COMP3850 Group 23 Analysis and Design Document

Revision History

Revision Number	Date	Person(s)	Changes
1.0	29/04/2024	Group 23	Initial version
1.1.	16/05/2024	Michael Yee	Added ffmpeg command example to Video pre-processing section Added details for UNet architecture in Model Backbone section Added track_id to output file structure definition

Feature Engineering

Feature engineering is a process in machine learning development which involves the addition, deletion, combination, and mutation of the raw features in a dataset with the purpose of improving the overall training of the machine learning model, thus resulting in better performance and accuracy. These raw features can take the form of any type of data that the model can learn from, such as numbers, texts, images or audio. In the context of Project APRA, our team's primary objective is to transform the raw data from ant behavioural videos into a format that coincides with our models[1].

In comparison to more conventional machine learning tasks, feature engineering in computer vision obtains its information by extracting visual features from images and videos using convolutional neural networks (CNN) rather than numerical and structured data like in feed-forward tasks. Additionally, the features are learned by the model itself, instead of being defined manually by the data engineer. Since this method involves analysing raw video data, different algorithms are also needed to identify any relevant patterns, shapes and colours, as opposed to feed forward tasks which use techniques such as feature selection, transformation, or dimensionality reduction.

Supervised convolutional neural networks are the most commonly used method for computer vision tasks such as pose estimation. This approach involves learning features associated with particular labels in an image, including edges, shapes, and colours associated with a particular label. Since feature engineering involves identifying patterns to improve the overall quality of our predictive models, some trends that may be identified in the input data for weaver ants in a pulling-chain could include the degree at which their legs bend, the curvature at certain points of their body as well as the distance between certain body parts.

To learn features, CNNs use a hierarchical network structure to process data, utilising a number of pooling and convolutional layers to identify elements of the input image consistent with each label. Through each passing layer, the detail of extracted features is incrementally improved with early layers detecting low-level elements which could range anywhere from edges, textures and colours. Accumulation of these elements further allows the CNN to have a better understanding of more abstract and complex patterns that may later present themselves.

Label definitions

Each ant body part is defined by an X-coordinate with a value between 0 and the width of the frame in pixels, and a Y-coordinate with a value between 0 and the height of the frame in pixels. As each part location must be resolved to a single point, the definitions for each part are defined as follows.

[1] Gomede, E 2023, 'Feature Engineering for Computer Vision', *Medium*, viewed on 24 April, accessed at: https://medium.com/@evertongomede/feature-engineering-for-computer-vision-f01a76d8058c

Part name	Technical name	Definition
Head	Head	The front-most point on the head, between the mandibles (if they are visible)
Neck	Neck	The point between the head and the front-most point of the thorax
Upper Thorax	UpperThorax	The point between the first and second segments of the thorax
Lower Thorax	LowerThorax	The rear-most point on the second segment of the thorax where the hind legs are joined to the body
Petiole	Petiole	The rear-most end of the segment joining the thorax to the gaster/front-most end of the gaster
Abdomen	Abdomen	The rear-most point on the gaster
Left Front Leg	LForeLeg	The knee joint of the front-most left leg
Left Front Foot	LForeFoot	The end of front-most left leg
Right Front Leg	RForeLeg	The knee joint of the front-most right leg
Right Front Foot	RForeFoot	The end of front-most right leg
Left Middle Leg	LMidLeg	The knee joint of the middle left leg
Left Middle Foot	LMidFoot	The end of middle left leg
Right Middle Leg	RMidLeg	The knee joint of the middle right leg
Right Middle Foot	RMidFoot	The end of middle right leg
Left Rear Leg	LHindLeg	The knee joint of the rear-most left leg
Left Rear Foot	LHindFoot	The end of rear-most left leg
Right Rear Leg	RHindLeg	The knee joint of the rear-most right leg
Right Rear Foot	RHindFoot	The end of rear-most right leg

Solution Architecture

The proposed architecture for building and training a working computer vision model from the videos of weaver ants utilises a combination of the SLEAP pose estimation software and the Jupyter computing platform, and can be categorised under the following steps which are explained in detail below,

1. Video pre-processing
2. Data labelling
3. Dataset combining (if required)
4. Training
5. Comparing models
6. Inference, tracking and data export

Video pre-processing

Before the videos can be used for training models, they must be cropped and re-encoded to simplify the labelling process and reduce the possibility for errors during training. All of the ant activity of interest occurs at the tip of the leaf, which is located on the left side of the shot in all of the videos. Cropping the videos to only include this region of the frame reduces the number of ants that need to be labelled per frame and decreases the dimensions of the video, which reduces the complexity of the trained models and makes them faster and less resource-intensive to train. The videos are specifically cropped such that the linear dimensions of the processed videos are half of the linear dimensions of the unprocessed videos (i.e. a 3840x2160 video resolution becomes a 1920x1080 video resolution) and the cropped area is the vertically centred and left aligned area of the unprocessed video frame.

The videos are also re-encoded into a ‘reliably seekable’ video format as this reduces the likelihood of errors arising due to inconsistent video data being retrieved using a particular frame index. Many modern video codecs compress video data by taking advantage of redundancy in pixel values that don’t change significantly between frames over time and saving space by ignoring the redundant values. As a result, displaying these compressed videos often requires each frame to be reconstructed during playback using the pixel data from the frames around it. Some codecs can reconstruct slightly different frames for the same index during a manual seek operation compared to normal playback and this can cause problems when training a model as SLEAP requires that the frame data are consistent for a given frame index.

Both the re-encoding and cropping operations are performed using the ‘ffmpeg’ video editing software, which is a command-line based application that can perform a range of editing operations on video, audio, and other multimedia files. To perform the required processing efficiently, a Jupyter Notebook file has been written that can process all of the ant videos in a single run, which will be provided to the client for future use.

An example of a command that uses ffmpeg to process a raw video containing ants for training is as follows,

```
ffmpeg -y -i <input_file> -vf "crop=in_w/2:in_h/2:0:in_h/4" -c:v
libx264 -pix_fmt yuv420p -preset superfast -crf 23 <output_file>
```

The `-c:v libx264 -pix_fmt yuv420p -preset superfast -crf 23` set of arguments converts the video into the reliably seekable format necessary for use with SLEAP and the `-vf "crop=in_w/2:in_h/2:0:in_h/4"` argument crops the video using a bounding box that is half the width and height of the original video dimensions and is located halfway between the top and bottom of the frame on the far-left side of the frame.

Data labelling

The data labelling step is performed entirely in the ‘sleap-label’ application, which is a GUI application that is included as part of SLEAP. This application is intended for use as a user-friendly tool for labelling the locations of body parts that SLEAP will use in the model training step to train and evaluate posture estimation models. The labelling process starts by creating a new SLEAP dataset and importing the videos to be labelled. The user must then define the model skeleton, which consists of each animal body part that the user wants to track and the relationship between them. This is represented as a tree structure with the body parts as the tree nodes and the logical connection between them as the tree edges. The skeleton can also be saved and loaded from a JSON file, which is useful for version control purposes.

When the videos have been imported and the model skeleton defined, the positions of each instance of an animal and their body part positions can be labelled. The frames to label can be manually selected from the imported videos but the ‘sleap-label’ application also provides functionality for selecting suggested frames to label, including random selection as well as choosing frames that scored poorly during previous inference attempts among other algorithms. To label a frame, an instance of the model skeleton is added for each occurrence of the animal to be tracked in the frame, which is displayed in the ‘sleap-label’ GUI. Each node in the model skeleton must then be dragged to its actual position in the frame.

Depending on the number of body parts to be tracked and the number of animals appearing in the frame, the labelling process even for a single frame can take a significant amount of time. In practice, it was found that for the skeleton used in this project consisting of 18 parts, the time to label one frame containing around 15 ants could range from 15-30 minutes, depending on the skill of the labeller and the complexity of the ants’ positions. There are no specific guidelines for how many frames or instances must be labelled in order to train an accurate model and reasonably accurate single animal tracking models have been trained with as few as 10-20 labelled frames, however multi-animal tracking models tend to require more frames to be accurate and for this project, over 100 frames were labelled with approximately 15 instances per frame prior to the model training phase.

Dataset combining

To reduce the amount of time required to label the videos, it is possible to have different labellers label a subset of the video data and combine the individual labelled datasets into a single main dataset at the end. The functionality to perform this dataset merge is available in the SLEAP Python module and a Jupyter notebook has been developed as part of this project that uses the Python module to present a user-friendly interface for the client to combine multiple datasets at once. In the notebook, the user must specify the path to a folder containing all of the datasets that they want to merge as well as the name of the output dataset that will be generated. They then only have to run the notebook, which will merge the datasets and output the combined file in the input directory.

Training

Once an appropriate number of video frames have been labelled with the instances of ant present in them, models can then be trained using these labelled frames for reference and evaluation. Models can be trained using the ‘sleap-label’ application, which allows users to specify the training mode (top-down or bottom-up), backbone network, and all associated hyperparameters for the backbone and model heads. Data augmentation can also be configured to adjust the input data during training to allow the model to effectively expand the training dataset without requiring additional labelling by rotating, scaling and flipping the input image to name a few operations. Given that the broad range of hyperparameters can be confusing, particularly for non-technical users, comprehensive user documentation will be designed and delivered as part of this project to educate the client on the meanings of each hyperparameter and the possible effects of changing their values on the performance of the model.

An important consideration during the training phase is whether to perform the required data processing using the training computer’s Graphical Processing Unit (GPU) and what options should be used to ensure that the training process does not exceed the available dedicated video RAM of the GPU. Training using the GPU is almost always faster compared to CPU training as the GPU’s design and processing model are more suitable for parallel data processing. However, depending on the size of the input and the number of input images processed per batch, a substantial amount of video RAM may be required. Therefore, the user may need to adjust settings such as the batch size and input scale to ensure they do not exceed their available resources in this regard, which may have an adverse effect on model accuracy.

Comparing models

After several candidate models have been trained in the previous step, it would be useful to compare the performance of each model by some metric(s) to determine which are most suitable for subsequent inference. As part of the training process for each model, SLEAP calculates several useful metrics that measure the model’s performance in terms of its instance and part location accuracy and part visibility accuracy. These metrics include the average

Euclidean distances between predicted and actual part positions, the confusion matrix values for the part visibility predictions, and the Object Keypoint Similarity and Percent of Correct Keypoint scores, the details of which are explained further below.

In order to make the process of comparing model performance easier, a Jupyter notebook has been developed by the project team that automatically extracts the calculated metrics from a set of trained models and displays them in a tabular and graphical format. This allows the user to quickly and easily compare models without having to manually extract the metrics from each model folder, which can be a time-consuming exercise when there are many models to compare. The user must first specify a parent folder path in the notebook, and then the code in the notebook will recursively search each folder under the parent folder and enumerate each model that it finds. It then extracts the metrics for each training phase and displays the raw values in a tabular format while also creating various useful and informative graphs that will help the user obtain a holistic view of each model's performance compared to the others.

Inference, tracking and data export

The final stage in the pipeline is to use the best performing trained model as determined by the user to infer the positions of ants in videos showing the behaviour of interest and export the inferred ant posture data in a format that is suitable for further analysis beyond what SLEAP can facilitate. The client has previously advised that they want to use the inferred ant postures to correlate the rear leg angles with measurements of the pulling force of the weaver ants, which would require a means of linking the positions of a given ant or ants with the measured force at a particular time point. While performing this type of analysis for the client is beyond the scope of this project, providing a method of extracting the posture data from the inferences carried out on a video file is within the project scope.

For performing inference on unlabelled videos using a trained model, the solution for this project uses the built-in functionality of SLEAP to run predictions on entire videos or sub-sections of them to verify that the model correctly identifies the presences and postures of ants in these videos. This step can also be carried out on unlabelled frames of partially labelled videos, which can be used to simplify the process of labelling more frames in the case where more training is required. Similarly to the training step, performing inference using the GPU does take less time in general, however the number of frames that can be inferred in a single batch will be limited by their available video RAM.

Individual instances of ants can also be tracked between frames across a video using the tracking functionality built into SLEAP. During inference, the positions of ant instances are compared between frames based on the relative positions of their body parts, centroids, or bounding box. Due to the temporal nature of video frames, it can be reasonably inferred that an animal's position will not be significantly different between adjacent frames unless the animal is moving very quickly. Therefore, SLEAP can link together nearby instances on subsequent frames and assign them to a track, which is used to identify a particular animal and its position over the course of a video.

Once inference and tracking (if required) have been completed, the final step is to export the predicted ant instances and body part positions for additional analysis. To maximise compatibility with whichever analysis software that the client wants to use, the position data will be exported in a comma-separated values (CSV) format, which is a commonly used data interchange file format that stores data in a tabular format with columns separated by commas and individual records on separate rows. SLEAP does not provide a straight-forward way of extracting posture data from the predicted labels, therefore the ant posture inference Jupyter notebook provides this functionality instead and can extract the required data in a useful format. In the notebook, the user must specify the file paths for the model and video they want to use for inference, and the output path where the posture data file should be saved on the filesystem. The notebook is then run, which produces the output file with the predicted posture locations in a comma-separated values (CSV) file format that can be readily loaded into any analytics software applications that support these types of files.

Algorithms/Models/Methods

Model backbone

In this project, a supervised Convolutional Neural Network (CNN) will be used to predict the estimated postures of weaver ants in the video that the client has provided. Rather than build a fully customised CNN model, which would likely require more time than is available for the project, a model built on top of one of the backbone models available in SLEAP. In the SLEAP framework, the choice of backbone models for pose estimation plays a critical role in determining the accuracy, speed, and resource requirements of the system. SLEAP includes several backbone architectures and the following backbones were considered during development:

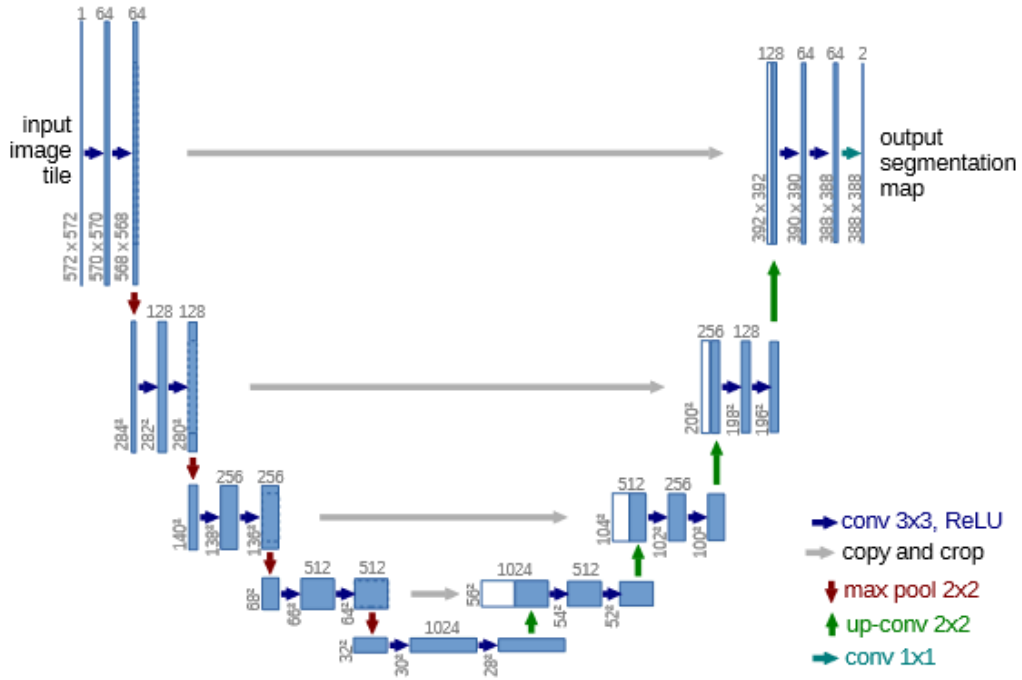
1. **UNet**:
    - UNet is a popular architecture for semantic segmentation tasks, characterised by its symmetric encoder-decoder structure.
    - The UNet architecture consists of a contracting path (encoder) to capture context and a symmetric expanding path (decoder) for precise localization.
    - UNet is particularly effective for multi-animal pose estimation tasks due to its ability to capture fine-grained details and spatial relationships between multiple instances.
    - Additionally, UNet tends to be faster compared to some other architectures, making it suitable for real-time applications.
2. **LEAP**:
    - LEAP (Locally Enhanced Alignment and Priors) is a specific architecture introduced in the original SLEAP paper for animal pose estimation.
    - LEAP is designed to leverage local spatial relationships and priors to improve the accuracy of pose estimation.
    - However, UNet was chosen over LEAP possibly because UNet demonstrated better performance or efficiency in multi-animal inference scenarios.
3. **ResNet**:
    - ResNet (Residual Network) is a deep convolutional neural network architecture known for its effectiveness in image classification tasks.
    - While ResNet can be adapted for pose estimation, it may not be the most suitable choice for multi-animal inference due to its single-stream architecture, which might struggle to capture the diverse spatial relationships between multiple instances in complex scenes.

Considering the specific challenges and requirements of multi-animal pose estimation, UNet emerges as the preferred choice among the available backbones in SLEAP. Its symmetric encoder-decoder structure, ability to capture fine details, and efficiency in multi-animal inference scenarios make it a well-suited backbone for the task at hand. Additionally, the issues

encountered with other backbones, such as errors in stacked hourglass and resource limitations with pre-trained encoders, have influenced the decision to choose a UNet backbone.

The UNet network architecture was originally developed in 2015 by Ronneberger et al. at the University of Freiburg in Germany as an improvement on pre-existing image localisation architectures[2]. A conventional approach to the problem of identifying the location of a label in an image is to apply a model in a “sliding window” over the full image, where each execution of the model only sees a small section of the image. This approach can be effective, but there is a significant amount of redundant processing since the windows must overlap to narrow down the location of a label.

The UNet architecture uses a different approach, where the initial contracting path used for feature recognition is followed by an expanding path, which is used to generate the confidence map of likely locations for the predicted label in the original image and results in a U-shaped model from which the architecture gets its name. This allows the model to generate an inference in a single pass and hence is more efficient than previous approaches. The modular structure of the UNet architecture means that the number of pooling layers can be increased, resulting in an increase in the size of the receptive field (effectively how much of the image that the model can see at once at the lowest level of the model) at the cost of an increase in model complexity and training time.



Example of a UNet architecture from Ronneberger (2015)[2]

[2] Ronneberger O, Fischer P, Brox T (2015) ‘U-Net: Convolutional Networks for Biomedical Image Segmentation’, *CoRR*, viewed on 12 May, accessed at: https://arxiv.org/abs/1505.04597

Training mode

SLEAP offers two modes for training models, a top-down approach and a bottom-up approach. The top-down approach involves training two models, one for locating individual instances of animals in a given frame and outputting the centroid location of each instance in the frame, and the second for inferring the posture of each instance. This is done by cropping the frame to only include the animal instance based on the centroid location and then identifying the position of each animal part in the cropped frame. The bottom-up approach instead trains a single model with two output heads, one for identifying animal part locations in the entire frame, and one for learning the part affinity field, which groups the individual animal parts into cohesive instances.

Both approaches can perform well, depending on the subjects of the videos used as input. However, both approaches have their respective advantages and disadvantages, which must be considered when selecting which approach to use. For top-down models, the centroid inference model can still produce reasonably accurate results when the centroid model input is scaled, reducing the amount of system resources required to train this model, whereas the centred instance model can use the full resolution input as it uses a cropped version of the frame as input hence reducing the amount of memory required to train. However, the overall accuracy of inferred animal positions is highly dependent on the performance of the centroid model, as any instance that is not recognised will not be passed on to the centred instance model for posture inference. Additionally, if the inferred location of the centroid for an instance is incorrect, this may result in certain body parts of the instance being cropped out of the frame and hence not being considered visible by the model.

The bottom-up approach can potentially produce more accurate inferences in cases where there are many densely-packed instances of animals as the model is not limited to finding animal body parts that are necessarily associated with a coherent instance. However, this can result in cases where the incorrect parts, particularly the limbs, of one animal instance are associated with another unrelated instance. Additionally, the bottom-up approach tends to perform poorly when the input is scaled, and training models using the full resolution input can require a large amount of memory that most consumer-grade computers do not have, which can make this approach infeasible for large input images.

Therefore, for this project we have chosen to use a top-down approach as the ant videos that have been provided are large enough to exhaust the video RAM of most consumer-grade GPUs when using the bottom-up approach and testing with the top-down approach has shown that accurate results can still be achieved using this approach. Note that hereafter the centroid and corresponding centred instance models will be referred to as a single model for brevity.

Hyperparameters

By carefully selecting these hyperparameters, we aim to achieve accurate and robust pose estimation results while maintaining computational efficiency and handling uncertainties inherent in real-world data.

Backbone hyperparameters

Hyperparameter	Value
Filter size	3x3
Centroid: Maximum stride	16
Centroid: Number of filters	32
Centred Instance: Maximum stride	32
Centred Instance: Number of filters	32

- **Filter size**
  - The filter size determines the spatial extent of the convolutional kernels used in the convolutional layers of the UNet backbone.
  - A larger filter size captures more global features but may lead to a loss of fine details, while a smaller filter size captures finer details but may increase computational complexity.
  - In SLEAP's UNet backbone, a filter size of 3x3 is commonly used. This size strikes a balance between capturing local features and preserving computational efficiency. It allows the model to extract meaningful features while maintaining a reasonable computational cost.
- **Maximum stride**
  - The maximum stride determines the receptive field of the model by controlling the number of pooling layers in the model, with more pooling layers effectively increasing the size of the receptive field at the cost of increasing the network complexity.
  - The receptive field is a measure of how much of the image that the model can “see” at any given time, and a larger receptive field allows the model to capture more spatial information at a time.
  - For this project, a maximum stride of 16 was chosen for the centroid model and 32 was chosen for the centred instance model based on testing with different maximum stride values. This corresponds to 4 and 5 pooling layers respectively and gives reasonably accurate results without introducing excessive complexity into the model.

- **Number of filters**
    - The number of filters for a UNet backbone refers to the number of filters in the first encoder block, with the number of filters in subsequent encoder blocks being increased by a factor of 2 by default.
    - Increasing the number of filters allows the model to extract more features during the training process, but increases the model size and can lead to overtraining.
    - A value of 32 for this hyperparameter in both the centroid and centred instance models balances the size required to capture the complexity of recognising the ant positions and postures while reducing the possibility of overtraining and improving the models ability to generalise to novel inputs.

Model head hyperparameters

Hyperparameter	Value
Sigma	2.5
Anchor part	LowerThorax

Centroid model

The centroid model predicts the location of the centre point of each instance in the image. The following hyperparameters were configured to produce the model with the best overall performance.

- **Sigma**:
    - Sigma refers to the standard deviation parameter of the Gaussian distribution used to generate the confidence heatmap for the centroid prediction.
    - A higher sigma value results in a broader and smoother confidence heatmap, while a lower value leads to a sharper and more localised heatmap.
    - In SLEAP, a sigma value of 2.5 is chosen for both the centroid predictions. This value strikes a balance between capturing the uncertainty in the predicted locations and maintaining spatial accuracy. It provides sufficient smoothing to handle noise and variability in the data while still preserving the distinctness of individual instances.

Centred Instance model

The centred instance model predicts the estimated pose for each instance, consisting of estimates for each body part location. The following hyperparameters were configured to produce the model with the best overall performance.

- **Sigma**:
    - Similar to the centroid, sigma determines the spread of the Gaussian distribution used to generate confidence heatmaps for each keypoint.
    - A higher sigma value results in smoother heatmaps, which are easier to learn but tend to produce less accurate predictions, while a lower value produces sharper, more localised heatmaps, which can be more accurate but are more difficult to learn and can take longer to train.
    - A sigma value of 2.5 is chosen for the centred instance predictions in SLEAP. This choice ensures that the predicted keypoints are well-defined and accurately localised while accounting for uncertainties in the predictions due to noise and variability in the input data.
- **Anchor part:**
    - Setting an anchor part as opposed to using the midpoint of the centroid bounding box as the reference point for each instance can improve model accuracy by establishing a consistent geometry of body parts relative to the centre of the cropped image.
    - This helps reduce the occurrence of prediction errors where body parts are predicted to be in physiologically improbable locations, which can be a common issue when using the top-down approach.
    - The lower thorax was chosen as the anchor part as this was found to produce more accurate models, which is likely due to the fact that the lower thorax is one of the most central parts of the ant's body.

Detailed Data Description

Ant videos

The videos of weaver ants provided by the client are the main raw data source used for generating models capable of tracking the postures of ants. The videos were captured as part of research conducted at the Macquarie University School of Natural Sciences by Masters student Madelyne Stewardson as part of Chris Reid's team. The primary focus of this research was to measure the pulling strength of weaver ants engaging in nest-building behaviours. The videos themselves capture the experimental setup that consists of numerous ants pulling on the tip of a piece of paper cut into the shape of a leaf typically used by weaver ants to construct their nests. These ants either pull individually or create "pulling chains" consisting of their own bodies when pulling on the leaf tip, and tracking the ants' postures and correlating them with the measured pulling force is one of the primary goals of this project.

The video data consists of 23 MPEG-4 video files with 19 files recorded in 3840x2160 pixel resolution, 2 files recorded in 1920x1080 pixel resolution, and 1 in 1280x720 pixel resolution. The videos range in duration from approximately 10 minutes to around 1 hour and 40 minutes and the file sizes range from 2.03GB to 19.70GB. The videos were recorded using a Panasonic Lumix GH-5 digital camera equipped with a macro 30mm lens and all videos were recorded at 24 frames per second.

Once the raw videos have undergone pre-processing as per the 'Video pre-processing' stage in the solution architecture, the processed videos are output at 25% of the original frame size in pixels. Therefore, the processed video data consists of 23 MPEG-4 video files with 19 files in 1920x1080 pixel resolution, 2 files in 960x540 pixel resolution, and 1 in 640x360 pixel resolution, and the processed video files range from 495MB to 2.61GB in file size.

Labelled datasets

When a user labels frames with the ground truth position of the different animal body parts visible in a frame, all relevant information related to the videos being labelled and the labels themselves is saved in a SLEAP-specific '.slp' file. Internally, '.slp' files use the Hierarchical Data Format 5 (HDF5) file format. The HDF5 format stores data in a user-defined hierarchy consisting of 'datasets', which are multi-dimensional arrays, and 'groups', which store datasets and/or other groups. SLEAP uses its own hierarchy in this file format consisting of datasets for the frames, instances, and label points in the file, as well as other metadata for the suggested frames to label and the instance tracks. While it is possible to open '.slp' files in any program or library that supports HDF5 files, SLEAP also provides a library that includes functions for reading from '.slp' files, which makes extracting information from these files more straight-forward.

The individual labelled datasets can also be merged into a single main labelled dataset, which can be the primary dataset used for training models as having all of the labelled data in a single file simplifies the model training process. The functionality for merging datasets is built into SLEAP and can be performed either through the GUI or using the SLEAP Python module. The resulting file is also output in the SLEAP file format and can be merged with other datasets if required.

Models

Each trained model produced by SLEAP is stored in its own folder on the filesystem with several key files that allow the model to be used for subsequent training and/or inference. The weights in each layer of the best performing model are stored in a ‘best_model.h5’ file, which is an HDF5 file which is necessary for reconstructing the exact model state for use in later inference. This file can be substantial in size depending on the structure of the trained model and preliminary training attempts in this project generated ‘best_model.h5’ files that were typically over 100MB in size. Therefore these files will be stored and provided to the client in Google Drive instead of the Github repository, as Google Drive is already available to Macquarie University students and staff, and Github has a maximum individual file size limit of 100MB without using Git Large File Storage, which would incur additional and potentially significant costs.

The model configuration is also stored in the model folder and consists of two files, ‘initial_config.json’ and ‘training_config.json’. The ‘initial_config.json’ file contains the settings provided by the user prior to the beginning of the training process and can be used as the basis for other training jobs in future. The ‘training_config.json’ file is largely similar to the ‘initial_config.json’ file in that it contains the settings selected by the user before training, however it also contains other metadata set by SLEAP immediately prior to training, including the split of training, validation, and test data used. This allows an analyst to view the exact settings used to train the model, inclusive of the user-defined settings and those set by SLEAP. Both files use the JavaScript Object Notation (JSON) file format, which is a commonly used text file format which can serialise complex data structures in a human and machine readable format.

The ground truth labelled datasets and predicted labelled datasets used and produced in each phase of the training process (training, validation, and testing) are also stored as part of the model, with ‘labels_gt.train.slp’, ‘labels_gt.val.slp’, and ‘labels_gt.test.slp’ containing the ground truth labelled data for the training, validation, and testing stages respectively, and ‘labels_pr.train.slp’, ‘labels_pr.val.slp’, and ‘labels_pr.test.slp’ containing the predicted label positions for the same frames in each split. These files are stored in the same SLEAP file format used for the labelled dataset files, hence they can be opened in the SLEAP GUI or using the SLEAP library to examine their contents. They are not strictly necessary for re-training or performing inference using the model, however these files may be useful for investigating why a particular model has performed differently in each stage of the training process.

There are also files stored with the model containing various metrics calculated during each training phase. The ‘metrics.train.npz’, ‘metrics.val.npz’, and ‘metrics.test.npz’ contain the metrics calculated for the training, validation and testing phases respectively. The metrics contained in each file include the average Euclidean distances of each node in the skeleton between the predicted position and the ground truth position, as well as the average distances of different percentiles of the predicted points. The metrics also include the mean Percent of Correct Keypoints (PCK) and mean Object Keypoint Similarity (OKS) scores averaged over all predictions in each training phase. The PCK score measures the percentage of correct predictions, with a prediction being considered correct if it is less than a given threshold value. The OKS score applies a statistical approach that takes the size and visibility of the body part into consideration when calculating the relative distance between a predicted body part position and the ground truth, which is then averaged to get the total OKS score. Finally, the confusion matrix values (true positive, false positive, true negative, false negative) for the trained model’s ability to correctly identify when a body part is visible in frame are stored in the metrics file. This also includes the precision and recall values, which are commonly used measures of accuracy when evaluating a model’s performance. The ‘.npz’ file format represents a NumPy serialised file which can be de-serialised using the NumPy library to obtain the original data structure containing each metric.

Finally, a log file named ‘training_log.csv’ is included that stores the loss function values calculated for each phase and epoch in the training process. These values can be useful for analysing the rate at which the model converges and the point where over-training starts to occur. The file is in a CSV file format, which represents individual records as separate rows with the column values delimited by commas. This file format can be readily loaded by many analytics software packages and even spreadsheet programs for quick analysis.

Exported position data

To facilitate further analysis of the ants’ postures once the positions of the ants’ body parts in the videos have been predicted, the predictions of locations of each body part in each instance of each frame will be exported from SLEAP in a CSV file format, as this is a common file format that can be loaded by many other analytics applications such as R and MATLAB. The output file will be generated using a Jupyter notebook (explained further below) and the structure of the output file is as follows.

Output file structure

Field name	Data type	Description
video_path	String	The absolute path to a video containing one or more instances of ants.
video_width	Integer	The width of the video in pixels.
video_height	Integer	The height of the video in pixels.
frame_idx	Integer	The index for a frame in the video given by video_path.
instance_idx	Integer	The index for an instance in a frame. Note that the index is only applicable for a particular frame and the same value does not correspond to the same instance between frames.
track_id	String	The name of the track that the instance belongs to, or blank if the instance does not belong to a track.
part_name	String	The technical name for a body part in an instance.
predicted	Boolean	True if the record is the predicted position of a body part, False if the record is the ground truth for a body part position.
x_pos	Float	The X-coordinate of the body part in pixels.
y_pos	Float	The Y-coordinate of the body part in pixels.

Jupyter notebooks

In cases where the SLEAP GUI is insufficient to meet a specific use case for the client, additional functionality will be provided by various Jupyter notebooks. Jupyter is a web-based application that allows Python code to be executed individually in separate cells with further explanation and information given in other cells using the Markdown markup language. The interactive nature of these notebooks makes them easier to use by non-technical users and avoid having to develop a separate GUI application for these same use cases, which would take a significant amount of development time and require someone with the technical knowledge and skills to maintain the application(s). For this project, in addition to the notebooks mentioned earlier that will be used to merge the labelled datasets and generate the prediction output file, there will be two additional Jupyter notebooks, one for summary and analysis of labelled datasets and one for model performance summary and analysis.

Labelled dataset analysis

The ‘Label analysis.ipynb’ Jupyter notebook can be used to summarise a specified labelled dataset in the form of a ‘.slp’ SLEAP file. The user must provide the path to a ‘slp’ file in the notebook itself and then, upon executing all of the cells in the notebook, several metrics will be

calculated based on the contents of the dataset file. Firstly, the videos referenced by the dataset and the total number of labelled instances for each video will be calculated and presented, which will provide a clear view of the label coverage for each video in the dataset. The total number of visible instances of each body part will also be calculated, which will help indicate which body parts tend to be more visible and may provide an explanation if certain body parts tend to be harder for the model to identify.

Finally, the total number of unverified labelled points in the dataset will be calculated. In SLEAP, when a new instance is added to a frame, by default the points are in an unverified state as they haven't had their position confirmed by a human labeller. These points are still considered in the training process so if the labeller inadvertently fails to put these points in the correct location for the body part (as the default position is almost always wrong) and thus verify them, the model may be trained using the incorrect parts of the image for a body part which would adversely impact the model's accuracy. Having a summarised view of which frames contain any unverified points would help labellers correct any mislabelled data points, improving the quality of the labelled dataset.

Model comparison analysis

When tuning the model hyperparameters, it is highly likely that multiple trained models will be generated with different sets of hyperparameter values. The 'Model analysis.ipynb' Jupyter notebook can be used to compare the summary metrics of different models to help determine which model would be most suitable and accurate. The user must specify the path to a parent folder containing all of the models that they want to compare and these models may be nested as deeply as the filesystem will allow. Once all of the models have been enumerated in the parent folder, the various model metrics (as explained previously) will be extracted from the metrics files and displayed in a summary table for straight-forward comparison.

Assumptions

The APRA project runs under the assumption that the client possesses the necessary hardware to run the required software including PCs or laptops with system specifications capable of handling SLEAP for ant posture analysis. Though both Windows and Macintosh operating systems are supported, it is assumed that Windows operating system will be used as the client has expressed previously that this is the primary operating system used in their lab. It is also assumed that the client has or can obtain an Nvidia graphics card to allow for GPU training of models, which is faster to perform compared to CPU training. The graphics card must be an Nvidia graphics card as SLEAP uses the Compute Unified Device Architecture library to perform GPU training and inference, which is an Nvidia proprietary technology and hence is only supported by Nvidia graphics cards. However, it is also assumed that the graphics card will be a consumer-grade model as workstation-grade models can cost in excess of $10,000, which is likely to be outside the client's budget. Therefore, it is assumed that the available video RAM is limited to a maximum of 8GB.

The project also assumes that the client's technical skill level includes an intermediate level of computer literacy which is sufficient to navigate the software tools and data analysis methods used in the project. The client must be able to install SLEAP through the command-line interface, given appropriate instructions by the project team, and understand the steps to initialise it as the integrated SLEAP GUI application must first be activated using the command-line interface before it can be used. Overall, it is expected that, given comprehensive user documentation and training manuals, the client will be able to effectively use and maintain the developed system without ongoing external support.

All of the code and model metadata will be stored in a Github repository that will be handed over to the client at the conclusion of the project and the full models will be shared using Google Drive due to their large file sizes. All of the project team members have had prior experience with Git and Github in previous unit studies so everyone should have the knowledge and skills necessary to manage their work using these applications without interfering with or overwriting the work of others in the team. It is assumed that, after handover, the client will be able to navigate the Github web interface to obtain a copy of the repository, though it is not necessary for the client to understand how to use Git itself unless they wish to take advantage of the version control features for any future development, which the client has not expressed any desire to do so.

SLEAP was chosen as the main computer vision framework for this project as it specialises in multi-animal pose estimation. This is important for the analysis and tracking of the complex movements of weaver ants in this project, as there may be upwards of 30 or more ants in a given video frame. This software can conduct a thorough analysis of the movement of the limbs of ants and a Python module is provided that allows for the inferred positions to be extracted into an output file for further analysis. Its selection over other pose estimation frameworks such as DeepLabCut was also decided by its ease of use, reducing the learning curve of installation

and enhancing the project's feasibility for use by non-technical users. The use of SLEAP does operate under the assumption that the video input is of sufficient visual quality to allow for precise labelling of ant postures to train an accurate model, however this would also be applicable for most other pose estimation frameworks.

For the model itself, the decision to use a top-down approach for the model design was primarily driven by computing resource constraints given the comparatively large file size and pixel resolution of the ant videos used as inputs. Bottom-up models perform best on the unscaled input frames, which can require a substantial amount of video RAM. The core assumptions around this are that any future videos of ant behaviour will be recorded using the same camera that was used for the video used in this project and therefore, will be of sufficient visual fidelity to be able to differentiate the body parts of each ant in the video, but also be large enough to make the use of bottom-up models impractical due to the high video RAM requirements for training models on the GPU with such large inputs. For a team of 6 people labelling, using 20 reference frames with up to 20 ants labelled in each frame per person was assumed to be sufficient for training reasonably accurate models for the purposes of MVP development.

Most of the additional functionality developed under the project (dataset merging, model comparison, and posture data extraction) is provided via Jupyter notebook files. The decision to use Jupyter as opposed to developing a custom-built GUI application was made due to the time, effort and knowledge required to build and maintain such an application. It was assumed that a custom solution would require more time than is available for the project and would require the client to maintain the software after the conclusion of the project, which requires technical knowledge that is outside their field of expertise. Jupyter is supported on all major operating systems, however the initial setup requires the creation of a Python environment, therefore it is assumed that the client will be capable of understanding and carrying out the setup instructions prepared by the project team.