Developing Machine Learning Methods For Rat Tracking in a Laboratory Setting
Ariel Feldman\textsuperscript{1,2,*} and Caleb Kemere\textsuperscript{3,4}

Abstract—We propose using a recurrent convolutional neural network to track long evans rats in various laboratory settings. Such a network would reduce the amount of time and human error involved in manually tracking, as well as account for occlusion of the rodent during experimental tasks, a feat which earlier attempts at rodent tracking through bifurcation were unable to perform. Accurate quantifications of time spent by the rat(s) in different areas or positions would thus be enabled with slight modification.

I . INTRODUCTION
Growing up, children learn to identify their surroundings beginning from a blank slate. Not yet having been exposed to the real world, or the language with which they are to discuss it, the learning process in infants is one easily recognizable to us all. Parents and teachers repeat words and point to corresponding objects, thus providing examples of the new words they are attempting to teach the child. Many problems seen in linguistic development, such as overextension—the tendency of a child to overgeneralize meanings, such as all furry, four-legged creatures to be cats—occur. With an increase in examples, however, the child will soon be able to distinguish between a cat and a dog.

It helps to consider the complex task of object detection in this manner. The same learning process and errors seen in the development of human object discrimination can be translated to neural network attempts at a solution. Image classification has long been an area of research within machine learning, with its own unique challenges as compared to other subfields. In recent years, interest in convolutional neural networks (CNNs) rose after Krizhevsky et al demonstrated high object detection performance on the ImageNet task, requiring a smaller number of connections and parameters to do so than other concurrent attempts utilizing other methodologies [5]. However, early applications tended to be computationally expensive. In addition, these implementations tended to apply unsupervised pre-training as compensation for a lack of sufficient data during training. The result was a relatively low mean average precision, when compared to later implementations, specifically the R-CNN.

II . NETWORK ARCHITECTURES
The network architecture used in our implementation is Faster R-CNN, for reasons explained later. However, detailed explanations of preceding and succeeding architectures are found below.

A. R-CNN
The Region-based Convolutional Network (R-CNN) method has since offered such improvement in object detection applications. As opposed to unsupervised pre-training followed by supervised fine-tuning [9], R-CNN implements supervised pre-training followed by domain-specific fine-tuning. Girshick et al. was able to demonstrate that such a manner is successful in training high-capacity CNNs when training data is lacking.

Instead of using a sliding-window approach for object localization, R-CNN generates about 2000 category independent regions proposals per input image. Linear support vector machines (SVMs) are then applied to classify a 4096-dimensional feature vector extracted from each of the proposed regions, thus allowing for more precise localization of objects within the images. With a feature matrix with dimensions of about 2000 x 4096, and an SVM with dimensions of 4096 x number of classes, indicating that R-CNN may be scaled to thousands of classes without relying on heuristic techniques (i.e. hashing). Additionally, this allows R-CNN to operate considerably faster than previous implementations of CNNs for object detection [2]. Figure 1 provides a visual representation of this methodology.

Once a region is labeled, the final step of R-CNN is to refine the classification to a more precise region using a linear regression model. Thus, R-CNN is able to discriminate objects relatively precisely. However, Girshick’s initial R-CNN exemplification takes considerable amounts of time, both during training and detection, as well as a large amount of memory. A forward pass of the CNN is required for each of the proposed regions in each frame inputted, which is rather inefficient. The proposed training takes place in multiple stages, which is far from ideal.
Since the training pipeline consists of 3 separate models to be trained—the CNN, the classifier and the regression model—the opportunity to make significant improvements in the training pipeline presents itself [1]. Girshick thus revisits his earlier work and improves it, aptly naming the newer implementation Fast R-CNN.

### B. FAST R-CNN

In an attempt to reduce the computational expense of training, Fast R-CNN introduces the concept of region of interest pooling (RoIPool). Simply put, the RoIPool greatly reduces the number times the CNN is run on an image from nearly 2000 times each to once per image. After running the CNN on an entire image, each region is assigned features from the image-specific feature map, based on the region’s location in the image [1]. This greatly reduces run time, as sharing the results of the CNN on an image to all regions within that image allows for accurate, fast classification of the possible objects within those regions, without needlessly running the CNN on overlapping regions that have already been assigned features, as was the case in the initial R-CNN instance.

Additionally, Fast R-CNN utilizes just 1 network during training—no longer are separate CNN, SVM and linear regression models required, but rather one CNN extracts image features, classifies regions and refines bounding boxes. Instead, the CNN takes in images with region proposals and produces a feature map. From said map, a feature vector is extracted and processed by several fully connected layers, eventually branching into 2 parallel output layers: a softmax layer, outputting classification, and a linear regression layer, encoding the coordinates of the bounding box in 4 digits [1]. *Figure 2* visualizes this process. Such a change greatly simplifies the network, as well as improving running time of the R-CNN implementation over the initial.

While Fast R-CNN makes considerable improvements in the training pipeline, the region proposal methodology—an application of selective search, which is notably slow—begs for improvement. Too speed up the overall process, this separate selective search algorithm was done away with in a new implementation.

### C. FASTER R-CNN

Perpetuating the abundance of creativity within the machine learning community, the next improvement of a Fast R-CNN instance was titled Faster R-CNN. In efforts to improve the efficiency of Fast R-CNN, Girshick once more revisited his previous work, this time with the intent to reduce the impact the region proposal process had on the overall runtime of Fast R-CNN.

Since the usefulness of proposed regions was determined by the features of the input image extracted through forward passes by the CNN, Girshick’s team tackles this issue by embedding the region proposition process within the CNN. Thus, regions would be proposed after the first step of classification at nearly no temporal cost [8]. The parameters of the CNN would be reduced to just the image, and still be able to perform accurate object detection.

Faster R-CNN is able to achieve such efficiency through a region proposal network (RPN). The RPN operates through a sliding window methodology—one that the original R-CNN had attempted to avoid. In this application of the sliding window technique, at each sliding window location, $k$ regions are proposed with varying scales and aspect ratios (parameters). In the paper, 3 scales and 3 aspect ratios were used, leading to 9 proposed regions per sliding window location. Of the proposed regions, each is scored based on likelihood that said region contains an image [8]. This, then, removes the time
sink that resulted from generating and classifying 2000 regions per image that was implemented in the initial R-CNN and Fast R-CNN.

To understand the importance of the aspect ratios, it may be useful think back to a child learning to discriminate objects for the first time. Among the most notable features of the object, perhaps the first encoded would be the relative size and aspect ratios of the object. With more and more examples of an object, the child would be able to build an expectation for the scale and shape of the object—thus ruling out objects with the wrong aspect ratios from falling under certain categories. The same can be said of the classifier in Faster R-CNN: during training, the classifier is able to associate certain aspect ratios with different classes, thus narrowing the possible classifications of each proposed region.

Faster R-CNN is essentially an implementation of Fast R-CNN combined with an RPN; the latter will generate region proposals, whilst the former will act as a detection network. However, since Fast R-CNN was implemented to depend upon fixed region proposals, training both Fast R-CNN and the RPN in parallel may conceivably cause failures in the performance of Faster R-CNN. To account for this possibility, convolutional layers are shared between the RPN and Fast R-CNN, and an algorithm for alternating optimization is applied for training purposes. This training is segmented into 4 parts. First, the RPN is trained on its own. Once the RPN is initialized, it is used to generate regions to train the detection network. At this point, the layers that are to be shared between the RPN and the detection network are fixed, and the layers specific to the RPN are fine-tuned. Finally, keeping the layers to be shared fixed, the fully connected layers specific to the detection network are fine-tuned [8]. In this way, the detection and region proposal networks are able to attain optimal performance while maintaining a consolidated network for object detection.

D. MASK R-CNN

Building upon earlier works, a new model entitled Mask R-CNN attempted to improve the precision of the earlier networks. Instead of drawing an imprecise bounding box, Mask R-CNN aims to identify the exact pixels belonging to an object in an image. However, realizing such a detailed discriminator proved more of a difficult task than simple modifications.

Rather, to attain such accuracy, yet another convolutional network was added to Faster R-CNN to determine whether or not a specific pixel belongs to an object. This network intakes the feature map generated by the RPN, and outputs a binary matrix corresponding to each pixel in the image, determining if each is a member of an object or not [4]. However, doing so introduces a new complication: pixel level specificity. As opposed to bounding boxes, image segmentation requires discrimination at the pixel level, whereas the RoIPooling introduced in Fast R-CNN does not achieve such precision.

As a solution, Mask R-CNN implements RoIAlign, which applies bilinear interpolation to extract a more accurate assessment of any given region in the original image. Essentially, to determine features at some pixel, linear interpolation will be performed twice, once in one direction, and then once in the other (i.e. x- and y-directions). In image processing, bilinear interpolation is typically used to map a pixel in an image onto a texture or feature map. In our case, this would be to determine a binary value of whether or not the pixel belongs to an object.

This allows us to infer what may be at, say, pixel 3.67 in the feature map, instead of rounding to what is at pixel 3, as was the case with RoIPool. Thus, RoIAlign allows for the pixel specificity required to
attain the precision Mask R-CNN sets out to achieve.

### III. BUILDING A DATASET

However, a network solution is only as good as its dataset—the larger and more diverse, the better. This allows the network to discriminate different objects in a wide range of scenarios, broadening its application immensely. For our purposes, LabelImg was used to draw ground truth bounding boxes. An example of this is depicted in Figure 5. LabelImg generates an xml file containing all of the relevant information regarding the locations of each object in the image and their corresponding labels. Initially, 6 classes were labelled: human, object, median, foraging rat, interacting rat, and rearing rat. A training set of just 420 images, and a validation set of 108 images, were more than sufficient to train a fairly accurate 6 class object detector.

Images in the training set were used from 2 videos in different contexts and with different items. As stated above, it is imperative there is diversity amongst images in the dataset—thus, images in both the training and validation sets contained a mixture of clear and blurry, occluded and not occluded images, as well as images that contained anywhere from 2-6 classes.

### IV. METHODOLOGY

We configured Tensorflow’s Object Detection API to create an instance of Faster R-CNN for rodent tracking purposes. Though Faster R-CNN detects slightly slower than other models in the model zoo, such as the Single Shot Multi-box Detector (SSD), it tends to attain higher accuracy in detection. Our priority was accuracy, and thus Faster R-CNN was chosen; however, it may be interesting to see how the SSD performs in terms of a trade-off between accuracy and speed.

After exploration of and experimentation with the network architectures detailed in (Ⅱ), it was determined Faster R-CNN achieved the best
combination of accuracy and training time. Training the Faster R-CNN implementation was temporally costly enough, requiring about 16 hours to complete a full training session. While Mask R-CNN distinguished itself in terms of precision, the added temporal cost of RoIAlign would be substantial, and the improvement in precision would be unnecessary for our purposes at the moment.

V. DISCUSSION AND FUTURE STUDY

Though our model of Faster R-CNN performs with rather high accuracy on videos not in the dataset, there are still some discrepancies requiring attention. For example, in the video from which Figure 8 was extracted, from about 0:11-0:12, when the rat is sniffing the object, the model switches between “foraging rat” and “interacting rat” 4 times. This can likely be corrected with small amounts of smoothening, as this is not a terribly common occurrence, and the correct classification is made for many of the frames. Thus, experimenting with different methods of smoothening classifications should be a next step in developing a rat tracking model.

The trade-off between accuracy and temporal cost with an implementation of Mask R-CNN begs exploration. The dataset, however, would need to be relabeled, as Mask R-CNN requires more specificity within training and validation sets than Faster R-CNN. Preferably, we would have trained both models and compared them, but due to the fact that the two datasets would have had to be labeled in different manners, and our interest was getting something functioning more quickly, it was not feasible. However, it would be useful to see how the two models perform in a head tracking task, as such a task would greatly benefit from a model known for its increased specificity.

Another model worth exploring would be the model proposed by Redmon et al., You Only Look Once (YOLO). YOLO achieves accurate object detection at a much faster rate than its competitors. In fact, its main claim of superiority over Fast and Faster R-CNN models is that it can detect objects in real time, a feat all of the models detailed earlier in this report fail to accomplish [8]. Real time detection would be extremely useful, in our case. The time required to analyze video data from an experiment would be significantly reduced, as it would be able to be performed as the experiment occurs.

As perfect as it sounds, YOLO is not without its flaws. The loss function YOLO implements to quantify accuracy in object detection does not consider the size of a bounding box when determining error. This can become an issue when considering a small error in a small vs. large bounding box. An error of any size would have a larger impact on the accuracy of a smaller bounding box than of a larger one, regardless of the size of the error. The issue begins in cases where the error is of a size that is inconsequential to a larger bounding box, yet has a great impact on the accuracy of a smaller one [8]. Localization by YOLO, thus, is not nearly as good as other models (see Figure 9 for a comparison with Fast R-CNN). Thus, YOLO introduces concern in terms of quantification of performance, as well as specificity. If we are to introduce head tracking, which requires more specificity than tracking the entire rat body, YOLO may not perform as well.

Due to the spatial constraints of YOLO’s bounding box predictions, the model struggles to predict small objects that appear in large groups, as well as objects that appear in unexpected sizes and aspect ratios [8]. Though this is unlikely to cause issue in our use case, it does cause some concern. If a rat is to knock over an object in an experiment, for example, would YOLO still be able to recognize it? It is a question worth exploring, as YOLO otherwise provides a promising solution to rodent tracking.

When compared to Fast R-CNN on the PASCAL dataset, Fast R-CNN produces an overall better performance. YOLO achieved a much lower error rate than Fast R-CNN in background detection, though Fast R-CNN outperformed YOLO on this specific task in each of the other categories (see Figure 9). Redmon et al. thus combined YOLO and Fast R-CNN to achieve higher accuracy. By combining the top Fast R-CNN model with YOLO, they were able to increase mAP from 71.8% to 75% on VOC 2007, a significant increase in accuracy unachieved by other models.
A feature worth looking into is user selection of a region of interest. Some commercial video tracking applications implement such specificity, but we have not yet added this to user capabilities. As I noted before, head direction seems to be an extremely useful feature worth adding. Quantification of time spent with each object, additionally, appears to have numerous applications in a laboratory setting, and thus should be realized in future advancements.

ACKNOWLEDGEMENT

This work was supported by an NSF BRAIN EAGER IOS-1550994, as well as the kindness of Joshua Chua and Amanda Wickens to share their video data for training, as well as James Webb for lending his GPUs and Shayok Dutta for helping me learn to SSH and use Colaboratory, as I traveled a bit through the semester. No rodents were injured in the making of this network :)

REFERENCES
