A Survey of Deep CNNs for Mouse Paw Location

Konstantin Kaganovsky
Stanford University
1050 Arastradero Rd., Stanford, CA
kkaganov[at]stanford.edu

Molly Lucas
Stanford University
401 Quarry Rd., Stanford, CA
mlucas9[at]stanford.edu

Abstract

In order to study how brain activity is linked to motion, researchers often record neurons while an animal, such as a mouse, moves. These recordings of mouse movement must be coded (marking foot location in each image frame), which is an onerous task prone to human error. Here, we tested four convolutional neural networks used for pose estimation and object detection in order to find which performed best at locating the mouse's left front paw. We tested DeepLabCut, the field standard, as well as a deeper version of this network (DeepLabCut using ResNet101). Additionally, we attempted to achieve DeepLabCut's performance with a faster object detection network, Faster R-CNN, as well as a hybrid between the two: Faster R-CNN using ResNet50. We found that DeepLabCut with ResNet101 had the smallest root mean squared error (2.9650 pixels), which is slightly worse than human expert performance (2.7 pixels) - a proxy for Bayes error. Faster R-CNN was significantly faster than DeepLabCut but with a much higher error rate.

1. Introduction

The neural basis of motion is not yet fully understood. In experiments with animal models, such as mice, researchers can record neural activity while a mouse moves on a treadmill. Historically, in order to link brain and behavioral data, experimenters manually label behavioral data (which is labor intensive and prone to human error). Here, we tested different convolutional neural network (CNN) architectures to detect a mouse's paw location. The inputs to our algorithm are grey-scale images of mice running on a ball. The output is an X,Y coordinate marking the left, front paw's location. We evaluate model performance by calculating root mean squared error (RMSE) of the predicted X,Y coordinate to the true (expert hand-labeled) X,Y coordinate. We tested two main model types: DeepLabCut (state-of-the-art for this task) and Faster R-CNN. While DeepLabCut is known to have high accuracy, the computational cost is very high (for both time and resources). We selected Faster R-CNN as a second model, as it is known to be much faster. DeepLabCut is very new (2018), so our aim was to see if we could improve the model accuracy through modifying the architecture and hyperparameters. With Faster R-CNN, our aim was to try to achieve DeepLabCut's accuracy but with shorter training/test time using a smaller GPU.

2. Related Work

DeepLabCut. In 2015, a research group used a Support Vector Machine classifier to detect paw location with a sliding detection window [1]; however, their model is specific for black mice on a white background, the images must have a specific resolution, and extensive preprocessing of the image is necessary. Critically, it is not taking advantage of recent developments in convolutional neural networks for image detection. Also in 2015, another group used a random forest classifier to detect mouse posture [2]; however, extensive segmentation, background subtraction, and alignment were necessary - and their classification was manually supervised. So far, the state-of-the-art for individual limb tracking is DeepLabCut [3] - a simplified pose estimation model adapted from [4]. DeepLabCut (DLC) was just published 08/2018 in *Nature Neuroscience* and made a huge impact on the field. DLC uses the feature detector portion of the best performing multi-person human pose estimation model (DeeperCut) [4]. DeeperCut uses the ResNet architecture but then feeds its output into a model that uses pairwise angles between joints as a constraint for learning. Finally, this model uses Integer Linear Programming with incremental optimization to split the detected body parts into individual humans (if multiple subjects are present in one image). While this model achieves very high accuracy, it is too complicated for simple limb tracking of one research subject, and the publicly-available labeled datasets for human pose estimation are large (~25k examples) - unattainable for a standard neuroscience research group.

Faster R-CNN. There has been a major push for image recognition and object detection algorithms to increase processing speed in recent years. Fast R-CNN [5] is an object detection algorithm that trains significantly faster than its predecessors. It uses an initial series of convolutional layers with a max pooling layer to create convolutional feature maps. A max pooling layer is used on each proposed region of interest. A fully connected layer is then fed into a softmax function, and object classification is performed. Multi-task loss is used, which simultaneously trains for object detection (classification) and regression of bounding box location. Faster R-CNN [6] is an object detection network that further increases the speed of Fast R-CNN by adding a novel Region Proposal Network (RPN) while maintaining the original convolutional layers for Fast R-CNN. This RPN is a fully convolutional deep network combining bounding box prediction with object scores, which eliminates the main computational bottleneck of Fast R-CNN, dramatically increasing the object detection speed without negatively impacting performance. This algorithm can be fine-tuned to detect specific object classes of interest.

3. Dataset and Features

Our images were obtained from a Stanford neuroscience research laboratory; they used a high-resolution camera under infrared light illumination at 150 frames per second with a resolution of 800x600 pixels (Fig1). The position of the mouse in the camera's field of view is relatively constant from one mouse to the next, and the illumination conditions are consistent. The data were not pre-processed or augmented as our error rate is sufficiently low. The data were labeled by a single researcher in the lab instructed to label a specific part of the paw. In our 2D object detection/pose estimation models, each frame is treated as a separate image. We have 973 images in the training set (60%), 325 in the development set (20%), and 325 in the test set (20%). The test set was saved until model hyperparameters were finalized. We only used labels for the front left paw. The same training, development, and test images were used in each independently tested model to allow for a fair comparison.

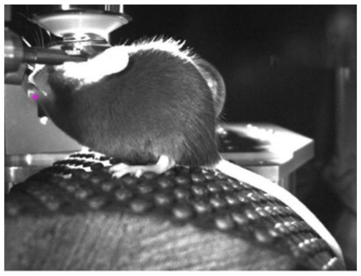
Figure 1: Example Labeled Data Sample

4. Methods

DeepLabCut. DLC uses a ResNet-50 [7] architecture (Table 1) with stride 8, "same" padding, and modifications as follows (Fig 2) - holes were added to the 3x3 convolutions in conv5 to increase receptive field size, the stride of conv5 was decreased to 1 pixel to avoid down-sampling, the final classification and average pooling layers were replaced with deconvolutional layers to up-sample the final output

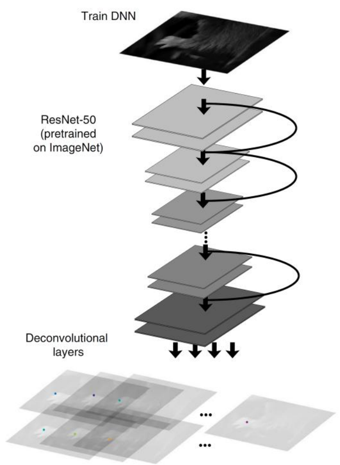

Figure 2: Simplified DeepLabCut Architecture, adapted from [3].

back to a similar size as the input image. This gave us a "score map" of limb probabilities in each pixel of the original image. The pixel with maximum probability was selected as the predicted limb location. As mentioned in lecture, there is generally more data available for object classification tasks than for detection tasks, so this network was pretrained on the ImageNet dataset. Sigmoid activations were used, all model weights were fine-tuned on the training set, and optimized with Stochastic Gradient Descent (SGD, batch size of 1 to allow for different sized inputs). The standard cross-entropy loss function (Eq. 1) was used to predict pixel-wise class probability and finally a Huber loss (Eq. 2, δ=1) was used to regress predicted location to actual location – a form of location refinement for the final coordinate. These losses were combined and optimized with SGD (Huber loss was weighed by 0.05, the optimal weight in [3]) Because the training data are labeled as a single X,Y coordinate, DLC dilates this positive labeled pixel to a radius of 17 pixels (these are all considered "positive" pixels). DLC was implemented in python [10-14] on 2 Ubuntu computers with a single NVIDIA GTX 2080 8GB GPU. Training ran for 12-36 hours, but inference on new images takes a few minutes.

$$\mathcal{L}(\hat{y}, y) = -(y \log \hat{y} + (1-y)\log(1-\hat{y}))$$

Equation 1: Cross Entropy Loss

$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for} |y - f(x)| \le \delta, \\ \delta\, |y - f(x)| - \frac{1}{2}\delta^2 & \text{otherwise.} \end{cases}$$

Equation 2: Huber Loss for Regression: note f(x) denotes predicted values

DeepLabCut with ResNet101. We also experimented with a deeper ResNet architecture; we replaced the ResNet-50 model described above with ResNet-101 (now the conv4_x block is repeated 23 times [7]). This ResNet model with 101 layers was also pre-trained on the ImageNet dataset, and we implemented the same modifications as above to the output layer in order to obtain a "score map" of limb probabilities. Because this is a deeper network, we decided to experiment with adding intermediate supervision. The intermediate supervision was implemented akin to GoogleLeNet [8]. In DLC, an intermediate detection/classification layer was added to the output of the largest conv block (conv4) of ResNet-101; in order to implement this, the network de-convolved the feature map and predicted with a sigmoid activation function similar to the final output. Then this prediction is subject to cross-entropy loss (Eq.1) and added to total loss described in Methods and subject to SGD. This helps ensure that earlier parts of the network are learning feature maps that have discriminate power for the final object detection. This additional part of the network is discarded during inference on new data.

Faster R-CNN. We used an input layer size of 32x32x3. There were two convolutional layers, each with thirty-two 3x3x3 convolutions with stride 1 and padding 1. Each of these convolutional layers was followed by a ReLU activation layer. Following this, max pooling was used (3x3, stride 2, padding 1). A fully connected layer (64) then fed into a third ReLU layer. The last fully connected layer (output 2, since only identifying one class) was followed by softmax and cross-entropy loss was calculated. Loss function uses binary classification evaluated by measuring intersection over union of the proposed bounding box and the true label bounding box (Eq. 3). This model was pretrained on PASCAL VOC. It is trained using SGD using mini-batch size 1 for 5 epochs. Faster R-CNN was implemented in Matlab [9].

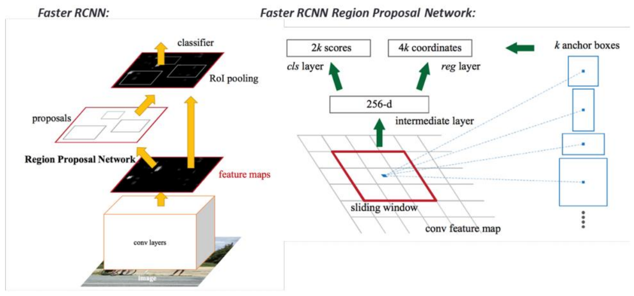

Figure 1. (Left) Faster R-CNN architecture. (Right) Region Proposal Network for Faster R-CNN. Both images adapted from Ren et al. 2016 CVPR.

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i,p_i^*) + \lambda\frac{1}{N_{reg}}\sum_i p_i^* L_{reg}(t_i,t_i^*).$$

Equation 3. Faster R-CNN Loss Function.

Faster R-CNN with ResNet50. We exchanged the convolutional layers of Faster R-CNN with a Residual Network (50-layers) pretrained on PASCAL VOC [7]. This uses 5 blocks of convolutional layers (conv1-conv5) followed by an average pool layer, a fully connected layer, and softmax layer for classification (for details, see Table 1). This architecture has been used effectively for image recognition tasks. Importantly, ResNet50 architecture is used in DeepLabCut, suggesting it is an appropriate model for this task. We used the same Regional Proposal Network and output layers (fully connected layer, classification, loss function) as in our Faster R-CNN model. This model was designed and run using Matlab [9].

Layer Name	Layer Details
conv1	7x7, 64, stride 2
conv2_x	3x3 max pool, stride 2 1x1, 512 3x3, 512 x3 1x1, 2048
conv3_x	1x1, 512 3x3, 512 x4 1x1, 2048
conv4_x	1x1, 512 3x3, 512 x6 1x1, 2048
conv5_x	1x1, 512 3x3, 512 x3 1x1, 2048
Final layers	Avg. pool, fc, softmax
FLOPs	$3.8\text{x}10^9$

Table 1. ResNet50 Layers (Adapted from He et al. 2016 CVPR). ResNet50 replaced the convolutional network in standard Faster R-CNN (Figure 3).

5. Experiments/Results/Discussion

We used root mean square error (RMSE) between predicted and true X,Y coordinate of left foot position as a single number evaluation metric to tune the hyperparameters, as this is the field standard for testing this type of model [3]. Human performance on this task is a good estimate of Bayes error and has been reported as RMSE of 2.7 pixels [3]. For perspective, the mouse paw has an area of about 1000-2000 pixels^2, and the human-labeled X,Y coordinate marks the outermost front edge of the foot.

Experiment 1: Hyperparameter tuning of DeepLabCut (with ResNet). Experiments began by varying the learning rate - we began with the published parameters: $5\text{x}10^{-3}$ for 10k iterations, $2\text{x}10^{-2}$ for 420k iterations, and $2\text{x}10^{-3}$ for 70k iterations for a total of 500k iterations. SGD converged to a similar cross-entropy loss for all learning rates within 500k iterations (Fig4). We varied the learning rate as shown in Table 2 and found a better solution than [3] based on RMSE of the development set. To further optimize the model and training time, we varied the size of the input images with the following scaling factors: (0.5,**0.8**,1.0,1.2,2.0) and obtained development set (dev) RMSE of: (5.0864, **4.4437**, 5.6170, 4.9365, 15.4678). The scaling factor of 0.8 was the best, but there is surprisingly no major difference from 0.5 to 1.2 scaling - indicating that at our resolution, detecting a simple white paw on a black background does not require a full-scale image. If training speed needs to be optimized then 0.5 may be best, but we continued with 0.8 as our goal was minimizing RMSE.

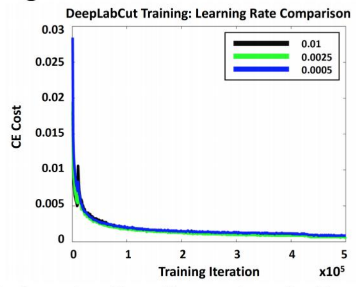

Figure 2: Learning Rate Comparison for DeepLabCut

Learning rate	Training RMSE	Dev. RMSE
0.0005,0.002,0.0002	1.285	6.1486
0.0025,0.01,0.001	1.0097	4.4437
0.005,0.02,0.002	0.9204	5.41
0.01,0.04,0.004	0.9081	7.3286

Table 2: DeepLabCut: Effect of learning rate on train and dev RMSE

Experiment 2: Testing a deeper network for DeepLabCut (with ResNet101). We then switched the CNN to a deeper ResNet-101 architecture, but did not observe a dramatic decrease in dev RMSE (Train RMSE: 0.6251, dev RMSE: 4.9553). Based on the difference between train and dev RMSE, the larger network is overfitting the training data, so increasing training data may help overcome this problem and further decrease the dev RMSE to below the standard DLC value. Further, adding intermediate supervision halfway through the network did not improve performance, but actually increased the overfitting problem (Train RMSE: 0.567, dev RMSE: 9.5173). It is possible that adding more training data would help reduce this high variance problem. However, the fact the train RMSE decreased is surprising, we expected the residual connections learned by ResNet and the intermediate loss function to almost be redundant – the Deep Learning community uses both to decrease the effect of vanishing gradients. However, this shows intermediate supervision can further improve performance given enough data.

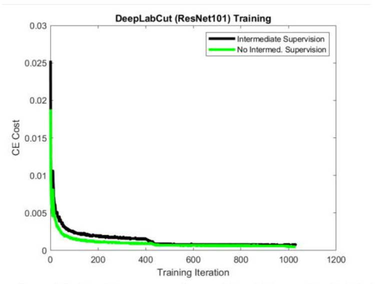

Figure 3: Learning Rate Comparison for DeepLabCut with ResNet 101

DeepLabCut ResNet101	Training RMSE	Dev. RMSE
No Intermed. Supervision	0.6251	4.9553
Intermediate Supervision	0.567	9.51730

Table 3: DeepLabCut with ResNet101: Effect of intermediate supervision on train and dev RMSE

Experiment 3: Faster R-CNN. While DeepLabCut is the field standard, it requires significant time and computational resources to train. We tested Faster R-CNN to see whether it could perform similarly in terms of accuracy. A main difference between DeepLabCut and Faster R-CNN is that DeepLabCut uses labels of X,Y coordinates, and Faster R-CNN uses bounding boxes (providing object classification within the box). Our first series of tests resulted in 0% accuracy (high bias). We completed an error analysis (looking at 100 incorrectly labeled images) and found that the model was labeling light/dark edges. We tested a series of bounding box sizes and eventually chose a box of 50x50 pixels and also centering it around the foot (shifting the X,Y label by 25 pixels up/left), which had the lowest dev set RMSE (80 pixels). From here, we trained this model using 5 different learning rates (Figure 6, Table 4). Looking at RMSE on the development set, learning rate of 0.0005 had the lowest RMSE (82.9 pixels). Variance across these tests was quite low (minimal difference between training and dev RMSE), but bias was high even with our best bounding box labels and learning rates. Therefore, in our next experiment, we changed the model architecture to resemble DeepLabCut more closely.

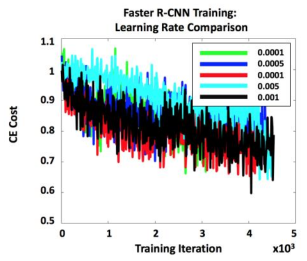

Figure 4. Learning rate comparison for Faster R-CNN. Learning rate of 0.0005 resulted in the lowest RMSE.

Learning rate	Training RMSE	Dev. RMSE
0.001	80.1	86.7
0.005	103.5	107.5
0.0001	80.1	83.6
0.0005	77.3	82.9
0.00001	174.3	166.6

Table 4. Learning rate comparison for Faster R-CNN. Learning rate of 0.0005 resulted in the lowest RMSE (82.9 pixels).

Experiment 4: Creating a hybrid between Faster R-CNN and DeepLabCut.

Here, we used a ResNet50 architecture integrated with Faster R-CNN's Regional Proposal Network to try to reduce bias in the Faster R-CNN model. We used the same optimal bounding box and X,Y label positions found in Experiment 3. We trained the model using 4 learning rates (Figure 7, Table 5) and found that learning rate of 0.005 had the lowest development RMSE (22.9 pixels).This model showed much less bias than the standard Faster R-CNN. There was still a fairly low variance (bias remains the predominant issue). For error analysis, the model consistently chose either the correct foot, a different foot, or the head (the three bright locations). However, almost all images labeled the correct foot. At this point, the margin of error is likely due to the way we systematically created bounding box labels from X,Y coordinates (instead of manually assigning bounding box labels). For future, we would go back and hand-label the test data set to minimize this error.

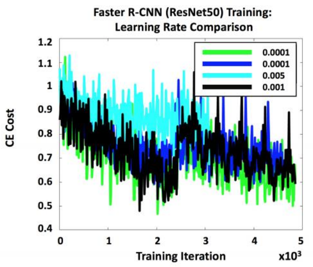

Figure 5. Learning rate comparison for Faster R-CNN with ResNet50. Learning rate of 0.005 resulted in the lowest RMSE.

Learning rate	Training RMSE	Dev. RMSE
0.001	31.8	32.4
0.005	20.3	22.9
0.0001	29.0	32.5
0.00001	39.7	42.7

Table 5. Learning rate comparison for Faster R-CNN with ResNet50. Learning rate of 0.005 resulted in the lowest RMSE (22.9 pixels).

Final Comparison across 4 models.
For our final comparison (Table 6), we used the satisficing metric of RMSE < 10 pixels and our optimizing metric was computational speed. DeepLabCut and DeepLabCut101 were the only model to reach our satisficing metric with a test RMSE of 2.965 pixels. This is only slightly worse than human performance (our estimate of Bayes error) at 2.7 pixels. Faster R-CNN using ResNet50 was the third-best performing model with a test RMSE of 19.9 pixels, and the computational speed was much faster than DeepLabCut (1.9 hrs to train vs 24 hrs to train DeepLabCut) even using a much weaker GPU. Upon inspecting the labeled frames, DeepLabCut located the proper paw in 100% of the frames! Indeed, human labels seemed to be more variable than the predicted paw label. Impressively, DLC detected the paw even when it was hard for us to detect the paw without looking at the previous frame and the next frame (Fig1,pink spot).

Model	Training RMSE	Dev. RMSE	Test RMSE	Average time to run	Hardware (GPU)
DeepLabCut	**1.0097**	4.4437	3.1686	24hr train 3 min test	GeForce GTX 2080
DeepLabCut (ResNet101)	0.6251	4.9553	2.9650	36hr train 3 min test	GeForce GTX 2080
Faster R-CNN	77.3	82.9	81.3	0.8 hrs training 6 min. testing	GeForce GTX 1050
Faster R-CNN (ResNet50)	20.3	22.9	19.9	1.9 hrs training 20 min. testing	GeForce GTX 1050

Table 6. Final comparison across four models. DeepLabCut had the lowest test RMSE (2.9650 pixels). Faster R-CNN had the lowest training time (0.8 hours).

6. Conclusions/Future Work

We were surprised to find that Faster R-CNN with ResNet50 was able to perform reasonably well with dramatically decreased training time and GPU resource needs. Future work would continue to hone this model to see if we can improve accuracy to the level of DeepLabCut while still maintaining the low resources of Faster R-CNN. This would make this tool more accessible to researchers with more limited computational resources. Using the ResNet50 layers improved the performance of Faster R-CNN significantly and its possible a deeper network would help more.

DeepLabCut performed the best, reaching almost human expert level performance. In previous iterations of DeepLabCut (not shown here) and in the RCNN models, we observed the model predicting the wrong paw when the front left paw was not visible or close to the front right, for example. Future work could integrate multi-target tracking algorithms to add priors about the possible sequence that a mouse limb can take - for example, the loss function can incorporate information about the previous predicted location and "punish" large changes from one frame to another. This would be reasonable at fast frame rates and would decreasing the likelihood that models would classify a random bright spot (or the wrong paw) as the target in select frames. Alternatively, a 3D version of this model could be implemented to include temporal information in detection. Overall, CNNs show promise for mouse paw localization and can already perform close to human level.

7. Contributions

KK ran the DeepLabCut and DeepLabCut ResNet101 experiments. ML conducted the Faster R-CNN and Faster R-CNN with ResNet50 experiments. Both authors discussed model bias and variance, training and development set results, and error analysis throughout the quarter to inform further decisions.

8. References

1. A. S. Machado, D. M. Darmohray, J. Fayad, H. G. Marques, M. R. Carey, A quantitative framework for whole-body coordination reveals specific deficits in freely walking ataxic mice. *eLife*. **4** (2015), doi:10.7554/eLife.07892.
2. A. B. Wiltschko *et al.*, Mapping Sub-Second Structure in Mouse Behavior. *Neuron*. **88**, 1121–1135 (2015).
3. A. Mathis *et al.*, DeepLabCut: markerless pose estimation of [user-defined body parts with deep learning. *Nat. Neurosci.* **21**, 1281 (2018).
4. E. Insafutdinov, L. Pishchulin, B. Andres, M. Andriluka, B. Schiele, in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, M. Welling, Eds. (Springer International Publishing, 2016), *Lecture Notes in Computer Science*, pp. 34–50.
5. R. Girshick. Fast R-CNN. In *ICCV,* 2015.
6. S. Ren, K. He, R. Girshick, J. Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *NIPS*, 2015.
7. K. He, X. Zhang, S. Ren, J. Sun, Deep Residual Learning for Image Recognition. In 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (IEEE, Las Vegas, NV, USA, 2016; http://ieeexplore.ieee.org/document/7780459/), pp. 770–778.
8. C. Szegedy *et al.*, Going Deeper with Convolutions. *arXiv:1409.4842 [cs]* (2014) (available at http://arxiv.org/abs/1409.4842).
9. MATLAB and Deep Learning Toolbox Release 2018b, The MathWorks, Inc., Natick, Massachusetts, US.
10. OpenCV. *Open source computer vision library*. https://github.com/itseez/opencv (2015)
11. Oliphant, T. E. Python for scientific computing. *Computing in Science & Engineering* 9 (2007).
12. Abadi, M. et al. Tensorflow: a system for large-scale machine learning. In OSDI, vol. 16, 265–283 (2016)
13. Hunter, J. D. Matplotlib: A 2d graphics environment. *Computing in science & engineering* 9, 90–95 (2007).
14. McKinney, W. pandas: a foundational python library for data analysis and statistics. *Python for High Performance and Scientific Computing* 1–9 (2011).