Mice Tracking Using The YOLO Algorithm

Richardson Santiago Teles de Menezes², John Victor Alves Luiz², Aron Miranda Henriques-Alves³, Rossana Moreno Santa Cruz¹, and Helton Maia²

¹Electrical Engineering Department, IFPB, Joao Pessoa-PB, Brazil
²School of Science and Technology, UFRN, Natal-RN, Brazil
³Brain Institute, UFRN, Natal-RN, Brazil

Corresponding author:
Helton Maia²
Email address: helton.maia@ect.ufrn.br

ABSTRACT

The development of computational tools is essential for the development of new technologies, including experimental designs needed for behavioral neuroscience research. The computational tool developed in this study is based on the convolutional neural networks and the You Only Look Once (YOLO) algorithm for detecting and tracking mice in videos recorded during behavioral neuroscience experiments. The task of mice detection consists of determining the location in the image where the animals are present, for each frame acquired. In this work, we propose mice tracking using the YOLO algorithm, running on an NVIDIA GeForce GTX 1060 GPU. We analyzed a set of data composed of 13622 images, made up of behavioral videos of three important researches in this area. The training set used 50% of the images, 25% for validation and 25% for the tests. The results show that the mean Average Precision (mAP) reached by the developed system was 90.79% and 90.75% for the Full and Tiny versions of YOLO, respectively. It has also been found that the use of the Tiny version is a good alternative for experimental designs that require real-time response. Considering the high accuracy of the results, the developed work allows the experimentalists to perform mice tracking in a reliable and non-evasive way, avoiding common system errors that require delimitations of regions of interest (ROI) or even evasive luminous identifiers such as LED for tracking the animals.

INTRODUCTION

In recent years, researchers in the field of computer vision have developed a large number of algorithms and techniques for intelligent image processing. Not long ago, some methods for identifying the objects in images overcame the human-level performance to categorize images. This was achieved by Microsoft in the ImageNet (Large Scale Visual Recognition Challenge, 2015) (Russakovsky et al., 2015), using Deep Neural Networks (DNN).

Convolutional Neural Networks (CNNs or ConvNets) are a type of neural networks widely used in computer vision problems. Inspired by biological processes, CNNs are designed to obtain a large number of connection patterns, which in combination can similarly respond to stimuli, such as the visual cortex (Cichy et al., 2016). Among the applications currently are the recognition of patterns in video and images (Feichtenhofer et al., 2016), diagnosis and analysis of medical images (Rajpurkar et al., 2017; Esteva et al., 2017), object detection in images (Liu et al., 2016; Redmon et al., 2016) and surveillance monitoring systems (Rasti et al., 2016).

Multidisciplinary research areas such as Neuroscience can benefit from the improvement of laboratory experiments to complex data analysis using modern machine learning techniques (Vu et al., 2018). In this way, it is possible to perform the detection and classification of brain patterns, correlations between behavioral tasks and electrophysiological recordings, social stress tests and animal interaction, among other experimental designs with rodents (Henriques-Alves and Queiroz, 2016; Frasch et al., 2017). Neuroscientists conduct a series of experiments in which the tracking of animals on video is essential for the research objectives (Menezes et al., 2018; Unger et al., 2017; Mathis et al., 2018). This tracking can be done in real time or even in an offline analysis after the experiment. Most of the software specialized
in rodent tracking have characteristics such as high acquisition cost and need facilitating environments 
for a good accuracy of the tracking algorithm, they are: different colors between animals, objects and 
the background, previous selection of a region of interest (ROI) to remove information that may confuse 
detection, use of a reference image of the environments before placing the animals for tracking, this 
allows comparison with subsequent images, controlled environment brightness, focus control and fixed 
distance between animals and the acquisition camera (Aitken et al., 2017; Romero-Ferrero et al., 2018). 

This work aims at the development of techniques for tracking mice with convolutional neural networks 
using the You Only Look Once (YOLO) algorithm. We used three different experimental data on 
behavioral tasks of mice, showing the ability of CNNs to perform mice tracking under adverse conditions, 
tasks that are often avoided during these types of experiments. The results obtained provide a fast and 
accurate alternative for mice tracking in complex scenarios, which include reflexes, obstruction of objects 
and even more than one animal.

METHODS

We use the C/C++ programming language and the YOLOv3 algorithm (Redmon and Farhadi, 2018) for 
object detection, in this case, for detecting mice in different datasets (Henriques-Alves and Queiroz, 2016; 
Jhuang et al., 2010; Burgos-Artizzu et al., 2012). The computational development was performed on a 
computer with CPU AMD Athlon II X2 B22 (2) @ 2.800GHz, RAM 4GB, GPU NVIDIA GeForce GTX 
1060 6GB, OS Ubuntu 18.04 LTS, CUDA 9.1, CuDNN 7.1.

Computational Development

All steps required in this work are described in the UML activity diagram shown in Figure 1.

- Step 1: Creation of a dataset from images used in mice behavioral experiments. Selection of the 
ground truth boxes for future comparison with the box predictor and obtaining the confidence score 
using the intersection over union (IOU).

- Step 2: Selection of the Yolov3 algorithm in the Full or Tiny version for mice tracking.

- Step 3: The Convolutional Neural Network performs the training, validation, and testing for mice 
tracking in the dataset images.

![UML activity diagram](image.png)

Figure 1. UML activity diagram of the developed system for video tracking and classification.

Image Dataset

To validate the computational development proposed in this work, a dataset was built with images from 
three researches that involve behavioral experiments with mice:

- Ethological Evaluation (Henriques-Alves and Queiroz, 2016): This research presents new metrics 
for chronic stress models of social defeat in mice. The authors built their image dataset and kindly 
provided a sample for our research.

- Automated home-cage (Jhuang et al., 2010): In this study, a trainable computer vision system was 
introduced that allows the automated analysis of complex mouse behaviors, they are: eat, drink, 
groom, hang, micromovement, rear, rest, and walk. This dataset has two parts - Full: with more 
than 10.6 hours of continuous video of various behaviors, and Clipped: with 4200 small videos 
with specific behaviors.
- **Caltech Resident-Intruder Mouse dataset (CRIM13)** (Burgos-Artizzu et al., 2012): It has videos recorded with superior and synchronized lateral visualization of pairs of mice involved in social behavior in thirteen different actions. Each video has a duration of \( \sim 10 \) min, totaling 88 hours of video.

Table 1 describes the sample size selected from each of the datasets used in this paper. For the Ethological Evaluation (Henriques-Alves and Queiroz, 2016), 3707 frames were used, captured in a top view of the arena of social interaction experiments among mice. For the Automated home-cage (Jhuang et al., 2010), a sample of 3073 frames was selected from a side view of behavioral experiments: eat, drink, groom, hang, micromovement, rear, rest, and walk. For the Crim (Burgos-Artizzu et al., 2012), a sample of 6842 frames was selected, 3492 of a side view and 3350 of a top view for the following experiments: approach, circle, sniff, walk away. The sampling rate for selection of the frames in all cases was \( \sim 1.5 \) s, avoiding similar images in the composition of the dataset.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Images</th>
<th>Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ethological Evaluation (Henriques-Alves and Queiroz, 2016)</td>
<td>3707</td>
<td>640 \times 480</td>
</tr>
<tr>
<td>Automated home-cage (Jhuang et al., 2010)</td>
<td>3073</td>
<td>320 \times 240</td>
</tr>
<tr>
<td>Crim (Burgos-Artizzu et al., 2012)</td>
<td>6842</td>
<td>656 \times 490</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>13622</td>
<td></td>
</tr>
</tbody>
</table>

We randomly selected the number of images used for training 50%, validation 25% and tests 25%, as described in Table 2 on the composite dataset for this work.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Training</th>
<th>Validation</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Images</td>
<td>6811</td>
<td>3405</td>
<td>3406</td>
</tr>
</tbody>
</table>

**MICE TRACKING USING CNN**

In Neuroscience, mice models of social stress (Henriques-Alves and Queiroz, 2016) are often used and require fast and accurate decision-making from the experimentalist to identify the correct response depending on the stimulus or the situation in which the animal is subjected. The essential element of such models has been a robust tracking algorithm. However, these algorithms are mainly limited by the computational cost, which hinders their usage in real-time and therefore limits the applicability for laboratory tests.

Commercial systems for video tracking of laboratory animals are often restricted to track the body or nose position, leading to inaccurate estimates of the head orientation or requiring manual artifices (Menezes et al., 2018; Kretschmer et al., 2012). Thus, large-scale studies emphasize the need for automated high-throughput systems providing a reproducible behavioral assessment of freely-moving mice with only a minimum level of manual intervention (Unger et al., 2017).

In this context, this work focuses on the development of a computational tool designed with deep convolutional networks, the **Darknet** framework (Redmon and Farhadi, 2018), the Open Source Computer Vision Library (OpenCV) and the C/C++ programming language for mice tracking during behavioral neuroscience experiments.

**Convolutional Neural Networks**

The idea behind CNNs is to use a cascade of convolutional and pooling layers to reduce the spatial dimension of an input image and combine local patterns to generate features that get more abstract as the data progress into the network. This cascade is called an encoder as raw input pixels are encoded into more abstract features (Lecun et al., 1998).

In this paper, \( f^{in}_{i} \) denotes the \( i \)-th input feature map, \( f^{out}_{j} \) denotes the \( j \)-th output feature map and \( b_{j} \) denotes the bias term for the \( j \)-th output feature map. For the convolution layer, \( n_{in} \) and \( n_{out} \) represent the...
number of input and output feature, respectively. For the Fully connected layer, \( n_{in} \) and \( n_{out} \) are the length of the input and output feature vector.

The convolutional layer takes a series of feature maps as input and convolves with convolutional kernels to obtain the output feature maps. A nonlinear layer, which applies nonlinear activation function to each element in the output feature maps, is often attached to convolutional layers. The convolutional layer can be expressed using (1):

\[
f_{i}^{out} = \sum_{j=1}^{n_{in}} f_{i}^{in} \otimes g_{i,j} + b_{j} \quad (1 \leq i \leq n_{out})
\]

Where, \( g_{i,j} \) is the convolutional kernel applied to \( j \)-th input feature map and \( i \)-th output feature map and \( b_{j} \) is the bias term for the \( j \)-th input feature map.

The pooling layer also referred as a downsampling layer, which takes an \( n \times n \) filter and a stride of length \( n \), is then applied to the input vector and outputs the maximum or average values of each subarea. The reasoning behind this layer is that once we know that a specific feature is an original input, its exact location is not as crucial as its relative location to other features, such invariance can be provided by this layer as seen in (Scherer et al., 2010; Zhou et al., 2016). Max-pooling can be expressed as (2):

\[
f_{i,j}^{out} = \max_{p \times p} \begin{pmatrix} f_{m,n}^{in} & \cdots & f_{m,n+p-1}^{in} \\ \vdots & \ddots & \vdots \\ f_{m+p-1,n}^{in} & \cdots & f_{m+p-1,n+p-1}^{in} \end{pmatrix}
\]

Where, \( p \) is the pooling kernel size. This non-linear “downsampling” not only reduces the feature map size and the computation for next layers but also provides a form of translation invariance.

In a CNN, the encoder is often followed by some fully-connected layers as in a classical multi-layer perceptron setup. These layers apply a linear transformation to the input feature vector (3):

\[
f^{out} = W \cdot f^{in} + b
\]

Where, \( f^{in} \) is the input feature vector resulted from previous convolutions, \( W \) is an \( n_{out} \times n_{in} \) transformation matrix, \( b \) is the bias term, and \( f^{out} \) is the output with the classes probabilities.

These layers take the highly abstracted encoded features as input, forwards then throughout the network using the feed forward algorithm and output global statistical predictions about the presence or absence of objects of interest in the input image (Ciresan et al., 2011).

**YOLO Object Detector**

You Only Look Once (YOLO) (Redmon and Farhadi, 2018) is an object detection algorithm targeted for real-time processing. It differs from other object detectors by using a single CNN for both classification and localization of the objects.

YOLO uses logistic regression for calculating the confidence score of an object in each bounding box. Another feature is to use a variant of Darknet, which has a 53-layer network trained on Imagenet dataset (Deng et al., 2009). However, 53 more layers are stacked on it to form a 106-layer fully convolutional architecture for detection tasks. The algorithm also includes many important elements like residual blocks, skip connections, and upsampling in its architecture.

The analysis of a frame in the YOLO framework consists of three steps. First, the input image is resized, then a single CNN is run on it, and in the last step, thresholds using non-max suppression are applied in the resulting detections as described in (Redmon et al., 2016). Figure 2 describes how an image of the dataset of this work is processed initially in the input, passing through the CNN, and finally in the output with mouse tracking.

**RESULTS**

Our approach used two versions of the YOLO network to detect mice within three different experimental setups. The results obtained in this work were based on the analysis of 13622 images, organized according to the dataset described in Table 2.
The first version of YOLO trained was the YOLO Full network which uses the Darknet-53 (Redmon and Farhadi, 2018) convolutional architecture that comprises 53 convolutional layers. Such a model was trained as described in (Redmon et al., 2016) starting from an Imagenet (Deng et al., 2009) pre-trained model. The model comprises a segment with convolutional layers and residual connections in a total of 65.290 × 10⁹ floating point operations. Each model requires 235 MB of storage size. We used a batch of 64 images, a momentum of 0.9, and weight decay of 5 × 10⁻⁴. The model took 139 hours to be trained.

We also trained a smaller and faster YOLO alternative, namely YOLO Tiny. To speed up the process this “tiny” version comprises only a portion of the Darknet-53 (Redmon and Farhadi, 2018) resources: 23 convolutional layers, resulting in 9.670 × 10⁹ floating point operations, almost seven times fewer operations than the bigger version. Each model requires only 34 MB of storage space. The network was trained as described in (Redmon et al., 2016), fine-tuning an Imagenet (Deng et al., 2009) pre-trained model. We used a batch of 64 images, a momentum of 0.9, and weight decay of 5 × 10⁻⁴. The model took 19 hours to be trained. Figure 3 shows some examples, resulting from mice tracking performed on the three different datasets used.

Table 3 shows the classification metrics resulting from the classification step, using both, YOLO Full and Tiny. These metrics are used to evaluate the quality of the classifier output. Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. Recall is the ratio of correctly predicted positive observations to all observations. F1-score is the weighted average of precision and recall, therefore, the maximum value assigned to it is 1, which means a perfect precision and recall.

<table>
<thead>
<tr>
<th>Dataset size</th>
<th>Precision</th>
<th>Recall</th>
<th>F1-score</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Full</td>
<td>Tiny</td>
<td>Full</td>
</tr>
<tr>
<td>13622</td>
<td>0.99</td>
<td>0.98</td>
<td>0.99</td>
</tr>
</tbody>
</table>

Still on Table 3, for the Tiny network, the average values of Precision, Recall, and F1-score were all equal to 0.987, which indicates a good relation between Precision and Recall, whereas for the Full network, the results were slightly better, with the average values of Precision, Recall, and F1-score equal to 0.99. More results related to mice tracking performance are shown in Table 4. First, it is shown the mean average precision (mAP), which is the mean value of the average precisions for each class, where average precision is the average value of 11 points on the Precision-Recall curve for each possible threshold, that is all the probability of detection for the same class (Precision-Recall are evaluated according to the terms described in the PascalVOC (Everingham et al., 2015)). Finally, the average intersect over union (IoU), which is the average of the intersection between the predicted bounding box and the ground truth box divided by the the total area of both boxes.

The Full network shows slightly better statistics for both the mean average precision and average IoU, differing from the Tiny counterpart by 0.04% and 1.86%, respectively, as shown in Table 4.
Figure 3. Output examples of the YOLO network. (a)-(c) refer to Ethological Evaluation (Henriques-Alves and Queiroz, 2016), (d)-(f) refer to Automated home-cage (Jhuang et al., 2010) and (g)-(i) to Crim (Burgos-Artiztu et al., 2012).

Table 4. Results for the epoch 40000, mAP and IoU

<table>
<thead>
<tr>
<th>Dataset size</th>
<th>Mean Average Precision (%)</th>
<th>Average IoU (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Full</td>
<td>Tiny</td>
</tr>
<tr>
<td>Full</td>
<td>90.79</td>
<td>90.75</td>
</tr>
</tbody>
</table>

In terms of training and testing time for the YOLO network, Table 5 shows the efficiency of using the Full architecture in relation to the Tiny one. The training using the Tiny architecture was performed in 19 hours, that is about seven times faster than the Full version. For the tests the use of the Tiny version was about four times faster than the Full counterpart.

Table 5. GPU time necessary to perform the training and test of the model

<table>
<thead>
<tr>
<th>Train Time (hours)</th>
<th>Test Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full</td>
<td>Tiny</td>
</tr>
<tr>
<td>139</td>
<td>19</td>
</tr>
</tbody>
</table>

Figure 4 shows the comparison between the two models used, YOLO Full and Tiny. It was presenting the same sample of 100 images from the set of tests, for both models. Figure 4(a) shows high accuracy of the Full architecture with small oscillations of the accuracy curve during the training. In Figure 4(b), the high accuracy is maintained from the earliest times, and remains practically unchanged up to the limit number of epochs. Both architectures reached high mean average precision values while successfully minimizing the values of the loss function. The Tiny version of the YOLO network presented better stability in precision, which can be seen by the smoothness in its curve.

Figure 4(c) is a bar graph showing the mean time spent on the classification of a single image in both architectures. The smaller size of the Tiny version gets a direct translation in execution time, having 0.08 ± 0.06 as the mean and standard deviation values, whereas the Full version has 0.36 ± 0.16s as the mean and standard deviation values, respectively.
The results presented in Tables 3 and 4 suggest a higher classification accuracy for the YOLO network when using the Full architecture, although the difference obtained with the Tiny version was not substantial. Given the aforementioned small difference between the two versions of the network, the possibility of a robust real time system for mice tracking is made a reality with the Tiny version of the YOLO network. Due to the smaller demand of computing power, this work opens the possibility of real-time tracking systems where actions can be taken during the experiment in an automated and intelligent way without the need for human intervention.

The computational development presented in this paper performed mice tracking in behavioral experiments setups using the deep neural network model called You Only Look Once (YOLO). The accuracy of the YOLO model and its performance showed better results using the Tiny architecture in comparison with the Full architecture. The results obtained in this work, which include high rates of accuracy and fast computation, encourage experimentalists to conduct new experimental designs where real-time decisions can be made.

ACKNOWLEDGMENT

This work was supported by the School of Sciences and Technology at the Federal University of Rio Grande do Norte (ECT-UFRN).

REFERENCES


