Deep Recurrent and Convolutional Neural Networks for Automated Behavior Classification

Zachary Nado

Advisor: Erik Sudderth, Reader: Thomas Serre

Abstract

In this thesis we investigate different methods of automating behavioral analysis in animal videos using shape- and motion-based models, with a focus on classifying large datasets of rodent footage. In order to leverage the recent advances in deep learning techniques a massive number of training samples is required, which has lead to the development of a data transfer pipeline to gather footage from multiple video sources and a custom-built web-based video annotation tool to create annotation datasets. Finally we develop and compare new deep convolutional and recurrent-convolutional neural network architectures that outperform existing systems.

1 Introduction

As the digitalization of society increases so does the amount of data available and the need to process it. This is best seen in the scientific community with studies that have grown in scale to where it would be unreasonable for humans alone to analyze all of the information produced. As computer vision techniques improve in accuracy and efficiency scientists can use them more often in their research, including autism diagnosis [1], medical diagnosis [24], and animal behavioral analysis [4]. These are all examples of areas where computer vision is improving the productivity of scientific study through the automation of classification.

This thesis focuses on behavioral analysis, a process that is a bottleneck in many areas of science but especially in biomedical research. Until recently humans were vital to the data gathering process. They would typically watch many hours of footage or comb through many pages of records, an error prone and laborious process that was still central to the experiments. An example of this bottleneck is that it currently takes twenty five minutes of human analysis to analyze one minute of mouse behaviors. With the experiments in the lab there are several dozen streams of continuous footage being recorded, generating far too much data for humans to process in a reasonable time. In contrast the models developed in this paper perform at human level and take approximately twelve seconds to process one minute of video, indicating there is great opportunity for automation in this area.

To truly leverage the power of new deep learning models one requires very large training datasets, such as the popular ImageNet database which contains over fourteen million images in twenty one thousand categories [21]. Unfortunately there do not exist any tools that are effective at annotating behaviors in videos; most video annotation interfaces [22] are focused on object classification, which make them ill-suited for the wide range of interactions and experiment setups that behavioral analysis requires. Thus in order to gather enough data to train deep models a web based annotation system was also developed that lets users from all over the world stream videos and detail what is happening on a frame by frame basis. While still much slower than a computer model, this tool has noticeably improved the speed at which human annotators can process videos.

2 Related Work

Animal behavioral analysis is being used more frequently for varying fields of research and the need for automating its monitoring and analysis is growing rapidly. While it is useful for many fields

of science including ecology, neuroscience, psychology, and pharmacology it is only recently that computer vision techniques have been developed to create truly robust systems.

Many animal species are commonly used in experiments, such as rodents which are very popular for modeling neurological functions in mammals [11] and zebrafish [2] which are often involved in pharmacology and genetic research [5]. Previously these studies required special hardware in order to record animal activity [7] or relied upon human observation. While there has been work in automating these practices with computer vision [8] often times the algorithms are very problem dependent ([4], [19]) and susceptible to failure from small changes in the experimental setup. For example, the current classification system [16] relies on background subtraction methods that many times must be tuned to the specific environment.

The work in this paper hopes to move beyond the many experiment-based restrictions and complications of current behavioral analysis systems by using a deep learning approach. By using a system that can adapt to the study's environment, it should not have the current difficulties of adapting to experimental fluctuations or, given new data, generalizing to new types of experiments. Instead of relying on multiple stages of analysis such as background subtraction, feature extraction, and classification it will be able to learn a single end-to-end classification system; thus in addition to increasing the robustness of setups it will also be easier for scientists that do not have as much knowledge about computer vision to feed their data into a deep learning blackbox rather than develop a novel pipeline for each of their studies.

3 Background

3.1 Neural Networks

Multi-layer perceptrons, more generally known as neural networks, go as far back as the 1960s [20] yet in recent years they have fueled an explosion of advances in the field of computer vision. Increases in GPU computing power to new neural network techniques have pushed these models to consistently do very well in large scale computer vision competitions for the last several years [21]. Composed of layers of neurons much like in the mammalian brain, it has been shown [14] that under certain conditions neural networks can approximate any function in $\mathbb{R}^n$ with arbitrarily accuracy.

The basic unit of a neural network is the neuron, represented by an activation function. Common choices for activation functions are the sigmoid function $\frac{1}{1+e^{-x}}$, the hyperbolic tangent function, and the rectifier function $max\,(0, x)$. Traditionally a layer in a neural network is thought of as a group of neurons, with layers stacked on top of each other where the bottommost layer is the raw data and the topmost layer is the output of the model. Connections between neurons are then represented by which activation functions in layer $L-1$ are used as inputs to activation functions in layer L. The input to the i^{th} neuron in layer L is then

$$x_L^i = f\left(W x_{L-1} + b\right)$$

where x_{L-1} is the concatenated outputs of all neurons in layer $L-1$ that connect to the i^{th} neuron and f is the chosen activation function. The matrix W is the weight matrix and b is a vector of biases unique to each neuron; the values of these structures are what the network learns in order to minimize the output error. Starting at the last layer in the network and using the chain rule from calculus, adjustments to each of these can be calculated to lower the model's error through a technique called backpropogation.

3.1.1 Convolutional Neural Networks

Convolutional neural networks (CNNs) are a special kind of neural network used for image processing and other spatial domain tasks. Instead of being arranged in rows, the convolutional layers in a network are arranged in an n-dimensional grid that convolves over the input. Each convolutional layer only has one weight matrix and bias vector shared among all the neurons, which means that the model learns weights and biases that are effective at detecting features in regions of the input; depending on how the input is arranged, these regions can be organized in space, time, or a problem-specific dimension. Often times these layers are combined with pooling layers which take the minimum or maximum value among several neurons, which in the example of image processing allows for spatial invariance of image features.

It is not uncommon for Gabor filters to be used as initial values for the weights of convolutional layers. Gabor filters are biologically inspired tensors that act as edge detectors when convolved over

an input. They are used in the HMAX system (Hierarchical Model and X, [23]), a special case of a CNN that does not involve backpropogation. The structure of HMAX was inspired by the visual cortex in the brain, and even though it is a feedforward-only network it is still powerful at classifying images.

3.1.2 Recurrent Neural Networks

Recurrent neural networks (RNNs) are another type of neural network architecture that have seen many exciting developments in recent years ([25], [6], [9]). Traditional networks treat subsequent samples as independent of others, but RNNs allow for a hidden state to be maintained inside the recurrent neurons that can be altered to affect future outputs. A special type of RNN unit, called a long-short term memory (LSTM, [13]) unit, has several mechanisms that can be learned to control this hidden state. By optimizing when to allow the hidden state to change, these units can learn to control how long to keep information for between inputs. The ability to learn when to retain and forget information turns out to be quite powerful in detecting patterns in sequential data, and can even be used as a generative model [10].

A technique called backpropogation through time is used to train RNNs. It works by taking the single neuron and "unrolling" it in time, making a layer for each time step that consist of the same neuron but with its hidden state at that step. This can substantially increase the depth of the network, which can lead to problems such as the unstable gradient problem [12]. Because of the multiplicative nature of the chain rule used to backpropogate the changes in weights and biases to layers in the network, small or large differences can explode or vanish exponentially fast. While LSTM units are much better at stabilizing these weight and bias changes than traditional RNNs, issues still arise in practice.

3.2 Current Behavior Classification System

The current classification system [16] was published with a 78.3% accuracy on eight classes on the expert annotated data, yet more recent improvements have increased this to approximately 80%. The model can be broken into three parts: image preprocessing, motion feature generation, and the classifier which is an SVMHMM implementation.

The first thing the system does with a new mouse video is perform background subtraction on all frames. This identifies the mouse pixels versus the cage so that the position, velocity, and acceleration vectors of the mouse can be computed, which is referred to as *position features* in this paper. Unfortunately this background subtraction is susceptible to failing if elements of the cage are slightly altered such as lighting or cage position; this has caused failures in mouse identification and often requires manual tuning.

The spatial-temporal feature maps are computed using the S1 and C1 layers of an HMAX implementation. In the S1 layer of the system there are four orientations of Gabor filters shifted over a window of nine frames to compute four directions of motion features, followed by max pooling in the C1 layer. In practice the frames are cropped to a tighter subwindow centered around the mouse to remove unnecessary information and speed up computation time. Using these croppings the S2 features are then computed. The S2 weights were developed by randomly cropping high motion pixels from human chosen times of interest in the videos and then performing feature selection to get approximately 300 features.

In addition to the temporal features generated by HMAX, the current system uses an SVMHMM model that adds another time dependence between video frames. The position features and motion maps from HMAX are then combined into feature vectors and used as input to the SVMHMM classifier. The output of this classifier can then be analyzed to determine behavioral patterns and compute ethograms of the mouse's activities.

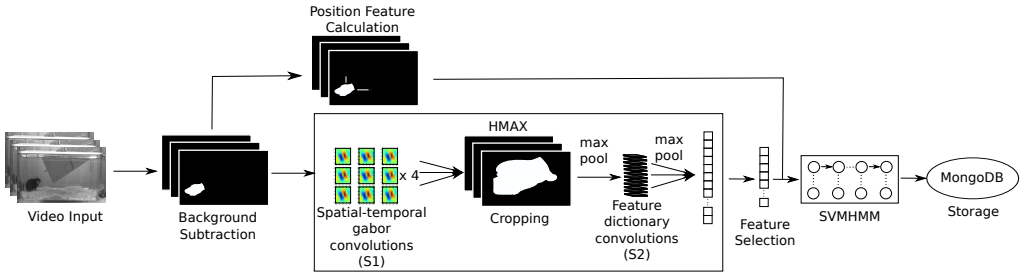


Figure 1: A visualization of the current classification pipeline.

4 Methods

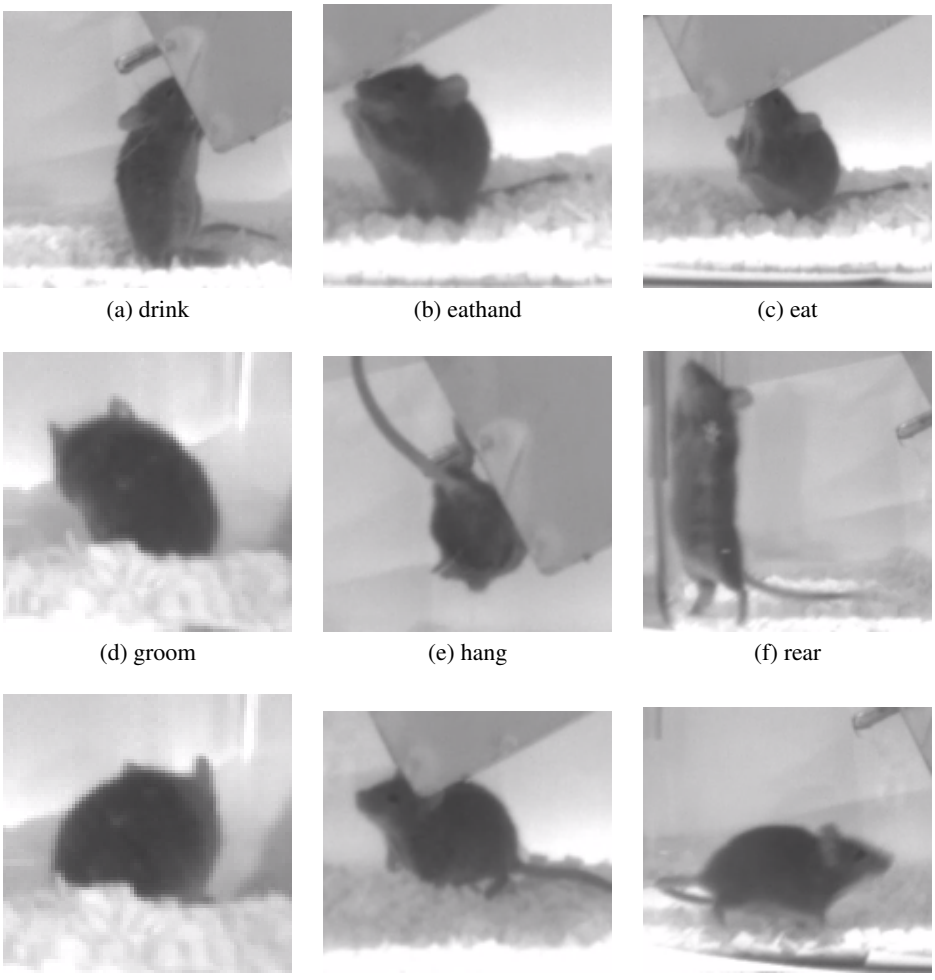
4.1 Web Annotation Tool and Pipeline

Over the course of several years a web-based system has been developed for storing and querying annotations on various video based projects. What started out as a simple HTML5 video player soon grew into a large part of the lab's infrastructure, and is now relied upon by several partner labs for data collection as well. After several iterations in design the current version of the tool was settled on, which has proved generic enough to work for multiple types of projects: ecologists annotate interactions between birds in the wild, neuroscientists analyze eye-tracking data from epileptic patients, and biomedical researchers classify behaviors in mice using the web interface. The lab also has several groups of students that work part time during the school year annotating mouse footage, in addition to several expert annotators that used the tool to generate data such as that used to train the models in this paper. The system consists of three main parts: a web-based user interface written in JavaScript and jQuery, a server that handles API and static file requests running in Node.js, and a central MongoDB database to store all of the data and analyzes.

Figure 2: An example of the tool being used to annotate faces in an episode of *Friends*.

4.2 Experimental Setup

Experiments were run on several desktop machines in the lab all running Ubuntu 14.04, using the Caffe [17] deep learning framework to leverage the power of the NVIDIA Titan X GPUs in each computer. For the neural network trials that did not have recurrent layers the master branch of Caffe was used, and for the remaining trials the LSTM layer in the recurrent fork by Jeff Donahue was included.



(a) drink (b) eathand (c) eat

(d) groom (e) hang (f) rear

(g) rest (h) sniff (i) walk

Figure 3: Representative frames of the mouse actions in the dataset

4.3 Data

All of the data used in this thesis was collected from rooms in the Brown Bio-Medical Center, where the lab has an array of infrared cameras setup in controlled environments that continuously record and save footage of mice. They are average lab mice that have not been subjected to any experiments. The data is recorded using custom software and Point-Grey Firefly cameras, stored as 30 FPS, 640x480 resolution, grayscale MP4 videos which are broken into frames to be analyzed by Caffe using FFMPEG.

The data is annotated with one of nine classes per frame: *drink*, *eat*, *groom*, *hang*, *sniff*, *rear*, *rest*, *walk*, and *eathand*. These represent simple actions the mouse performs that are useful for behavioral analysis, and representative frames of each class are shown in Fig. 4.3.

4.4 Defining Accuracy

The traditional definition of accuracy is the number of samples that were correctly labelled divided by the total number of samples. While this is a valid measure, it is not as effective for datasets that have very uneven distributions of classes. The classification systems developed in this paper are better analyzed with *balanced accuracy*; this is defined as the average of the per-class accuracies, which gives more weight to classes that are underrepresented in the data. For example, in the training and validation dataset the *rest* and *eathand* classes each represent more than 25% of the data, while *drink* and *walk* classes each occur less than 1% of the time.

Another metric considered in the assessment of model accuracies is the ability for humans to agree on classifications. Several of the classes are defined by the type of small movements the mouse is making, and many times humans have difficulty deciding exactly what kind of motion is occurring. Additionally sometimes the mouse is performing many actions simultaneously which leads to further discrepancies and has motivated an effort to redefine the classification framework. Labelling disagreements between humans have shown there is only an approximately 80% *human agreement* for the same videos in the datasets. This sets the target accuracy for the models, because even if the model achieves a higher accuracy on one video in general this is the level at which humans can consistently annotate.

4.5 Network Architectures

4.5.1 CNN

In order to see how effective recurrent architectures are a convolutional only model is trained without a recurrent layer for comparison. Alexnet [18] is a CNN that is often times the basis for many convolutional architectures. It was among the first papers to clearly demonstrate the power of deep learning combined with GPU computing architectures. It uses a technique called *dropout*, a popular method for preventing overfitting that works by randomly removing neurons from a network at any given training iteration. It does this in an attempt to make a model that is more robust by forcing it to learn weights and biases that are not as dependent on other activations. Using dropout and rectified linear units (ReLU) amongst other techniques, AlexNet achieved winning top-1 and top-5 error rates at the ILSVRC-2012 image classification competition [21].

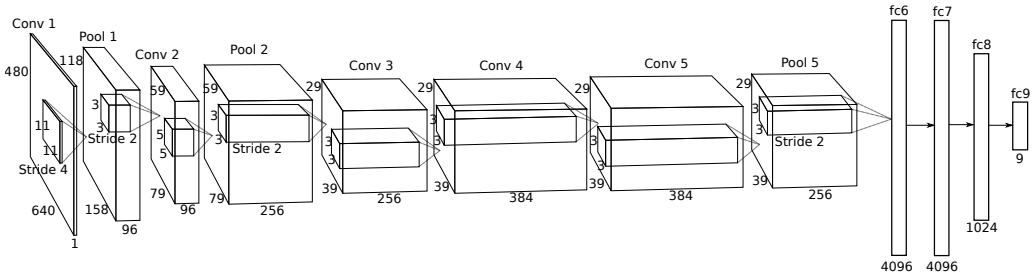


Figure 4: The CNN architecture used for time-independent classification, inspired by AlexNet.

As shown in Fig. 4, the model is similar to the original AlexNet; there are five convolutional layers using the original filter and output sizes with pooling in between to learn spatial features, followed by several fully connected layers. All neurons are rectified linear units (ReLU), but batch normalization [15] was used instead of local response normalization after the *pool1* and *pool2* layers; this has been shown to improve learning rates and removes the need of dropout neurons. There is also an additional fully connected layer added before the final output layer, so that features could be extracted at this layer to use in other classifiers. This is referred to this as the shape model.

Combining Features The current classification system saw a noticeable improvement when using position features of the mouse compared to its motion-only alone. In order to see how much explicit position information affects the performance of the neural networks the pre-computed position features from the current system are taken for use in the new models. Position and velocity are diagnostics for many of the actions used, especially when the mouse is resting and eating. Eating is always defined as when the mouse is touching the feeder spout, so a classifier will have a much easier time if given the explicit position information compared to having to extract and learn it. Position features were inserted into the CNN by concatenating them to the *fc8* layer in Fig. 4.

The other type of features used in the current classification system is a hierarchy of motion features generated starting from spatial-temporal Gabors in several orientations. While these are hard-coded into the HMAX system, in order to use these in the neural network models the architecture was modified to that of in Fig. 5.

The motion stream uses four orientations of Gabor filters that convolve over nine frames at a time to produce a single motion image, which is then used as input for a condensed version of the shape stream from the CNN. In practice the motion images were precomputed due to the format that the data is read in by the deep learning framework, so the network definition starts at the *Motion Conv1* layer. The two-dimensional outputs from the motion and shape streams are then flattened and

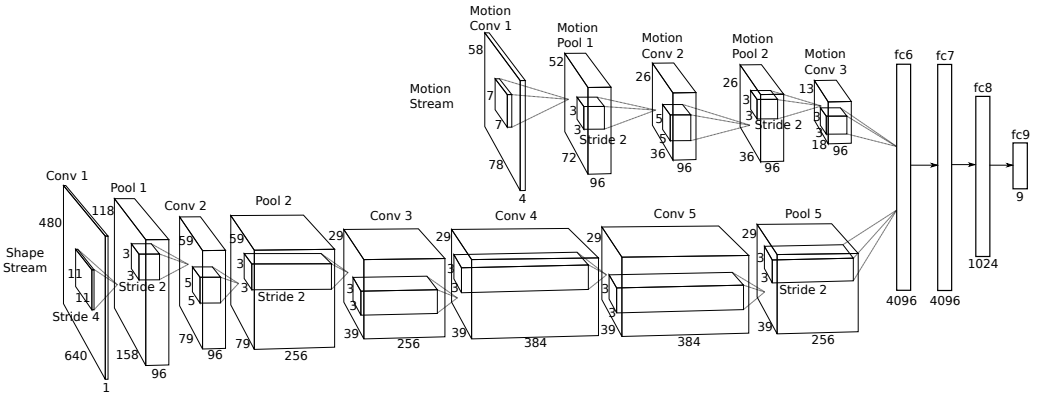


Figure 5: The CNN architecture incorporating spatial-temporal filters using a motion stream. The convolutional layers of the motion stream represent the motion based component of the current system.

concatenated before being sent to *fc6*. Some actions that the CNN has difficulty classifying are very dependent on motion, such as when the mouse is grooming or sniffing; the only difference is small movements in particular parts of the mouse that shape alone has difficulty conveying.

Pretraining A common practice with neural networks is to load weight and bias values from networks that have already been trained on large sets of data relevant to the purpose of the model. The intuition is that if they have been exposed to a large amount of data then they will already have parameter values that are general enough to apply to the particular purpose of the model, and when new training data is given to the network it will *finetune* the model for the given dataset.

A common set of weights and biases to use are from AlexNet trained to iteration 310,000 on the ImageNet dataset [21]. The parameters for the convolutional layers from the pretrained model were used as initializations for the equivalent layers in the new models, while the fully connected layers' parameters were initialized from Gaussian distributions.

4.5.2 Long-Term Recurrent Convolutional Networks

The current classification system uses a temporal dependence between frames in its SVMHMM component, so in order to better replicate its results a recurrent neural network on top of the convolutional model was used to learn a relationship across time. Particularly, an LSTM layer was used in order to avoid unstable gradient issues that often plague recurrent architectures. As seen in Fig. 6, it is very similar to the shape model except there is a recurrent layer between the second and third fully connected layers.

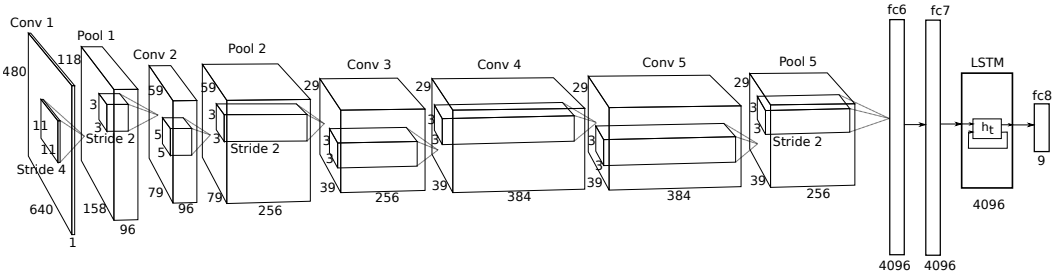


Figure 6: The recurrent convolutional (LSTM) architecture used to combine spatial and temporal models.

Combining Features As with the shape model, different feature combinations with the LSTM models were also tested. However now there were two possibilities for concatenating position features, to a fully connected layer before or after the LSTM layer. Motion features were incorporated into the LSTM model the same way as with the shape model in Fig. 5 except with an LSTM layer instead of *fc8*.

5 Results

5.1 Network Hyperparameter Tuning

Deep learning models often have many hyperparameters that require careful tuning, such as those that control learning rate, parameter updates, and normalization. There has been a great deal of work in developing improvements to these elements including batch normalization, gradient clipping, and sophisticated learning rate policies; these were all tested and tuned to improve the learning efficiency and validation accuracy of the models.

5.1.1 Data Normalization

Until recently many neural network models used local response normalization (LRN) [18] which normalizes activation function outputs across different output maps; for example, in a convolutional layer that produces many output maps the activations in each map at coordinates (i, j) would be normalized across all maps. This was inspired by the lateral inhibition in neurons observed in the brain, and prevents any one map from producing activations that overwhelm others so that their effect on learning is not mitigated. However in 2015 a paper was published detailing batch normalization [15]. Batch normalization works by normalizing all samples in a mini-batch so that inputs to neurons from different mini-batches are from the same distribution; this makes it easier for them to learn because they do not have to constantly adapt their parameters to different ranges of values during training.

Using batch normalization and removing dropout neurons, a significant speedup in model learning in observed as seen in Fig. 7.

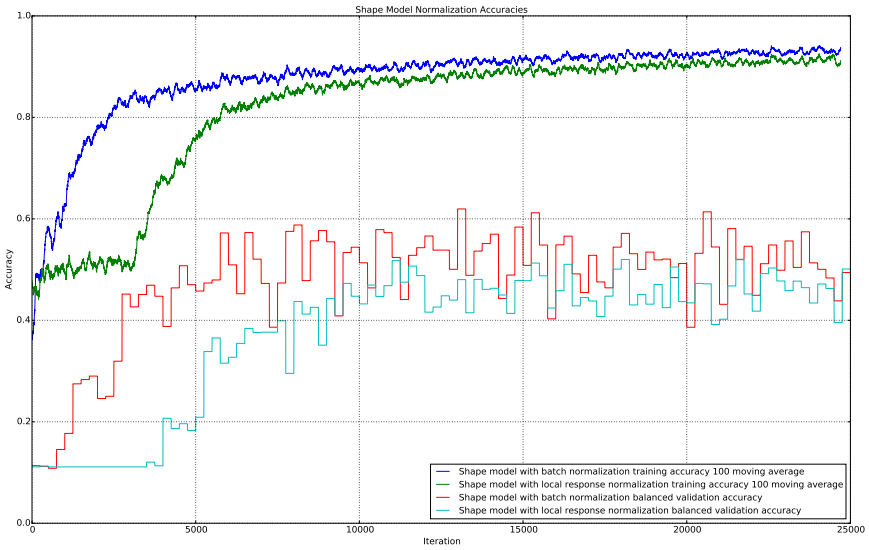


Figure 7: A shape model trial with local response normalization and batch normalization.

5.1.2 Solver Types

When training neural networks, the model error is minimized using a numerical solver such as stochastic gradient descent (SGD). SGD works by using the gradient of the model error with respect to the parameters to adjust each parameter in a way that reduces the overall error. The basic equation for the weights of layer L is

$$w_L = w_L - \frac{\eta}{N} \sum_i \frac{\partial E_i}{\partial w_L}$$

where η is the learning rate hyperparameter, N is the batch size, and E_i is the error for training example i. Note that it only calculates the change in parameters from the gradients with respect to

the current mini-batch, which significantly reduces computation time compared to calculating it on the entire training population. However, often times SGD is prone to overshooting local minima in the error surface of the model because the error gradients are too large. One solution to this is to use *momentum*, which is a technique inspired by the physical concept that resists changes in the gradient vector by keeping a running sum of the past gradient vectors. The original SGD equation can then be modified to get

$$v_L = \xi v_L - \frac{\eta}{N} \sum_i \frac{\partial E_i}{\partial w_L}$$
$$w_L = w_L + v_L$$

where ξ is the momentum hyperparameter that tunes how much the new gradient vector affects the weight updates. ξ is typically between 0.9 and 0.99.

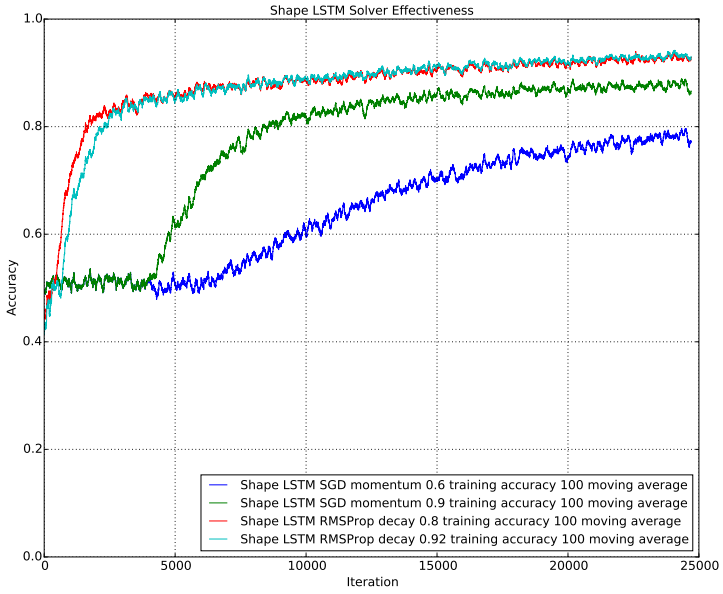


Figure 8: The shape LSTM model using the SGD solver with a typical momentum value of 0.9 and a low momentum value of 0.6, and using the RMSProp solver with a typical decay rate of 0.92 and a low decay rate of 0.8. Clearly RMSProp learns more efficiently than SGD does, especially when comparing the solvers with lower hyperparameter values.

Another popular and effective solver that expands on the momentum concept is called RMSProp [26]. The formula is similar to the momentum SGD,

$$c = \mu c + (1 - \mu) \left(\frac{1}{N} \sum_i \frac{\partial E_i}{\partial w_L} \right)^2$$
$$w_L = w_L + \eta \frac{\frac{1}{N} \sum_i \frac{\partial E_i}{\partial w_L}}{\sqrt{c}}$$

where c can be thought of as a running cache of the gradient vectors and μ is the cache decay rate hyperparameter. Typical values of μ are between 0.9 and 0.999. For the new models RMSProp was much more effective than SGD as seen in Fig. 8 and Fig. 9.

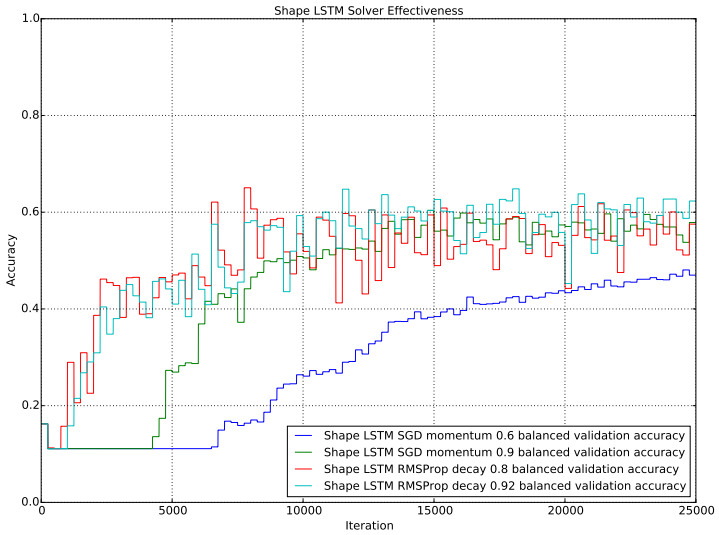


Figure 9: The shape LSTM model using the SGD solver with a typical momentum value of 0.9 and a low momentum value of 0.6, and using the RMSProp solver with a typical decay rate of 0.92 and a low decay rate of 0.8. Even the lower decay rate RMSProp solver reaches a higher validation accuracy than SGD does.

5.1.3 Learning Rates

The learning rate hyperparameter η controls how much the new error gradients affect the model parameters. While this could be a constant value, it is almost always a value that decreases with training iterations. This is so because when the model is starting it should update the parameters more to get away from the random initializations, but once it has gone through many iterations it should have found a minimum in the error surface of the model and therefore only be finely tuning the parameters.

An inverse learning rate policy was used as the function controlling this hyperparameter, given by the formula

$$\eta = \frac{b}{(1+\gamma t)^{\epsilon}}$$

where b is the base learning rate, t is the training iteration, and γ, ϵ are hyperparameters that control the rate and shape of learning rate decay. Adjusting these hyperparameters can have a significant impact on learning performance, as evidenced by Fig. 10.

5.1.4 Gradient Clipping

One of the issues that many networks run into as they increase their depth is the *unstable gradient problem* [12]. The basic intuition for why gradients can become unstable goes back to the derivation of the backpropogation algorithm: using the chain rule the gradient of the error with respect to any parameter in the network can be derived as a product of derivatives and parameter values. However, as the number of layers in between the parameter and the output layer increases so do the number of terms in the product. Considering the sigmoid activation function has derivatives in the range $\left(0, \frac{1}{4}\right]$ and $\tanh$ activation function has derivatives that are in the range $(0, 1]$, many of these terms can be below 1 and drive the gradient toward zero at an exponential rate. A similar argument can be made with combinations of parameter values for the case where the gradient explodes.

In order to combat the exploding gradient problem, different *clipping values* for the gradient were tested. If the network tries to change its parameters by a gradient with a norm larger than the clipping value, the gradient is scaled to have a norm equal to the clipping value. By tuning this hyperparameter appropriately the learning efficiency and accuracy of the models were increased, as seen in Fig. 11 and Fig. 12 respectively.

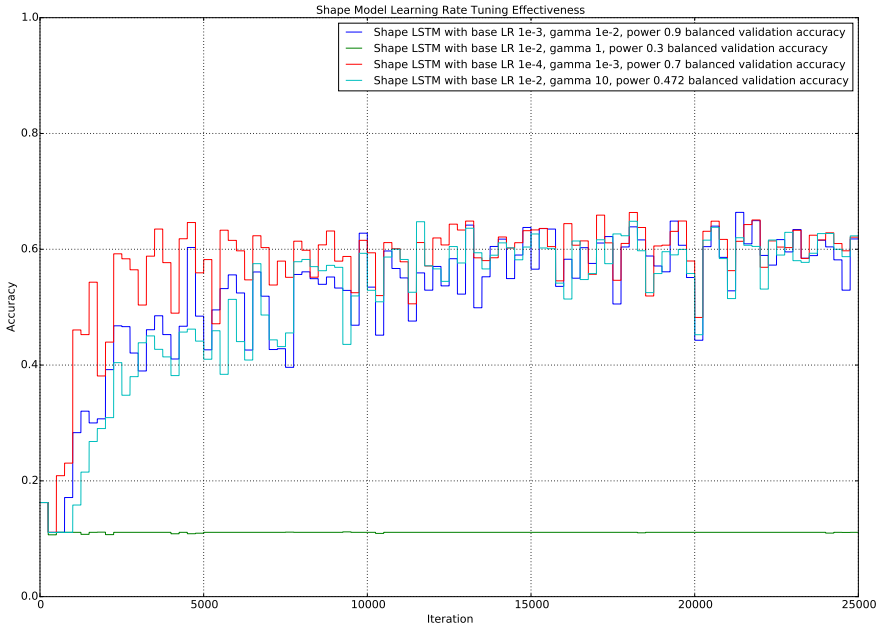


Figure 10: Several choices of hyperparameters for the inverse learning policy. The green line is for a set of hyperparameters that made the learning rate too large, resulting in a model that could only predict as well as chance. The others are the result of careful tuning in order to balance learning magnitude and rate decay to achieve better performance.

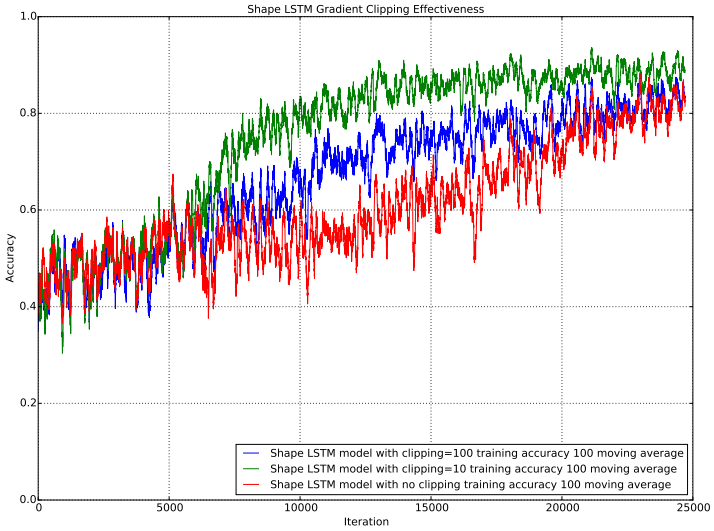


Figure 11: Training accuracy on the shape LSTM model. By tuning the clipping value of the gradients it was able to learn much faster compared to poorly tuned or no clipping values.

5.1.5 Random Initializations

Unless pretrained values are specified all of the model weight parameters are initialized to values sampled from a Gaussian distribution with zero mean and standard deviation 0.01, while all the bias parameters are initialized to zero or one. Because of the highly nonlinear nature of the error space

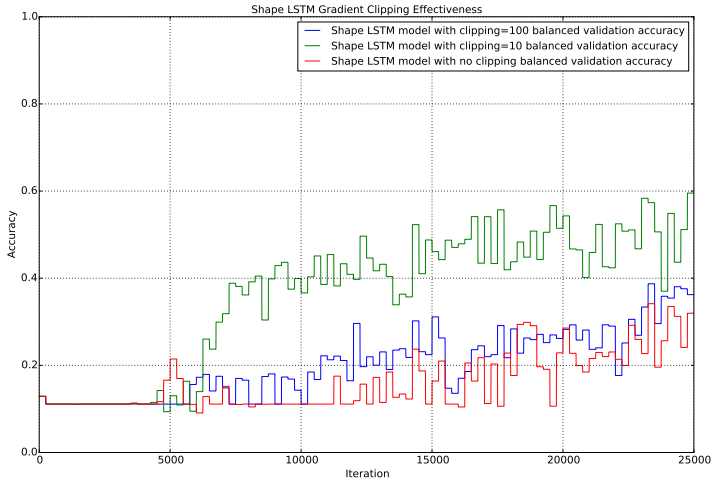


Figure 12: Balanced validation accuracy on the shape LSTM model. By tuning the clipping value of the gradients it was able to achieve a higher balanced validation accuracy compared to poorly tuned or no clipping values.

of these models, starting in a new random location can place the model near different local minima and thus significantly impact learning performance. This is clearly the case as Fig. 13 shows.

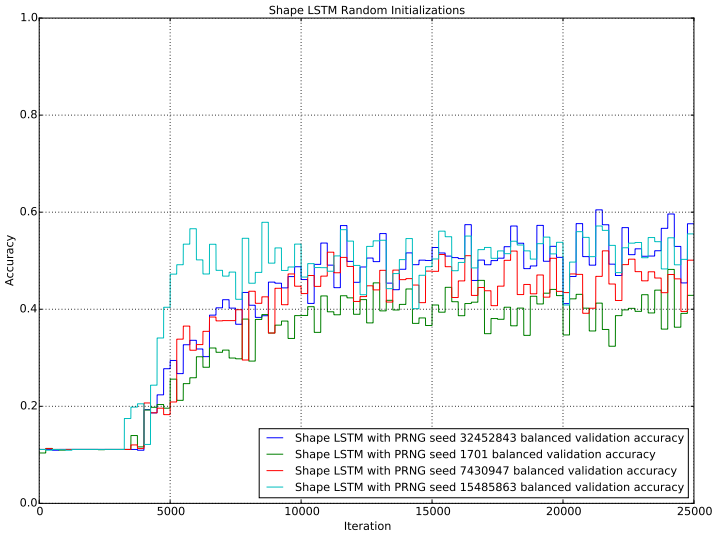


Figure 13: The balanced validation accuracies for several trials of the shape LSTM model with different random initializations. Depending on where the solver starts it can be confined to a lower accuracy for a great length of time or quickly reach a higher one.

5.2 Architecture Effectiveness

In order to select the best hyperparameters cross validation on the eight expert-annotated videos was used. For all of the results in this section the following hyperparameters were used: an inverse learning policy with $b = 0.01, \gamma = 0.01, \epsilon = 0.472$, a gradient clipping value of 10, a random seed of 15485863, the RMSProp solver with $\mu = 0.92$, and batch normalization. While several models did not perform nearly as well as the current system, others were able to achieve a higher balanced validation accuracy. The accuracies reported are taken at the training iteration with the highest validation accuracy.

5.2.1 The Effectiveness of Pretraining

As seen in Table 5.2.1, pretraining has a significant boost in accuracy over randomly initializing the model parameters. Additionally it outperforms the existing classification system when pretraining with shape and motion features. Having already been trained on so many other images, the parameters should be at a local optima in the error surface of the model that can be used to boost the accuracy of the shape component of the model.

	random initialization	pretraining
shape	62.9%	**72.0%**
shape & position	60.9%	**68.6%**
shape & motion	78.2%	**83.6%**
shape, motion & position	**77.1%**	76.8%

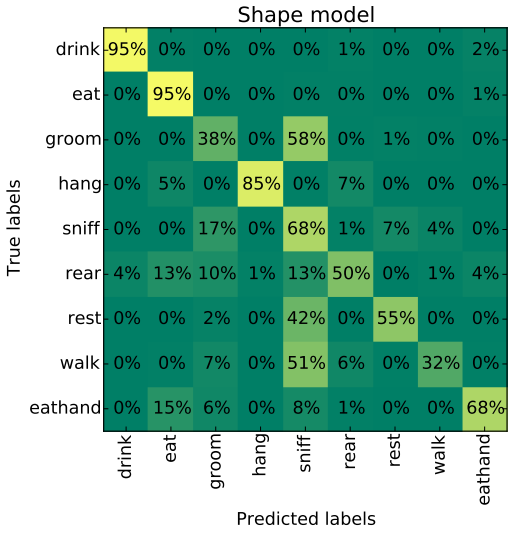


Figure 14: A confusion matrix for the shape model. While eating and drinking are very accurate it often confuses grooming and sniffing. This is expected because these misclassified behaviors are very dependent on the motion of the mouse, which this model does not explicitly know. Additionally, the model struggles to classify more basic, shape based actions as accurately as it does when using pretraining, which is again expected because we are randomly initializing the weights here instead of starting from known, useful values.

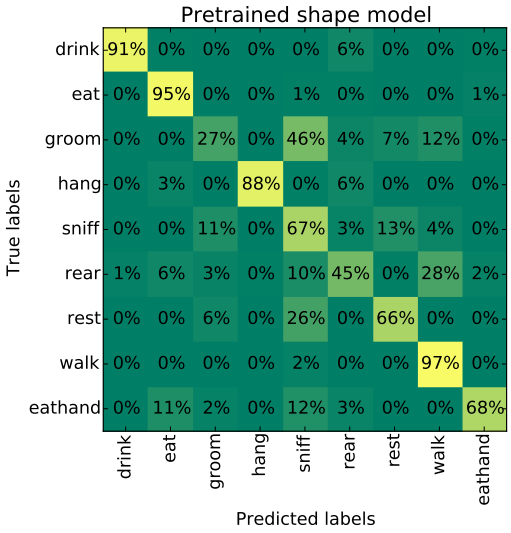


Figure 15: A confusion matrix for the shape model initialized with convolution layer weights from a pretrained AlexNet network. Clearly this has a substantial improvement over the randomly initialized shape model, yet it still fails to properly classify many motion based actions such as groom and sniff.

Even when pretraining, some actions are still commonly confused, such as sniffing versus walking and grooming and sniffing. They pose a challenge for the dataset because the mice often perform both at the same time or they look quite similar, so it can be difficult to distinguish when a human would label the action as one versus the other.

5.2.2 Shape versus Motion

Given that several mouse actions differ only in the small movements it is performing, the overall shape of the mouse will not give much useful information for these classes. Fig. 17 shows that once the motion features are included the model can better differentiate sniffing and grooming, significantly improving the accuracy of classification overall and outperforming the current system.

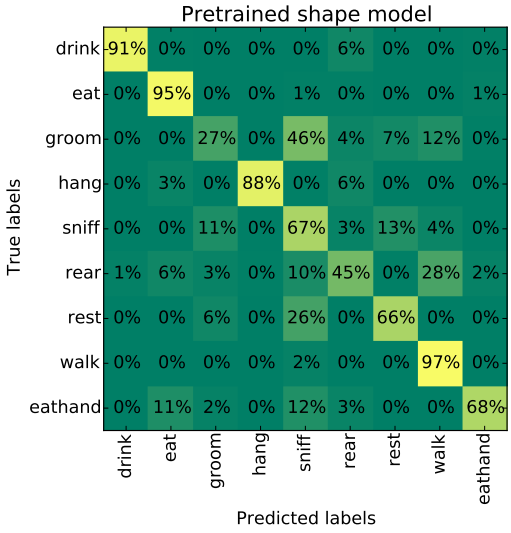


Figure 16: A confusion matrix for the shape model initialized with convolution layer weights from a pretrained AlexNet network. While it achieves a balanced validation accuracy of 72.0%, most of its accuracy is concentrated in a few classes that are easier to discern from shape alone.

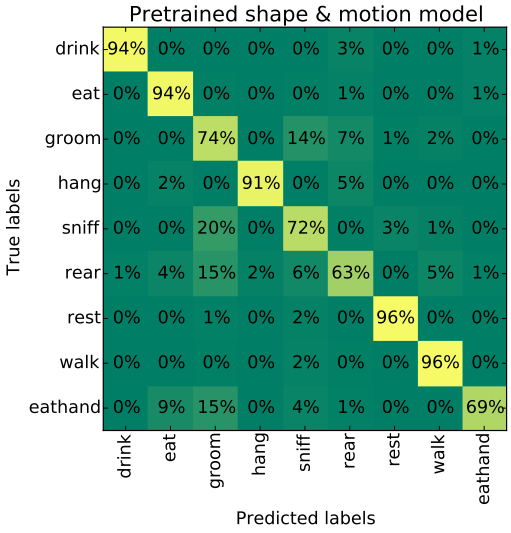


Figure 17: A confusion matrix for the shape model combined with motion features, initialized with convolution layer weights from a pretrained AlexNet network. This model achieves a balanced validation accuracy of 83.6%, scoring better than the existing classification system with high accuracies across all classes. While it does still make some mistakes between sniffing and grooming these are often also confused by human annotators.

5.2.3 The Effectiveness of Position Features

While it was expected that as with the current system the model would see an increase in accuracy when combining it with position features, there was actually some decrease in performance. Compared to a balanced validation accuracy of 72.0% with shape alone as seen in Fig. 15, there is a dip in performance to 68.8% when inserting position features as seen in Fig. 18. This is believed to be because the model is overfitting to the position training data; it is becoming accustomed to the mouse performing actions in certain positions in the cage, and when these position change in the test data the system fails to classify properly. A similar scenario is seen when including both motion and position, comparing the 83.6% accuracy seen in Fig. 17 with the 76.8% seen in Fig. 19.

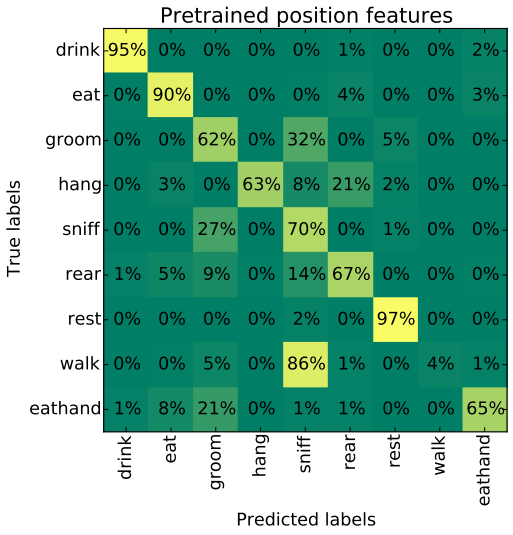


Figure 18: A confusion matrix for the shape model combined with position features, initialized with convolution layer weights from a pretrained AlexNet network. While this still is quite accurate for several actions, it still fails to classify many motion based classes as accurately as using motion features alone. It can still differentiate actions such as resting from others when the mouse has the same shape but different large scale movements, yet cannot accurately classify actions with the same shape and large scale movements but different smaller motions such as sniffing versus walking.

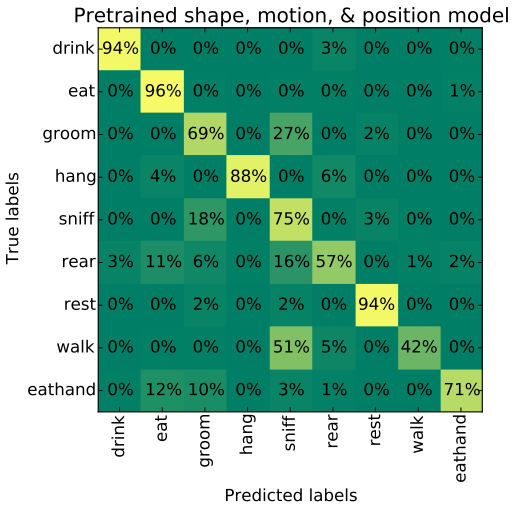


Figure 19: A confusion matrix for the shape model combined with motion and position features, initialized with convolution layer weights from a pretrained AlexNet network. It particularly fails to discern sniffing from walking, which the shape and motion model did very well. This is clear evidence that the position features are confusing the system, as the motion information to indicate that sniffing is occurring is present yet because the mouse is walking and sniffing in many different locations and directions the model fails to classify it well due to overfitting.

5.2.4 Non-recurrent versus Recurrent Architectures

In general the LSTM models did not perform as well as expected. In some trials that did not have pretrained weights the recurrent models did outperform the non-recurrent ones, yet more often they was surprisingly less effective. See Table 5.3 for a complete comparison of model types. This is believed to again be because the models were overfitting to the training data; it takes massive amounts of samples to properly train the many pieces inside LSTM units, and because they are randomly initialized it is unlikely that there are many meaningful values to start with. More training data is being collected to satisfy this requirement, at which point it is expected they will perform much better. The LSTM confusion matrices seen below are using a sequence length of two, in order to mimic the order two HMM used in the current system. Sequence depths of ten and thirty were also used, but did not show any significant impacts on performance.

Additionally, in general it appears that inserting the position features before the LSTM layer has a greater effect compared to inserting them after. This could be because the LSTM layer is actually learning the transitions between the larger scale motions of the mouse, which would improve classification accuracy on several action classes.

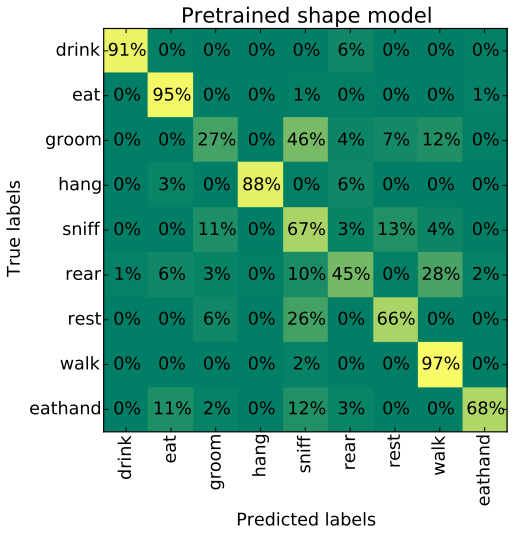


Figure 20: The shape model initialized with convolution layer weights from a pretrained AlexNet network, replotted for comparison with the LSTM architecture.

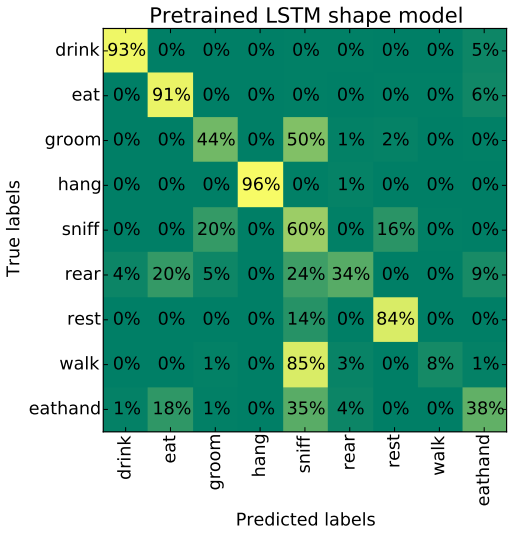


Figure 21: The shape LSTM model initialized with convolution layer weights from a pretrained AlexNet network. With a balanced validation accuracy of 61.5% it is significantly worse than the 72.0% achieved without the LSTM, which is believed to be from overfitting to the training data.

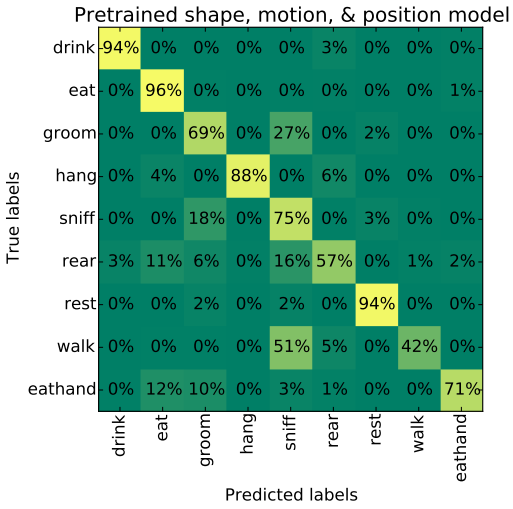


Figure 22: A confusion matrix for the shape model combined with motion and position features as well as initialized with convolution layer weights from a pretrained AlexNet network, replotted for comparison with the LSTM architecture.

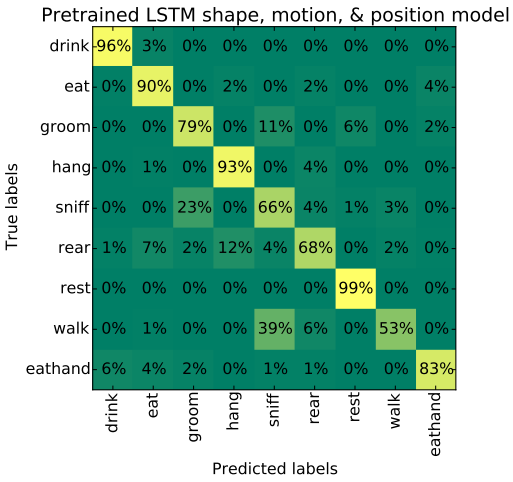


Figure 23: A confusion matrix for the LSTM model with shape, motion and position features initialized with convolution layer weights from a pretrained AlexNet network. The position features here were inserted after the LSTM layer. While this model is still accurate across most classes and even outperforms the same model without the recurrent layer, it has an overall lower balanced validation accuracy of 81.4% compared to the 83.6% achieved by the non-recurrent shape and motion model.

5.3 Summary

Table 5.3 summarizes the balanced validation accuracies of the model trials. The LSTM models were run with a sequence length of two, and the $LSTM(before)$ and $LSTM(after)$ columns specify if the position features were included before or after the LSTM layer of the model.

	CNN	LSTM (before)	LSTM (after)
shape	65.5%	–	69.2%
motion	79.8%	–	79.3%
motion & position	77.8%	–	75.6%
shape & motion	78.2%	–	72.3%
shape & position	60.9%	70.9%	70.4%
shape & pretrain	72.0%	–	61.5%
shape, motion & position	77.1%	78.9%	74.0%
shape, motion & pretrain	**83.6%**	–	79.4%
shape, position & pretrain	68.6%	75.4%	70.4%
shape, motion, position & pretrain	76.8%	71.3%	81.4%

6 Discussion

This paper demonstrates how new developments in deep learning and neural networks are both powerful and difficult to optimize. As the results show, given enough data the new models outperform the current system used for behavioral classification, and it appears that there is still more room to improve. In addition to accuracy, the neural networks developed in this paper are able to analyze the same minute of mouse footage eight times faster than the current classification system; this will have a huge impact on research productivity. Even though the presence of position features could improve accuracies given more training data, moving forward they will not be included; this is because while sometimes effective, the position features rely upon a fragile background subtraction preprocessing step that often required human tuning. Thus the shape and motion model discussed in this paper which had the highest validation accuracy can be run without any human intervention, enabling a much more scalable and parallelizable pipeline. This model also accomplished the goal of creating a system that outperforms the current models and learns a single, end-to-end processing stage without the need for fragile preprocessing steps.

It is suspected that the LSTM models did not perform as well as the non-recurrent architectures because there was not enough training data to properly learn the parameters for their many gates. This resulted in the recurrent layers learning the intricacies of the training data in order to increase accuracy, instead of general patterns that could also perform well on the validation dataset. This could be resolved by switching to either a layer of Gated Recurrent Units (GRU, [3]) or traditional RNN neurons that require less data to train. Their lacking performance is most likely not from using a sequence length of only two with the LSTM models, as longer sequences were also tested and did not noticeably improve the validation accuracy. In order to more rigorously determine the performance of the LSTM layer, the feature vectors produced by the current system were used to train an LSTM model. This model was simply two fully connected layers, an LSTM layer, and a final fully connected layer as seen in Fig. 6 after *pool5*. Unfortunately as seen in the appendix section the model was not able to learn much from these vectors, indicating that it was truly an issue with the LSTM layer in general rather than an error in the convolutional-recurrent models.

Future work will be focused on techniques such as boosting, object location, and image segmentation which are believed to have great promise for improving results. Boosting techniques such as Adaboost that work well with improving results on unbalanced datasets are particularly promising, because as mentioned previously these videos have a very uneven distribution of actions. Additionally an automated hyperparameter optimization routine such as a Tree of Parzen Estimators implementation could be very helpful with optimizing models. From the experiments in this paper the lab also gained numerous new performance metrics on how neural networks can be applied to its work, which will prove valuable in the coming semesters as more data is generated and in need of analysis.

In addition to mouse footage the lab also works with several other types of animal videos including zebrafish, crickets, and wild birds. Given the flexibility of the new models discussed in this paper they will also be applied to each of these datasets, something that the current annotation system would not be able to accomplish as easily.

7 Acknowledgements

I would like to specially thank *Ali Arslan*, *Sven Eberhardt*, and *Youssef Barhomi* for their advising and mentorship throughout my thesis. Additionally I would like to thank Professors Sudderth and Serre for their guidance on my thesis, and the Brown Computer Science and CLPS departments for their excellent facilities and resources.

References

[1] Zillah Boraston and Sarah-Jayne Blakemore. The application of eye-tracking technology in the study of autism. *The Journal of physiology*, 581(3):893–898, 2007.

[2] Jerry J Buccafusco. *Methods of behavior analysis in neuroscience*. CRC Press, 2000.

[3] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[4] Heiko Dankert, Liming Wang, Eric D Hoopfer, David J Anderson, and Pietro Perona. Automated monitoring and analysis of social behavior in drosophila. *Nature methods*, 6(4):297–303, 2009.

[5] Tristan Darland and John E Dowling. Behavioral screening for cocaine sensitivity in mutagenized zebrafish. *Proceedings of the National Academy of Sciences*, 98(20):11691–11696, 2001.

[6] Jeffrey Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2625–2634, 2015.

[7] Kimberley S Gannon, James C Smith, Ross Henderson, and Paul Hendrick. A system for studying the microstructure of ingestive behavior in mice. *Physiology & behavior*, 51(3):515–521, 1992.

[8] Evan H Goulding, A Katrin Schenk, Punita Juneja, Adrienne W MacKay, Jennifer M Wade, and Laurence H Tecott. A robust automated system elucidates mouse home cage behavioral structure. *Proceedings of the National Academy of Sciences*, 105(52):20575–20582, 2008.

[9] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[10] Karol Gregor, Ivo Danihelka, Alex Graves, and Daan Wierstra. Draw: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623*, 2015.

[11] Nathaniel Heintz. Bac to the future: the use of bac transgenic mice for neuroscience research. *Nature Reviews Neuroscience*, 2(12):861–870, 2001.

[12] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, and Jürgen Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

[13] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[14] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[16] Hueihan Jhuang, Estibaliz Garrote, Xinlin Yu, Vinita Khilnani, Tomaso Poggio, Andrew D Steele, and Thomas Serre. Automated home-cage behavioural phenotyping of mice. *Nature communications*, 1:68, 2010.

[17] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[19] Sean D Pelkowski, Mrinal Kapoor, Holly A Richendrfer, Xingyue Wang, Ruth M Colwill, and Robbert Creton. A novel high-throughput imaging system for automated analyses of avoidance behavior in zebrafish larvae. *Behavioural brain research*, 223(1):135–144, 2011.

[20] Frank Rosenblatt. Principles of neurodynamics. perceptrons and the theory of brain mechanisms. Technical report, DTIC Document, 1961.

[21] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[22] Bryan C Russell, Antonio Torralba, Kevin P Murphy, and William T Freeman. Labelme: a database and web-based tool for image annotation. *International journal of computer vision*, 77(1-3):157–173, 2008.

[23] Thomas Serre, Lior Wolf, and Tomaso Poggio. Object recognition with features inspired by visual cortex. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 994–1000. IEEE, 2005.

[24] Edward Shortliffe. *Computer-based medical consultations: MYCIN*, volume 2. Elsevier, 2012.

[25] Ilya Sutskever, James Martens, and Geoffrey E Hinton. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1017–1024, 2011.

[26] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4:2, 2012.

8 Appendix

8.1 Current System Features

Unfortunately even after testing a large number of different learning rates and other hyperparameters, little was able to be learned using this model. In Fig. 25 the learning rate was drastically lowered, and while there was some initial learning that occurred it was very soon forgotten. Most often the model simply did not learn anything, as seen in the almost horizontal lines in Fig. 24 where the validation accuracy is always at the same level as chance.

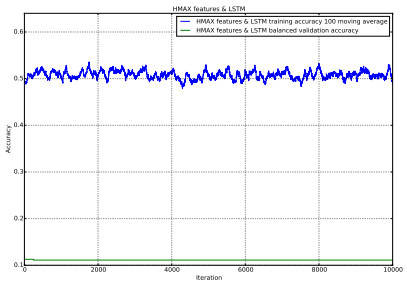


Figure 24: An example of no learning taking place in the model.

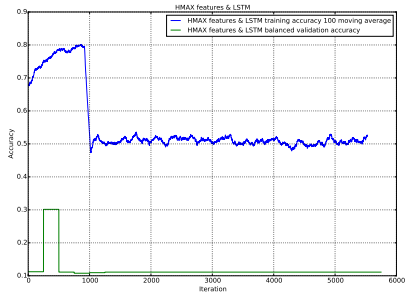


Figure 25: An example of unlearning, possibly due to overshooting and exiting a local minimum of the error surface.