Placeholder

Taliah Horner

MInf Project (Part 2) Report
Master of Informatics
School of Informatics
University of Edinburgh

2019

Abstract

Acknowledgements

Table of Contents

Chapter 1

Introduction

1.1 Project Overview

The goal of this project was to create a system that could analyse and detect whether rodents were grooming from a visual feed. The system would be developed to be used in a live pipeline on rodents in a lab in the Home Cage Analysis (HCA) system (Section 1.2.1) developed by Actual Analytics. Initially we would build a system that could detect whether rodents were having a seizure under the assumption that the model, when trained on grooming rodents, would perform reasonably well. This was due to the immediate availability of annotated and bounded data of rodents having seizures, whereas it would require the development of tracking system (a separate undergraduate project) to extract bounding boxes for grooming rodents. Due to unforeseen circumstances, we were unable to explore the grooming data as well as initially intended. We did, however, reach satisfactory levels of accuracy in detecting seizures.

The process for analysing the seizure data was initially working off of an MSc thesis described in Section **??**. There were several problems with the implementation of this work which we repaired and improved upon. The main idea behind the project was pre-processing the video data, as described in Section **??**, and using the LibLinear [**?**] package to obtain a classification model. We later describe the problems with this approach and how we improved upon it. This process is described in Chapters **??** and **??**

Next, we turned to the more powerful classification tool of neural networks (NNs) and the python packages tensorflow [**?**] and keras [**?**] which we disscuss in **??**. We used attempted similar pre-processing techniques as before as well as using convolutional neural networks (CNNs) and also combining the two. These methods proved to be much more effective at solving the problem, ultimately with the histogram data providing little improvement over simply using the CNNs yet requiring not an insignificant amount of processing and memory in live classification. This process is discussed in Chapters **??** and **??**

We tested potential NNs on a held out test video, resulting in (...) proving most effective. The outlines of the testing and results can be found in Chapter **??**

Following the slightly more abstract data science project of classifying rodents, we then proceeded with the engineering problem of creating the full pipeline with the techniques proven best by the test results. That is, building architecture that take in a video feed and detects seizures in real time. The implementation of the pipeline is described in Chapter **??**.

Finally, we attempted to train the best NNs on the grooming data that we had also, Chapter **??**.

All of the code for the project can be found on the github repository: ... Much of the code there is for processing the data and is somewhat messy. The code for the pipeline to be used is well structured and commented.

To do Maybe a bit more here (**??**)

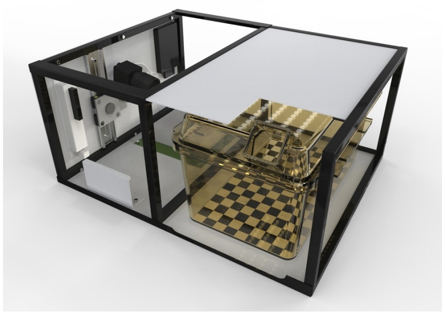
1.2 Previous Project

Our project last year was the development of a camera environment as part of a larger project of migrating the HCA system to a Linux based operating system. While this went reasonably well, we lost significant marks due to a lack of evaluation. Moving forward from this, we have made evaluation an integral part of our work, aiming to evaluate progress at many points during our project.

To do more here (**??**)

1.2.1 HCA

Figure 1.1: The HCA system: the right section contains a rodent cage along with a checked baseplate that is capable of reading off inputs of RFID tags; the left section houses the computer along with a camera.



The HCA, Figure 1.1, is a product developed by Actual Analytics that holds a cage for rodents. In the case is a computer and a camera while underneath the cage is a baseplate that can read off of the RFID tags implanted in the animals. The baseplate

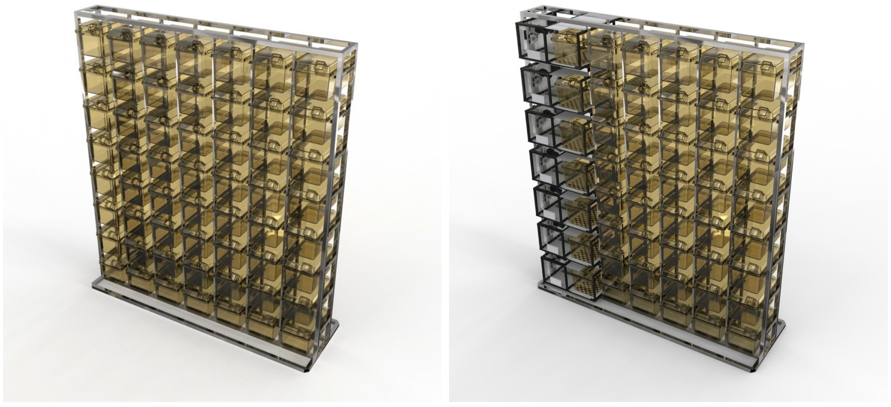

(a) A full rack of cages (b) Full rack with recording cases

Figure 1.2: Full racks

has a checker board pattern such that the location of the animals can be identified. The cage is lit by infra-red lighting strips from the roof of the case and often holds general equipment that you would expect from a rodent cage - water bottle, food bowl, etc. These cases can then be stored in racks, Figure 1.2 such that multiple cages can be monitored in a network.

There is a day and night cycle that is an instantaneous switch every 12 hours, the infra-red lighting meaning that footage is recorded for all 24 hours of each cycle. Each cage can also be viewed from a live stream. Experiments of up to 2 weeks could be requested, where the machine would capture video segments of a desired length for the duration of the experiment.

The products was already functional in some form on Windows and was in use by several companies. Part of the specification was retaining the functionality such that the switch would be without trouble. While part one of our project involved hands on interaction with the product itself, this year we were to be working with the data already obtained from the HCA by the companies using them. This meant that we had practical data from a source that we were well acquainted with, a situation which can be somewhat uncommon in data science.

1.2.2 Building Camera environment

As previously stated, our work involved designing the camera interface of the Linux HCA implementation. This also involved undergoing market research to select and purchase a camera. We decided on an ELP model that performed on camera H-264 encoding which proved to be reasonably efficient. A later discovered problem however was that, while the camera interfaced well with the Linux system, using it with Windows caused some strange issues due to the unstandardised H.264 encoding.

Constructing the environment proved to be a nice engineering problem, however it suffered from a lack of evaluation. One very successful aspect of the project was a development of a guide to compiling opencv on Linux to work with the ffmpeg H.264 codecs, a guide that we ourselves used when encountering the same problems.

To do Write about building the camera environment, areas let down was in evaluation. Have done much more evaluation here (**??**)

1.3 Visual Analysis of Rodents

1.3.1 Seizure

Specifics on the seizure data is covered in Chapter **??**

To do Describe the problem, include MSc project here? (**??**)

1.3.2 Grooming

To do Describe the problem, only did small amount, if any? (**??**)

1.4 Related Work

To do MSc here? Related HCA research. Also anything relevant on analysis (**??**)

1.5 Process Outline

For the sake of clarity, it is beneficial to explicitly outline the entire process of this academic year, during which the company stakeholders were consulted as necessary. This is provided along with a brief description of the work done at each step.

To do List of stuff (**??**)

Chapter 2

Seizure Data

In this chapter we will give an overview of the seizure data that we obtained and the potential problems that we initially identified with the data set. We will also discuss the approaches of the MSc project to the same problem and how we aimed to improve upon these.

2.1 Overview

The seizure data consists of five (labelled from 0), hour long videos of a cage containing two baby rats along with two sets of hand written annotations pertaining to one of the rats. During each of the videos, one of the rats has at least one bout of seizures, this rat being the one that the annotations relate to. One set of annotations is a list of features, as in Table **??**, for each frame of each video. Of the list of features, the ones that were relevant here were any that included seizure in the name. The other set of annotations is the coordinates of the rat that has a seizure at each frame along with a bounding box covering the rat, these had been hand drawn by the MSc student using a software they had implemented. The videos have a resolution of 1280x760 and are grey-scale.

Normal behaviour for the rats during the videos is general activity: running around, eating, some grooming and interaction. While have a seizure the rats will usually rear up onto their hind legs and their arms will start to spasm while they continuously fall over and rear up again.

To do More here? (**??**)

2.2 Potential Problems

The first problem to note is that the data set is not particularly large. While five hours may be reasonable, there are seizures present for approximately 13% of the videos. This could potentially mean that there is not enough data to create a solution that generalises well. Furthermore, to create a sufficient measure of generalisation, it is of course necessary to withhold some of the data from training so as to use as a test set. Potential solutions to this problem is data augmentation [**?**] and using cross fold validation [**?**].

Next, there is the issue that there are two rats in the video. While often not a problem as the other rat is in another part of cage, sometimes there are two rats present within the bounding box and, even worse, the other rat is in front. This means that some of the data is not as effective as training data when the model is generalising to a situation with one rat but it is also likely that the model will be employed in a similar environment with two rats. In this case, the data is still providing value.

To do Other Issues? (**??**)

2.3 MSc Project

To discuss the work done by the MSc project, they achieved an accuracy of ..., with a .... They first cut the videos into lengths of five minutes, to allow for ease of separation of data into training and test sets and pre-processed the data using the methods detailed in [**?**]. These methods, which we discuss in depth in Chapter **??**, involve taking gradients of optical flow and ... across frames. They then used the LibLinear package to obtain a linear fit on the data using cross fold validation.

Regrading the implementation of the pipeline, there were a few errors that we could detect, according to the code we received. Firstly, which we believe was quite significant, instead of extracting the seizure features from the annotations, they added all of the features together and used this as a target (with zero being changed to -1). We believe that using all of the features was intentional whereas summing them may have been a mistake. Regardless, using all of the features includes much that does not involve seizures at all, which would result in many false positives when the model is used for real. Secondly, adding all of the features together has the result of creating a separate class each representing how many of the features the rat is displaying, with no distinction between which features. Furthermore, as it is not using one hot encoding, it implies that the number of symptoms a rat is displaying is a continuous problem, for example, with grooming and tonic seizures, being an extrapolated case of death, which is clearly false. This problem was one that was easily rectified however.

Our second criticism of the work was the use of the LibLinear package which is from 2008, making it reasonably old in terms of computer science. The field of data science has long moved onto expansive NNs which can be used for classification problems, among others. Moving to this architecture, even if following the same pre-processing techniques, it would still likely be beneficial to implement even a simple NN.

Thirdly, the lack of any data augmentation seemed like a missed opportunity. Particularly with the size of the data set being such a primary concern, even small efforts to increase the amount of data would go a long way. We discuss what augmentation we performed in Chapter **??**.

Finally, it seemed remiss that no attempt at recurrence was attempted. By this, we mean exploiting the fact that the seizures appear in bouts and that if a seizure was more likely to be present in a frame if one had occurred in a previous frame. We attempted to implement this by way of using recurrent neural nets (RNNs) and also a more rudimentary system when that did not prove as effective as expected, Chapter **??**.

Overall, having read the thesis and after looking through the code, we felt optimistic that we could improve performance while having a base from which to start our own work. We also made the decision to not use any of the code from the previous project but to write it all afresh.

To do Look over (**??**)

Chapter 3

Pre-Processing

In this chapter we will discuss the methods that we used to pre-process the data such that we could train classifiers.

3.1 Cutting and Cropping

Firstly, similar to the previous project, we divided the videos into 5 minute segments. We believed that this seemed sensible in order to easily split the data. While we are not entirely sure how the bounding boxes were used before (we believe we are missing the code), what we decided to do was use opencv to crop the videos and save them as separate videos. As the boxes where rectangular, we used the longest side of the box to form a square about the same centre and then resized this new frame a resolution of 64x64 pixels. We deemed it reasonable to standardise the resolution of the new videos such that the training and testing process would be more reliable [], while 64x64 was the closest standard size to the actual size in the original videos. We made good use of multi-threading to speed up this process.

3.2 Data Augmentation

Data augmentation is a way the prevent over-fitting in a machine learning problem []. Over-fitting being where a model is too fitted to a training set and so does not generalise to other data, under-fitting being where a model in not fitted well enough and so cannot make any reasonable classification. The optimal solution to a problem lies in between these two situations.

Over fitting is a common problem where there does not exist enough data, a trained model will only learn about a specific few instances of an event (in this case seizures) and will only be able to identify instances that are so similar to those trained upon as to render the model useless in the general case. The idea behind data augmentation is that you can emulate having more data that you actually do under the assumption that instances will look similar (indeed the assumption required for the model to work at all) by transforming the data in some way such that a classifier sees it as a different instance altogether. In fact, the methods regarding optical flow described shortly are in fact data augmentations themselves (a fact we will later exploit). Here however we are talking about affine transformation, that is transformations affected by applying a 3x3 transformation matrix to an image.

The transformations that we used to augment were flipping the image horizontally, as well as rotating both the original and flipped video clockwise and anticlockwise by 1 and 2 degrees. What makes these reasonable augmentations to perform is that it is very much possible to have a rat performing the same actions but flipped horizontally or at a slightly different angle. Of course, simply rotating by too much of an angle would mean that the angle of perspective is no longer realistic, hence the limit to 2 degrees. We could have also performed perspective shift although this would have required the intrinsic parameters of the camera which we did not have. Ultimately we deemed it too much work to be truly worth it.

Again, we used multi-threading to speed up the augmentation process.

3.3 Histogram Data

To do All histogram stuff here (**??**)

3.4 Splitting Data and LibLinear

Initially, the way that we split the data was to randomly shuffle the videos and histogram matrices into training (80%), validation (10%), and test (%10) data sets for preliminary testing without cross fold validation. We treated each of the augmented videos as a separate entity in the splitting, a decision that would shortly prove to be problematic in that it resulted in validation accuracy unrepresentative of actual performance, discussed in Chapter **??**.

When then deemed it reasonably to store each data set in a single file to save time on loading during training, a decision we also later changed but was necessary to test the LibLinear package. This presented us with our first programming issue: storing the sets (approximately 17GB) that do not fit in memory on a file. Initially we attempted to build the array in memory and the save it to disk but this clearly did not work. Unfortunately the input for LibLinear is very specific as it supports sparse feature sets such that each feature must be labelled. With each frame consisting of 6272 features, this was frustrating and slow, as we had to edit and save each line to a text file.

We then attempted to run the LibLinear scripts on the data sets but kept getting segmentation faults. We ultimately decided that, rather than continue this effort, it would be more useful to continue with building NNs which could in fact emulate LibLinear if so desired.

Throughout the project we repeatedly restructured how the data was organised on disk as we learned of better ways to do so. We did, however, learn of the package PyTables [], that would allow us to create HDF5 files consisting of arrays that are specifically designed to store information that does not fit in memory. Using these files, we were able to move onto fitting NNs to our data.

Chapter 4

Neural Net Baselines

In this chapter, we will discuss the packages keras and tensorflow that we used to construct and train the NNs. We will then discuss the initial models that we trained as baselines along with the results of these models using cross fold validation. For clarification, if a model is evaluated across 4 folds, it is the same 4 folds as other models are, so that they directly compare.

4.1 Tensorflow and Keras

Tensorflow is a very powerful python package developed by Google for dataflow and differentiable programming. The package tracks propagation of tensors through operations such that the gradients can be calculated backwards through them. This is particularly useful for training neural nets as it allows the package to take care of gradient calculations automatically, gradient calculations being the part of machine learning problems perhaps hardest to calculate efficiently. It thus removes a huge barrier to creating various NNs architectures and allows for easier experimentation with network architecture.

Moreover, tensorflow also takes care of the physical side of operations, with regards to distributing across multiple CPUs or GPUs and managing other resources, such as memory.

Keras is a second abstraction on top of tensorflow, adding even more flexibility and ease of use to the process. Keras provides a significant level of modularity such that users can easily combine modules to create and train NNs. This includes layers, optimisers, activation functions, etc. that can be combined to create new model architecture. It also allows for the creation of new modules and for these to be easily interfaced with pre-existing ones.

As we used keras we discovered more of its functionality that we will comment on in the relevant section.

4.2 Initial Tests

The first networks we used were very simple, consisting of two fully connected layers with 512 hidden units each. We used relu [] activation functions and dropout [] layers, with dropout probability of 0.2, and a batch size of 128. Finally as in all networks we trained, a fully connected layer to one output with a sigmoid [**?**] activation function would provide the estimated probability of the frame containing a seizure. The loss we used was Binary Cross Entropy [**?**] and the optimiser was Adam []. We decided to use Adam due to it's popular and recommended application [].

We initially attempted to train the networks for 20 epochs, however the networks almost always reached their optimal validation accuracy during the first few epochs and so we implemented early stopping to provide the best attained accuracy. It was of note that the models peaked very early, reaching very low levels of loss very quickly. The models learned all that they could from the data set in any particular training cycle, this could mean that, given a certain performance, the model was not complex enough to deliver a better result or the data was not sufficient to generalise from.

The first validation accuracy scores we achieved were incredibly high, 99.7%, and we were immediately suspicious. This level of accuracy is completely unrealistic, even in well solved machine learning problems, this accuracy was usually unreachable. At first, we thought that perhaps the network was learning to distinguish between when the rats were upright and when not, something relatively easy to distinguish but not generalisable. In such a case, we would need to procure data of rats standing upright and run the model on such data.

Even this thought was optimistic though, as the problem, pointed out by our supervisor, likely lay in the fact that the way we separated our data meant that the validation data was almost identical to the training data. As such we decided to restructure the way we split the data set for future experiments, we would partition each 5 minute video and its augmentations into the same set.

We also encountered a second problem when attempting cross folds validation: we were currently loading the entire training set into memory (videos 0-3, 20GB), splitting it into training and validation sets and then shuffling each of the sets in place. Using the *fit* function of keras models, which we were using, requires the whole training set to be loaded into memory. Something that we thought was strange but manageable. While there was, for some reason no issue (except speed) when loading the data, attempting to index the array for splitting resulted in a memory error. At this point, our solution was to individually partition each fold's train and validation set, using up 140GB of space for 7 folds! While clearly a non sustainable solution, at this point in time we deemed it satisfactory.

Now we were finally in a position to conduct a useful experiment, using the same parameters as listed before. The results we obtained can be seen in Table **??** and appear to be reasonable. However approximately 13.8% of the total frames are that of seizures, the percentage for these folds being ...%, meaning that these results could be achieved by simply never saying that a seizure was occurring.

In light of this premise, we decided to balance the folds such that approximately 50% of each fold (training and validation) contained seizure frames. The obvious draw back here is that we are reducing the size of an already small data set, however we simply were interested to see the results. Indeed, from these experiments, we found that the accuracy rate was ...%, proving that something more sophisticated would be required.

4.3 Recurrent Neural Nets

The next experiments we decided to try were to use RNNs, specifically using Long Short Term Memory (LSTM) layers []. The logic here, as mentioned previously, was that the recent presence of a seizure was likely to affect the probability of a seizure in the current frame. LSTM layers have internal states that feedback as input along with the next set of features. At the start of each batch, the internal state is reset. It can be configured such that the states only reset when manually instructed to do so, however we deemed that this would have little benefit given that the chunks where not continuous anyway.

The experiments we ran used models with similar architecture as before except using LSTM fully connected layers instead of the dense layers, again with 512 hidden units. The final layer was not LSTM layer and was only a fully connected layer, again with a sigmoid layer. We also switched to using batch normalisation [] instead of dropout as a form of regularisation. This was always the intent as its use is well supported [], we simply wanted to get earlier models up and running sooner. This model obtained a mean accuracy of ...% across 4 folds.

We expected that we would obtain a higher accuracy, it wasn't until later that we realised that perhaps our error was in using LSTM layers for the hidden layers and not for the output layer. The logic behind this being that features extracted where there was a seizure might not necessarily be the same in other frames where there are seizures, however the hypothesis would hold at the highest level if nowhere else. We then attempted an experiment using the original fully connected layers but using an LSTM layer at the end, outputting the estimated probability. This gave us a mean accuracy of ...%.

4.4 Convolutional Neural Nets

We now moved onto testing the performance of CNNs on the raw frame data. If we could build CNNs that outperformed or performed similarly to the histogram data, this would be a preferred option. Using a CNN would removed the otherwise necessary preprocessing from the live pipeline and save computing resources.

Since we were no longer using the histogram data, the flipped data was now of use to us and so we included this in the training data. Unfortunately, this now resulted in our huge (now 22GB) files causing memory errors upon load. We reasoned that, sine this was a relatively small data set, it was ludicrous that keras would not be able to handle such a set, an intuition that proved correct.

The *fit_generator* function allows you to pass a generator instances to keras that is called each time a new batch is required. This had obvious implications in offering us much more control over how we structured our data on disk and how it was used to train. We decided to store the data relating to each chunk in its own hdf5 file, with a separate earray for each transformation and for raw frame data and histogram data, a final array for the targets of the video. This decision not only saves space on disk by avoiding repetition, it also means that minimal data is needed to be stored in memory, leaving sufficient space for the training process. Since the data can be retrieved while a training on a different batch, storing it in a queue, the impact on time is minimal.

The generator class that we designed would take a list of files and transformations, taking the cartesian product and for each produced combination distribute the index of frames that form the batch, there is the option of taking contiguous indices or not. These are then all shuffled to create a list of batches, ensuring that there are no repetitions of batches per epoch. Of course, there is the draw back that each batch only comes from one chunk and transformation but we deemed that accessing multiple files for one batch would be inconvenient. For some reason, when more than one hdf5 file is opened at a time, a segmentation fault occurs, suggesting perhaps that the library is not thread safe. To get around this we passed the same lock to the training and validation generators. Other than this problem, the solution was well suited to our problem.

With this problem overcome we trained a model consisting of .... This model achieved an accuracy of ...% which is of course much higher than the previous models. This suggested to us that we either needed significantly more complex models to train on the histogram data or that perhaps CNNs were the way to go.

Regardless, with these results, we were confident that we could improve upon them with some tweaking and the development of more complex models.

Chapter 5

Combined Neural Nets

In this chapter we will discuss the more complex NNs that we developed to attain a higher accuracy in detecting seizures. As before all of the folds consist of 10% of the training data used as a validation set, and the models are tested over the same folds.

To do Graphs (**??**)

5.1 CNN with LSTM layers

The first experiment we attempted was attaching an LSTM instead of the fully connected dense layer at the end of the CNN model. This performed reasonably predictably with an average accuracy of ...%, over 4 folds. As before, we also trained the CNN models with a single LSTM layer as the output layer which performed with an average accuracy of ...%, again over 4 folds.

To do Discussion (**??**)

5.2 Combined Models

Taking advantage of the fact the optical flow preprocessing is a non linear transformation, we next attempted to build a model that consisted of two inputs. One would be the raw frames into convolutional layers, and the second input would be to densely connected layers. These models would then be be merged together and fed through a final couple of layers. This was an attempt to augment the data that hopefully would effectively double the available data. The dataflow of the model can be seen in ...

To build this model, we were required to make use of the functional model API provide by keras [**?**]. This is a very powerful tool offered by keras that allows a model to have multiple paths, including multiple inputs and outputs at any point. The dataflow can be split and restructured in any way and the back propagation will be handled by keras as normal. The API functions by effectively applying models as functions to inputs subsequent outputs, to form a fully formed model.

Using this new combined model and using a LSTM layer after the merging of the two models, we achieved an average best accuracy of ...% across 4 folds. Without the LSTM layer we obtained an average best accuracy of ...%.

We also vary the number of layers around the model attempting to increase performance ...

To do what modifications (**??**)

5.3 Histogram Data

After using the combined model, we attempted some more experimentation on the histogram data. These experiments were much easier to perform as the required much less time to train. The architecture changes that we used varied the number of layers as well as the number of hidden units per layer. Results of these experiments can be seen in ...

Bibliography

Appendix A

Logs

Appendix B

Code