UNIVERSITAT DE BARCELONA

Treball final de grau

GRAU DE INFORMÀTICA

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

AUTOMATED MOUSE BEHAVIOUR RECOGNITION FOR NEUROSCIENCE RESEARCH LABS

Autor: Sara Albarran Berlanga

Director: Dr. Eloi Puertas i Prats
Realitzat a: Departament de Matemàtiques i Informàtica

Barcelona, 10 de juny de 2024

Abstract

Research studies in diverse fields, like neuroscience, usually employ experimentation on mice. Recording them is one of the most common ways of monitoring their actions and behaviours to certain stimuli. However, this videos later require professional analysis, which is a tedious and time consuming task. Current computer vision methodologies provide us with the necessary tools to build automated models that could perform this laborious and repetitive task.

This project aimed to research and develop computer vision and machine learning approaches enough to perform behaviour classification at frame level on laboratory mice records. The primary objective was to develop a tool for researchers to analyze mouse behavior, alleviating the repetitive and time-consuming manual analysis. Our focus was to align our methodologies with real-world scenarios where data varies significantly.

By examining state-of-the-art sequence analysis techniques, we identified key challenges and limitations in our data. This led to the development of tools such as frames-per-second ratio regulation and automatic mouse position detection, which improved our models' ability to handle diverse video inputs.

Through extensive experimentation and benchmarking, we designed a robust pipeline for sequence classification, achieving precision and recall rates exceeding 90% across various recording conditions. Our tool effectively adapts to different lighting, camera placements, and orientations, enhancing its applicability in real-world settings.

Furthermore, we integrated these automated models with an intuitive User Interface, providing researchers with easy access to this tool.

Acknowledgements

I would like to express my sincere gratitude to all to all of my supervisors and contributors to this work for their exceptional assistance, dedication, and expertise in bringing this project to fruition.

To Dr Eloi Puertas for giving me the opportunity of working in this project and guiding me through the course of it.

To Dra Mercé Masana, Melike Kucukerden and Maryam Givehchi for letting me work with them and provide me with the data, equipment and knowledge required for the development of this project.

Lastly, to my friends and family who have been supporting me during this journey.

Contents

Chapter 1

Introduction

1.1 Project context and motivation

Experimentation on animals, particularly mice, is a common practice employed in various fields such as neurology, histology, psychology, genetics, and pharmacology. Most scientific and preclinical studies develop their methodologies around these animals, testing different drugs and stimuli to achieve desired outcomes. The most common method of monitoring the animals' reactions is by recording them in specific environments. In most cases, the involvement of an expert is later required to properly analyze the videos and draw conclusions from the experiments. However, this process is tedious, repetitive, and time-consuming, preventing experts from focusing on other tasks.

Recent advances in computer vision and machine learning offer a solution to this issue. By implementing automated models, the task of video analysis can be delegated away from experts, freeing them from this labor-intensive process and allowing them to focus on other critical duties. Research in this field has advanced significantly in recent years. Currently, various studies have proposed machine learning-based techniques to track behavior in videos across different domains. Examples include work on human action recognition [28] and sentiment analysis [29]. There have been studies as well focused on practical animal behavior recognition [30], such as research on laboratory mouse activity [31][32], where researchers classified mouse behaviors from video clips.

This project aims to apply state-of-the-art models to develop a tool that enables scientists to analyze animal videos, at frame label, in an efficient and user-friendly manner. More specifically, the projects builds itself around the real use case of mice videos utilized by a neuroscience research group at the Hospital Clínic, Barcelona, lead by Dr Mercé Masana.

1.2 Objectives

The global objective of this project is to develop a tool for neuroscience researchers that automatically classifies mouse behaviors in videos at a frame level. For that we wanted to conduct research on current sequence analysis and classification methodologies, so we

could implement them. Also, this project follows up the previous work realized by Xavier de Juan on his Master's thesis *"Automated Mouse Behavior Recognition using LSTM and TCN Network"*. Given his works and results, we had the purpose of improving them under the use case and data provided by the research team in Hospital Clínic. We wanted to develop a pipeline that would adapt to their investigations and boost up models performance. One of our goals was to develop models that would generalize effectively across different recording conditions, making them adaptable to real-world applications.

To make the machine learning based pipeline accessible for laboratory researchers, we wanted to implement it on an easy to manage user interface. To make it easy to use, we wanted to deployed it as a Docker container, so it becomes easy to install and execute in any computer with the required computing capacities.

1.3 Document structure

The following document collects and explains all the work undertaken on this automated mouse behavior recognition tool. Its content is organized as follows: Chapter 1 serves as the introduction. Chapter 2 reviews the previous work done on the project as well as it explores the theoretical bases of the methodologies used. Chapter 3 presents the data used. Chapter 4 and 5 explain the preprocessing techniques applied to the data. Chapter 6 talks about the sequence processing architectures applied for video frame classification. Chapter 7 exposes the obtained benchmarking results among preprocessing techniques and classification models. Chapter 8 discusses the implementation of the user interface and its deployment using Docker. Finally, Chapter 9 presents a global conclusion of the work done.

Chapter 2

Related works

On this section we will first explore the previous work completed on the project. We will review the methodologies that were applied and analyse the extracted results and conclusions. After that, we will dig in in the theoretic bases of the methodologies we will apply and test for feature extraction and sequence classification, in Chapters 5 and 6.

2.1 Previous state of the project

2.1.1 Context

This project builds upon the previous work realized by Xavier del Juan on his Master's Final Thesis, *"Automated Mouse Behavior Recognition using LSTM and TCN Networks"* (X.de Juan, 2022). On his project, he explores different Recurrent Neural Network based architectures to perform behaviour classification at frame level on mice videos recorded by a research team at the Hospistal Clínic of Barcelona, led by *Dr. Mercé Masana from the Deparment of Biomedicine and the Institute of Neurosciences, Faculty of Medicine and Health Sciences, University of Barcelona*. On his thesis, X. de Juan builds a first pipeline that serves as a base for the current project.

X. de Juan counted with 17 videos, each of 11 minutes. Each video recorded a mouse placed in an empty white box, as it can be appreciated on *Image 2.1*. They were all took on the same conditions, having the same camera placement, lighting and frames per second rate. The purpose of his work was to develop a machine learning based pipeline that, given a video, was able to return a csv file with the mouse behaviour at each frame.

On his project, he distinguished up to 3 different behaviours: Grooming, Mid Rearing and Wall Rearing. We consider the mouse to be grooming when it licks its fur, it grooms with the fore-paws, or it scratches with any limb, as shown in 2.2A. On the other hand, we consider a rearing occurs when the mouse puts its weight on its hind legs, raise its forelimbs from the ground and extends its head upwards. In Xavier's project, he differentiated between Mid rearing and Wall rearing, the first one taking place in the middle of the box, as seen in 2.2.B, and the second one being at one of the the box's walls, as in 2.2.C. He also considered a neutral case, where the mouse was neither performing any

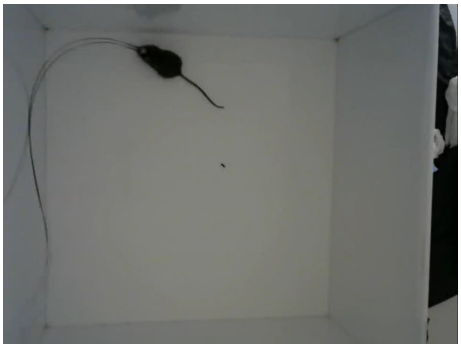


Figure 2.1: Frame extracted from a video on X.de Juan's thesis.

of the listed behaviours. The labels for each video were provided in CSVs created by the researchers at the Hospital Clínic team.

(A) Grooming (B) Mid Rearing (C) Wall Rearing

Figure 2.2: Mouse Behaviours, extracted from X.de Juan's thesis.

In addition to the videos and their corresponding labels, X. de Juan counted with data extracted from the software program *DeepLabCut*. *DeepLabCut* provides a computer vision package for animal pose estimation. Given a video, for each frame, the position of some parts of the mouse are stored in a CSV file.

2.1.2 Methodologies

X. de Juan started his pipeline by preprocessing the given videos. For that, he cropped each frame in a window of 160x160x3 pixels, being the last ones the RGB channels, with the mouse centered, as we can appreciate in the images of *Figure 2.2*. To get the mouse positioned in the center, X. de Juan used coordinates given by the *DeepLabCut* CSV files. In these files, there were two specific columns that gave the X and Y position of the center body of the mouse at each frame. That way, he used them as a reference point and cropped each original frame around them. The idea behind this is to work with smaller and more relevant data.

After the frame dimensionality reduction with cropping, he used a pretrained Convolutional Neural Network to express each video as a 1 channel feature vector. For that,

he explored 2 pretrained architectures: ResNet50 and InceptionResNetV2. Both models were loaded with weights previously trained on Imagenet datasets. On the following subsections they would be explained in more detail. The idea behind this is to, once again, reduce data dimensionality. For instance, by processing a frame on ResNet50 we reduce its dimensionality from $\mathbb{R}^{160x160x3}$ to $\mathbb{R}^{2048}$, making sure that only the most relevant information stays on the 1 channel vector. By doing this, we will later on achieve faster calculations which would allow our Deep Learning models to converge into good results faster during training. This will also avoid possible memory problems.

Once all the preprocessing was done, X. de Juan performed a random train-test split among the videos and labels. 12 of them were used for training the models and 5 were left unseeing during training. These 5 videos will be later used to compare models and choose the best performing one.

He then defined what would be the Deep Learning architectures responsible of classifying video frames. He opted for 2 sequence classification alternatives: Long-Short Term Memory (LSTM) based Recurrent Neural Network and Temporal Convolutional Network (TCN). Given a sequence of vectors, both models output a prediction for each element. To do so, they consider the data on an element as well as the data on the previous elements in the sequence. Because of this characteristic both architectures become a great alternative for video frame classification, since they take into account the temporal factor. Both architectures will be better explained on the following subsections. What X. de Juan does with them is, given a video and its 1 channel feature vectors, he splits them in sequences. Then he passes these sequences through either the LSTM or the TCN architectures. As an output he obtains the classification for each vector, that corresponds to one frame. In other words, the sequential architectures will give us the classification for each frame of the inputted sequence. After passing all the sequences that conform a video, we will obtain the classification of each frame in the video. This would be our desired output. Previously, we mentioned that X. de Juan considered up to 3 different mouse behaviours, other than the neutral mouse state. It is worth to mention that he developed 3 different classification models, one specialized for each behaviour. Therefore, each model performed a binary classification, where label 1 corresponded to the action and 0 to the mouse neutral state.

2.1.3 Results and conclusions

X. de Juan performed a benchmark of the following 4 architectures: ResNet50 + LSTM, ResNet50 + TCN, InceptionResNetV2 + LSTM, InceptionResNetV2 + TCN. In other words, he extracted feature vector of each frame with both of the pretrained Convolutional Neural Networks. Having data processed in both ways, he tested fitting them into the 2 sequence classification alternatives: LSTM and TCN. To define the LSTM and TCN networks architectures he performed an hyper-parameter tuning using the Grid Search technique. We understand hyper-parameter tuning as the process of optimizing the the settings that control the training of a machine learning model to improve its performance. Unlike model parameters (e.g. weights in a neural network), which are learned during training, hyper-parameters are set prior to the training process and control how the model is trained. Grid Search is a hyper-parameter tuning technique used in machine learning to find the best combination of hyper-parameters for a given model. It defines a grid with all possible val-

ues for each specified parameter. That, creates a matrix where all possible combinations are represented. Then, the algorithm iterates over all of them. As a result we obtain the combination that gave the best model's performance. Some of the parameters that X. de Juan included in his hyper-tuning are the number of TCN or LSTM layers, the sequences length, the batch size, etc.

After finding the best parameters for all 4 models, he trained each one of them for 50 epochs. Once trained, X. de Juan benchmarked them by analysing their performance on the 5 unseen videos, as we previously mentioned. On the following tables there are displayed the metrics results for all 4 models on the 3 different behaviours. The analyzed metrics, as appreciated on the tables, are: Binary Accuracy, Precision, Recall and PRC, which stands for Precision-Recall Curve. For a better explanation of their meaning you can check Chapter 7 where we discuss the meaning of all the metrics used for model evaluation.

	ResNet+LSTM 2	ResNet+TCN 3	Inception+LSTM	Inception+TCN
Binary Accuracy	0.902	0.909	**0.950**	0.942
Precision	0.412	0.426	**0.681**	0.577
Recall	**0.946**	0.876	0.537	0.635
PRC	0.679	**0.690**	0.588	0.586

Table 2.1: Grooming results

	ResNet+LSTM 2	ResNet+TCN 3	Inception+LSTM	Inception+TCN
Binary Accuracy	0.897	0.902	0.887	0.**912**
Precision	0.518	0.535	0.415	**0.592**
Recall	**0.797**	0.748	0.104	0.596
PRC	**0.702**	0.691	0.201	0.602

Table 2.2: Mid Rearing results

	ResNet+LSTM 2	ResNet+TCN 3	Inception+LSTM	Inception+TCN
Binary Accuracy	**0.918**	0.914	0.840	0.888
Precision	**0.671**	0.667	0.433	0.569
Recall	**0.785**	0.737	0.570	0.739
PRC	**0.792**	0.770	0.411	0.716

Table 2.3: Wall Rearing results

After the analysis, X. de Juan concluded that ResNet LSTM seem to be the most suitable model. He stated that although both kind of networks, LSTM and TCN, showed similar results, the LSTM one was chosen by its ability to minimise the number of false positives, as appreciated by precision values.Regarding the pretrained models, the difference between results was also quite small favoring ResNet most of the times.

2.2 ResNet 50

ResNet, which stands for Residual Network, is a kind of Convolutional Neural Network (CNN) architecture first presented by Microsoft Research in 2015 on the paper *"Deep Residual Learning for Image Recognition"* by K. He, X. Zhang et all [1]. ResNet architectures are widely used in image classification, object detection, and segmentation tasks, significantly improving accuracy in computer vision applications. They are applied in many domains, like medical imaging, autonomous driving, and facial recognition due to their ability to handle deep networks effectively. What differentiates ResNet from classical CNN, is their architecture features that addressed the vanishing gradient problem, present in classical neural networks. On the following subsections we will explore what the vanishing gradient problem is and how ResNet addresses it.

2.2.1 Vanishing Gradients

The Vanishing Gradient is a phenomena caused during models error backpropagation. Backpropagation is the most common method used during model training. Its purpose is to update the weights and biases of neurons in neural networks enough to make them converge into good results. Converging into good results implicates minimizing the loss function of the output. To do so, we calculate the gradient of the loss with respect the output of the network. The algorithm then adjusts the output layer weights in regard this gradient. This same procedure is applied iteratively to the previous layers, until reaching the first one. We call this error propagation, where each layer regulated it's weights regard the gradient of the loss respect to that layer's parameters. By repeating this process we achieve the minimization of the error.

This methodology however presents a limitation, gradient can become extremely small as it propagates from output layer to the earlier ones. This is what we call the Vanishing Gradient problem. When this happens, adjusting parameters on the first layers becomes very hard, consequently causing a major slow down of the model's convergence into good results. This problem is proportional to the deepness of our network. The more layers it has, the more prone it is to develop vanishing gradients. Reason to this is the fact that by measuring the gradient of the loss regard one layer, we are considering the impact that layer has on the model's output. It becomes obvious that last layers will be the more relevant than the earlier ones. Therefore, the deeper our model becomes, the less impact will the first neurons have on its output and the smaller the gradients will be, reaching values very close to 0.

2.2.2 ResNet architecture

ResNet architectures confront this problem by adding what we call *skip connections*. A skip connection, also known as residual connection, is a feature where the input of a layer is added directly to the output of a deeper layer, bypassing one or more intermediate layers. These intermediate layers could be convolutions, poolings, activations, etc. To express this more formally, we can consider x as the input of a given layer in the network. We define function F as the series of transformation x goes through the intermediate

layers. In a classical CNN, the output after the intermediate layers would be $F(x)$, instead with a skip connection we would have that the output is $F(x) + x$. By doing this we address vanishing gradients, since we are allowing gradients to flow more easily through the network during backpropagation.

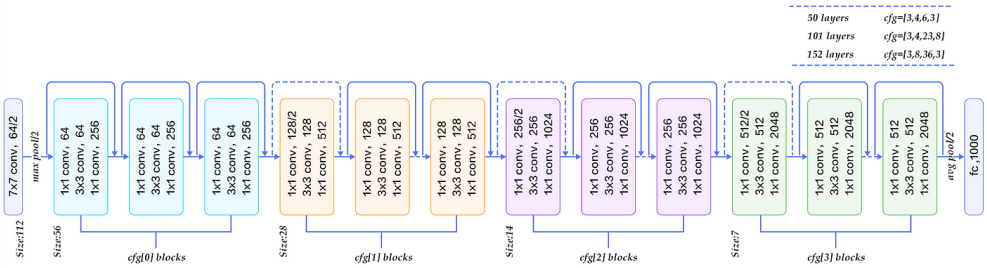


Figure 2.3: ResNet50 architecture's schema, extracted from [6]

ResNet 50 builds itself around this methodology. On figure 2.3 we can find a graphic representation of its architecture. It has up to 5 convolutional blocks, where the first one is in charge of padding the input so it has an specific shape that can be later processed by the following blocks. These following blocks follow a similar structure. They all start with a skip connection block, where instead of adding to the output the identity function of the input, as we have just explained, it performs a convolution to the input before adding it. This block is then followed by a variable number of classical residual blocks, where we add the identity of the input. In all cases the intermediate layers consist of to 3 convolutions followed by pooling and normalization. We say the number of 'identity' skip connections is variable since it is different for each convolutional block, e.g the second block has only 1 identity connection, while the third has 3. ResNet50 takes it's name from the number of total layers it has, which is 50. Through Tensorflow we can access a pretrained version of the architecture, with weights trained on ImageNet RGB images. As an output from ResNet50 we obtain a feature vector in dimension $\mathbb{R}^{2048}$. On the following project we will use it as a feature extractor for each video frame.

2.3 Long Short-Term Memory (LSTM)

2.3.1 Recurrent Neural Networks (RNN)

Long Short-Term Memory(LSTM) are a kind of Recurrent Neural Network(RNN). What differentiates a RNN from a classical feed-forward neural network is the presence of feedback connection. A feedback connection refers to the use of a hidden state from a previous time step as an input to the current time step, as represented in *Figure 2.4*. This mechanism allows the network to perform tasks that involve memory, since it allows the flow of previous inputs. This feature enables RNN to process sequential data and capture dependencies between elements. For this reason, RNN are widely used in tasks that involve temporal dependencies, such as video analysis.

RNN, however, present a major limitation when processing sequential data. They

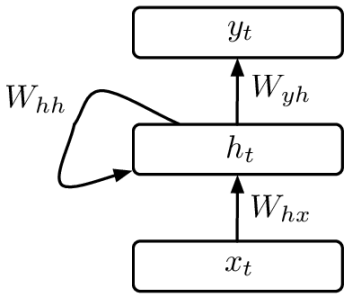


Figure 2.4: Simple RNN architecture, extracted from [10]

become ineffective when processing long sequences. Because of its mechanism, simple RNN are only able to considerate dependencies between elements that are very close from each other in the sequence. If the gap between 2 correlated elements is big enough, RNN will become very ineffective.

2.3.2 LSTM architecture

Long Short-Term Memory emerged as a solution to RNN limitations. They were first introduced by Sepp Hochreiter and Jürgen Schmidhuber on their paper *Long Short-Term Memory* at 1997 [7]. LSTM consist of 1 memory unit made up of 4 fully connected feed-forward neural networks, where each of them consist of one input and one output layer. Three of them are responsible of selecting the information of previous time steps that would be passed to the subsequent steps. They are called *forget, input and output gates*. The fourth NN is the responsible of creating new information to be inserted in the unit's memory. It is called the *candidate memory*.

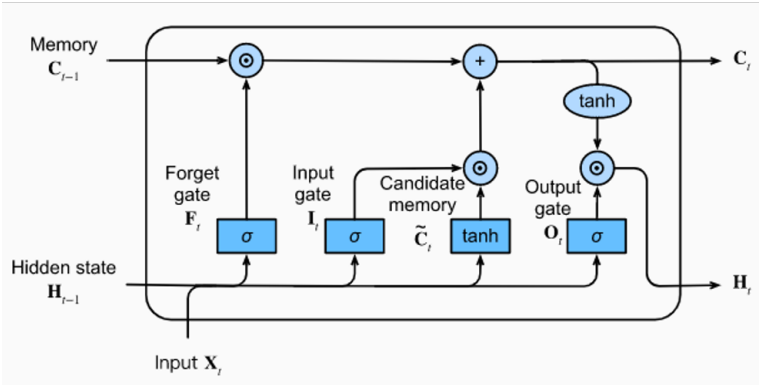


Figure 2.5: LSTM unit architecture, extracted from [8]

On *Figure 2.5* we can see a graphical representation of an LSTM unit. As we can appreciate, a unit takes 3 different inputs. Two of them come from the unit itself. On the graphic we see them represented as the cell state, also called memory, C_{t-1} and the

hidden state H_{t-1}. C_{t-1} is the vector responsible of maintaining the long-term memory. Meanwhile, hidden state H_{t-1} is the vector that contains short-term memory data. The third input an LSTM receives is X, which is the new data received by the unit. Given these 3 vectors the LSTM regulates, through the gates, the internal flow of information and transforms the values of the cell state and the hidden state vectors so they can be passed to the unit at the following time step, $t+1$. Hidden state H_{t-1} will be the output of the LSTM unit at a certain time step t.

Knowing how information is passed, we can now understand the function each of the 4 internal neural networks have. We mentioned that we had up to 3 gates. The first to be executed is the *forget gate*. Given the short-term memory vector, H_{t-1}, ad the input vector X, the forget gate computes the relevance of long-short term memory elements. This relevance is expressed through what we call a selector vector. This selector vector would have the same shape as C_{t-1} and all of its values would be inside the range [0,1]. An element with a value close to 0 means that its respective element in C_{t-1} has no longer relevance. That way, when C_{t_1} gets multiplied by the selector vector, only relevant information will remain.

The second gate to work is the *input gate*, along with *candidate memory* network. They are both in charge to add new information to long-term memory vector. Both networks receive vectors H_{t-1} and X as inputs. Candidate memory creates a candidate vector with information candidate to be added to C_{t-1}. Input gate computes a selector vector that would determinate which information of the candidate vector will have more or less relevance. Its values would be in a range from 0 to 1. Both vectors will be multiplied, in a way that elements with values closer to 0 on the selector vector will stay less relevant. The result of the product will be added to vector C_{t-1}. This way, new information is added to the long-term memory.

Last gate to operate is the *output gate*. As its names indicates, output gate is in charge of generating the output of the LSTM unit, which is equivalent to the hidden state H_t. To calculate it, output gate generates a selector vector after receiving C_{t-1} and X as inputs. The selector vector is then multiplied by a normalized cell state vector C_{t-1}. We normalize C_{t-1} so it's values are in between the range [-1, 1], to do so we can use an activation function like Tahn, as we can see in *Figure 2.5*. The result of the product will become the output of the LSTM unit.

Through this mechanism, LSTM units are able to regulate the flow of information, being capable of defining correlations between different elements in a sequence. Each element of the sequence will be analysed at a certain instant of time t,. That way, the previous sequence will be always considered, enabling us to take into account the time factor of a video.

2.3.3 LSTM advantages and limitations

LSTM present some advantages regard RNN. Previously, we mentioned one: LSTM are able to find dependencies between far distanced elements in a sequence, something that RNN struggle doing. This is due to their mechanism of relevance regulation in long-term memory, through selector vectors. Another major advantage LSTM present is the regulation of the vanishing gradient problem. On section 2.2.1 we talked about the meaning,

causes and implication of vanishing gradient. By they architecture, RNN have a bigger tendency of having this issue. This would also limit the capacity RNN have on finding element wise correlations. Meanwhile, we found out that through its control of the flow of information on its gates, LSTM contrast vanishing gradients, as explained in [11], making this problem much less probable.

However, LSTM present limitations as well. On 2016, Google Brain presented a study [12] where it was found out than when training LSTM on data with log-term dependencies, the models struggled to learn the task and generalize on new examples. This could be explained by the fact that when deciding which information to keep or remove through the forget gate, the gate would have access only to information from a few steps back, through the H_{t-1} vector. Another limitation presented by LSTM units it's their computing time. Since information needs to be analyzed sequentially, LSTM architectures can be much slower than other alternatives like Temporal Convolutional Networks or Transformers.

2.3.4 Bidirectional LSTM (BI-LSTM)

Bidirectional LSTM, or BI-LSTM, were first presented at 2015 on the paper *"Bidirectional LSTM-CRF Models for Sequence Tagging"* by Z. Huang, W. Xu and K. Yu [13]. They are a variant of LSTM units. The main difference lies in the order in which the sequence is analyzed. While understanding how LSTM works, we have seen that the inputted sequence is processed forward. What this means is that given an element at position i, not only the data at this position will be taken into account, but also the data in the $i-1$ previous elements. This is possible due to the long short-term memory the LSTM unit has, through its cell state and hidden states vectors.

On the other hand, BI-LSTM don't only process sequence data in a forward way, but also in a backward direction. This makes the model be able to capture past and future context data and dependencies. The architecture consists of two LSTM layers, where one will be responsible to process data in forward direction, and the other one will do it backwards. Each layer maintains its own hidden states and memory cells, in other words, they don't share long short-term memory. They would process data simultaneously. To combine both results, the hidden states, of both layers, at each time step will be combined. One of them most common ways of doing this is by simply concatenating both hidden states.

2.3.5 BI-LSTM advantages and limitations

The main advantage BI-LSTM present regarding LSTM, is that, by considering both previous and following context, it can capture richer dependencies in the input sequence. By doing this, the performance of classification improves. Specially in cases like ours, behaviour analysis, where future frames can help determining previous ones that at first could appear as ambiguous. However, BI-LSTM also present some cons regard LSTM. Since instead of having one layer, it needs two, BI-LSTM have a higher computational cost. They are often harder to interpret and can be seen as "black boxes". This becomes

specially a problem on applications where interpretability becomes an important factor, e.g medical diagnosis.

2.4 Temporal Convolutional Networks (TCN)

2.4.1 TCN architecture

Temporal Convolutional Networks or TCN are convolutional models capable of taking into account temporal properties. They consist of dilated causal 1D convolutional layers, where the input and output have the same length. They were first introduced as an alternative to LSTM and RNN on sequence processing, by Colin Lea et all on their paper *"Temporal Convolutional Networks: A Unified Approach to Action Segmentation"* at 2016 [16].

To understand how TCN works, we first need to understand what are dilated causal 1D convolutional layers. To start with, we can define what 1D convolutional layers consist in. 1D convolutional layers take a 3 dimensional tensor as an input, where the first dimension is the batch size and the second one is the length of the input sequence. Therefore, what we are passing to the layer is a sequence of elements, elements that we will find in the third dimension. It as well outputs a 3 dimensional tensor of the same sequence length. Therefore, only the last and third dimension changes.

To process the input, one 1D convolutional layer looks at sub-series of consecutive elements in the input. These sub-series have a given length n, which we will call the *kernel size*. For instance, to obtain the output of one specific element, 1D layer takes the product of the the n values around this element and a kernel vector of learned weights, which will also have a length n. Then, to calculate the output of the following element in the input, the kernel size sub-sequence would be shifted to the right as many times as the stride determines. Usually the stride is 1. That way, we would be able to process all elements i the sequence considering the other n values around them. One important characteristic is that the kernel vector of learned weights won't change among elements, it will always remain the same.

In case our input has more than one channel on its last dimension, this exact same process will be repeated for each individual channel, except the kernel vector will be different for each of them. If the number of channels at the output is not the same as in the input, we will combine the intermediate outputs to achieve the same number as in the output. One common way to do this is by summing them up. Therefore, we call this type of mechanism 1D convolution since the kernel window only moves along a single side, which is the inputs length.

The second characteristic we said TCN layers had was causal convolutions. This characteristic is what makes TCN able to consider time dependencies among elements. For a convolutional layer to be causal means that the element i of the output will only depend on the $i - 1$ previous elements and the same i element on the input. This way we ensure that only previous state will be contemplated, making models capable of finding time dependencies. Since the kernel size is static, we will need to pad as many zeros as needed at the left of the input sequence. Therefore, by ensuring this characteristic we will be able to find dependencies among elements following a time wise forward direction.

Dilation, which is the last characteristic, regulates the receptive field considered while predicting one element's output. We understand the receptive field as the set of entries of the original input that affect a specific element of the output. As we have just seen, these elements from the input will always be prior to the current output position. For instance, given a kernel size of 3, when looking at the 5th element in the input sequence, its output would depend on elements 3,4, and 5 of the input. As we can see in *Figure 2.6* if we stack multiple 1D layers we can wide the receptive field of one output element.

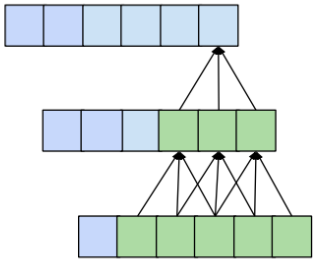


Figure 2.6: Stacked 1D causal layers, extracted from [17]

In fact, we can calculate the receptive field size r with the formula $r = 1 + n * (k - 1)$, where n is the number of layers and k the kernel size. Given a fixed kernel size, the amount layers required for a full coverage receptive field is linear to the length of the input tensors. With long tensors, this would cause networks to be very deep. As seen previously, this can lead to some struggles such as vanishing gradients. Models would slow down its convergence in good results and it might even lead to memory problems.

A solution to this problem is dilation. By applying dilatation we can achieve a wide coverage without having to have that many layers in our network. We understand dilation as the distance between the elements of the input sequence that are used to compute one element of the output. If dilation is 1, we would have that we take the adjacent elements in the input to the output element placement. This was our case in *Figure 2.6*. A higher dilation would give us a wider receptive field. In fact, we can calculate the length of our receptive field given a dilatation d and a kernel size k as: $1 + d * (k - 1)$.

However, if we had a fixed d, it will still be necessary to add numerous layers to achieve a wide coverage. To solve that we can consider a variant d. The most common way to do this is to increase it exponentially as we move through layers, as in *Figure 2.7*. We can then consider a base $b \in \mathbb{Z}$, which we will use to compute the dilation d at layer i as $d = b^i$.

However, using this methodology can cause gaps. We understand gaps as those elements in the input that have not been considered when calculating the output. To avoid this, we have to choose a kernel size k such that is bigger or equal to the dilation base b. By knowing this we are able to calculate the minimum number of layers we would need to achieve a full coverage. Therefore, by adding the right number of 1D dilates causal layers we can create a TCN model that would be able to consider temporal dependencies among elements in our sequences.

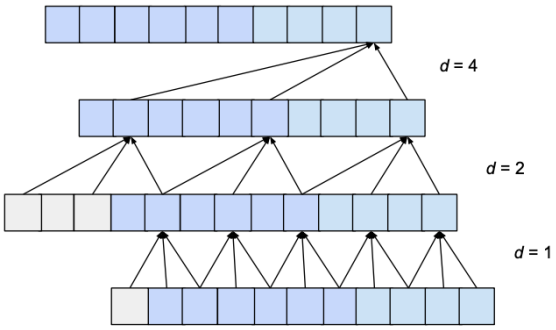


Figure 2.7: Stacked 1D layers with exponential dilation, extracted from [17].
In grey, those elements added by the padding.

2.4.2 TCN advantages and limitations

TCN present some advantages regarding LSTM layers. As stated in Bai et all paper, *"An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling"* [18], since they are fully convolutional models, they can be less resource expensive than LSTM and can be easily parallelizable. Also, they don't need to save memory vectors as we saw in LSTM, so they have lower memory requirements. Because of this, TCN can be faster to train and therefore converge faster into good results.

However, TCN also present some limitations. Different domains can have different requirements on the amount of history the model needs in order to predict. Consequently, transferring a model from a domain with minimal memory requirements (i.e., small k and d) to one that necessitates much longer memory (i.e., significantly larger k and d) can result in poor performance due to not having a sufficiently large receptive field. Another limitation that might occur is that, despite the use of dilation, TCN might still struggle with capturing non-local dependencies. Specially compared to some transformer-based models, which are designed explicitly to handle long-range dependencies using attention mechanisms.

2.5 Transformers

Transformers were first introduced in 2017 on Google Brain's paper *"Attention is all you need"* [19]. They emerged as a new neural network architecture that over-passed LSTM and TCN performance in sequence processing. Originally, they were developed for Natural Language Processing(NLP), but during the course of years they have also demonstrated to achieve excellent results in domains like Computer Vision. What differentiates Transformer from LSTM or TCN is that their architecture is completely build from attention mechanisms. On the following subsections we will first analyze what exactly is an attention mechanism. After that, we will dig in the Transformer architecture, its applications and the advantages and limitations.

2.5.1 Attention mechanisms

Attention mechanisms were first presented by Bahdanau D. et all in the paper *"Neural Machine Translation by Jointly Learning to Align and Translate"* in 2014 [22]. The idea behind it was to improve an encoder-decoder model performance on machine translation tasks. To achieve this attention mechanisms allowed the decoder to only focus on what were considered the most relevant elements in the data.

Attention mechanism took inspiration from human perception. Different studies in the field of psychology and neuroscience [25, 26] define attention in human beings as a complex cognitive function than allows us to selectively choose the information we want to put a focus on. When receiving an stimuli from the external worlds, our brains don't process the whole information. Instead they learn to discard irrelevant data and only focus on those parts that seem to be more useful.

In humans, we can distinguish between two types of attention, depending on its generation manner. The first one is called *saliency-based attention*. We say this kind of attention is bottom-up unconscious, since it is driven by external stimuli. For instance, it is responsible of making us more perceptible to hear loud voices when having a conversation. Although it was not our intention, we ended up paying more attention to those stimuli because of the task we were performing. The second attention type is what we call *focus attention*. We refer to this as the top-down conscious attention, since it has a predetermined purpose and relies on specific tasks. It enables us to focus on a certain object consciously and actively, which will improve our performance on a task.

Focused attention is what most attention mechanism in deep learning have tried to resemble. They aim to improve model performance by allowing the network to weigh the importance of different input elements when performing a task like translation, video classification and more complex sequences in natural language processing and computer vision.

At the present time, there are many attention variants that adapt to specific tasks and domains. Even though this, they follow a common schema. On this subsection we will try to understand the basic components of an attention mechanism. For that, we will use *Figure 2.8* as a reference. In this schema we can see represented a self-attention mechanism, which is the one used by transformers and shows us what the general architecture of the mechanisms looks like.

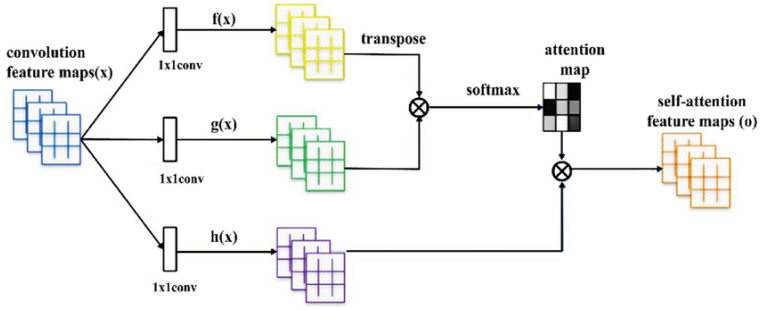


Figure 2.8: Self-attention mechanism schema. Extracted from [27].

Given an input X an attention mechanism will start by computing the attention distribution in its information. For that, attention mechanism first encodes X into a feature vector k, called *key*. Key can be represented in different ways, depending on the task (e.g. feature vectors representing an image sequence, embedded words, etc.). In *Figure 2.8*, k would correspond to the yellow matrix. It is usually necessary to introduce a task-related representation vector q, which is refereed as the *query*. The query tends to be a transformed version of the input. In our schema, we can see it represented in green.

Given the key vector k and the query vector q, the attention mechanism will compute the correlation between them. It will do this through a score function f, obtaining as a result what we call the energy scores e, such that: $e = f(q, k)$. The energy scores represent the importance of queries with respect to keys in deciding the model's next output. In the case of transformers, since they use self-attention mechanism, the score function is the dot-product of k and q as represented on the schema.

Once we have the energy scores e we map them to attention weights α. We do this through what we call the attention distribution function g, such that: $\alpha = g(e)$. The attention distribution function can vary depending on the type of mechanism. In the case of self-attention we use softmax function, which gives us a probability distribution, such that $\alpha \in [0, 1]$.

After getting the attention weights we will it is often necessary to apply them into a new data feature representation v, called *value*. Each element in v will correspond to one element in k. In fact, in some mechanism both vectors are equal. Although in most cases, as in self-attention, v will be a different representation of the input X. In fact, in self-attention we have that $x, qandv$ are all different representations of the same input X. Once we have the values and attention weights, we combine them through a function ϕ and obtain what we call the *context vector* c such that: $c = \phi(\alpha_i, v_i)$. The most common implementation of ϕ, also used in self-attention, is the weighted sum of v: $c = \sum_{i=1}^{\text{len}(v)} \alpha_i v_i$. Once c is calculated, this will be the output of our attention module, a transformation on the input where only relevant information was preserved.

2.5.2 Transformers architecture

As mentioned previously, Transformers were first presented in the paper *"Attention is all you need"* [22] as an alternative to RNN and TCN. They solely base their architecture in self-attention, which leads to better quality predictions, are easily parallelizable and take less time to process inputs.

Transformers follow an encoder-decoder architecture, as represented in *Figure 2.9*. The encoder is responsible of mapping an input sequence $(x_1, x_2, ..., x_n)$ to a sequence of continuous representation $z = (z_1, z_2, ..., z_n)$. On the original paper, the encoder is composed by a a stack of $N = 6$ identical layer. Each of them is composed by a *multi-head* self-attention encoder and a simple position wise fully connected feed-forward network. In both of them, we have a residual connection followed by a normalization layer. Something to consider is that when receiving the input $(x_1, x_2, ..., x_n)$, this has been went through positional encoding. Positional encoding provides information about the position of each element in the sequence, allowing the model to capture the order and structure of the input data since the model itself lacks inherent sequential awareness.

On the other hand, we have the decoder stack, which is as well composed by $N = 6$ identical layers. Each of them have the first 2 sub-layers as in the encoder, additionally they add a third sub-layer which performs multi-head self-attention over the encoder stack output on each corresponding layer. Each sub-layer has a skip connection followed by a normalization layer. Additionally, the decoder can modify self-attention to prevent positions from considering information from subsequent positions. This way, it get ensure than an element i will only depend on the $i - 1$ previous elements, keeping the time distribution.

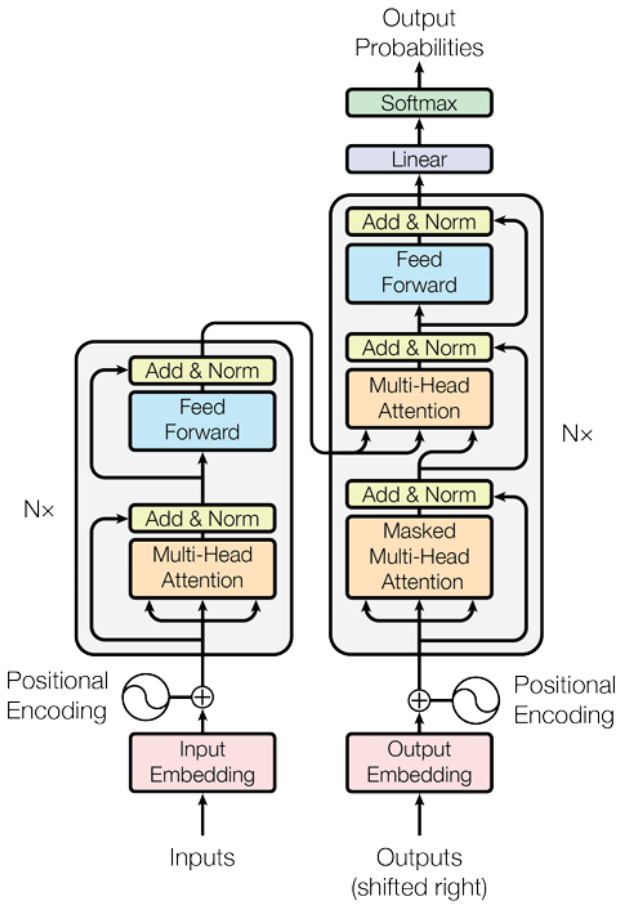


Figure 2.9: Transformer architecture. At left, one encoder layer.
At right, one decoder layer. Extracted from [22].

We say transformer models have auto-regressive steps. This means that for each step it consumes the previously generated symbols as additional input when generating the next.

On the previous subsection we talked about self-attention, but did not mention multi-head attention. Multi-head attention is a mechanism in transformer models where multiple attention heads operate in parallel, each learning to focus on different parts of the input

sequence. Instead of having a single set of attention weights, multi-head attention splits the input into multiple subspaces and applies the attention mechanism independently to each. The outputs from all heads are then concatenated and linearly transformed to produce the final result. This approach allows the model to capture various relationships and patterns in the data more effectively than a single attention head could. By attending to different parts of the input simultaneously, multi-head attention enhances the model's ability to understand complex dependencies and improves performance on tasks.

2.5.3 Transformers advantages and limitations

During transformers' architecture explanation, we have been mentioning a few of transformers advantages regarding other sequential processing models like LSTM and TCN. Their self-attention mechanism allows them to capture long-range dependencies and contextual relationships within the data more effectively than traditional models like LSTM. Transformers also facilitate parallel processing of input sequences, speeding up training and inference compared to sequential models. Additionally, their architecture is highly flexible, allowing for easy adaptation and fine-tuning across a wide range of applications and domains, which we saw was a limitation in TCN.

However, transformers also present a few disadvantages. They need a lot of computational power and memory, especially with longer sequences, making them expensive to run. Transformers also can require a lot of of training data to achieve good performance without leading to an early overfitting. While they are great at understanding long-range relationships in data, they sometimes miss out on local details and may require extensive hyperparameter tuning and regularization to prevent overfitting.

Chapter 3

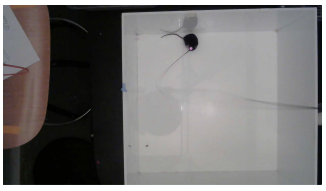
The Data

As mentioned previously, this project collaborates with a neuroscience research group at Hospital Clínic, which provided the dataset used. The project utilized the data previously used by X. de Juan in his Master's thesis, as explained in Chapter 2. This dataset consisted of 17 videos, each one being 11 minutes long. All videos featured a mouse placed in an empty white box. All of them were recorded under the same conditions: camera position, lighting, mouse color, etc. The frame rate for each video was 10 fps. For now on, we would refer to them as videos from *condition 1*.

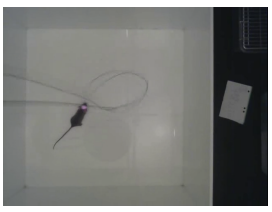
In addition to these initial videos, the research group provided us with seven more. These new videos had different conditions compared to the older ones. Five of them were recorded with the same camera placement and lighting. Although their duration varied, they all had a frame rate of 30 fps. We will refer to them as videos from *condition 2*. The other two videos, had identical recording settings between them, with a frame rate of 10 fps and a duration of 10 minutes each. These last group of records would be refereed as videos from *condition 3*.

On *Figure 3.1* we can see displayed 3 frames, one for each class of video. As it can be appreciated, all 3 frames were taken under different recording settings. It is worth to mention as well, that videos under conditions 2 had a different shape.

Frame extracted from a video of condition 1



Frame extracted from a video of condition 2



Frame extracted from a video of condition 3

Figure 3.1: Frames extracted for each class of video

In addition to the videos, the research team provided us with the behaviour labels for each frame. On chapter 2, we mentioned that on X. de Juan's thesis, up to 3 behaviours were considered. These were Grooming, Mid Rearing and Wall Rearing. We described

Grooming as the action that a mouse does while licking its fur, grooming with the forepaws or scratching with any limb. We also said that a rearing occurred when a mouse puts its weight on its hind legs, raise its forelimbs from the ground and extends its head upwards. On X. de Juan's work, he differentiated between two types of rearing. One that took place in the center of the box (Mid Rearing) and the other that was performed on the box's edges (Wall rearing). After discussing about it with the research team, it was decided that for this project we would only consider a general Rearing behaviour. On *Figure 3.2* we can observe a representation for each of the considered behaviours, we have also included what a neutral state look like.

Mouse on a neutral state | Mouse Grooming | Mouse Rearing

Figure 3.2: Mouse Behaviours

When talking about X. de Juan's previous work, we mentioned that he also used data extracted from *DeepLabCut*. *DeepLabCut* is a software with a computer vision package for animal pose estimation. From this software, the researchers at the Hospital Clínic team were able to extract, for each frame, the position of some parts of the mouse and stored them in CSV files. In our case, we also had at our disposition the CSV files for the old video set, the one used by Xavier. Unfortunately, that was not the case for the newly provided videos. As a result, we discarded the old position files and developed our own algorithm within the pipeline to automatically detect the mouse's center position. In Chapter 4, we will delve deeper into this process.

On the two following subsections, we will briefly explain a study that was performed to understand better the insides of the provided data.

3.1 Data study on video conditions

While explaining the project's data, we classified videos into 3 different categories, depending on their recording conditions. We named them as videos in condition 1,2 or 3. A majority on a video condition would lead to a bad generalization of the models. We have to keep in mind, that due to its final application in laboratories, the ideal case would be to have models able to generalize to any camera conditions. Therefore, we want to avoid a specialisation on the majority type of video. Because of this, we wanted to better understand which was the proportion of each class inside the total number of frames that we had in the whole dataset.

To get an idea of how many frames we had for each condition class video, we plotted a box plot on the total frame length for each class. On *Figure 3.3* we can see displayed the results for each condition type of video. We have to state that this box plots have been taking after correcting all videos so they all share the same frames per second rate, 10fps. On Chapter 3 we will explain this with more detail.

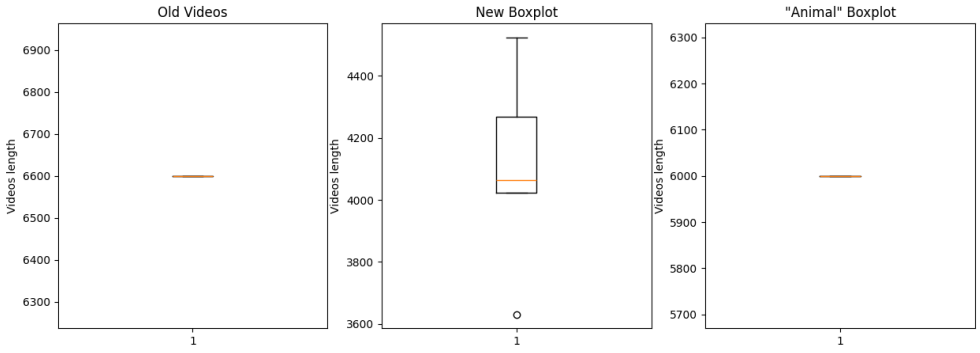


Figure 3.3: Box plots for each video condition class. Starting on the left, we first have the box plot corresponding to condition 1, followed by condition 2. On the left, condition 3.

We as well calculated the total number of frames for each configuration, so we could get an idea of the balance regard the total number of frames:

. *Configuration 1 number of frames: 112200*
. *Configuration 2 number of frames: 20507*
. *Configuration 3 number of frames: 12000*

From this, we can see clearly that Configuration 1 is the main configuration among all frames. Its videos are the longest, having each of them 6600 frames. Not only that, but since they are 17 out of the 24 videos we have, it becomes by far the most dominant among the dataset. By its corresponding box plot, we can see that videos under Configuration 2 have different lengths, although they all have between 4000 and 4500 frames. They become the second most dominant video type on the data set. Lastly, we have Configuration 3, that although its videos having more frames individually than the ones under Configuration 2, since there are only 2 videos, become the least predominant kind. This study will be considered later during the pipeline.

3.2 Data study on labels

After studying video configurations, we studied briefly behaviours. To start with, we looked at how many frames in the total dataset corresponded to each of the considered behaviours: Grooming and Rearing. On the following, we can find the number of frames for each and the percentage they represented regarding the whole dataset:

. *Grooming samples: Positive samples: 14886 (10.287%)*
. *Rearing samples: Positive samples: 29797 (20.5913%)*

Total number of frames in the dataset: 144707

As we can see, neither Grooming or Rearing are dominant among the dataset. Therefore, when training our models it would become important to set sample weights, to avoid they only consider the most frequent class, that in both cases would be the neutral state, with label 0.

We as well studied for how long each behaviour usually least. To consider it a behaviour sequence, we put the requirement that at least 5 consecutive frames should be labeled as such. We calculated the mean length of each behaviour's sequences, as well as the maximum one. To understand better if most sequences lasted the same or there was a considerable variance, we plotted the box plots at *Figure 3.4*.

	Grooming	Rearing
Mean Duration	102.5 frames	555 frames
Max Duration	22.4 frames	6 frames

Table 3.1: Sequences duration for both Grooming and Rearing

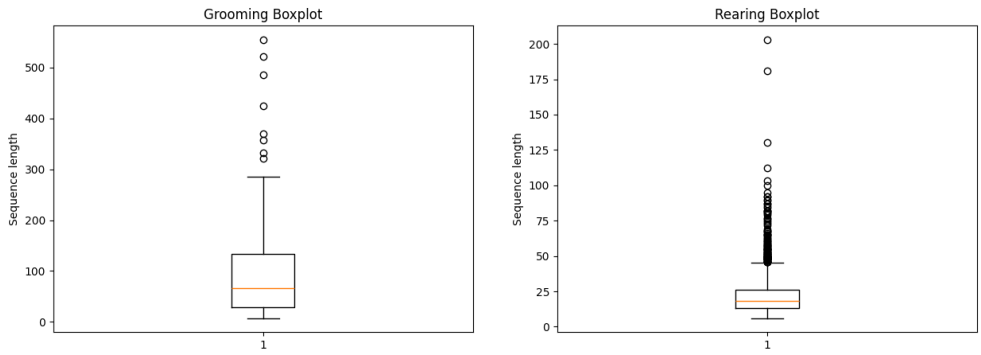


Figure 3.4: Box plots for each behaviour sequences lengths. At right, the one corresponding to Grooming. At left, the Rearing one.

Based on the results, it is evident that Grooming sequences are significantly longer than Rearing sequences. We can see by the box plot that most Grooming sequences take between 50 and 150 frames. However we can also appreciate some outliers, where a few sequences take more than 300 frames, being 555 frames the longest. On the other hand, most Rearing sequences seem to least between 20 and a little bit less than 30 frames. On its box plots we can appreciate many outliers. Most of them are between 30 and 100 frames. This information would be used to choose suitable sequence lengths when splitting the videos, as we would see on feature steps.

Chapter 4

Data preprocessing

On the following chapter we will explore the different methodologies and techniques applied to preprocess our data. Some of them had a more experimental character and eventually were discarded after model benchmarking. However, they would be briefly explained on this chapter. On Chapter 7, the results of benchmarking would be analyzed and conclusions regarding these preprocessing techniques would be drawn.

Our data preprocessing has its base on a frame extraction loop. Given the directory with all our data, the videos with their corresponding CSV files, we iterate and process each couple of them. The processing of the CSV remains quite simple. As previously explained, all of them contain 2 or 3 columns with the behaviour labels. On old CSVs, corresponding to the ones used by X. de Juan, we find 3 columns: one for Grooming and 2 for Rearing, since he considers mid and wall rearing. For that we unify the rearing columns into a single one. We clean the CSVs, standardize the same column names for each of them and make sure labels are expressed as 0 or 1.

In the case of videos, preprocessing becomes more complex. With the *OpenCV* library, we iterate through each video by reading its frames separately. The idea is to process the frames in a way we can treat them as images and save them individually, so later it becomes easier to represent a video as a sequence of images. On the following sections we will explore some of the techniques and methodologies applied to the frames while reading them, before saving them as images. We will explore why they may become beneficial to later stages of the pipeline and explain briefly how they were implemented.

4.1 Frames per second standardization

First thing we took into account when iterating through a video's frames was to set a standardized frames per second rate. As we mentioned on the previous chapter, videos on our data set don't have a consistent frames per second rate. While most of them have a rate of 10fps, five videos, the ones under recording conditions 2, have 30fps. At first, we didn't take this into consideration, but after experimentation we realized it makes struggle the models on generalizing. We understand the frames per second (fps) rate as a measure of video quality. Essentially, what it represents is the quantity of photograms taken in a

time stamp of a second. We directly relate a higher fps rate with a higher video quality.

To understand why different frames per second ratios would cause a struggle on our models, we can think about two identical videos, except that one would have a higher fps rate. What we would find is that the sequence representing a behaviour, like for instance Grooming, would take much more frames in the higher fps video rather than in the lower one, eventhough in reality both actions had taken the same time. Since our classification models only receive sequences of frames as inputs and they have no clue of the fps, they would interpret that the behaviour in the video with higher fps has been longer in time than the one on the low fps video. As we have seen on Chapter 3, behaviours tend to have a unique length that sets them apart. For instance, Grooming actions take way longer than Rearing. This information can become key when deciding which action is the mouse performing. Therefore, it becomes important to maintain a constant frames per second rate so the time factor can be properly considered by the models.

To achieve this, we have included on our data preprocessing pipeline an algorithm that allows us to set a standard fps rate and adjust those videos which have a different one. Since most of our videos in the dataset have a rate of 10 fps, we decided that to be the standard. Also, because it is a low rate, it would also help us with memory consumption when applying the pipeline in the clinical application for laboratory research.

The algorithm starts by checking the video fps rate, this can easily be done by the *OpenCV* library. If it matches the default one, we simply iterate through each frame and save it after the other procedures have been done. If it is not the case, we will iterate each frame but only save certain ones, so the fps rate matches the standard one. To know which frames to save and which to discard we will follow the algorithm proposed by the pseudo-code in *Algorithm one*.

Algorithm 1 Frames per second standarization Algorithm

Input: A sequence of frames s with a fps rate f. A standard fps $f_{standard}$
function StandarizeFPS($s, f, f_{standard}$)
    if $f = f_{standard}$ **then**
        process and save each frame in s
    else
        Initialize z as the counter of last processed frame that follows $f_{standard}$ rate
        for frame in s at postion i **do**
            calculate the correspondent frame in position j in a $f_{standard}$ sequence
            if position $j > z$ **then**
                process and save frame in s at position i
                update $z = j$
            else
                skip frame in s at position i
                delete from label's CSV the row corresponding to frame in position i
            end if
        end for
        Update label's CSV so lines that correspond to discarded frames are delated
    end if

4.2 Frame cropping

At Chapter 2, we mentioned that on his work, X. de Juan cropped each frame in a 160x160x3 picture where the mouse was always positioned at the center. He did this during video preprocessing, with the purpose of setting an standard frame size, saving memory and only keeping relevant information. To do so, he used the CSV files provided from *DeepLabCut*, where he had the coordinates of mouse's center at each frame. For this project, we have used the same strategy, the only difference is that we have no longer used *DeepLabCut*'s data. Instead, we have developed our own automatic mouse position detection algorithm, by applying computer vision techniques.

As we mentioned at Chapter 3, although we have the CSVs X. de Juan used for position extraction, we don't have this data for the new provided videos. This motivated us to develop the following automatic position detection algorithm. On application, this also presented a major advantage. By doing this, we free professionals at the research laboratory of having to previously process videos on *DeepLabCut*'s software. This saves them time and makes the clinical application of the models free of other software's dependencies.

On the following subsection we will explore how the automatic mouse position detection algorithm works.

4.2.1 Automatic mouse position detection

The whole automatic mouse position detection algorithm has been build around the computer vision tools provided by the library *OpenCV*. We can define the methodology on the following steps:

1. First, we convert our RGB frame to gray scale.

2. We convert the gray scale frame to a binary image. To do so we set a threshold, so any value higher or equal becomes 255 and, on the opposite, any value lower to it becomes 0. We apply Otsu's binarization method [15] to set the threshold automatically. Through the histogram of the gray scale values on the frame, Otsu's method is able to determinate the most optimal threshold. By using this methodlogy, we ensure that binarization is suitable for any lighting condition. We are sure that after the procedure the figure of the mouse will still be visible, since all recorded videos in the laboratory are done so there is a big contrast between the mouse and the box where it is displayed.

3. Once the image is binary, we search for the box on the frame. In some cases, as in the videos under condition 1, the box where the mouse is displayed occupies most of the frame size. However, other times, as in videos under conditions 2, by the camera placement, we have that box only represents a smaller percentage of the total frame. This last situation often leads to have many different objects present on the edges of the frame. This can led to mistakes when detecting the mouse's position, since it may detect an object instead. To avoid this situations, we first try to detect the box's shape and position. To do so, we would apply *OpenCV* library to detect those objects

that follow a rectangular shape. In all most every case, the box will correspond to the biggest rectangular shaped object detected. We can then obtain its coordinates. If the box represents most of the frame's size, there won't be any need of cropping it. If that's not the case, we would use the box coordinates to cut the frame around it, so nothing can interfere when detecting mouse's position.

4. Once the frame is centered on the box, through compute vision tools provided by *OpenCV*, we look for contours inside it. This would give us the mouse's silhouette. Sometimes, other things can be detected as well, for example, the mouse's faeces. To ensure we are getting the mouse, we only consider the biggest detected object. We also check that the detected silhouette is not situated very close to the frame's edges, since that would mean we are detecting an external object instead.

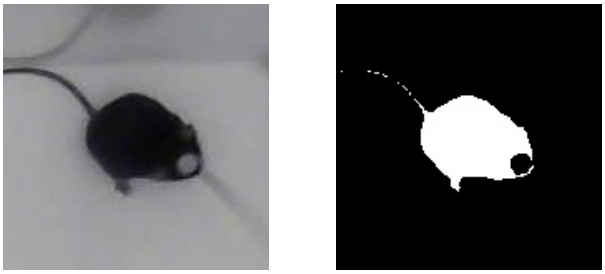
5. Having our mouse located, we create a bounding box around it. With the bounding box vertices coordinates, we can calculate the center coordinate. This would give us a grate approximation on where the mouse's center is located. Therefore, we will use this coordinates to crop the image around it.

4.3 Frame normalization

While processing frames, we also performed some value normalization. On X. d Juan's original project , originally frames were saved as RGB images after the cropping. On a RGB image we have up to 3 channels(red, green and blue), were values can go from 0 to 255. Although we used this technique, we also created to alternatives data sets where values were normalized.

For the first normalized data set, we reduced the values range to $[0,1]$, instead of $[0,255]$. To achieve this, we simply divided each frame by 255 which in all cases was the maximum value the frame could reach. The idea of this procedure is to simplify later calculations on machine learning models. By having a smaller value range, we speed up calculation, which can help the models to converge faster into good results. On Chapter 7 we will explore the results on this dataset and see it it really supposed an improvement regarding the standard RGB frames.

Another alternative we tried as well was working with binary images rather than RGB. On the previous section, we saw that when finding the mouse position, we would first convert frames to binary. To do so, we first passed the image to gray scale and then applied Ostu's threshold technique. What we have done then to generate a Binary dataset, is a slight modification on the preprocessing code, where instead of applying the cropping on the regular RGB frame, we applied it on the binary representation of the frame. We would then save this binary cropped image, rather than the RGB one. To avoid noise, like the one generated by the cables or faeces on the box, after cropping the binary frame, we apply erosion. Erosion of a binary image is an operation that removes pixels on object boundaries, shrinking the objects in the image. With that, we can make small objects almost disappear. On *Figure 4.1* we can see a the difference of one same frame expressed in RGB values and in Binary.



RGB frame Binary frame

Figure 4.1: Comparison between a same frame, before and after applying binarization.

The main idea behind this modification of the original frame data set is to reduce the variance on images produced by lighting and camera quality conditions. By doing this, we ensure than every frame, no matter the conditions in which it was recorded, is expressed with the exact same range of values. This might help models on generalization. On Chapter 7 we would explore the results on both the RGB and the Binary datasets.

4.4 Data Augmentation

Finally, the last preprocessing technique we applied consisted on Data Augmentation. Data augmentation is a technique used to increase the diversity and amount of data available for training machine learning models. It involves creating new training examples by applying various transformations to the existing data, such as rotations, flips, translations, scaling, cropping, and adding noise. This helps improve the model's generalization ability, making it more robust to variations in real-world data.

To perform data augmentation we have made use of Keras' ImageDataGenerator. ImageDataGenerator is a utility within the Keras library, designed for real-time data augmentation and preprocessing of images. By applying a range of transformations such as rotations, shifts, flips, zooms, and brightness adjustments, ImageDataGenerator helps create varied versions of existing images. In our case, we designed ours so given a video it would process all its frames through the same random changes. The transformations to be applied were brightness adjustment, zoom in or zoom out and aleatory horizontal flipping. We did not wanted to include rotations since that could change the way the walls of the box were orientated and, as far a we knew, they always tend to be parallel to the frame edges.

Having our random ImageDataGenerator defined, we processed all the videos on our dataset. We configured the script so transformations inside the same video would follow the same configurations, but between videos were always different. By these we were able to generate up to 24 new videos imitating different recording conditions. We included all of them on our train set, so our data would be more varied.

4.5 Data Split

Once we had our data preprocessed and loaded, we proceeded to perform a split into a train, a test and a validation set. To do so, we divided our videos into these 3 data sets. We would like to note that we have adopted the following terminology:

1. Train set: data set used on training to adjusts model's parameters
2. Test set: data set used on training on the Checkpoints to save model's best performance
3. Validation set : dataset never seen while training, with the purpose of performing benchmarking between models

At first, we thought about creating a random distribution of the videos to generate the splits. However, after experimenting with this idea, we realized it would give us a wrong perception of the models' abilities to generalize. Since in all sets had a majority of videos recorded under conditions of type 1, as seen in Chapter 3, we found that models tended to overfit these conditions, and the validation set metrics did not properly represent the other video types. For that reason, in the end, we opted for a custom-made dataset split. We then followed this strategy:

Videos recorded under conditions of type 3 had the fewest frames. In fact, we only had 2 available videos of this kind. For this reason, we decided to put one of them in the validation set and the other in the training set. With this, we wanted to judge our models' ability to generalize to videos recorded under conditions not considered during training. With this approach we wanted to reassemble a real use case, since in the laboratory video always tend to be recorded under different camera placements and lighting conditions.

Videos recorded under conditions of type 2 were the second category with the fewest frames. Since there were only 5 videos of this kind, we decided to save the shortest one for validation, the second shortest for testing, and the rest for training. In other words, the longest videos were in the training set. Lastly, videos recorded under conditions of type 1 were the most common, with up to 17 videos in the set. We saved one of them for the validation set. For the rest, we performed a random 70-30 split for the training and test sets. This meant that 70% of them went to training, while the other 30% went to the test set.

With this approach we wanted to make the datasets as balanced as possible, making sure that we would have an equal representation of all 3 conditions on the validation metrics.

Chapter 5

Feature extraction

On chapter 2, we mentioned that X. de Juan processed frames through pretrained Convolutional Neural Networks. That way he could be express each of them as feature vectors. Later on, those feature vectors would be grouped in sequences and would serve as input's for the sequence classification models. The idea behind that was to keep only relevant information of the frames in a more concise way, so it becomes easier and more effective to generate the sequences. By processing frames through pretrained models we are also making sure we are only keeping relevant data, since through the convolutions and pooling system the pretrained models discard any irrelevant information on the image and encode low-level features present in the video. On this project, we have followed the same approach.

On his thesis, X. de Juan tested 2 pretrained models: ResNet50 and InceptionResNet V2. After experimenting with both, he saw that ResNet50 gave the best classification results. For that reason, we have chose to use ResNet50 as the feature extractor of this project. We have used the pretrained architecture available on *Tensorflow* library, with the weights trained on *Imagenet* datasets. *ImageNet* is a large-scale visual database designed for use in visual object recognition software. It contains millions of labeled images across thousands of object categories.

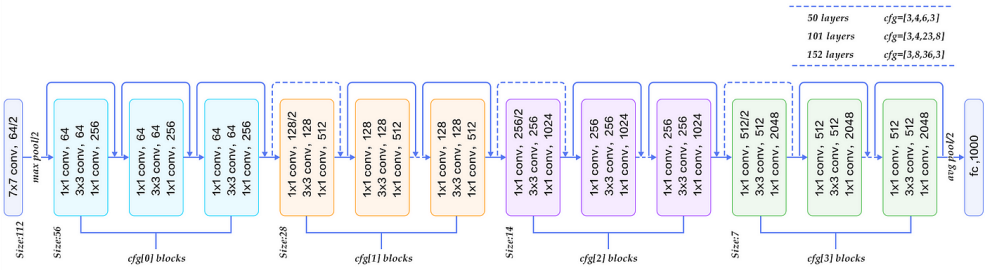


Figure 5.1: ResNet50 architecture's schema, extracted from [6]

At *Figure 5.1*, we have a graphical representation of ResNet50's architecture. As we mentioned during Chapter 2, ResNet50 counts with different convolutional blocks. On X.

de Juan's thesis, he used all them, obtaining for each frame a feature vector representation of 2048 elements. For the complexity and size of our processed frames, that number of features seemed to be too big. In fact, it is quite probable that the majority of them contained irrelevant data for the classification task. A smaller feature vector would ease classification models training and would also reduce the quantity of memory required for processing any video. For this reason, we decided to simplify the feature vectors. For that, we analysed the architecture of ResNet50 and decided that instead of using the full architecture, that was giving us vectors in dimension $\mathbb{R}^{2048}$, we were going to cut the model to an earlier layer, where the extracted feature vector was in dimension $\mathbb{R}^{512}$.

To corroborate that by doing this we were not losing any information that was crucial to determinate classification, we performed an experiment where the exact same architecture was trained on the same conditions 2 times. The first time, the model was trained on a dataset conformed by $\mathbb{R}^{2048}$ feature vectors and the second time the vectors were in dimension $\mathbb{R}^{512}$. On Chapter 7 we will display and analyse the results.

Chapter 6

Sequence classification modules

In this chapter, we will explore the architectures we have developed and used for sequence classification. In Chapter 2, we discussed the strategy followed by X. de Juan, where he performed video frame classification based on LSTM and TCN models. He built up to two models, one for each approach, capable of taking into consideration temporal dependencies between elements. On video classification, capturing high-level temporal information becomes crucial. As we saw during our data exploration on Chapter 3, behaviours tend to appear on relatively long sequences. Knowing data about previous or even subsequent elements, can give crucial information when distinguishing between classes.

On his work X. de Juan prepared the data to be analyzed by the sequence classification architectures. As we have seen on the previous chapter, he represented a video as a succession of feature vectors, where each of them encoded data from a frame. Since videos tend to have a large number of frames, he divided them into feature vector subsequences. This sub-sequences were the inputs for the classification models. Each of them was processed by the models. After predicting each subgroup, he combined the results to obtain the final total video classification at frame level.

For this project, we have adopted a similar approach as the one on X. de Juanâs on his previous work. However, having the new data that Hospital Clínc gave us, we wanted to extend his methodology and develop model's that were capable to generalize different recording conditions. For that we have experimented with sequence prediction architectures built around four different approaches: LSTM layers, BI-LSTM layers, TCN layers, and Transformers.

In the following sections, we will explore the nature of each of these models, how they were adapted to our pipeline, and examine their main differences. Before this, although, we will briefly talk about the tool that has been used enough to determinate most of the architectures.

On Chapter 2 we mentioned that X. de Juan performed Grid Search enough to determinate the models he used. Grid Search is a hyperparameter optimization technique that systematically evaluates a specified range of hyperparameters for a machine learning model. The goal is to identify the combination that gives the best performance. It involves exhaustively searching through a manually specified subset of the hyperparameter space

of the learning algorithm. On his work, X. de Juan specified hyperparameters such as the number of LSTM or TCN layers, the size of the sub-sequences, etc.

To determinate our architectures, we decided to apply a similar approach. We developed as well a hyperparameter space where we could find specifications of our models, such as the number of layers, the size of sequences and batches, etc. Each subspace was adapted to its corresponding model and its requirements. We will explore this deeper on the following sections. However, the main difference regarding X. de Juan approach has been that instead of implementing Grid Search, we have applied *Bayesian Optimization*. Bayesian Optimization is an alternative hyperparameter optimization technique. The main difference regarding Grid Search is that instead of exhaustively evaluating each parameter combination, it iterates through a fewer number of possibilities. Through probability theory, it is available to determinate the combinations that most probably will improve the performance, based on the results on previous iterations. This way we can ensure we will converge into good results without having to search through all options, which can be a costly and very time and resource consuming task.

To perform Bayesian Optimization and store its result we have used the tool *Weights and Biases*. Weights and Biases is a machine learning platform that provides tools for experiment tracking, model visualization, and hyperparameter tuning. It helps to keep track of experiments and visualize performance metrics in real-time. Through this tool we have been able to store all the tried hyperparameter combinations and the metrics results they gave. We have also been able choose the ones that boost up any desired metric. At the annex of this document you can find attached the tables with the parameter combinations for each architecture that gave the best PRC (Precision-Recall Curve) with their corresponding metrics. This results were used to define the final architectures of our models.

6.1 LSTM

The first architecture we developed and tried is fully based on LSTM layers. We followed X. de Juan's proposed architecture. At *Figure 6.1* we can see an schema of the LSTM based model. On the schema we can appreciate the different layers of the model, as well as some of the parameters that would be determined through Bayesian Optimization. First we find the input of size (b, s, f), where b stands for the batch size, s for the sequence length and f for the number of channels on the feature vectors (2048 or 512). Firstly, the input goes through a Dropout layer that aims to avoid overfitting on the network. After taht, it goes through the first LSTM layer. As we can see on the schema, it is said to be a number N of LSTM layers. This is because the number of stacked LSTM layers is also a parameter we would include on our hyperparameter tuning. We also included on the hyperparameters the number of LSTM units each layer would have. Each time, we would reduce to half the number of units, this would cause a reduction of third dimension. This third dimension will be the one carrying out the probabilities in the output. To convert the last LSTM layer output into probabilities we will apply to each element in the sequence a Dense layer with sigmoid as the activation function. To be able to apply the Dense layer to each element separetly we will use Keras' TimeDistributed layer.

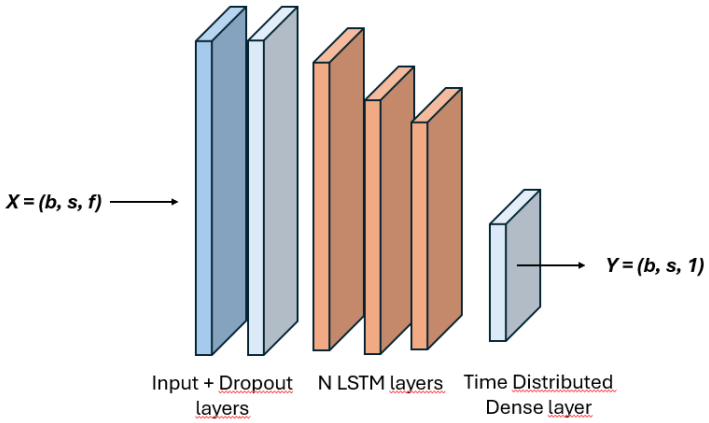


Figure 6.1: LSTM based classifier schema

6.2 BI-LSTM

After testing the LSTM based architecture we addapted it to become a Bidirectional LSTM model. Essentially, the only remarcable changes were done on the LSTM layer stack, where through Keras' layer Bidirectional, we converted LSTM to BI-LSTM layers. The rest of the architecture remained the same, as well as the parameters tested during Bayesian Optimization.

6.3 TCN

Due to its advantages in computational requirements compared to LSTM layers, we also decided to test a TCN-based architecture. This architecture closely resembled the one developed for the LSTM-based model, with the only difference being the replacement of the LSTM stack with a TCN stack. Since TCNs have a completely different architecture from LSTM, we slightly changed the parameters on Bayesian Optimization. We removed the number of units calculated for LSTM and introduced the number of filters and the kernel size the convolutions on TCN will use. The rest of parameters remained the same, included N that, in this case, would give us the number of TCN layers the architecture will have.

6.4 Transformers

The transformer based classification model is the one that differs the most from the other architectures. We defined our transformer classifier based on the work realized by Pual Sayak on his article *"Video Classification with Trasformers"* [21]. On his article, Sayak elaborates a transformer based architecture for video classification. We took it as a base and adapted it to video classification at frame level.

On *Figure 6.2* we can see a graphic representation of our Transformer based classifier. The model recives an input (b, s, f), where again b stands for the batch size, s for the sequence length and f for the number of channels on the feature vectors (2048 or 512). This inputs goes through a positional embedding. This becomes primordial since this embedding will make our transformer block aware of temporal order. Once all elements in the sub-sequences are embedded, they go through a Transformer Encoder block. To define this block we will have to specify as a parameter number of heads we want in our multi-head self-attention layers. The output of the transformer block goes through a Dropout layer, so we reduce the possibilities of overfitting. Finally, to convert our features into probabilities, we apply a Dense layer with sigmoid activation to each element through the TimeDistributed layer. As an output, we obtain the probability for each element in the sub-sequences to be one behaviour. It is worth to say, that due to transformers having a grater number of parameters than a usual neural network, performing hyper-parameter tuning was not possible, since it required too many memory and computing capacities.

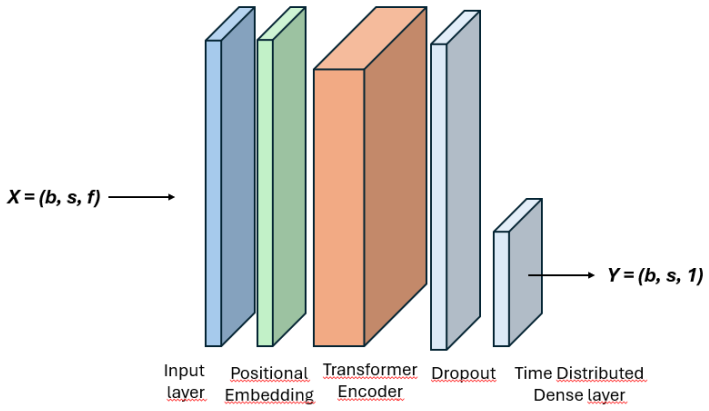


Figure 6.2: Transformer based classifier schema

Chapter 7

Experiments

On the following chapter we will discuss about the results obtained on benchmarking the different approaches presented during Chapters 4, 5 and 6. Before that we would first like to clarify the methodologies we have followed enough to state if the performance of a model is good. Therefore, the first section of this chapter will focus on explaining all the implemented evaluation metrics. After that, we will evaluate the different variations on the pipeline presented by previous chapters.

7.1 Evaluation methodology and metrics

As we stated on previous chapters, enough to compare models performances we have made use of a validation set. This validation set is conformed by 3 videos, each one of them representing a different video recording condition. One of the main objectives on this project has been to develop models that would generalize among different recording settings, so they could be applicable to real use cases in research laboratories. Therefore, what would determinate the great performance of a model will be its ability to achieve good results for each of the 3 recording conditions represented in the validation set.

Enough to measure the model performances, we have defined a set of evaluation metrics that would serve us as indicators. In the following subsections we will dig in the meaning and interpretability of each of them.

7.1.1 Binary accuracy

In classification tasks, accuracy measures the percentage of predictions that were correct regarding the true labels. To obtain it we simply divide the number of correct predictions by the total number, such like:

$$Attention = \frac{\#correct predictions}{\#predictions} \tag{7.1}$$

On binary classification, where labels are either 0 or 1, we have binary attention. Given the prediction probabilities, binary attention considers any value equal or superior to a

given threshold as 1, else the value will be evaluated as a 0. This threshold is usually set at 0.5, but can vary depending on the preferences.

However, accuracy only results useful when the data has an equal distribution of classes. If one class is significantly more frequent than the other, a model can achieve high accuracy by simply predicting the majority class all the time, without actually learning to distinguish between the classes. For that reason, we will consider other metrics that will give us a grater comprehension on the models performances. Although this, we will maintain accuracy, as it provides a simple baseline and context for model performance

7.1.2 Precision

Precision focuses on the proportion of true positive predictions among all positive predictions. It is calculated as the number of true predicted positives divided by the number of positive predictions:

$$Precision = \frac{TP}{TP + FP} \tag{7.2}$$

Where TP equals to True Positives and FP to False Positive.

We can interpret precision as the number of instances predicted as positive that are actually positive. Its value will always be between 0 and 1. Being 0 the worse case, where no true positives have been predicted, and 1 the best case, where the model has no false positives.

On binary classification we can apply a threshold to precision, as we did with accuracy. Given this threshold, precision would interpret as 1 those values in the prediction that are greater or equal to it and as 0 those which are lower.

7.1.3 Recall

Recall evaluates the proportion of true positive predictions among all the ground true positives. We calculate it as the following way:

$$Precision = \frac{TP}{TP + FN} \tag{7.3}$$

Where TP equals to True Positives and FN to False Negativves.

We can interpret recall as the measure on how well the model identifies all positive instances in the dataset. The minimum value its can reach is 0, meaning none true positives were predicted at all, and the maximum value is 1, meaning all real positives were properly labelled by the model. We can apply a threshold in Recall as well.

7.1.4 Precision-Recall Curve (PRC)

The precision-recall trade-off refers to the inverse relationship between precision and recall in classification models: as you increase precision by being more conservative in predicting positives (reducing false positives), recall often decreases because more true positives are missed, and vice versa. Balancing this trade-off is essential for optimizing

model performance, especially in applications where the costs of false positives and false negatives differ significantly. Specially in cases where classes are very imbalanced, the trade off between precision and recall measures the success of of a prediction.

The precision-recall curve (PRC) illustrates the trade-off between precision and recall across various thresholds. A larger area under the curve signifies both high recall and high precision, where high precision indicates a low false positive rate, and high recall indicates a low false negative rate. Achieving high scores in both metrics demonstrates that the classifier is accurately identifying relevant instances (high precision) and capturing the majority of positive instances (high recall).

A system with high recall but low precision produces many results, though most of its predicted labels are incorrect compared to the true labels. Conversely, a system with high precision but low recall returns few results, but most of its predicted labels are correct. An optimal system with both high precision and high recall will yield many correct results, accurately labeling all instances. The precision-recall curve allows us to determinate if our predictions are laying in any of this cases and therefore determine how good these models are.

On this project we will also work with the Area Under the Precision-Recall curve(PRC-AUC) metric. This metric summarizes the trade off between precision and recall across different thresholds into a single value. A higher PRC-AUC indicates better model performance.

7.1.5 Confusion Matrix

Given binary classification, a confusion matrix is nothing more than a matrix that contains the true negatives(TN), false positives(FP), false negatives(FN) and true positives(TP) of a prediction given its label. Each row in the matrix would represents the true and false predictions for a class. By applying this matrix on a visual heat map, as we will see on the following sections, we will get an easy to look graphic representation of how good or not the predictions of a model is.

7.2 Adaptation of the previous approach

The first experiment we conducted aimed to replicate the best model obtained by X. de Juan's previous work. After experimentation, X. de Juan stated that the best method for feature extraction was using the pretrained ResNet50 architecture, giving vectors of dimension $\mathbb{R}^{2048}$. He also found that LSTM layer based classifier would suit the best our objective. Here we applied the same approach and tested a LSTM based classifier on sub-sequences of ResNet50's feature vectors.

The idea behind recreating his model was to have a reference point where we could develop other approaches around. One of the main objectives in this project is to improve the previous work and adapt it to the new data. With this, what we wanted to do in this experiment was to evaluate how would the previous best approach adapt to the new conditions.

X. de Juan defined his LSTM classifier architecture based on the results of Grid Search based hyper-parameter tuning. Some of the parameters he tested during this process were the number of LSTM layers or the units each layer would have. In our case, since we counted with a different dataset and different computing conditions, we decided to redo the hyper parameter tuning on our PC through Bayesian optimization. This way, we would be able to adapt X. de Juan approach to the current conditions. On the annex of this project you will be able to find the results after hyper-parameter tuning.

Once we defined our classifier based on the results of Bayesian optimization, we trained a model for each behaviour during 50 epochs. On the following tables, *Table 7.1* and *Table 7.2*, we have displayed the prediction metrics results for Grooming and Rearing on the validation set. To compute them, we have set a neutral threshold at 0.5.

Condition 1 column represents the results for the video in the validation set recorded under the most dominant recording setting, which corresponded to old videos. *Conditions 2* column corresponds to the video on the second most dominant recording conditions. Finally, *Conditions 3* column corresponds to the video recorded under the camera setting with less frames in the total dataset. If we remember from the data split, this last group had no representation on the train set.

	Condition 1 video	Condition 2 video	Condition 3 video
Binary Accuracy	0.933	0.971	0.972
Precision	0.868	0.0	0.929
Recall	0.79	0.0	0.559
PRC-AUC	0.901	0.744	0.807

Table 7.1: Grooming results on LSTM based architecture with feature vectors $\in \mathbb{R}^{2048}$

	Condition 1 video	Condition 2 video	Condition 3 video
Binary Accuracy	0.889	0.909	0.987
Precision	0.68	0.343	0.705
Recall	0.624	0.952	0.581
PRC-AUC	0.738	0.791	0.67

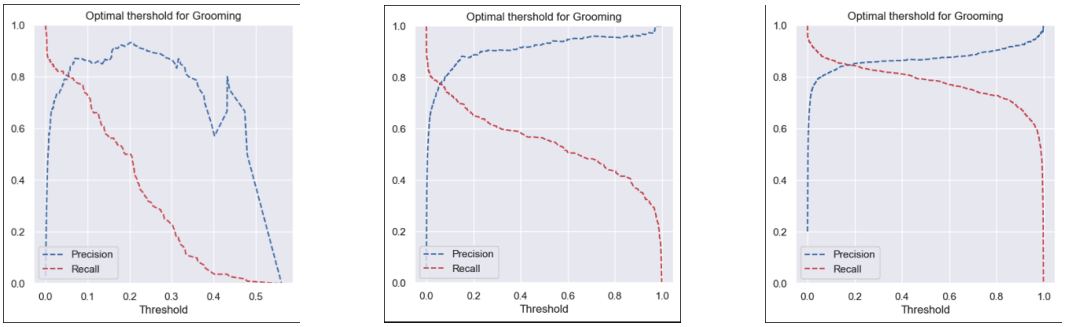
Table 7.2: Rearing results on LSTM based architecture with feature vectors $\in \mathbb{R}^{2048}$

By looking at the results, we can see that Grooming's model performance was specially good at the video under recording condition 1. Since this video corresponds to the majority type in both train and test set, it makes sense that such good results were achieved. We can see that both Recall and Precision are quite similar, meaning we have a great balance between false positives and false negatives.

However, results for conditions 2 and 3 are not as good As we remember, condition 2 has up to three videos on the training set. Even though this, it seems they were not enough for the model to adapt to them. As we can see on the results, both precision and recall are 0. This means that no positive frames were predicted with a probability

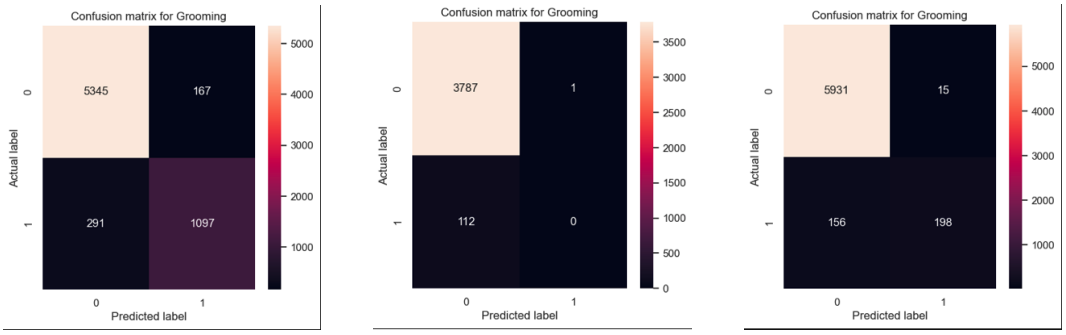
over 0.5. Although this, if we look at the PRC-AUC is quite high, being at 0.791. The precision-recall curve in *Figure 7.1* explains this. It seems that on very small thresholds, the precision-recall trade off is quite high. Although that, such small thresholds cannot be considered, since we need to find a value that is equally applicable to all recording conditions. In fact, having such small values can be caused by a lack of generalization. What is probably happening is that the model is overfitting on videos under recording condition type 1.

If we look at the results of the video under condition 3, we can see that they were not as bad as in condition 2. This could be explained by the fact that recording conditions of type 3 are more similar to conditions of type 1. However, we still have a low recall, meaning that most of the real positive frames are not being predicted. In fact, if we again look at *Figure 7.1*, we can appreciate that the PRC only reaches high values on very low thresholds, which again demonstrates us that the model is not genralizing enough.



Condition 1 video Condition 2 video Condition 3 video

Figure 7.1: Grooming Precission-Recall curve for each video in the dataset

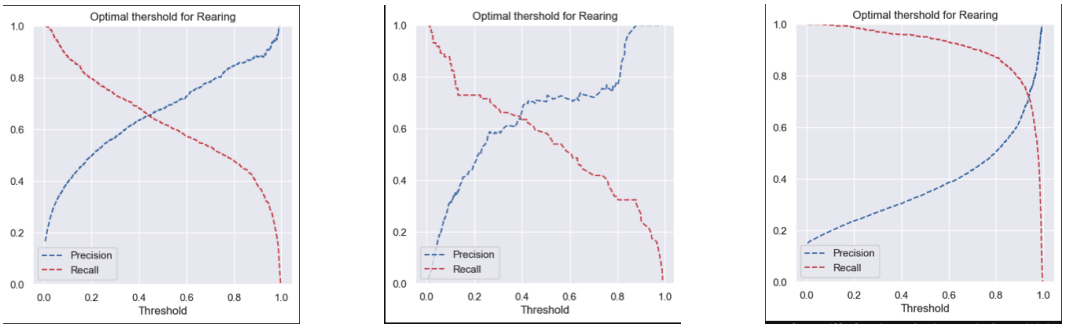


Condition 1 video Condition 2 video Condition 3 video

Figure 7.2: Grooming confusion matrix for each video in the dataset (threshold set at 0.5)

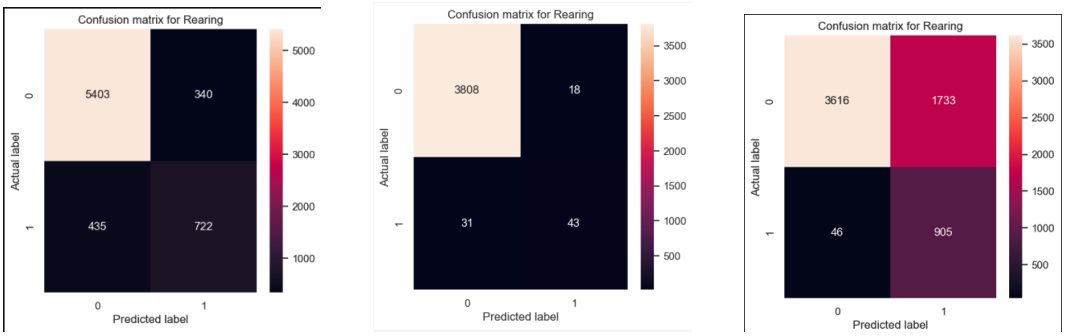
In regard results for Rearing, we can see a similar pattern. Video under conditions 1 holds the most balanced Precision-Recall rates. Although in this case, they have lower values. This could be due to hyper-parameter tuning performed around grooming metrics. Video under conditions 2 have a very low precision while having very high recall. We

understand from this that the model has classified very few frames as positive, but from those few almost all of the, where correct. Again this is very dis-balanced and shows us a poor performance of the model on minority video settings. Finally, if we look at the video in condition 3, results seem to be decent. Still all metrics remain very low.



Condition 1 video Condition 2 video Condition 3 video

Figure 7.3: Rearing Precission-Recall curve for each video in the dataset



Condition 1 video Condition 2 video Condition 3 video

Figure 7.4: Rearing confusion matrix for each video in the dataset (threshold set at 0.5)

What we can conclude from this first experiment is that even though the previous approach gives good results for videos under conditions 1, it does not do well on adapting to different recording settings. This gives us room for improvement.

7.3 Feature vector dimensionality

On Chapter 5 we talked about reducing feature vectors dimensionality. We proposed using an earlier layer in ResNet 50 that would give us vectors in dimension $\mathbb{R}^{512}$, instead of $\mathbb{R}^{2048}$. The principal advantages of doing this are memory requirements reduction. By saving less data to represent each frame we avoid memory problems during training and, specially, on real use case where researchers would be using the models on PC of variable computing capacities. Not only that, but by having less data we would also speed up the training and prediction speed of models. The only risk this approach presents is that, by

reducing features this much, we end up loosing crucial information for the classification task.

The second experiment we conducted had the purpose of demonstrating if reducing data dimensionality was worthy or supposed a huge loss of information. For that, we applied the same strategy as in experiment 1. We trained two LSTM based classifiers, one for each behaviour, during 50 epochs. Since the data changed, we performed a new hyper-parameter tuning through Bayesian Optimization, to determinate our models architecture and fixed parameters. On the following tables, *Table 7.3* and *Table 7.4*, we can find the obtained results.

	Condition 1 video	Condition 2 video	Condition 3 video
Binary Accuracy	0.932	0.981	0.944
Precision	0.911	0.909	0.0
Recall	0.736	0.357	0.0
PRC-AUC	0.902	0.789	0.202

Table 7.3: Grooming results on LSTM based architecture with feature vectors $\in \mathbb{R}^{512}$

	Condition 1 video	Condition 2 video	Condition 3 video
Binary Accuracy	0.889	0.976	0.5823
Precision	0.716	0.4	0.252
Recall	0.559	0.541	0.9
PRC-AUC	0.678	0.563	0.6

Table 7.4: Rearing results on LSTM based architecture with feature vectors $\in \mathbb{R}^{512}$

If we compare the resulting metrics, there isn't much differences between the patterns we found on the previous experiment while using vectors of 2048 features. Again performance on video under conditions 1 is good, in fact values between both experiments are quite similar. But if we look at results for videos under conditions 2 and 3, once again we see a poor performance. We either see very distinct precision and recall values, very low metrics or even zeros. Therefore, we can state that changing the feature vectors dimensionality from 2048 to 512 does not imply a worse performance.

Something to remark as well is that, while in the previous model training one epoch took around 45 seconds, with this approach on average an epoch lasted only 4 seconds. This made the training process much faster and less computational consuming. For these reasons, we ended up choosing frame representation through 512 features rather than 2048.

7.4 Data normalization and augmentation

What we have seen through experiments 1 and 2 is that the LSTM based classifier performs bad on data generalization. Instead, it only learns how to properly predict

frames on videos under recording condition 1. The reason tho this is most probably the quality and diversity of the data given while training. The amount of frames in both, train and test sets, that belong to condition 1 is huge in respect with the other filming settings. Therefore, it is normal that our models tend to specialize only in this kind of videos. To avoid this situation we performed experiment 3, where we implemented and benchmarked the different preprocessing techniques proposed during Chapter 4 to standardize values along data so models would generalize them.

7.4.1 Maximum value normalization

The first approach we tested was the one proposed during data preprocessing, where we said to reduce the range of values in each frame from [0,255] to [0,1]. The maximum value a RGB image can reach is 255, however in some occasions reducing this range to a smaller scale can give some benefits. A smaller range implies smaller calculations, which means less computing complexity, specially on deep learning layers where dot products of large values can result into very large quantities. Easier calculations can lead models to converge into good results faster.

One common technique to reduce an RGB scale is simply dividing its values by 255. This way, we ensure each value on the image will be between 0 and 1, while we preserve the distribution. We first applied this technique to each frame before going through ResNet50. However, once we tested the dataset on the classifier architecture it gave horrible results. We associated this to the encoding ResNet50 did. Since ResNet50 is trained on ImageNet datasets, which have images on the current RGB range, it is not used to handle data with lower ranges.

We then tried to implement normalization after ResNet 50 encoding. What we did was to normalize each feature vector by its maximum value before going through the LSTM based classifier. After training each model for 50 epochs, we obtained the results on *Table 7.5* and *Table 7.6*.

	Condition 1 video	Condition 2 video	Condition 3 video
Binary Accuracy	0.931	0.971	0.944
Precision	0.919	0.0	0.0
Recall	0.722	0.0	0.0
PRC-AUC	0.885	0.487	0.189

Table 7.5: Grooming results on LSTM based architecture with normalized feature vectors $\in \mathbb{R}^{512}$

As seen by the metrics, this technique did not work. All metrics remained having the same pattern, they even got worse values. A reason to this might have been that by reducing values to such a small range, during operations most of them may have tended to 0, causing by that a loss of information. Therefore, we ended up discarding this methodology.

	Condition 1 video	Condition 2 video	Condition 3 video
Binary Accuracy	0.892	0.98	0.75
Precision	0.813	0.3	0.347
Recall	0.468	0.04	0.749
PRC-AUC	0.7	0.123	0.643

Table 7.6: Rearing results on LSTM based architecture with normalized feature vectors $\in \mathbb{R}^{512}$

7.4.2 Binary normalization

A second approach proposed during Chapter 3 was to convert images from RGB to Binary. The intention behind this was to reduce the impact of lighting on frame variance among different recording conditions. A binary image only has two possible values: 0 and 255. As seen during image preprocessing, we edit every frame so mice silhouettes would be zeros and the background would be 255. With that we created an alternative dataset that was later processed by ResNet50 to encode it into feature vectors. After that, we followed the usual procedure of training for 50 epochs each model. On *Table 7.7* and *Table 7.8* we can find the obtained metrics for the validation set.

	Condition 1 video	Condition 2 video	Condition 3 video
Binary Accuracy	0.919	0.982	0.967
Precision	0.743	1.0	0.658
Recall	0.912	0.357	0.859
PRC-AUC	0.921	0.887	0.891

Table 7.7: Grooming results on LSTM based architecture with binary feature vectors $\in \mathbb{R}^{512}$

	Condition 1 video	Condition 2 video	Condition 3 video
Binary Accuracy	0.818	0.956	0.553
Precision	0.466	0.15	0.2445
Recall	0.588	0.284	0.94
PRC-AUC	0.582	0.14	0.752

Table 7.8: Rearing results on LSTM based architecture with binary feature vectors $\in \mathbb{R}^{512}$

By the obtained results, we can observe that this approach has improved the performance of Grooming on video under condition 3. However, results of Grooming on video under conditions 2 remain equally bad. Results on Rearing are not better neither. From this we extract the conclusion that regularizing color values caused by lightning conditions is not enough. In fact, as we can appreciate in *Figure 7.5*, there are more factors involved like the size of the mouse. Depending on the camera position, this would change. This approach then is not enough to reduce variance among data and boost up the generalization

of models.

Figure 7.5: Comparission between cropped frames and original frames. On top a frame from a video under condition 2. Bellow, a frame from a video of condition 1.

7.4.3 Data augmentation

Dragged by conclusions on last experiment, we opted for performing data augmentation on our data set. That way, we would have enough data to cover more camera position and lighting states. As explained in Chapter 3, we implemented a Keras' ImageDataGenerator that given a video, would randomly change the brightness, zoom range and orientation of all of its frames. Applying this technique to all of our videos, we were able to obtain 24 new videos under totally random settings. We added them to our train set and trained our LSTM based classifiers for 50 epochs. Results on the validation set are displayed on tables *Table 7.9* and *Table 7.10*.

	Condition 1 video	Condition 2 video	Condition 3 video
Binary Accuracy	0.957	0.983	0.971
Precision	0.882	0.912	0.913
Recall	0.908	0.464	0.537
PRC-AUC	0.96	0.889	0.695

Table 7.9: Grooming results on LSTM based architecture with feature vectors $\in \mathbb{R}^{512}$ after data augmentation

As we can appreciate on the results, all conditions have achieved around a 90% precision on Grooming, which means almost no false positives were detected in non of the video types. This supposes a major advance regarding our previous metrics. Recall is neither bad, although it has room for improvement. Both videos, the one under recording conditions 2 and the one on conditions 3, have recall values close to 50%. We can understand this as at least half of the positive frames have been properly predicted. Rearing results have also improved in regard our previous results on the LSTM based classifier.

	Condition 1 video	Condition 2 video	Condition 3 video
Binary Accuracy	0.885	0.986	0.858
Precision	0.647	0.602	0.518
Recall	0.687	0.78	0.869
PRC-AUC	0.762	0.829	0.785

Table 7.10: Rearing results on LSTM based architecture with binary feature vectors $\in \mathbb{R}^{512}$ after data augmentation

Precision and recall are quite similar among all conditions. Although values can be higher, this supposes a major improvement. We can then state that data augmentation has enabled the model to generalize more among video recording conditions. Therefore, this step would be included on our finale pipeline.

7.5 Sequence classification module

The last stage of our experiments consisted on testing which sequence processing approach suited best our problems. As explained during Chapter 6 we have developed up to 4 different sequence classification architectures. The first of them was the LSTM based one, which we have been seeing during previous experiments. The other three are based in BI-LSTM, TCN and Transformers. The purpose of the following last experiment was to determinate which of the 4 architectures would suit best our problem. We trained all of them with our augmented dataset during 50 epochs per each behaviour, so we can equally compare them.

We will not displayed LSTM results since they can be found on the previous sub section, at tables *Table 7.9* and *Table 7.10*. On the following tables *Table 7.11* and *Table 7.12* we can find the results obtained for the Bidirectional LSTM based classifier.

	Condition 1 video	Condition 2 video	Condition 3 video
Binary Accuracy	0.953	0.995	0.948
Precision	0.92	1	0.689
Recall	0.842	0.848	0.144
PRC-AUC	0.944	0.941	0.47

Table 7.11: Grooming results on BI-LSTM based architecture with feature vectors $\in \mathbb{R}^{512}$ after data augmentation

As we can observe on the tables, BI-LSTM has given outstanding results on Grooming for videos under conditions 1 and 2. In fact, they very high precision values indicates us only very few frames where predicted as false positives. Recall as well stands pretty high, indicating us that all most every positive frame was properly labeled. However, we can see by the recall that almost no frames where classified under condition 3. As for results on Rearing, except for the video under condition 1, by the recall we can see that the model only predicted few positive frames. We could think then that by its design, BI-LSTM works

	Condition 1 video	Condition 2 video	Condition 3 video
Binary Accuracy	0.917	0.986	0.895
Precision	0.841	0.824	0.783
Recall	0.651	0.19	0.421
PRC-AUC	0.812	0.479	0.663

Table 7.12: Rearing results on BI-LSTM based architecture with binary feature vectors $\in \mathbb{R}^{512}$ after data augmentation

better for predicting long lasting actions like Grooming, rather than sporadic behaviours like Rearings. Also, from the Grooming results we can state that BI-LSTM adapt very well to the recording conditions seen while training, but don't generalize to new ones.

Next architecture we evaluated was the TCN based one. On *Table 7.13* and *Table 7.14* we can find the results obtained for the TCN based classifier.

	Condition 1 video	Condition 2 video	Condition 3 video
Binary Accuracy	0.941	0.995	0.962
Precision	0.832	0.942	0.671
Recall	0.937	0.875	0.582
PRC-AUC	0.91	0.955	0.683

Table 7.13: Grooming results on TCN based architecture with feature vectors $\in \mathbb{R}^{512}$ after data augmentation

	Condition 1 video	Condition 2 video	Condition 3 video
Binary Accuracy	0.902	0.99	0.962
Precision	0.702	0.895	0.69
Recall	0.673	0.459	0.686
PRC-AUC	0.754	0.743	0.78

Table 7.14: Rearing results on TCN based architecture with binary feature vectors $\in \mathbb{R}^{512}$ after data augmentation

Results for TCN keep balanced precision and recall values, for both Rearings and Groomings. Grooming results have surpassed all the previous approaches. Videos under conditions 1 and 2 keep having very high metrics, as in the BI-LSTM classifier. Although this, condition 3 now keeps a great balance between precision and recall, with quite high values considering that is the minority recording condition. Results on Rearing are also considerably good. Despite rearing being a harder action to detect, it keeps a great balance between false positives and false negatives. This demonstrates that the TCN based architecture is capable of generalizing between different recording settings, making it a very suitable option for real case uses.

Lastly, we tested the Transformer based architecture. *Table 7.15* and *Table 7.16* contain the obtained results.

	Condition 1 video	Condition 2 video	Condition 3 video
Binary Accuracy	0.94	0.995	0.958
Precision	0.885	0.97	0.882
Recall	0.779	0.857	0.297
PRC-AUC	0.902	0.939	0.691

Table 7.15: Grooming results on Transformer based architecture with feature vectors $\in \mathbb{R}^{512}$ after data augmentation

	Condition 1 video	Condition 2 video	Condition 3 video
Binary Accuracy	0.888	0.981	0.873
Precision	0.719	0.588	0.567
Recall	0.545	0.135	0.682
PRC-AUC	0.693	0.284	0.685

Table 7.16: Rearing results on Transformer based architecture with feature vectors $\in \mathbb{R}^{512}$ after data augmentation

While results for Grooming on videos under conditions 1 and 2 are high and balanced, it seems that the Transformer based model struggles on generalizing videos under condition 3. As it happened with Bi-LSTM layers, the recall on video 3 lets us know that very few frames where predicted as true. Results on Rearing are neither good, specially on the second video where the recall stands very low. What might cause this is the fact that Transformers, due to their complexity, usually require bigger amounts of data and computing power enough to converge into good results. Just the simple model we defined, had three times more parameters than the previous described approaches. Therefore, it becomes natural to think that more data and time should be putted to properly train the model.

7.6 Final approach discussion

After the analysis of the conducted experiments, conclusion can be extracted about what are the approaches that best suit our pipeline. First experiment aimed to see how a reduction on feature vectors dimensionality would affect classification metrics. We saw that in fact, reducing dimensionality from 2048 to 512 did not affect models performance. Therefore, due to its benefits in memory and computing time, we state that using an earlier stage of ResNet50, that gives us a smaller dimensionality, as the CNN extractor works better than using its whole architecture. Therefore, our pipeline will use this pipeline where encoding frames.

On the second conducted experiment, we wanted to see which data prepossessing methodologies where worthy to implement. We ended up discarding value normalization and binary images rather than RGB, due to they poor performance. On the other hand, we saw that performing data augmentation on our train set improved our models capacities in generalizing, which was a feature we wanted our architectures to have. For that reason,

we decided to use a dataset extended with data augmentation for training

Lastly, we have conducted a benchmark through different state of the art approaches on sequence processing. Although almost every approach surpassed the results obtained by the previous methodology presented on X. de Juan's thesis, there was one that achieved almost all the objectives we had regarding the project. We wanted an architecture that maintained a great balance between false positives and false negatives. The architecture also needed to be capable of generalizing among video recording conditions, since we wanted it to be applicable to a real research laboratory environment. Given these requirements, TCN based classifier seemed to be the most suitable.

On the following Figures we can find a graphic representation on how TCN classifier has performed on the validation set. In orange, we have represented the ground true labels, in blue, the predictions outputted by the classifier. We have as well attached the extracted confusion matrix and PRC graphics, that allows us to better compare this final pipeline with the initial one that aimed to recreate the previous state of the project. We can see that now PRC graphics have higher and more stable curves. It is also worth to say that the number of frames, if we compare the previous confusion matrix with the new ones, has a slightly change. This is due to the way frames are subdivided in groups. On the previous experiment we had sequences of 300 frames and with TCN we have sequences of 600 frames. Since the number of frames per second is not always dividable by the sequence, sometimes on the last subgroup we get the frames previously seen on the previous subgroup, so the remaining ones can be fitted as well. This way of processing it caused the new videos to have 300 frames more, they are duplicated frames on the 2 last sub sequences. Although this we can use the confusion matrix to compare how the proportions of false positives and false negatives has changed regarding the beginning.

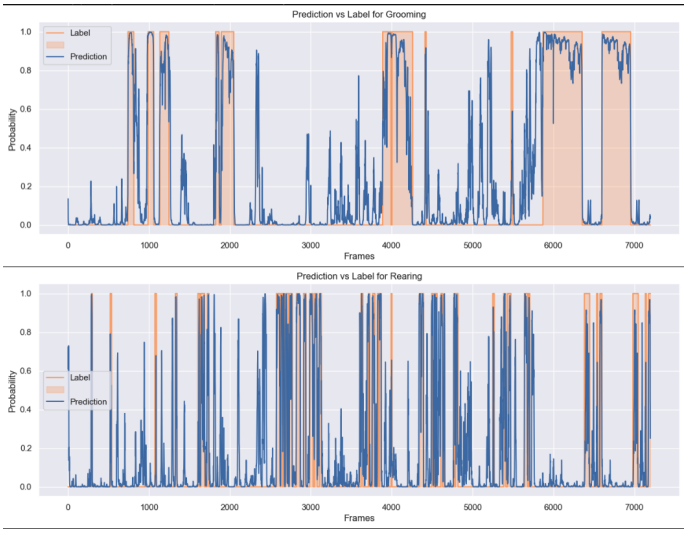


Figure 7.6: Grooming and Rearing predictions on video under conditions 1.

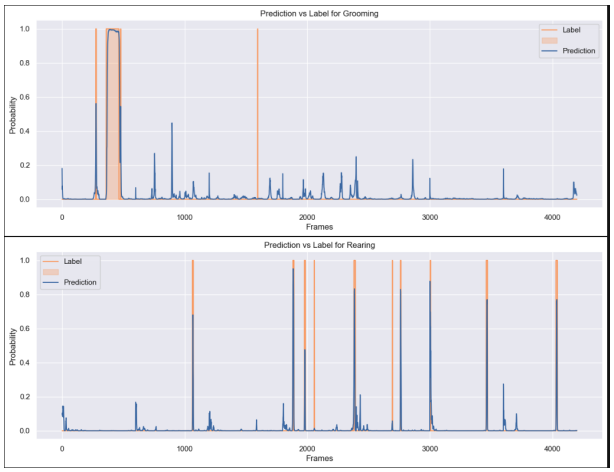


Figure 7.7: Grooming and Rearing predictions on video under conditions 2.

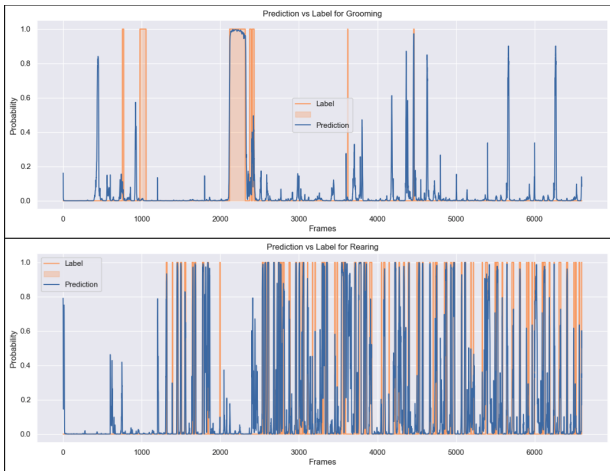
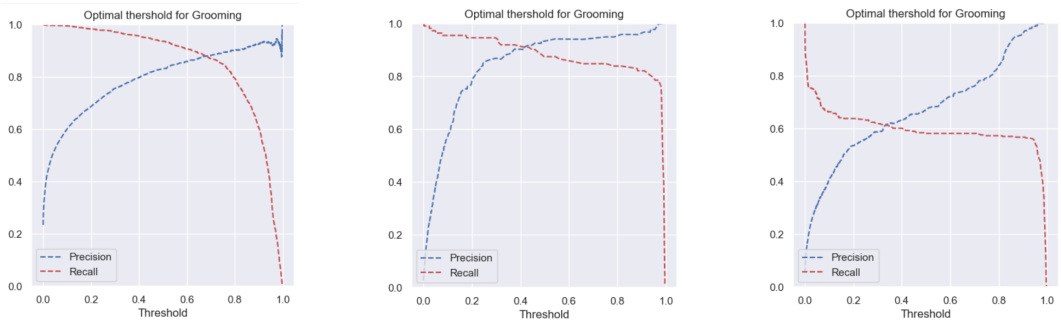
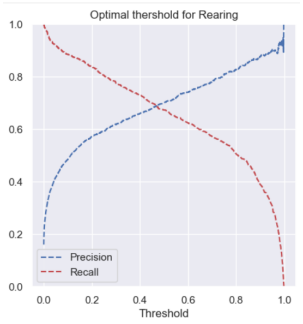


Figure 7.8: Grooming and Rearing predictions on video under conditions 3.

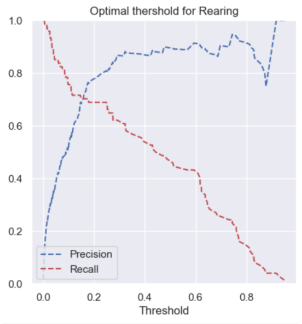


Condition 1 video Condition 2 video Condition 3 video

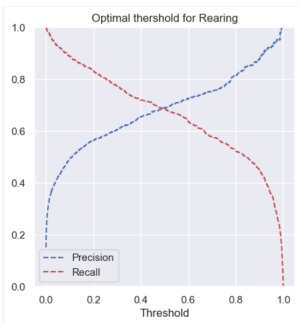
Figure 7.9: Grooming Precision-Recall curve for each video in the dataset



Condition 1 video

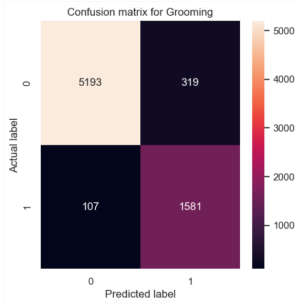


Condition 2 video

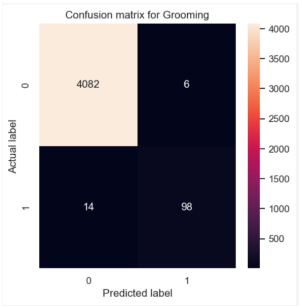


Condition 3 video

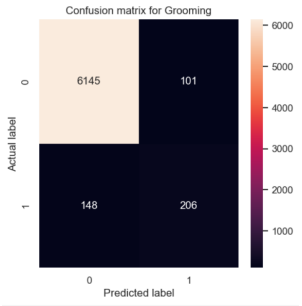
Figure 7.10: Rearing Precision-Recall curve for each video in the dataset



Condition 1 video

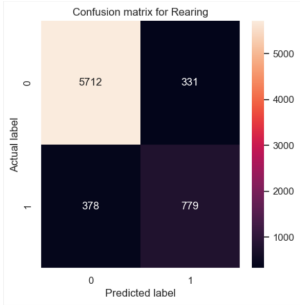


Condition 2 video

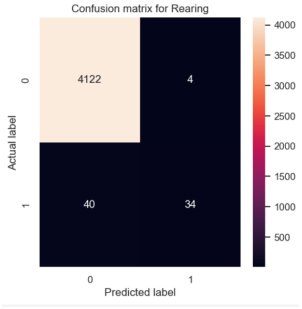


Condition 3 video

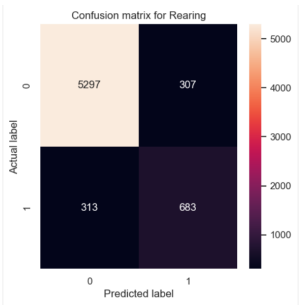
Figure 7.11: Grooming confusion matrix for each video in the dataset (threshold set at 0.5)



Condition 1 video



Condition 2 video



Condition 3 video

Figure 7.12: Rearing confusion matrix for each video in the dataset (threshold set at 0.5)

Chapter 8

Clinical Application

At the begging of this project we mentioned that the main purpose of developing this machine learning models was so they could be applied in real use cases, such as the neuroscience laboratory in Hospital Clínic. For that, a user friendly interface was required. A space where researcher could upload the videos and obtain, in an easy way, their labels. For that, this project has also worked on the integration of models in a front-end web page.

Originally, the project counted with a web page developed, with Python's Streamlit library, by the student Eli Barlow. He developed an interface than given a video and a CSV with its labels, would write at each frame the corresponding action so while visualizing the video one could know what action that was being taken. Not only that, but given the mouse position at each frame, he would also perform a study of the total distance traveled, as well as the speed increase or decrease during time. All these functions where accessible from the tab presented in *Figure 8.1*. Additionally, in this project we have added the option of filtering results by behaviours, as well as some adjustments on the way stadistics were calculated and displayed in graphics. We have also added our automatic position detection algorithm, so distance and speed calculations could be done without *DeepLabCut* data.

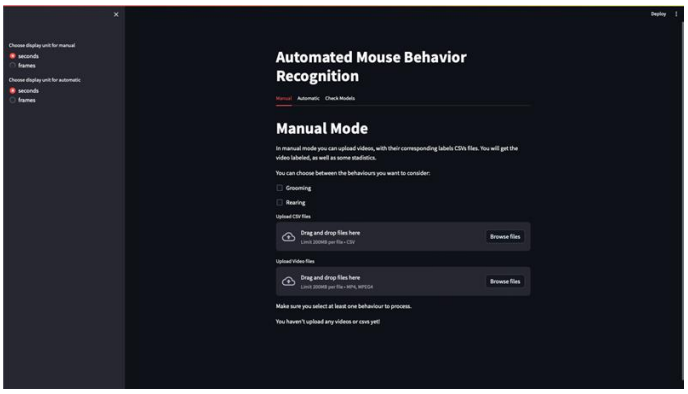


Figure 8.1: Manual Mode on the User Interface.

We have also created a new tab that only receives as an input videos. This tab connects with the developed machine learning pipeline, where given a video it preprocess it as we have seen during Chapter 4 and 5. After preprocessing, the data is passed to the developed TCN classifiers, which will return the labels. This way, researchers can easily get videos automatically labeled. Data studies in speed and distance are also displayed in this tab.

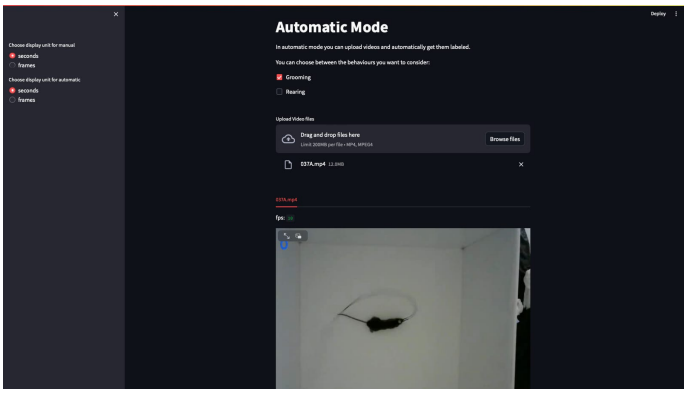


Figure 8.2: Automatic Mode on the User Interface.

The whole User Interface was deployed on Docker, so it would be accessible and easy to install in any PC that had the required computing power. By the way the docker image is done, it automatically gets connected to GPUs, so machine learning models can run more effectively. The docker image of the project can be found on the DockerHub: `https://hub.docker.com/repository/docker/albarransara/mouse_behaviour/general`

Chapter 9

Global conclusions

From the begging, we had the purpose of continuing with the work X. de Juan left us with his Master's thesis. Given his investigations and developments, we wanted to expand them and bring them closer to real case applications. We wanted to create a tool that would enable researchers at laboratories to easily analyse mice behaviours in videos at frame level, freeing them of doing this repetitive and time consuming task. For that we wanted to adapt our approaches as much as possible to real use cases, where data goes through multiple variant factors.

Through the research and understanding of the current state of the art approaches for sequence analysis, we have been able to identify the major challenges on this project. We have taken rigorous studies on our data, enough to understand the limitations it presented. We have seen how imbalanced our data was regarding labels and regarding the conditions in which videos were taken. We have developed tools such as the frames per second ratio regulation or the automatic mouse position detection that have enable our models to treat and standardize different kinds of video inputs.

Through experimentation and benchmarking we have been able to develop a pipeline that has enabled us to build sequence classification architectures capable of reaching great performances, with precision and recall rates over 90%, on different recording settings. We have been able to develop a tool that would adapt to different lighting, camera placement and orientations, bringing the state of project closer to real world use cases.

Not only that, but also we have been capable of unifying this automated models with an easy and intuitive User Interface that will enable researchers to access this technology in a comfortable manner.

However, there is still remaining work on the project to be done. Here we will list some ideas on what future steps can be considered:

1. Experimenting with multi-class models. On our current approach, we treat each behaviour separately through binary classification. Although this results comfortable from a research point of view, in application is tedious to have one model for each action. In a future, if new behaviours want to be considered, having a separate architectures for each of them will be complex to implement. Doing this also requires a post-processing algorithm that chooses between outputs. A bad performance of

this algorithm can easily lead on a degradation of the predictions quality.

2. Try other feature encoding architectures. On this project we haven't explore any alternatives to ResNet50. It could be nice to consider other pretrained convolutional neural networks, as well as experimenting with architectures trained from scratch. This could lead to an improvement of frame encoding, making it more orientated to the classification task. Implementation of attention blocks could be considered to boost up the encoding process.

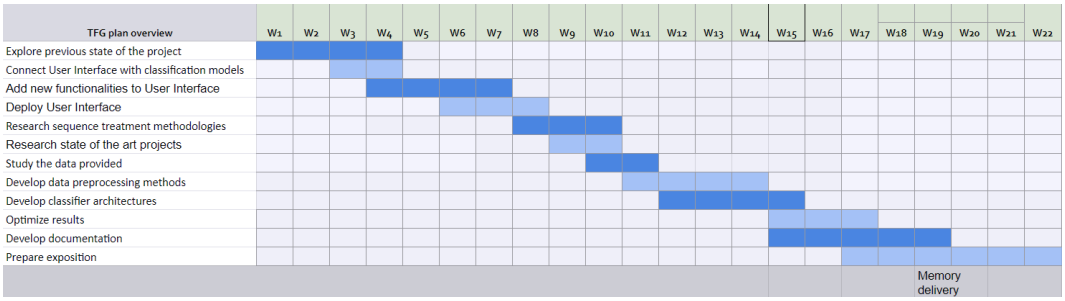
3. Improve Rearing metrics. Although they are not bad, rearing metrics still have room for improvement. It would be worthy to perform an exhaustive hyper-parameter tuning focused on boosting up Rearing metrics.

4. Perform a more exhaustive training. Due to the time and computing capacities we had, we were not able to train models for a larger number of epochs. It would be worthy to train them for longer time until being sure of reaching a point of overfitting.

5. Create a larger dataset applying data augmentation. Data augmentation boosted up a lot the performance of our models. Even-though that, a larger set could be considered so a more exhaustive training can be performed. Especially, in cases like transformers, we have seen that a small data set can easily lead to overfitting. Experimenting with more diversity and quantity of data could boost up some architectures.

6. Considering pretrained classifiers. There are some pretrained models like *Timesformer*, designed for sequence processing and classification tasks. It could be interesting to experiment with them.

To finish with this document, we will attach an schema of the followed project planing. The schema counts with 22 weeks, since it is considered that each week an amount of 20-25 hours was dedicted to the project on its different taks.



Bibliography

[1] He K., Zhang X., Ren S. et Sun J. (2016). Deep residual learning for image recognition. In 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 770-778). Las Vegas, NV, USA. https://arxiv.org/pdf/1512.03385

[2] Bangar S.(2022). ResNet Architecture Explained. Medium. https://medium.com/@siddheshb008/resnet-architecture-explained-47309ea9283d

[3] Kundu N. (2023). Exploring ResNet50: An In-Depth Look at the Model Architecture and Code Implementation. Medium.

[4] Scarff B. (2021). Understanding Backpropagation, A visual derivation of the equations that allow neural networks to learn. Medium. https://towardsdatascience.com/understanding-backpropagation-abcc509ca9d0

[5] Wang CF. (2019) The Vanishing Gradient Problem. The Problem, Its Causes, Its Significance, and Its Solutions. Medium. https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484

[6] Shinde S., Kulkarni U., Mane D. and Sapkal A. (2021). Health Informatics: A Computational Perspective in Health. Chapter 2.

[7] Hochreiter S., Schmidhuber J. (1997). Long Short-Term Memory. Neural Computation, 9(8), 1735-1780. https://deeplearning.cs.cmu.edu/F23/document/readings/LSTM.pdf

[8] Calzone O. (2022). An Intuitive Explanation of LSTM. Medium. https://medium.com/@ottaviocalzone/an-intuitive-explanation-of-lstm-a035eb6ab42c

[9] Sugandhi A. (2024). What is Long Short-Term Memory (LSTM) - Complete Guide. upGrad. https://www.knowledgehut.com/blog/web-development/long-short-term-memory

[10] Talathi S et all. (2015). Improving performance of recurrent neural network with relu non linearity.

[11] Weber N. (2017). Why LSTMs Stop Your Gradients From Vanishing: A View from the Backwards Pass. https://weberna.github.io/blog/2017/11/15/LSTM-Vanishing-Gradients.html

[12] Jozefowiez R., Vinyals O., Schuster M., Shazeer N., Wu Y. (2016). Exploring the Limits of Language Modeling. Google Brain. https://arxiv.org/pdf/1602.02410

[13] Huang Z., Xu W., Yu K. (2015). Bidirectional LSTM-CRF Models for Sequence Tagging. https://arxiv.org/pdf/1508.01991

[14] Anishnama. (2023). Understanding Bidirectional LSTM for Sequential Data Processing. Medium. https://medium.com/@anishnama20/understanding-bidirectional-lstm-for-sequential-data-processing-b83d6283befc

[15] Otsu N. (1979). A Threshold Selection Method from Gray-Level Histograms. IEEE Transactions on Systems, Man, and Cybernetics, vol. 9, no. 1, pp. 62-66. https://engineering.purdue.edu/kak/computervision/ECE661.08/OTSU_paper.pdf

[16] Lea C. Vidal R. Reiter A. Hager G.D. (2016). Temporal Convolutional Networks: A Unified Approach to Action Segmentation. https://arxiv.org/pdf/1608.08242

[17] Lassig F. (2021). Temporal Convolutional Networks and Forecasting. Unit8. https://unit8.com/resources/temporal-convolutional-networks-and-forecasting/

[18] Bai. S, Holter JZ., Koltun V. (2018). An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling. https://arxiv.org/pdf/1803.01271

[19] Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A.N., Kaiser L., and Polosukhin I. (2017). Attention is all you need. Advances in Neural Information Processing Systems. https://arxiv.org/pdf/1706.03762

[20] Giacaglia G. (2019). How Transformers Work. Medium. https://towardsdatascience.com/transformers-141e32e69591

[21] Sayak P. (2023). Video Classification with Transformers. Keras documentation. https://keras.io/examples/vision/video_transformers/

[22] Bahdanau D., Cho K. and Bengio Y. (2014). Neural machine translation by jointly learning to align and translate. https://arxiv.org/abs/1409.0473

[23] Niu Z., Zhong G. and Yu H. (2021). A review on the attention mechanism of deep learning. https://www.sciencedirect.com/science/article/pii/S092523122100477X

[24] Cristina S. (2023). The Attention Mechanism from Scratch. https://machinelearningmastery.com/the-attention-mechanism-from-scratch/#:~:text=The%20attention%20mechanism%20was%20introduced%20by%20Bahdanau%20et%20al.,information%20provided%20by%20the%20input.

[25] Rensink R.A. (2000). The dynamic representation of scenes, Visual Cognition. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=751e384c376dd2f6ccd50ef13db19dff13d22f3a

[26] Corbetta M. and Shulman G.L. (2002). Control of goal-directed and stimulus-driven attention in the brain. https://www.nature.com/articles/nrn755

[27] Wang Z., She Q. and Ward T.E. (2021). Generative Adversarial Networks in Computer Vision: A Survey and Taxonomy. https://www.researchgate.net/publication/349189619_Generative_Adversarial_Networks_in_Computer_Vision_A_Survey_and_Taxonomy

[28] Yeung S., Russakovsky O., Jin N., Russakovsky M, Mori G. and Fei-Fei L. (2017). Every Moment Counts: Dense Detailed Labeling of Actions in Complex Videos https://arxiv.org/pdf/1507.05738

[29] Abdu S.A., Yousef A.H., Salem A. (2021). Multimodal Video Sentiment Analysis Using Deep Learning Approaches, a Survey. https://www.sciencedirect.com/science/article/pii/S1566253521001299

[30] Souza J.S., Bedin E., Hirokawa Higa G.T., Loebens N., Pistori H. (2024). Pig aggression classification using CNN, Transformers and Recurrent Networks. https://arxiv.org/pdf/2403.08528

[31] Kopaczka M., Tillman D., Ernst L., Schock J., Tolba R. and Merhof D. (2019). Assessment of Laboratory Mouse Activity in Video Recordings Using Deep Learning Methods. https://www.lfb.rwth-aachen.de/bibtexupload/pdf/KCZ19b.pdf

[32] van Dam E.A., Noldus L.P.J.J and van Gerven M.A.J. (2020). âDeep learning improves automated rodent behavior recognition within a specific experimental setupâ. In: Journal of Neuroscience Methods 332, p. 108536. https://www.sciencedirect.com/science/article/pii/S0165027019303930

Annexed Documentation

W&B Bayesian Optimization results

. 2048 LSTM

seqı	batc	nun	nun	Accı	P▾	Prec	Recı
600	16	256	1	0.9261	0.6951	0.5338	0.77
600	32	256	3	0.9314	0.6712	0.5692	0.6662
1650	32	64	1	0.9166	0.6653	0.4944	0.8001
1650	32	64	3	0.8979	0.6611	0.4365	0.8484
600	16	256	2	0.9089	0.6578	0.47	0.8817
1650	16	128	3	0.9262	0.6569	0.5376	0.7077

. 512 LSTM

seqı	batc	nun	nun	Accı	P▾	Prec	Recı
900	32	128	1	0.9253	0.7895	0.5861	0.7832
1200	8	64	3	0.9168	0.7683	0.5443	0.875
300	32	256	1	0.9082	0.7671	0.5158	0.8814
900	8	256	1	0.8764	0.7669	0.4373	0.9549
900	32	64	3	0.8803	0.7631	0.4432	0.9113
1200	16	128	3	0.8916	0.7611	0.4705	0.9347

—-

. 512 BI-LSTM

seqı	batc	nun	nun	Accı	P▾	Prec	Rec
2100	8	64	1	0.9145	0.8017	0.5279	0.8708
900	32	256	3	0.9261	0.7976	0.5729	0.8437
900	32	128	3	0.8464	0.7639	0.3776	0.9729
900	32	256	3	0.8849	0.7615	0.4455	0.9082
900	32	128	2	0.8821	0.7611	0.442	0.9614
2100	32	256	2	0.8963	0.7594	0.4736	0.9099

. 512 TCN

batch_	kernel	num_l	num_u	sequer	Accura	PRC▾	Precisi	Recall
32	3	2	512	600	0.8962	0.6104	0.4803	0.8496
8	3	1	1024	600	0.8712	0.5954	0.4153	0.8023
32	3	2	512	600	0.8616	0.5603	0.4025	0.8809
32	5	3	1024	300	0.8246	0.5503	0.3424	0.8783
8	3	1	6256	600	0.848	0.548	0.3756	0.8561
32	3	2	512	300	0.8329	0.5469	0.3516	0.8558