items():
    id.append(frame_id)
    all_frames.append(get_dict(results))

mouse1 = pd.DataFrame(all_frames, index=id)

Sum_of_squared_distances = []
K = range(1, 10)

for num_clusters in K:
    kmeans = KMeans(n_clusters=num_clusters)
    kmeans.fit(mouse1)
    Sum_of_squared_distances.append(kmeans.inertia_)

plt.plot(K, Sum_of_squared_distances, "bx-")
plt.xlabel("Values of K")
plt.ylabel("Sum of squared distances/Inertia")
plt.title("Elbow Method For Optimal k")
plt.show()

# Choose the number of clusters
num_clusters = 5

# Apply the clustering algorithm
kmeans = KMeans(n_clusters=num_clusters)
kmeans.fit(mouse1)

# Step 7: Evaluate and interpret the results
labels = kmeans.labels_
centroids = kmeans.cluster_centers_

def perform_silhouette_analysis(X, range_n_clusters):
    """
    Perform silhouette analysis for K-means clustering.

    Args:
    X (array-like): Input data to be clustered.
    range_n_clusters (list): A list of integers specifying the range
                                of number of clusters to
                                evaluate.

    Returns:
    best_n_clusters (int): The optimal number of clusters based on
                                silhouette score.
    silhouette_scores (list): Silhouette scores for each number of
                                clusters.

    """
    silhouette_scores = []
    best_avg_score = -1
    best_n_clusters = -1

    for n_clusters in range_n_clusters:
        # Initialize K-means with the current number of clusters
        kmeans = KMeans(n_clusters=n_clusters)
        cluster_labels = kmeans.fit_predict(X)

        # Calculate the silhouette score for each sample
        silhouette_avg = silhouette_score(X, cluster_labels)
        silhouette_scores.append(silhouette_avg)

        # Update the best number of clusters if the current score is
                                   higher
        if silhouette_avg > best_avg_score:
            best_avg_score = silhouette_avg
            best_n_clusters = n_clusters

    return best_n_clusters, silhouette_scores

def plot_silhouette_analysis(range_n_clusters, silhouette_scores):
    """
    Plot the results of silhouette analysis.

    Args:
    range_n_clusters (list): A list of integers specifying the range
                                    of number of clusters evaluated
                                    .
    silhouette_scores (list): Silhouette scores for each number of
                                    clusters.

    """
    plt.plot(range_n_clusters, silhouette_scores, marker='o')
    plt.xlabel('Number of Clusters')
    plt.ylabel('Silhouette Score')
    plt.title('Silhouette Analysis for K-means Clustering')
    plt.show()

# Generate random data for clustering
np.random.seed(42)
X = mouse1

# Specify the range of number of clusters to evaluate
range_n_clusters = [2, 3, 4, 5, 6]

# Perform silhouette analysis
best_n_clusters, silhouette_scores = perform_silhouette_analysis(X,
                                   range_n_clusters)


# Plot the results
```

```
plot_silhouette_analysis(range_n_clusters, silhouette_scores)

## Clustering of consecutive frames
data = mouse1.to_numpy()
consec_frames = data[:int(data.shape[0] / 20) * 20].reshape(-1, 20 *
                                   data.shape[1])
Sum_of_squared_distances = []
K = range(1,10)
X = consec_frames

for num_clusters in K :
    kmeans = KMeans(n_clusters=num_clusters)
    kmeans.fit(mouse1)
    Sum_of_squared_distances.append(kmeans.inertia_)

plt.plot(K,Sum_of_squared_distances,"bx-")
plt.xlabel("Values of K")
plt.ylabel("Sum of squared distances/Inertia")
plt.title("Elbow Method For Optimal k")
plt.show()

# Generate random data for clustering
np.random.seed(42)
X = consec_frames

# Specify the range of number of clusters to evaluate
range_n_clusters = [2, 3, 4, 5, 6]

# Perform silhouette analysis
best_n_clusters, silhouette_scores = perform_silhouette_analysis(X,
                                   range_n_clusters)

# Plot the results
plot_silhouette_analysis(range_n_clusters, silhouette_scores)

# Choose the number of clusters
num_clusters = 4
X = consec_frames

# Apply the clustering algorithm
kmeans = KMeans(n_clusters=num_clusters)
kmeans.fit(X)

# Evaluate and interpret the results
labels = kmeans.labels_
centroids = kmeans.cluster_centers_

## Sliding window frames clustering
sliding_window = []

for i in range(0, (data.shape[0] - 20), 5):
  sliding_window.append(list(data[i:(i+20)].reshape(1, -1)[0]))
```

```
sliding_array = np.array(sliding_window)

Sum_of_squared_distances = []

K = range(1,10)
X = sliding_array

for num_clusters in K :
    kmeans = KMeans(n_clusters=num_clusters)
    kmeans.fit(mouse1)
    Sum_of_squared_distances.append(kmeans.inertia_)

plt.plot(K,Sum_of_squared_distances,"bx-")
plt.xlabel("Values of K")
plt.ylabel("Sum of squared distances/Inertia")
plt.title("Elbow Method For Optimal k")
plt.show()

# Generate random data for clustering
np.random.seed(42)
X = sliding_array

# Specify the range of number of clusters to evaluate
range_n_clusters = [2, 3, 4, 5, 6]

# Perform silhouette analysis
best_n_clusters, silhouette_scores = perform_silhouette_analysis(X,
                                   range_n_clusters)

# Plot the results
plot_silhouette_analysis(range_n_clusters, silhouette_scores)
```

List of Figures

References

[1] Zexin Chen. “AlphaTracker: A Multi-Animal Tracking and Behavioral Analysis Tool”. In: (2020).

[2] ZexinChen. *AlphaTracker*. `https://github.com/ZexinChen/AlphaTracker`. 2022.