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 the plan was that 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 from an MSc project completed previously, whereas it would require the development of tracking system (a separate undergraduate project) to extract bounding boxes for grooming rodents. We then intended to explore the grooming data using the techniques that we found to be effective when analysing the seizure data, however we found that the project that we were building upon had several extensive errors that we believe devalued the results, see Chapter 2. As this was the case, we reevaluated the problem and provide results that exceeded the performance of the MSc project. Unfortunately, we did not get the time to explore the grooming data, however we believe that with the pipeline we have developed it would be simple to provide a working model given the bounded grooming data.

The main idea behind the MSc project was preprocessing the video data, as described in Section **??**, and using the LibLinear [**?**] package to obtain a classification model. We discuss the problems with the project and how we adapted and improved upon the project in Chapter **??**.

We instead used the more powerful classification tool of neural networks (NNs) and the python packages tensorflow [**?**] and keras [**?**] which we discuss in Chapter 4. We investigated the same preprocessing techniques used by the MSc alongside using convolutional neural networks (CNNs) and further combining the two. The techniques and models we used proved to be much more effective at solving the problem (...). This process is discussed in Chapters 4 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 abstracted data science problem of developing classification models, 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. We discuss the design and implementation process of the pipeline in Chapter 6.

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 (**??**)

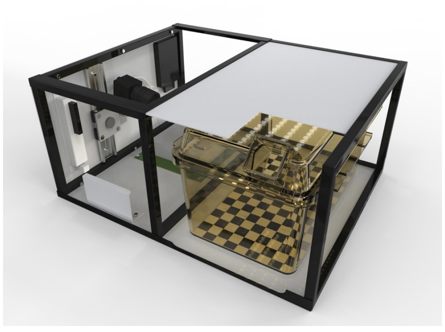
1.2 Previous Project

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, our feedback was that 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 key decision points during our project.

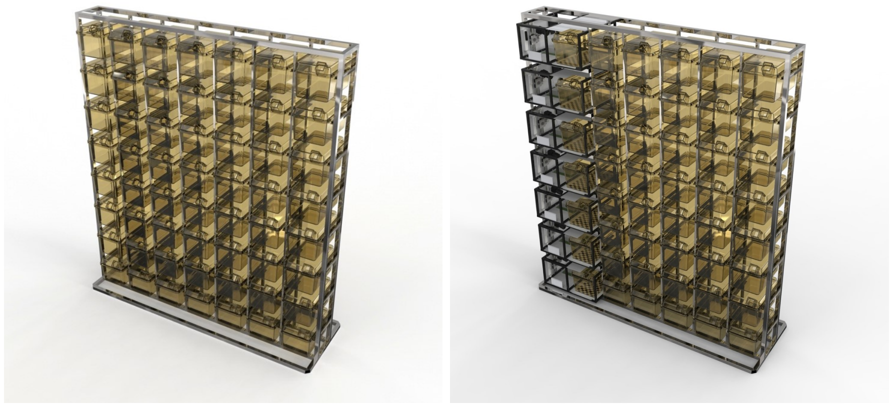
To do more here (**??**)

1.2.1 HCA System

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 has a checker board pattern such that the location of the animals can be identified. The



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

Figure 1.2: Full racks

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 needed to be catered for, where the machine would capture video segments of a desired length for the duration of the experiment.

The product 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 the 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 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 explicit evaluation. Regardless of this lack, the project was successful overall and we achieved what was set out to do. One 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.

1.3 Process Outline

For the sake of clarity, it is beneficial to explicitly outline the entire process of this academic year. This is provided along with a brief description of the work done at each step.

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

- Initial analysis of MSc project and seizure data
- Preprocessing of seizure data
- Baseline experiments
- Data restructuring
- Development of models
- Testing of best models
- Development of pipeline to process live videos

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, the second rat is not considered in annotations. 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 having a seizure the rats will sometimes rear up onto their hind legs and their arms will start to spasm while they continuously fall over and rear up again; other times they will be completely still.

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 seem reasonable, there are seizures present in the annotated rat for approximately 13% of the total frames. This could potentially mean that there is not enough data to create a solution that generalise well to unseen data. 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 preprocessed the data using the methods detailed in [**?**]. These methods, which we discuss in depth in Chapter **??**, involve taking gradients of image brightness, optical flow and motion boundaries across frames. They then used the LibLinear package to obtain a linear fit on the data using cross fold validation.

Regarding 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). 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. For example, no distinction is made between a rat that is grooming and a rat that is dead, and both would be considered to be having seizures.

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 on to expansive NNs which can be used for classification problems, among others. Moving to this architecture, even if following the same preprocessing 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 **??**.

It also seemed remiss that little 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. Recurrent models would maintain some information of the frames across time to aid in classification.

We attempted to implement this by way of using recurrent neural nets (RNNs), as discussed in Chapter 4.

Further more, they only ever used one set of histograms and only the histogram of gradients (HOG) and histogram of optical flow (HOF). The paper [**?**] also suggested extracting the motion boundary features which we decided to include, concatenating all three sets of features to form a larger one for each frame.

Lastly, our main criticism of the work was the way that they evaluated their results, regarding the preprocessing methods used to extract the histogram of gradients and optical flow. The code allows for the histograms to be averaged and grouped across frames. One drawback of this is that it makes it less useful when implementing in a live context, as you must wait for several frames before processing. More importantly they do not state how they adapted the targets to match this coalescing. For example, if the features have been averaged across 10 frames, must all of them contain a seizure or just one? Or is the target an average? In their best performing model, they have indeed averaged over 10 frames at a time. In our opinion, without specifying what the targets are, it is not possible to evaluate what the results actually mean.

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

Preprocessing

In this chapter we will discuss the methods that we used to preprocess 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 to 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 across a vertical mirror line, as well as rotating both the original and flipped video clockwise and anti-clockwise 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

The method we used to extract features from the data was as in the paper [**?**], namely extracting HOG/HOF/MBH feature blocks using the code that they provide in the paper. We modified the code such that one line in the block would correspond to a single frame such that we could easily use the provided targets. The process involves treating a video as a group of blocks, in our case we used block sizes of 8 x 8 x 1 pixels.

Calculating the HOG of an image first involves calculating the brightness gradients for each pixel. In this case, this is approximated by convolving the image with the Sobel kernel, Figure **??**, with the image to find the horizontal and vertical gradients which are used to find the magnitude and direction of the gradient. Within each block, each pixel casts a weighted vote into a number of histogram channels based on orientation. The magnitude is divided between the two bins that the direction falls between, linearly weighted to the closest bin.

Calculating the HOF of an image involves estimating the optical flow of an image, that is to say finding the solution to the constrained problem:

$$... \tag{3.1}$$

Where ...

The method used in the paper, as we also used, is the Horn-Schunck method. This involves computing I_x, I_y and I_t using convolution, then compute the average velocity at each pixel, assuming 0 initially, iteratively solving for u and v.

Now the magnitude and orientation of the total optical flow for each pixel is obtained, which are once again binned into histograms as before.

Motion boundary histograms are similar to HOF except that they account for camera motion and other variables between frames. Although the camera is static for these videos, as the video is cropped, we believe that it could still provide some benefit.

Now having obtained these histograms, ”Let R be an $N \times M$ matrix containing responses in a single orientation (be it gradient magnitude or optical ow magnitude). Let B_N and B_M be the number of elementary blocks from which HOG/HOF features are composed. Now it is possible to construct (sparse)matrices O and P of respectively $B_N \times N$ and $M \times B_M$ such that $ORP = A$, where A is a $B_N \times B_M$ matrix containing the aggregated responses for each block. O and P resemble diagonal matrices but are rectangular and the lled in elements follow the diagonal of the rectangle instead of positions (i,i).” [**?**]

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 4.

We 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. After multiple attempts, 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. Learning of the package PyTables [**?**] was helpful as it would allow us to create HDF5 files, consisting of arrays, that are specifically designed to store information that would 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.

We further discuss our error in not balancing the data sets such that there are drastically more negative targets than positive, how we remedied this, and subsequent baseline tests. We do not display our initial results in a table or graph due to this fact, see Section 4.5

For clarification, each model is evaluated across four folds (unless otherwise specified), the folds initially each contain data from 10% of the files (later 10% of training data when specified). The folds are the same over each experiment so that they directly compare. The reason that we used four folds was simply to save time training.

Results are reported as the mean of the best performance over each fold, across 3 trials, along with error bars of standard deviation across folds. Occasionally performance on individual folds is discussed when we deem it noteworthy.

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 a relu [?] activation functions and batch normalisation [?] 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 found that the optimal learning rate appeared to be around $1e-3/\sqrt{E+1}$, where E is the current epoch number. We also utilised early stopping to evaluate the performance of a given model on its best performance over training.

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. With this model we attained an accuracy of $88.90\pm8.33\%acrosstwoexperim$

4.3 Convolutional Neural Nets

We moved onto testing the performance of CNNs on the raw frame data. If we could build CNNs that outperformed or performed similarly to the models trained on the histogram data, it would remove the requirement of 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 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, see Figure **??**, that achieved an accuracy of $88.51\pm6.71\%whichisonlyslightlylessthanthebasicmodelsonthehistogramdata, butwithlessvar$

4.4 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

Fold	Chunks Contained	Number of frames
0	000000-52500	4272
1	000002-30000, 000000-75000	4272
2	000003-60000, 000002-52500, 000003-67500	4806
3	000001-75000, 000001-45000, 000001-00000	4806

Table 4.1: The balanced folds that models were validated on. Chunks are described by their [video-number]-[Beginning frame].

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. This model obtained a mean accuracy of 88.74±7.61%.

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 best accuracy of 87.83±6.95%*acrosstwotrials*.

At this point, we became incredibly suspicious that all the results were so similar and we subsequently realised why the recurrent element was not aiding the model. After looking at our true positive rate, it was only 3.94±2.64%*forthemodelwiththeLSTMoutputlayer.Withahitrateth*

4.5 Data Balance

Following the realisation of the deceptive performance, we decided to cull our data set even further. Even though this would mean an even smaller data set, if left without such a change, the loss from the frames without seizure would massively outweigh the loss from the positive frames during training, resulting in the model simply being trained to say that there is no seizure in any frame.

We wanted to restructure the data such that accessing contiguous frames would still be possible so that we could still use RNNs, and we of course wanted to keep all of the frames containing seizures. The method we used was to divide each chunk into sections of length 534 then keep each section that contains at least one seizure and additional random sections such that approximately 50% contain seizures.

Model	Fold 0	Fold 1	Fold 2	Fold 3	Mean
Dense Last Layer	0	28.09	48.28	59.50	33.36±26.13
Full LSTM	1.74	16.20	28.21	36.71	20.71±15.19
LSTM Last Layer	24.82	56.10	24.24	40.10	36.31±15.08
SimpleRNN Last Layer	11.68	20.63	55.92	47.39	33.91±21.11
CNN	1.50	18.09	51.10	39.22	27.48±22.05

Model	Fold 0	Fold 1	Fold 2	Fold 3	Mean
Dense Last Layer	47.74	58.39	75.76	73.82	63.93±13.30
Full LSTM	47.42	52.95	70.49	67.09	59.49±11.06
LSTM Last Layer	66.82	55.52	68.12	63.00	63.37±5.66
SimpleRNN Last Layer	52.69	53.56	75.93	74.07	64.06±12.66
CNN	48.97	52.93	72.08	64.33	59.58±10.58

Table 4.2: Performance of various models on histogram data to two decimal points, the first four models all consist of 2 layers of 512 units trained on the histogram data, see Figure **??** for the CNN architecture. The top table is the true positive rate, the bottom is the accuracy.

Using this new balanced data set, we evaluated the initial NN consisting of 2 fully connected layers of 512 units. Again over 4 folds, as expected, our accuracy was much lower at 66.83% but our true positive rate was 47.242%, a huge improvement. We also noticed that the variation of performance across folds was very large with a standard deviation of 18.28% for accuracy, and 36.32% for the true positive rate. Although we could be sure that the sets were balanced, we realised that the folds could now differ in size which we decided to correct by ensuring that the folds also contain the same amount of data. The files present in these folds can be seen in Table **??**

After balancing the folds, we ran the same experiment twice on four new folds. The true positive rate was 33.36±0.89% across the mean result of both experiments, with the std dev across fold being 26.38%, still extremely high. Most significant that on both experiments, the true positive rate on the first fold was 0%. Looking at the fold, the data only comes from a single file, we reasoned that perhaps this contains footage of a rat having a type of seizure that was not found in the rest of the data, in which case, training a model with some element of recurrence would be absolutely required.

Using the same CNN model as before we obtained a true positive rate of 27.48±22.05%, *slightlyworse*
*zeroperformanceonfold*0.

We also performed the same RNN tests as before, the results of which can be seen in Table **??**. Similar to before, using full LSTM layers did not seem to be effective, whereas using an recurrent layer as the last layer had clear benefit. Most importantly on fold0, the last layer LSTM achieved 24.82% while the SimpleRNN cell achieved 11.68%, suggesting our intuition to be correct.

When we looked at the details of the seizure in fold0, we found it to be an absent seizure, characterised by the rat being effectively still. It was the longest bout of an absent seizure that we had data for but not the only one. This fold would continue to be the most difficult to evaluate, often the models would either perform reasonably well or terribly across multiple test.

4.6 Discussion

After some initial blunders, we had restructured our data and obtained some baseline results. We could now subsequently move onto optimising our models. Even given the misleading performance, we were confident that we had space to develop models that could greatly improve upon these results.

Chapter 5

Continued Experiments

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. Again, the result reported are the mean of the best performance, of 3, over each fold along with error bars of standard deviation across folds.

To do Graphs (**??**)

5.1 More Complex Models on the Histogram Data

Moving forward from our initial tests, we attempted to vary the complexity of the models trained on the histogram data by varying the number and size of the hidden layers. We maintained the final LSTM output layer as it had proven to be somewhat essential.

The results of these experiments can be seen in Table **??**. After repeated tests, it appeared that using larger layers resulted in much better performance. However, it was difficult to make out what architecture choices would result in consistently better performance as it tended to vary reasonably wildly across tests, especially if you discount the one off good performances on fold 0. We believe that, should the model detect some seizure frames correctly, this would carry over to accurately predict subsequent frame, the issue being that the initial detection often didn't occur.

We also tried lowering the learning rate in an attempt to stabilise the results, however this results in significantly worse results across the board.

5.2 Tackling Absent Seizures

The models that we trained repeatedly attained approximately 0% true positive rate on fold 0. With the results from the baseline, we knew that it was possible to obtain

MODEL	FOLD 0	FOLD 1	FOLD 2	FOLD 3	MEAN
2x512	24.82	56.10	24.24	40.10	36.31±15.08
3x512	0.00	41.73	27.17	32.22	25.28±17.90
4x512	31.23	15.43	36.11	26.95	27.43±8.83
5x512	3.95	48.03	45.88	35.00	33.217±20.33
2x1024	56.30	45.92	27.18	81.18	52.65±22.52
3x1024	23.98	58.89	19.20	49.29	37.84±19.27
2x2048	24.27	65.28	25.60	74.24	47.34±26.14

MODEL	FOLD 0	FOLD 1	FOLD 2	FOLD 3	MEAN
2x512	66.82	55.52	68.12	63.00	63.37±5.66
3x512	48.36	70.40	64.04	65.92	62.18±9.59
4x512	65.53	57.19	73.30	63.12	64.79±6.67
5x512	48.19	64.10	73.85	73.52	64.92±12.03
2x1024	80.92	55.20	67.32	70.01	68.36±10.56
3x1024	52.60	68.27	70.22	76.35	66.86±10.11
2x2048	60.27	71.53	63.91	80.69	69.10±9.04

Table 5.1: Performance of models on histogram data to two decimal points, the model columns tells the number of layers and the size of each layer. The top table is the true positive rate, the bottom is the accuracy.

at least 24.82% and we deemed it reasonably necessary to train a model that could reliably produce non zero results.

We realised that perhaps the most telling factor of these absent seizures were the fact that the rats did not move within the cage, an aspect that was not captured by our current feature set. To remedy this, we would experiment with feeding the difference in coordinates from the previous two frame, normalised to the interval [0,1], into the NN.

To do this, we made use of the functional model API provided by keras [**?**]. This is a very powerful tool 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.

We ran two sets of experiments using a 2x1024 model: one where where the coordinate features are concatenated with the output of the NN into a fully connected layer of 256 units before the final layer; another where they are appended to the initial feature set, while keeping in the 256 unit layer for posterity. The results of the models are visualised in Figure **??** where they are compared against a similar model that does not use the coordinate features. Clearly, using the new features does help to some extend, most notably there was never a 0% true positive rate, while attaching them at the end of the model appeared to be slightly more beneficial.

At this point, we also realised that during training for evaluation on fold 0 the true positive rate on the training set would also be incredibly low. In light of this, we raise the learning rate by a factor of ten. This unfortunately did not lead to any noticeable stability increase, again with failed training occasionally cropping up. We decided ultimately that the models would occasionally get stuck in local minima during training and when training the models for the final testing, we would simply need to retrain if this appeared to be the case. This is the reason that we decided to report the best performance on each fold, luckily we had saved full logs of each experiment.

Regardless, we would maintain the addition of the coordinate features in future experiments.

5.3 CNN Models

Next we wanted to further explore the viability of using CNNs on the raw frame data, using some of the knowledge we had gained from the previous experiments. Our first attempt at this was using a similar model architecture as in the baseline experiments except feeding the coordinate features into the model after flattening the data. Additionally, we used LSTM layer as the output layer and used the new learning rate of 1e-2. This was quite successful across all folds, giving a true positive rate of

We also attempted increased the complexity of the CNN by adding more layers and increasing the number of features being detected. The results from these tests can be seen in

The comparison between best accuracy achieved can be seen in Table **??**. Taking this into account suggested that CNN models were perhaps the superior choice to the histogram models, convenient given that it would avoid the use of the preprocessing in the live pipeline.

5.4 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 our best performing linear model on the histogram data. These models would then be be merged together along with the coordinate features and fed through a final couple of layers. This was an attempt to augment the data that hopefully would effectively double the available feature set. The dataflow of the model can be seen in ...

Using this new combined model, we achieved an performance of ...%.

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

Chapter 6

Constructing the Pipeline

In this chapter, we will discuss how we built a pipeline to take a video input feed and to detect in live time whether a seizure or grooming behaviour was present, depending on the weights used. We first discuss our intentions and the possible problems, then our implementation decisions, and finally any problems that arose and how we approached them.

6.1 Requirements Analysis and Design

To design the full pipeline, there were several main features that we identified as core requirements:

- The system must be able to process a video stream in live time
- The system should will likely be easily used in a slightly different context - we are using prerecorded video and the bounding boxes may be in a different format etc
- The system should be as light weight as possible as will be used alongside other applications on the HCA system

Given these core aspects, we considered the main concern to be ensuring that the pipeline was able to keep up with the frame rate of the video input while not being intensely resource heavy. This is mainly due to the fact that the histogram features must be extracted from each frame before being fed into the prediction model. We did believe that it important to make sure it did not lag behind the feed above all else and so reasoned to make it as efficient as possible.

We also realised that it would be necessary to translate the Matlab code used to extract the histogram features into Python. Calling the Matlab scripts as external processed on each frame would be far to expensive, this led to several more problems which we discuss shortly.

Another feature that we deemed useful would be the recording of bouts of seizures.

This would mean making note of when a seizure starts and finishes and recording the time between the two. To make this system more robust, it would perhaps be wise to consider monitoring a window of frames and start and stop the bout when certain thresholds are met within the window. For example, declare the start of a bout when 3 of 5 frames are detected to be seizures, and declare the end of a bout when 4 of 5 frames are detected to not be seizures.

It would be important to make the pipeline as modular as possible, as is good practice regardless, such that any future use could plug in slightly separate models to fix the context. Additionally, a problem specific to the context we are using it in is that we only have bounding boxes for specific chunk that also have some large gaps in between. This means that special care would be required when managing frames. Compounding this, using the optical flow also requires using two frames.

Given these problems, we reasoned that it would be very important to make use of efficient multi-threading. This means dividing the processing into separate parts that can each be handle by a thread, managed by a main process. Making use of FIFO stacks, we could ensure two things: if any one thread is slowing the system down, the other threads would still be able to function by providing a buffer of frames, even skipping frames if necessary; the threads could work of an infinite loop, avoiding the costly overhead of spawning new threads. The separate parts that we identified were as follows:

- Retrieving frames
- Retrieving bounding boxes
- Cropping frames given bounding boxes
- Extracting histogram of gradients
- Extracting histogram of optical flow
- Extracting motion boundary histogram
- Detecting seizures or grooming
- Processing input using models, one thread for each model
- Display event detection
- Splitting of any outputs to multiple destinations, eg. frames to models and histogram processing

We reasoned that the retrieving and cropping frames could be performed by the same thread, however each of the other tasks should be performed by a dedicated thread to ensure that the live processing is achieved. An added benefit of specifically separating the tasks as such is that each one can more easily be replaced given a different context. Using FIFO stacks would allow for easy integration between the threads.

We decided that the program would load information about the model to used, along with weights, from the directory. This could be provided as an option from a list, where the program would attempt to load the default model for the option, or a list of names of models could be provided, where the pipeline would load each model and feed the frame through each of the models. This would allow users to detect any number of features that they had trained models to detect simultaneously without processing the stream multiple times.

6.2 Implementation Process

The first part of the process that we decided to tackle was translating the Matlab code into Python code. We did not anticipate this to be difficult and mainly an issue of translating individual lines of code. This proved to be incorrect, as some of the code was hidden behind Matlab engine and so was difficult to emulate exactly. This was the case in the *Diag_Linear_Matrix.p* code but this was easily avoided by hard coding the required matrix for a 64x64 image.

Where this posed a particular issue was for estimating the optical flow between two images. The method that we used to process the chunks was the Horn-Schunck method [], and although the Matlab page for *OpticalFlowHS* [**?**] describes the parameters used in the Matlab implementation, we could not get the same results for a given pair of frames. We attempted to modify the implementation slightly, in an attempt to find a similar results by way of trial and error but this also proved fruitless and we determined that there must be some additional processing going on beneath the hood. The decision we ultimately came to was resorting to the method proposed in the original paper as this provided the closest results that we could obtain. Unfortunately this would surely impact the accuracy of the model predictions given that the feature set would be slightly different, however there wasn't much we could do at this stage. An obvious fix would be to reprocess the data and retrain the models on the new features, however this would prove to be too time consuming to complete in time. Instead we used the method proposed in the original Horn-Schunck paper and hope that the features were close enough as to still be applicable, at least we would still have the completed architecture to be used. Fortunately, it would still be possible to use the CNN models to display the full pipeline in action.

On a more positive side, we we able to make the code more efficient by making use of the numpy functions *einsum* and *as_strided*. Where the Matlab code would iterate over multiple matrices performing similar operations in a loop, we would perform the entire set of operations in one swoop, making use of the matrix optimisation in the numpy library.

Another problem that we uncovered during the testing of our implementation was that as the decoders used by opencv and Matlab were slightly different, the pixel values they extracted from the same video were slightly different. There was again nothing that we could do here save retrain our models on the new data. However, as the difference was quite small, we remained hopeful that this would not impact detection accuracy too much.

We decided that outputting the number of seizures detected in the past x frames to a graph would present the easiest way to visualise the procedure. Having implemented and tested the full pipeline, we now had to find the optimal parameters to detect bouts of seizures using a detection window. Using a cnn model trained on one of the folds, we found that using a window of size ...

Chapter 7

Conclusion

At the end of this project, we had developed a system that could detect the seizures in rodent with an accuracy of ...%. By considering the detection over a group of frames, we were able to detect bouts of seizures with an accuracy of ...%. This result was a clear improvement over an MSc project that we based our studies upon and efficiently critically deconstructed, as documented in Chapter **??**.

We were able to analyse the problem and established baseline results that further clarified the problem, Chapter 4. We identified key problems that arose from these initial experiments and devised solutions to tackle them, using scientific practice, evaluating our own work and understanding at each step. Moving forward from these experiments, we further explored possible solutions to increasing the accuracy in seizure detection, Chapter **??**. Finally we tested our most promising models on our withheld test set in Chapter **??**.

Having obtained our final set of models, we developed a system that could perform the detection in a practical context, in real time. We analysed the requirements of the system and implemented a design that we believed captured these requirements. Using this system, we were able to improve upon our detection accuracy to ...% by considering bouts of seizures as opposed to individual frames.

7.1 Reflection

Looking back over this year, I think that while we achieved an accuracy rate over ...%, there is a lot more that we could have explored with regards to deeper models. We did attempt to make use of some GPU clusters but we were unable to make our models work properly and decided to prioritise on using what we already had. Perhaps given the time again, we would not have taken this route.

We believe that we maintained a high level of critical analysis of our own experiments throughout the project. Not only did we look at results in an abstract manner, we considered the context of the results by looking at the data itself and proposed solutions based on this deeper understanding. These solutions proved to be effective through further experiments, demonstrating our understanding of the problem.

It is a shame that we did not get the time to explore the grooming data, we think that we could have made good progress with this data also, given what we had already learned. Given the time, this is definitely the area in which we would progress, alongside considering deeper models, as is the current trajectory for deep learning.

Considering the two year project, we have learnt a great deal, particularly about low level camera functions and design, which is an area that we would perhaps not have otherwise delved into. Learning how this integrates with the higher level aspects of computer vision problems has certainly improved our understanding of the topic as a whole.

Bibliography

Appendix A

Logs

Appendix B

Code