UNIVERSITAT DE BARCELONA

FUNDAMENTAL PRINCIPLES OF DATA SCIENCE MASTER'S THESIS

Automated Mouse Behavior Recognition using LSTM and TCN Networks

---

Author:
Xavier DE JUAN

Supervisor:
Dr. Eloi PUERTAS

A thesis submitted in partial fulfillment of the requirements for the degree of MSc in Fundamental Principles of Data Science

in the

Facultat de Matemàtiques i Informàtica

June 30, 2022

UNIVERSITAT DE BARCELONA

Abstract

Facultat de Matemàtiques i Informàtica

MSc

Automated Mouse Behavior Recognition using LSTM and TCN Networks

by Xavier DE JUAN

Research studies in the neuroscience fields use to employ mice in their experiments which require a massive behavior analysis work. Due to the high requirement of time and effort, an automated solution is sought to reduce dedication in these tasks. Then, this report seeks this solution to automate the identification of mice behaviors in videos. Therefore, the problem being faced is a common computer vision problem also known as motion recognition which means the automatic identification of behaviors from video sequences. Specifically, the problem consists in, given a set of mice videos with their corresponding annotated behaviors, automatically identify these behaviors at frame level by using a deep learning model. Related studies have addressed this problem by using the most common computer vision action classification architectures such as Two-Stream Convolutional Networks, 3D Convolutional Networks or I3D Convolutional Networks among others. The architecture presented in this work aims to combine well-known Convolutional Neural Network (CNN) architectures, which generate an embedding from each frame, with Long Short-Term Memory (LSTM) networks or Temporal Convolutional Networks (TCN) to model the behaviors based on the embeddings generated by the CNN. These models have been tested in a dataset provided by the *Institute of Neurosciences* from the *Faculty of Medicine and Health* of the *University of Barcelona*. The results obtained in this dataset show scores around 90% of accuracy for the follwing mice behaviors: grooming, mid rearing and wall rearing. According to the experimental results, both LSTM and TCN models perform comparably showing great results but LSTM networks are the models that best suit to the problem and data given. The code generated in this work can be found in the following Github Repository.

Acknowledgements

I want to express my gratitude to all of my supervisors and contributors to this work for their outstanding assistance, hard work, and knowledge in getting this project completed.

To Dr Eloi Puertas for offering this project and guiding me through the course of this work.

To Dra Mercè Masana for giving me the opportunity to work in this project and providing me the data required for the development of the project.

And, finally, to Dr Sergio Escalera for giving me the technical support and guiding me to explore novel methods.

Contents

Chapter 1

Introduction

Several scientific and preclinical research studies in the fields of neurology, psychology, genetics and pharmacology widely employ rats and mice for their experiments. These trials need a significant amount of behavior analysis study. The most common way require human domain experts to examine each video carefully and manually annotate the animal behaviors at frame level. This method is tedious, labor-intensive, slow to complete, subjective and sometimes challenge to duplicate.

With the development of computer vision, computing power and machine learning techniques, researchers have turned to automated analytic approaches for assistance after spotting these imperfections. Consequently, this work will focus on the automation of annotating the mouse behaviors in videos at frame level.

This problem have been researched during the last years by applying the most common action classification architectures from computer vision. For instance, to address this problem, Simonyan and Zisserman, 2014 used the Two-Stream Convolutional Networks, van Dam, Noldus, and van Gerven, 2020 and Bohnslav et al., 2020 employed 3D Convolutional Networks, Ngoc Giang et al., 2019 made use of I3D Convolutional Networks and Zhang, Yang, and Wu, 2019 applied an stacking of a Convolutional Neural Network with an LSTM Network.

Then, the main goal of this work is to build a deep learning model to address the behavior recognition problem. To achieve this objective, the following specific ones are proposed: review the models already applied to the problem, build different models to deal with the problem, compare the performance of each model and, finally, select the best one.

In order to achieve these goals the following methodology will be employed. Split the multi-class behavior classification problem into independent binary problems. Apply some data transformations to the raw videos such as cropping frames and split the video into smaller sequences. With respect to the model, build different feature extractors to build embeddings from each frame and build powerful temporal backend models for the classification. For each model perform a hyperparameter search to find the best configuration and evaluate using different metrics to take into consideration the class imbalance from the problem. Finally, make a comparison of the models based on the metrics defined in order to choose the model that best suits the problem stated.

To sum up, this work presents a multi-class problem based on the behavior classification of mice given a set of annotated videos. To address this problem, computer vision models will be used to generate embeddings from each frame which will be used by a temporal deep learning model that will classify the behavior of each frame.

The rest of this work is organized as follows: Chapter 2 reviews the related work applied to behavior and action recognition. The methodology to address the problem is presented in Chapter 3 followed by the experimental setup and procedure in Chapter 4. After that, the results of the case study are presented and discussed

in Chapter 5. And, finally, Chapter 6 concludes the work and highlights the future work.

Chapter 2

Related Work

Computer vision tasks has gained popularity and research due to the appearance of modern algorithms based on deep learning, in particular, the Convolutional Neural Networks (CNN) which provided a dramatic improvement in terms of performance compared to the traditional image processing techniques. The performance and efficiency of a CNN is determined by its architecture for which different structures have been presented during the last years such as ResNet50 (He et al., 2015) and InceptionResNet (Szegedy et al., 2016) which will be used later in the experiments.

The course of this work focuses on motion recognition tasks for animals which have also achieved great results and has gained popularity as a consequence of its application in the automation of laboratory and medical tasks. Motion recognition is defined as the automatic identification of behaviors from images or video sequences. In the same way as before, motion recognition have been boosted by the introduction of deep learning enabling the usage of complex CNN architectures in image and video processing.

An example application of these new computer vision techniques is the DeepLabCut™toolbox (Nath et al., 2018). DeepLabCut™is an efficient method for 2D and 3D markerless pose estimation based on transfer learning with deep neural networks. In relation to this work, it takes the videos as input an outputs a csv document with the position of each part of the mouse body.

The case study from this report addresses a frame by frame classification from mice recordings. The videos studied present an open field with one mouse inside with the aim to identify different behaviors during the course of the video. This computer vision problem is identified as a many to many video classification as each frame of the video is tagged with a label, the behavior being performed at that specific moment. As the data analyzed are videos, each frame has a dependence with the previous and the following ones. So, in order to preserve the temporal continuity of the video, the problem should not be addressed as an independent image classification problem.

In this section, an overview of different approaches to deal with this kind of problem will be given. The following subsections present different techniques already applied to this problem as well as the ones applied in this work.

2.1 Two-stream ConvNets

This architecture is based in the idea of decomposing a video into its spatial and temporal components. The spatial part contains information about scenes and objects depicted in the video while the temporal part holds the movement of the observer and the objects. The video recognition architecture is divided into two streams where each stream is a CNN that combine their output by late fusion. The first stream takes as input each frame individually while the second one receives as input

the Optical Flow (Efros et al., 2003) of multiple frames. Simonyan and Zisserman, 2014 applied this technique for human action recognition on the standard video actions benchmarks of UCF-101 (Soomro, Zamir, and Shah, 2012) and HMDB-51 (Kuehne et al., 2011). Kopaczka et al., 2019 used the same methodology to recognize some mouse behaviors achieving a good result of 86.4% accuracy.

2.2 3D ConvNets

3D Convolutional Networks extend the layers and pooling kernels from 2D CNN architectures to handle the time factor of a video. The network inputs are sequences of images where 3D kernels are applied allowing to model local spatio-temporal information. van Dam, Noldus, and van Gerven, 2020 and Bohnslav et al., 2020 used this approach to automatically classify different mice behaviors achieving great results matching expert-level human performance.

2.3 I3D ConvNets

The Inflated 3D ConvNet (I3D ConvNet) architecture combines the strengths of 3D and Two-stream ConvNets. The 2D ConvNet filters and pooling kernels are expanded to three dimensions and in parallel has the Optical Flow stream. Ngoc Giang et al., 2019 used this technique, along with pre-trained human action datasets models, to a mouse behavior recognition task which showed better performance in comparison to previous research. They also showed good promises of transfer learning to other animal behavior recognition tasks without significant modification to the models' architecture.

2.4 Transformer-like architectures

Transformer (Vaswani et al., 2017) architectures had a great impact in deep learning during the last years. Although they were thought to be used in sequence to sequence tasks, they have been applied to other tasks and areas, such as computer vision, showing impressing results in this field. Neimark et al., 2021 propose Video Transformer Network that allows to encode spatio-temporal inputs as well as they benefit from self-attention mechanisms showing competitive performance evaluating them in different datasets.

2.5 ConvNet + LSTM

This architecture combines the strengths of the convolutional and the recurrent neural networks (RNN) model the spatial and temporal information of the videos. The network is composed by a CNN model, usually a common pre-trained CNN architecture, and some recurrent layers at the top such as Long short-term memory (LSTM) (Hochreiter and Schmidhuber, 1997). With this architecture, individual frames are input to the CNN and sequences of outputs are then fed into the LSTM network that finally generates the classification label. Zhang, Yang, and Wu, 2019 presented a method that extracts features of the mouse agents by using a CNN as an embedding network and then an LSTM network that, based on that embeddings, model

the behaviors taking into account the temporal component of the video. They concluded that their embedding method outperforms the traditional hand-crafted feature based methods and the results showed a better performance compared to the two-stream networks.

2.6 ConvNet + TCN

This last architecture is almost the same as the previous method with the exception that the model at the top of the CNN is different, that is the one called Temporal Convolutional Networks (TCN) (Lea et al., 2016). These kind of networks were presented as a unified approach between convolutional networks and RNN based networks that hierarchically captures relationships at low-, intermediate-, and high-level time-scale. In that sense, these networks work quite similar to the RNN ones but can be trained in a fraction of the time it takes to train the RNN. Scherer et al., 2020 tackled a hand gesture recognition problem by applying a 2D CNN combined with a TCN for the time sequence prediction. They obtained great results comparable to the current state-of-the-art networks on that field while using this TCN-based network that is much smaller than these state-of-the-art methods.

Chapter 3

Methodology

The research objective is that given a mouse recording identify different patterns of that mouse at frame level, that is to identify some mouse behaviors such as grooming, mid rearing or wall rearing and tag each frame of the video according to that identification. Figure 3.1 illustrate these three behaviors. During a grooming the mouse lick its fur, groom with the forepaws, or scratch with any limb as shown in 3.1a. A rearing occurs when the mouse put its weight on its hind legs, raise, its forelimbs from the ground, and extend its head upwards. A mid rearing consist in a rearing in the middle of the case as shown in 3.1b and a wall rearing when the rearing takes place on the wall of the case as shown in 3.1c.

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

FIGURE 3.1: Mouse Behaviors

So, in that sense, this is a common computer vision problem known as a many to many video classification. That means a sequence of frames is processed and each of them is tagged according to the behavior identified by the model at that frame. As there are more than one behavior, this is a multi-class problem, each frame has more than two options to be labelled with. To simplify the model and keep the behavior classification independent, the problem will be split into different binary classification problems, one for each behavior.

To address this problem, a common deep learning approach in video classification will be tuned. In this approach, the model architecture consists of a pre-trained convolutional network as backbone and the addition of some layers at the top to model the temporal factor of the video. The model backbone receives as input a 2D image and is responsible of extracting features of that image and outputs a 1D vector, then the top layers models the temporal factor by using multiple frames encoded as 1D vectors as input and outputting a label for each frame. On the following sections a deeper analysis is made by describing the different backbones used such as ResNet50 or InceptionResnetV2, the training strategy used, the selection of hyperparameters and the kind of top layers used such as LSTM or TCN. Finally, the

evaluation metrics will be introduced which will be used to compare the results of each experiment.

3.1 Data

The dataset has been provided by the research group led by *Dr. Masana, Professor of the Department of Biomedicine, Institute of Neurosciences, Faculty of Medicine and Health Sciences, University of Barcelona*. They provided seventeen mice recordings manually labelled at frame level with the corresponding behavior. In addition, they also supplied the outputs of the DeepLabCut™software. This software provides a computer vision package for animal pose estimation so, for each frame, the position of some parts of the mouse are stored in a csv file.

Each video is provided as an mp4 containing a recording of a mouse inside a square white box, the open field box as shown in the left of 3.2. The video is 11 minutes long with a frame rate of 10 frames per second, that means 6600 frames per video.

The DeepLabCut data are csv files, one for each video, containing the animal pose estimation using ResNet50. For each frame in the video, there is a row with the position of different body parts of the mouse such as the crown, the tail, the midbody or the paws, but in this research only the midbody position will be used. In the same file, the manually annotated labels for each behavior can be found.

Once having all of these files, the first step has been to make a first processing of them such as split the videos into its frames and extract the labels from the DeepLabCut files. Then, in order to focus the attention to the mouse, each frame of the video has been cropped around the mouse, as shown in the right of 3.2, by taking as the center the midbody position given by DeepLabCut.

After that, the second step has been to do a brief data analysis of the dataset to describe the data and detect characteristics which may affect the later model. First, examining the labels distribution within the frames of all videos, as shown in table 3.1, there is an imbalance between the positive and negative samples, the positive samples only represent one tenth of the total samples. To tackle that, a possible approach could be to augment the positive samples but, as each sample represents a frame from a video, this approach could affect the temporal factor of the sequences.

FIGURE 3.2: Left: Mouse open field. Right: Mouse cropped at the center.

	Grooming	Mid Rearing	Wall Rearing
Positive Samples	12077 (10.76%)	11881 (10.59%)	14759 (13.12%)
Total Samples	112200	112200	112200

TABLE 3.1: Class distribution for each behavior. Each sample represents a frame from a video

For that reason, the strategy used to face this problem has been to assign weights to each frame giving more importance to the positive samples in the loss function computation. Another characteristic that can be extracted from the data is the duration of each behavior in the video. In this case, the mean duration of the grooming, mid rearing and wall rearing behaviors are 107, 24 and 21 frames respectively. The purpose of this information is to get a minimum length of frames to cut the videos in smaller sequences. The videos itself have 6600 frames being too long to process them entirely so it is important to have a minimum length to avoid cutting the behaviors in the middle of them. For this reason, each video is cut in smaller sequences such as 150, 300 or 600 frames each resulting in 44, 88 and 11 sequences respectively.

The last step on this data has been to split the data into two different sets: training and testing. The training set is used for training and validation but the strategy used will be introduced in the following sections. The training-test partition will be used on the next stages, in order to train and tune the different hyperparameters the training set will be used. Then, the test set will be used to perform a final evaluation of the model that will be included in this report. The split has been done randomly by choosing 5 entire videos for testing and leaving the rest, 12, for training. The table A.1 from the annex A has the name of each video and the set which it belongs to.

3.2 Methods

The methods used to address the problem follow a similar structure as said previously. The architecture of these models have as backbone a pre-trained CNN and then some other kind of layers at the top of this backbone. As the data being used are images, a CNN must be used to extract features of them. For this task, the TensorFlow (Abadi et al., 2015) library provides several well-known architectures such as ResNet50 or InceptionResnetV2 and, in addition, the pre-trained weights in the ImageNet dataset (Deng et al., 2009) are also available. The backbone receives an 2D image as input and outputs a 1D vector that will be fed in the top layers. The aim of these top layers is to model the temporal factor of the data, the sequentiality of the video frames. For this reason, the kind of layers which will be used are LSTM layers or TCN layers that allow to model it.

LSTM, a recurrent deep neural network, predicts a sequence of data using feedback connections and loops in the network and is able to learn long-term dependencies. Generally speaking, LSTM consist of a chain of repeated cells (i.e. main layers or moduals). Each cell in an LSTM model typically consists of three interacting layers including *forget gate layer*, *input gate layer* and *output gate layer*. These gates are made of a sigmoid neural net layer followed by a point-wise multiplication operation, which enables the LSTM model to remove or add data from or to the flow of information through each cell. In each LSTM cell, the forget gate layer is the initial layer. This gate determines whether parts of the information are disregarded. The input gate layer followed by a tanh layer filters and updates the new input data and produces the update for the cell state. The tanh layer then aggregates the data from

the previous layers and updates each state value. Finally, the last layer called the output gate layer generates the output based on the cell state.

To sum up, LSTM is a Recurrent Neural network capable of capturing temporal properties by remembering the previously observed data which makes it suitable for the problem being addressed.

Temporal Convolutional Networks (TCNs) are a type of time-series model that overcomes previous constraints by capturing long-term trends via a hierarchy of temporal convolutional filters. TCNs are divided into two categories: 1) Encoder-Decoder TCN (ED-TCN) uses only a hierarchy of temporal convolutions, pooling, and up-sampling while successfully capturing long-range temporal patterns. The ED-TCN has a few levels (three in the encoder), but each layer is made up of a series of long convolutional filters. 2) Instead of pooling and up-sampling, a dilated TCN uses dilated convolutions and adds skip links across layers.

A dilated convolution is a convolution in which a filter is applied across a larger region than its length by skipping input values with a specified step. In time series, the model must memorize enormous amounts of data, which the causal convolution cannot handle, but a dilated convolution with a bigger filter created from the original filter by dilation with zeros is far more efficient to do the task.

At the end, TCN is a Convolutional Neural Network that through some stack of layers captures the temporal properties of time series data which, again, makes it suitable for the problem being addressed.

3.2.1 CNN + LSTM

As said previously, the structure of this model is composed by a pre-trained CNN as backbone and LSTM layers at the top of it. For this task, the TensorFlow library and the keras interface have been used to build the model described.

Regarding the backbone, two different backbones have been proposed to be used: ResNet50 and InceptionResnetV2. These two architectures can be instantiated from TensorFlow by preloading the weights pre-trained on the ImageNet dataset.

On top of the backbone, different layers have been added to refine the predictions of the model. This model uses LSTM layers to model the temporal factor, Dropout layers to regularize and a fully connected layer as output for classification purposes.

The figure 3.3 displays a diagram of the architecture described by using the LSTM network. On section 4 there is a deeper description of this architecture as well as the hyperparameters used for each experiment.

3.2.2 CNN + TCN

This method is quite similar to the previous one, it uses a pre-trained CNN as backbone and in the top of that appends other kind of layers. The difference in this case is that the LSTM layers are replaced by TCN layers.

In the same way, the model has been tested with two different backbones: ResNet50 and InceptionResnetV2. Then at the top of this backbone, the network is similar to the previous one but using TCN layers. In this case, the TCN layer is not available in TensorFlow but the TCN implementation by Remy, 2020 offers compatibility with the Keras API and has been used. The implementation can be found in the following Github repository *keras-tcn*[1]

[1]https://github.com/philipperemy/keras-tcn

The figure 3.3 displays a diagram of the architecture described by using the TCN network. The section 4 also contains different experiments for this method by using different combinations of the model architecture and hyperparameters.

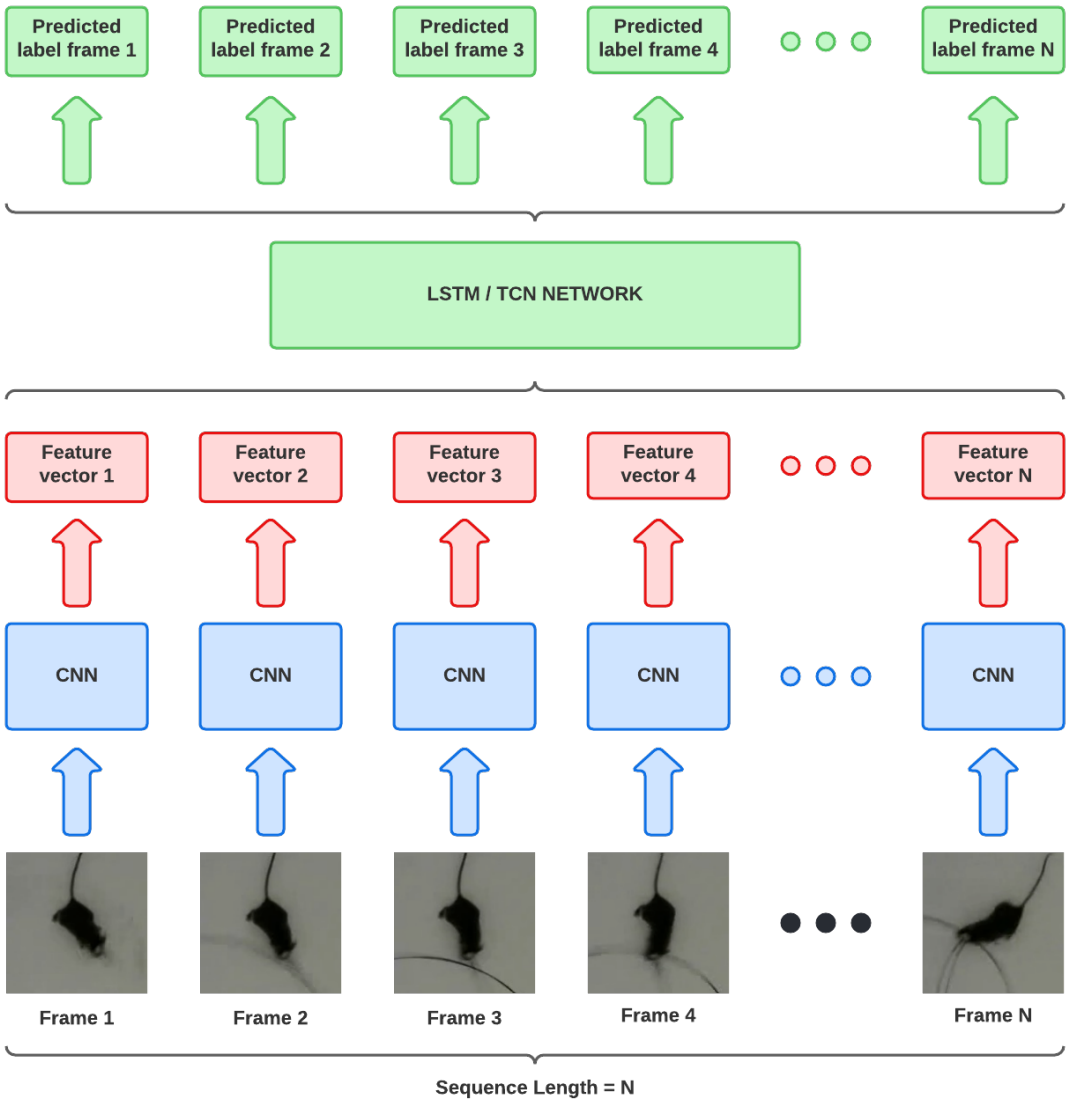

FIGURE 3.3: Architecture diagram of the models CNN + LSTM/TCN. The CNN represents one of the pre-trained models: ResNet50 or InceptionResNetV2.

3.3 Training Strategy

At this point is important to define the strategy that will be followed during the experiments. Each experiment included in the following section summarizes a list of executions by permuting the configuration of layers and hyperparameters. So, in order to find the final and best setting, the strategy will be to cross-validate each model by using the training set.

The main idea is that for each experiment the training set will be partitioned into two different sets: training and validation. With these two, the model is trained

by using the data in the training set and then evaluated with the validation data that has not been used in the training stage and select the best model among each configuration.

However, there is a problem with this approach, the selection bias. This bias may occur when splitting the initial training set into training and validation in such a way that proper randomization is not achieved and the results of each experiment has a dependence on such split. To tackle this problem the cross-validation is introduced, this strategy performs several splits into train and validation such that each model is trained and evaluated in different partitions of the data. At the end, the results from each partition are grouped and compared with the other experiments avoiding the selection bias.

As a result, an adaptation of the **leave one out** cross-validation strategy will be applied to each intermediate experiment in order to get the best configuration for each one. The leave one out strategy consists of making a partition of the dataset by leaving only one sample in the validation set and the rest for training. The adaptation made in this work considers an entire training video as the sample left out instead of a single sequence of a video. In fact, the samples of the dataset should be entire videos but in order to reduce the memory requirements each video is cut into smaller sequences that are interpreted as samples. But, as a real world application, the samples given are entire videos that then are processed by splitting them into smaller sequences.

To summarize, the strategy followed by the experiments will have a cross-validation and an evaluation phase. The cross-validation phase will take each video from the training set as validation, evaluating the model with it at the end of training, and using the rest for training. After training and evaluating the model into each video independently the results will be grouped and the best model will be selected. Finally, in the evaluation phase, the selected model will be trained using all training videos without making any split and then it will be evaluated by using the testing set which will output the results that will be reported in the results section.

3.4 Evaluation

Lastly, another important component on this configuration is the definition of a common set of metrics to evaluate the experiments under the same conditions. As exposed previously, the entire dataset is split into two different sets: training, and test. This split will benefit the evaluation of each experiment and also the comparison of their results. Each experiment defined in section 4 will run different configurations of their base setting by modifying the number of layers, the optimizer or the hyperparameters. Then the metrics defined below will be used to chose the best configuration among the different combinations by evaluating each one with the cross-validation strategy defined in the previous subsection. Finally, once having the best setting the model will be evaluated on the test set and the results of each metric will be outlined in this report. Having defined this procedure the following metrics will be used to evaluate the experiments.

Binary Accuracy. The accuracy is the typical metric used to give an overview of how well the model is performing. Basically, the accuracy is computed as the number of samples classified correctly divided by the total number of samples. However, knowing that the dataset is imbalanced, the value of this metric does not give enough information to evaluate the model performance so other metrics should be used in addition of this one.

Confusion Matrix. The confusion matrix is not a metric by itself but it is a tool that allows to visualize the overall performance. The confusion matrix reports the following information: **true positives** and **true negatives** being the positive and negative samples, respectively, that were correctly classified, and the **false negatives** and **false positives** being the positive and negative samples, respectively, that were incorrectly classified.

Precision. The precision metric is the percentage of predicted positives that were correctly classified. This metric is defined in terms of the previous concepts as the number of true positives divided by the sum of the number of true positives and the number of false positives.

Recall. The recall metric is the percentage of actual positives that were correctly classified. This metric is also defined in terms of the previous concepts being the number of true positives divided by the sum of the number of true positives and the number of false negatives.

Area under the curve of the Precision-Recall Curve. The precision-recall curve shows the tradeoff between precision and recall for different threshold values. The maximization of tha area under this curve will represent both high recal and high precision, in other words, by the definition of recall and precision, the minimum number of false negatives and false positives.

Chapter 4

Experiments

The aim of this section is to give further detail about the experiments configuration: model architecture and best hyperparameters. The following subsections describe each of the experiments giving specifying the final model architecture and hyperparameters and also which other configurations have been tested in the experiment to reach this final setting. Notice that each experiment is being run for each binary problem defined previously so, in order to simplify the hyperparameters selection, the results for each binary problem will be aggregated and the best model performance in that aggregation will be chosen. Another important detail of the model is that due to different machine limitations the construction of the full model, backbone with top layers, and its run has been impossible to perform. For this reason the model has been split into its two parts which also denies the possibility of performing a two stage training that consists of freezing the backbone layers in the first stage and unfreezing them in the second one. Leaving aside this training strategy, having the partition of the model into its parts, the procedure in the experiments will be the following. Firstly, input each frame of each video into the backbone model getting as output a 1D representation of the images. And secondly, create a model with the top layers and input the sequences of 1D vectors got from the previous step and outputting the classification label for each frame. It can be noticed that the output of the backbone is independent of the model used at the top so, both backbones will be run separately storing their output to disk and then the top model will load that data for training. Having this in mind, the following subsections are describing the experiments run[1].

4.1 ResNet LSTM

This first model combines a ResNet50 with and LSTM network at the top. The implementation uses the ResNet50V2 from keras library as an image feature extractor using the pre-trained weights on the ImageNet dataset using an average pooling layer instead of its classification layer.

Once having the output of the pre-trained CNN, the experiment training starts. Firstly, the data is loaded and each video is split in sequences of a fixed length. Then, by using the *leave one out* strategy the LSTM model is cross-validated through each video.

The LSTM model has been built by using the *Sequential* class from keras. It has an input layer with shape the sequence length by the number of features from each frame. Next, there is a *Dropout* layer with rate 0.5 followed by some LSTM layers

[1]The experiments in this section enumerate some of the hyperparameters used in the experiments but not all of them are listed or included in the table 4.1. It is assumed that the ones that are not included use the default value given by the Python library where they are implemented.

and with a classification layer at the end. The classification layer is composed by a fully connected layer with shape 1 and activation function sigmoid wrapped by a *TimeDistributed* layer to output a classification label for each frame. In order to balance the binary problem, the sample weights are computed by using the following formula:

$$\begin{aligned} \text{weight for negative samples} &= \frac{\#\text{ samples}}{2 \cdot \#\text{ negative samples}} \\ \text{weight for positive samples} &= \frac{\#\text{ samples}}{2 \cdot \#\text{ positive samples}} \end{aligned} \tag{4.1}$$

Then, the model is trained using *Adam* as optimizer with its default parameters, the *binary cross-entropy* as the loss function for 50 epochs. In order to prevent overfitting, the *Early Stopping* callback has been used which monitors the area under the precision-recall curve with a patience of 5.

For this experiment, the tuned hyperparameters are the following: the sequence length with values 300, 600 or 1650; the number of LSTM layers being 1, 2 or 3; the number of LSTM units with 64, 128 or 256 reducing them to the half at following layers, and the batch size with values 8, 16 or 32.

Finally, by using the metrics defined in the previous section, the results from each behavior and cross-validation partition are aggregated. Then, the best hyperparameter configuration have been chosen and reported in the Table 4.1.

4.2 InceptionResNet LSTM

This second model combines a InceptionResNet with and LSTM network at the top. The implementation uses the InceptionResNetV2 from keras library as an image feature extractor using the pre-trained weights on the ImageNet dataset using an average pooling layer instead of its classification layer.

The procedure in the experiment is exactly the same as the first experiment with the exception of the input shape to the LSTM model due to the backbone output shape. The hyperparameters tuned are identical as the experiment 1 and the best configuration is reported in the Table 4.1.

4.3 ResNet TCN

This third model is similar to the first one but uses a TCN model at the top. The ResNet output from the first experiment has been reused as the input for the TCN model.

The scheme of the experiment is the same as in the previous experiments but the model built uses TCN layers instead of the LSTM ones. That means the hyperparameters to be tuned are also different. These are the number of TCN layers being 1, 2 or 3; the number of TCN filters with values 256, 512 or 1024; the TCN kernel size being 3 or 5, and the type of normalization applied in the residual layers: batch, layer, weights or no normalization. The other training and evaluation hyperparameters tested are the same as in the previous experiments. And, again, the best configuration is reported in the Table 4.1.

4.4 InceptionResNet TCN

This last experiment combines InceptionResNet and TCN layers at the top. The output of the backbone has been reused from the second experiment and the hyperparameter space searched is the same as the third experiment as it also uses the TCN model at the top. Again, the best configuration is reported in the Table 4.1.

	Experiment 1	**Experiment 2**	**Experiment 3**	**Experiment 4**
Backbone	ResNet	InceptionResNet	ResNet	InceptionResNet
Sequence Length	300	600	300	300
Dropout Rate	0.5	0.5	0.5	0.5
# LSTM layers	3	2	-	-
# LSTM units	64	64	-	-
# TCN layers	-	-	1	1
# TCN units	-	-	256	256
TCN Kernel Size	-	-	3	3
TCN norm	-	-	batch	batch
Optimizer	Adam	Adam	Adam	Adam
Learning Rate	0.001	0.001	0.001	0.001
Loss	Binary Crossentropy	Binary Crossentropy	Binary Crossentropy	Binary Crossentropy
Batch Size	8	16	32	8
Epochs	50	50	50	50
ES monitor	Validation PRC	Validation PRC	Validation PRC	Validation PRC
ES patience	5	5	5	5

TABLE 4.1: Experiments best hyperparameters configuration.

Chapter 5

Results

This section aims to present the experimental results introduced in the previous sections along with the metrics evaluation in the test set. Additionally, a discussion of each of the experiments is included as well as a comparison between them in order to select the best model performance for this problem.

The following tables and figures present the results from the experiments on each of the problems split by the behavior to be analysed: groomings, mid rearings and wall rearings. The tables show the numeric metrics defined previously: binary accuracy, precision, recall and the area under the precision-recall curve; and the figures display the confusion matrix which contains the number of frames correctly and incorrectly classified.

Firstly, the results for the grooming results are shown in the Table 5.1 and in the Figure 5.1. From table 5.1 it can be seen that the experiments 3 and 4, the ones using a TCN network, outperform the accuracy of the LSTM ones by $0.04 - 0.05$. With regard to precision, experiments 3 and 4 are also achieving higher results than the first two experiments. But when looking at the recall values, it can be observed that the first two experiments obtained very high results in contrast to the last two experiments beating them by a difference of $0.3 - 0.4$. Then, as the area under the precision-recall curve tries to maximize both precision and recall, the results favor the first two experiments due to the large difference in the recall values.

Now, looking to the confusion matrices in figure 5.1, the previous values of precision and recall can be understood. As a recap, the precision value stands for $\frac{TP}{TP+FP}$ where TP are the true positives (top left corner) and FP the false positives (top right corner). So, as the figure shows, the first two experiments report a larger number of false positives than the last two experiments and that is why these last two are producing greater precision values. Regarding the recall, which stands for $\frac{TP}{TP+FN}$ where FN are the false negatives (bottom left corner), the previous behavior is now reversed, the first two experiments report a lower number of false negatives than the last two which produce greater recall values for the first two as a consequence.

Secondly, the resulting metrics of mid rearings are presented in the table 5.2 and the confusion matrices in the 5.2. By looking at the table 5.2, the accuracy shown is similar along the four experiments with values around 0.9 but having the last experiment as the best value for this metric. Regarding the precision values, the third experiment presents a lower value than the other three, all around $0.5 - 0.6$, and, again, the last experiment is achieving the best value. When looking at the recall metric, it can be observed that the experiment 3 has the worst value among the experiments while the first two almost achieved the 0.8 and the last experiment reports a slightly worse value than these first two. As a consequence, the area under the precision-recall values benefit the first two experiments which achieve the

best values, followed by the fourth experiment and finally by the third experiment because of the poor precision and recall values reported.

Again, when looking at the confusion matrices in figure 5.3, the precision and recall metrics are self-explained. The first two experiments have a similar number of true positives, false positives and false negatives. Then, for these two, as the number of false positives is high the precision value is low and, as the number of false negatives is small, the recall value is high. Regarding the third experiment, the precision and recall values were both small and the small number of true positives is one of the causes. It is observable that the number of false positives is not too high but is greater than the number of true positives leading to a precision value smaller than 0.5. And, when looking at the false negatives, it can be seen that is ten times greater than the true positives producing a recall value of 0.1. The last experiment shows a similar behavior of false positives and negatives, both below the number of true positives, which results in similar values of precision and recall around 0.6.

Lastly, the wall rearing results are presented in Table 5.3 and in Figure 5.3. Looking at the table 5.3, it can be noted that the first experiment has the best results for all four metrics. In terms of accuracy, all four experiments achieve slightly similar values but when looking at the precision and recall values the third experiment behaves different from the rest. In both metrics, the values from the experiments 1, 2 and 4 are quite similar, around $0.6 - 0.8$ while the values from the experiment 3 are not reaching this range. Regarding the area under the precision-recall curve, the behavior matches the previous two metrics, similar values between experiments 1, 2 and 4 and a lower value for the experiment 3.

Finally, by looking at figure 5.3, the difference between these last metrics is explained. Experiments 1, 2 and 4 have similar values of true positives which are between 3000 and 3500 while the experiment 3 reports 2500. Regarding the number of false positives, experiments 1 and 2 have values between 1600 and 1700 which produce an almost equal precision. The experiment 4 has a higher number of false positives which produce a lower precision value. But, the experiment 3, has a higher number of false positives combined with a lower number of true positives results in the worst precision value among the experiments. And, with regard to the recall value, it can be observed that the behavior in the number of false negatives is quite similar to the previous one. Experiments 1 and 2 report similar values, experiment 4 is also near to these values and experiment 3 has a higher number of them. This way, the recall values between experiments 1, 2 and 4 are almost equal because they have similar number of true positives and false negatives, but the recall value from experiment 3 is worse due to the difference in the number of true positives and false negatives with the other experiments.

	Experiment 1	**Experiment 2**	**Experiment 3**	**Experiment 4**
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
AUC PRC	0.679	**0.690**	0.588	0.586

TABLE 5.1: Grooming Results. **Experiment 1**: ResNet50 + LSTM. **Experiment 2**: InceptionResNetV2 + LSTM. **Experiment 3**: ResNet50 + TCN. **Experiment 4**: InceptionResNetV2 + TCN

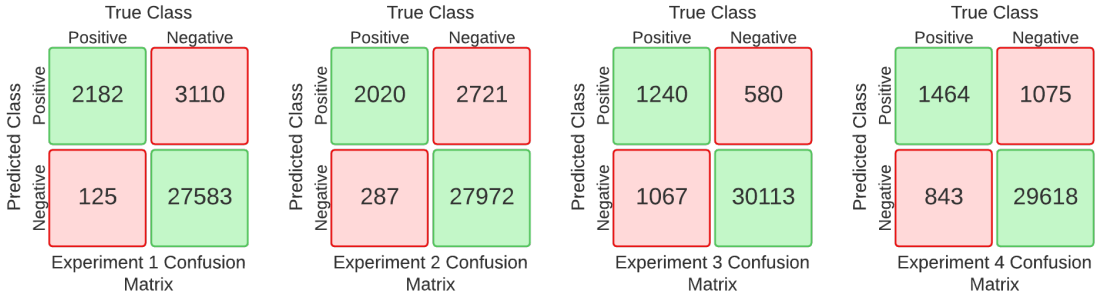

FIGURE 5.1: Grooming Confusion Matrices. **Experiment 1**: ResNet50 + LSTM. **Experiment 2**: InceptionResNetV2 + LSTM. **Experiment 3**: ResNet50 + TCN. **Experiment 4**: InceptionResNetV2 + TCN

	Experiment 1	**Experiment 2**	**Experiment 3**	**Experiment 4**
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
AUC PRC	**0.702**	0.691	0.201	0.602

TABLE 5.2: Mid Rearing Results. **Experiment 1**: ResNet50 + LSTM. **Experiment 2**: InceptionResNetV2 + LSTM. **Experiment 3**: ResNet50 + TCN. **Experiment 4**: InceptionResNetV2 + TCN

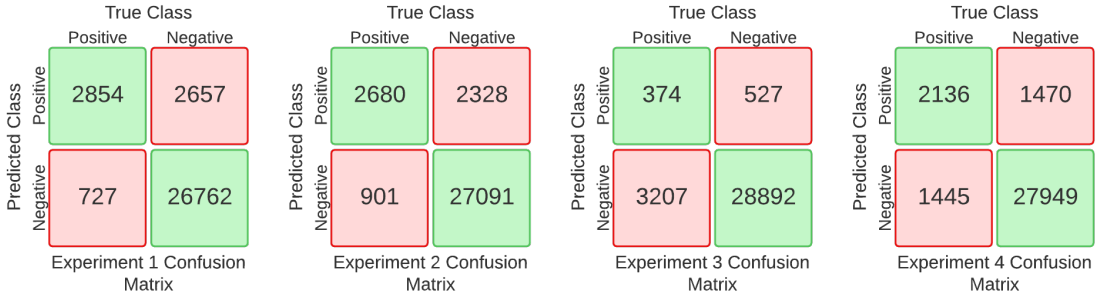

FIGURE 5.2: Mid Rearing Confusion Matrices. **Experiment 1**: ResNet50 + LSTM. **Experiment 2**: InceptionResNetV2 + LSTM. **Experiment 3**: ResNet50 + TCN. **Experiment 4**: InceptionResNetV2 + TCN

	Experiment 1	**Experiment 2**	**Experiment 3**	**Experiment 4**
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
AUC PRC	**0.792**	0.770	0.411	0.716

TABLE 5.3: Wall Rearing Results. **Experiment 1**: ResNet50 + LSTM. **Experiment 2**: InceptionResNetV2 + LSTM. **Experiment 3**: ResNet50 + TCN. **Experiment 4**: InceptionResNetV2 + TCN

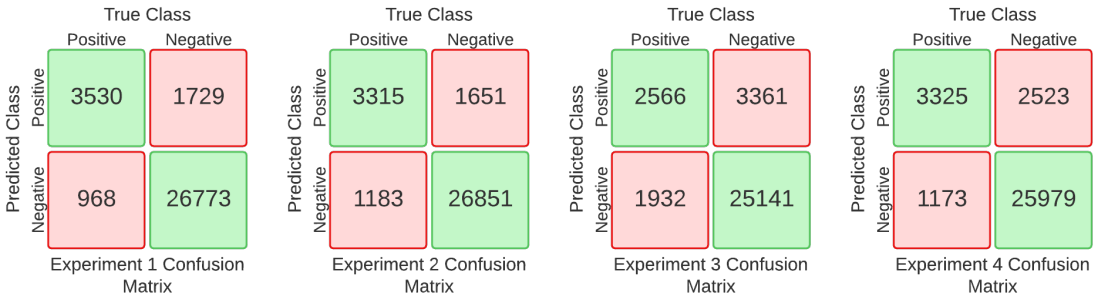

FIGURE 5.3: Wall Rearing Confusion Matrices. **Experiment 1**: ResNet50 + LSTM. **Experiment 2**: InceptionResNetV2 + LSTM. **Experiment 3**: ResNet50 + TCN. **Experiment 4**: InceptionResNetV2 + TCN

5.1 Discussion

After looking at the results from each experiment, it is important to compare the performance of each experiment in terms of the results obtained and the characteristics of each model, that is the kind of layers used and the pre-trained CNN used.

In this setting, two different splits can be made: the first one groups the experiments by the backbone used and the second one that groups them by the kind of layers used (LSTM or TCN layers). Based on these splits, the following paragraphs will analyse the similarities and differences between the performances of each group with the aim of giving a well-founded conclusion.

Before delving into the comparison, it is important to remember that the problem faced has an important imbalance between the true and the negative class. As said previously, the accuracy metric is not a significant metric to rank the experiments and that is why precision, recall, area under the precision-recall curve and the confusion matrices are introduced. Another important detail is which kind of error is preferable to commit: false positives or false negatives. As the problem consists in identifying the mouse behaviors, false positives will be more tolerable while false negatives should be minimal. In this way, even if misclassified patterns appear, the vast majority of true patterns will be identified. For this reason, higher recall values will be preferable over the precision ones.

Now, moving to the comparison, the backbone split will be considered first to look for similarities and differences and then the split by the kind of layers used at the top.

For the backbones split, that is the pre-trained CNN, used to build the model the experiments are split divided as follows: experiments 1 and 3 used a pre-trained ResNet and experiments 2 and 4 used a pre-trained InceptionResNet. Remembering that the models from experiments 1 and 2 consist of LSTM networks at the top and experiments 3 and 4 consist of TCN networks, the comparison can be made on these two groups by considering that each experiment in the group uses one backbone or the other.

On this comparison, the expected result would be that InceptionResNet experiments outperform the ResNet ones due to the evolution of this architectures. The InceptionResNet is a newer and more complex CNN architecture built on the Inception family combined with the usage of residual connections as the ResNet architecture

does.

Results illustrate that, in most cases, InceptionResNet outperform the ResNet metric results. However, this rule is not always fulfilled. Grooming results show that the best accuracy, precision and recall are achieved by ResNet models. Mid rearing results have the best recall and area under the precision-recall curve achieved by a ResNet model. And, wall rearing results, the best results of all metrics are achieved by a ResNet model. Bearing in mind that the outputs of ResNet and InceptionResNet are vectors of 2048 and 1536 features respectively, a valid hypothesis could be that vectors with more features give a more accurate representation of the image resulting in better metrics results.

Then, although InceptionResNet models outperform ResNet ones in most cases, ResNet models achieve most of the best results which makes this last ones the more suitable for the problem addressed.

Regarding the kind of layer split, that is if the network uses LSTM or TCN at the top, the experiments are divided as follows: experiments 1 and 2 used LSTM layers and experiments 3 and 4 used TCN layers.

Looking at the results, it can be observed that TCN models use to have better performances in terms of accuracy and precision while LSTM models achieve better results in terms of recall. That happens for groomings and for mid rearings but for wall rearings, LSTM networks outperform TCN ones in all metrics.

These results can be explained by looking at the confusion matrices, in particular, by looking to the number of errors or misclassified frames. On the one hand, LSTM models have a large number of false positives while keeping a low number of false negatives. On the other hand, the number of errors of TCN models is lower and are quite balanced between false positives and negatives. That results in higher accuracy and precision for TCN models, and higher recall for LSTM models. Notice that, when addressing the wall rearing detection, this behavior is not fulfilled and the results favor the LSTM networks in all aspects.

Remember that LSTM networks capture temporal properties by remembering previous observed data and TCN networks capture the temporal properties through some stack of layers. Then, LSTM networks recurrently predict these patterns resulting in a large number of true predicted labels correctly and incorrectly classified. And TCN networks apply some kind of causality in the stack of layers to predict expected outcome which results in a balance between false positives and negatives.

The conclusion in this split is that although TCN networks achieve the best results in most cases, LSTM networks are also achieving great results not too far to the TCN ones and, in addition, LSTM networks minimise the number of false negatives. For these reasons, the LSTM networks seem more suitable for the problem being faced.

To conclude, from the discussion above, the most suitable model that is able to address all three problems should be the ResNet LSTM. Both kind of networks, LSTM and TCN, have shown similar results and the LSTM one has been chosen by their ability to minimise the number of false positives. Regarding the backbone, the difference between results is also quite small favoring ResNet most of the times.

Chapter 6

Conclusions

The automation of identifying and tagging the behaviors of mice in videos is necessary in research studies in several scientific fields. For this reason, given a set of mice videos with their respective annotated labels with the behavior at each frame, the problem consisted in automatically classify each frame with the respective behavior.

To address this multi-class problem computer vision and deep learning techniques have been applied to classify each frame with the behavior happening at that time. Then, the main goal of this work has been to build this deep learning model which makes this classification. This objectives has led to the definition of more specific objectives to achieve the main goal: review the existing literature and models, build different models to deal with the problem, compare the performance of these models and keep the best one.

In order to achieve all these goals, the article has been split into chapters to cover them. First of all, Chapter 2 has overviewed the related work and different models applied to this task such as Two-Stream ConvNets, 3D ConvNets or I3D ConvNets. Then, Chapter 3 presents the methodology and procedure to address this problem. Starting by dividing the multi-class problem into multiple binary ones, applying transformations to the data, defining the models used for the classification, the training strategy and metrics used to evaluate the models. After that, Chapter 4 has defined the experimental setup and also the models hyperparameter tuning. Finally, Chapter 5 has presented the results and the discussion of each of them to conclude on which model is the most suitable for the problem.

Focusing on the models, the presented ones have consisted in a feature extractor which generates embeddings from each video frame, the ResNet and Inception-ResNet CNN architectures, and a temporal deep learning model that based on the embeddings have made the frame classification. These temporal models have been LSTM networks or TCN networks. LSTM is a Recurrent Neural Network (RNN) capable of capturing temporal properties by remembering the previously observed data. On the other hand, TCN is a Convolutional Neural Network (CNN) that through some stack of layers captures the temporal properties of time series data.

From the discussion provided in the results discussion, it has been concluded that the model suiting the best is the combination of a ResNet50 with an LSTM network. This model have not reported the best results in each behavior but have resulted to be the best in average and it minimizes the missed patterns. As a conclusion, this model have performed a good generalization across behaviors reporting a 90% accuracy on the classification of Groomings and Mid Rearings and a 92% accuracy on the Wall Rearings classification.

To sum up, the automation of this task has been achieved by the ResNet50 combined with the LSTM network. The specific objectives have also been achieved through each of the chapters by reviewing the existing models, building TCN and

LSTM models, comparing their performances and selecting the said model as the best for the problem addressed. However, although the perfect classification is impossible, there has been some limitations and difficulties that resulted in a poor performance of the models. For instance, the available hardware has some limitations which led to split the model into a feature extractor and the LSTM or TCN network. The main idea was to build the model together in order to perform to stages of training: the first stage training only the top layers by setting the CNN layers as non-trainable and the second one training the whole model allowing the CNN layers to learn the specific representations from the problem. The other difficulty faced has been the camera point of view from the videos. Videos have been recorded from the top of the box which makes the behaviors more difficult to identify even by the human eye.

6.1 Future work

Finally, before concluding this report, there are some improvements to the work performed in this research that could let to better results or allow to tackle the problems and difficulties presented. The following list summarizes the possible future work for this research:

- There are several models that can be applied to this problem as shown in the Related Work chapter. In particular, the Transformers models have shown an incredible performance in the Natural Language Processing area. The adaptation of these kind of models to the problem given could be a revolutionary application that could led to astonishing results.
- Build the model together, the feature extractor CNN model with the LSTM or TCN network. Using this model perform the two stage training in such a way that the feature extractor CNN can specialize in the given problem. As said previously, this extension would require some hardware characteristics that have not been available in this work.
- Record the videos using multiple cameras capturing different points of view. The usage of multiple cameras would allow to train different models, each one using a different point of view, whose predictions could be combined by using an ensemble method, for instance Bootstrap Aggregation.
- As this work presented four different models, an ensemble method could be used to combine the predictions of each of the four models, again for instance a Bootstrap Aggregation.
- In order to improve the model performance, one could increase the hyperparameter space in order to find a better configuration. Furthermore, instead of using the simple Grid Search technique, using a bayesian hyperparameter optimization could be a great alternative.
- Leaving the model apart and focusing on the predictions, there are situations where a pattern or non-pattern is predicted leaving isolated labels inside. For example, considering the binary prediction: $[0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1]$, there is a 0 in the eighth position surrounded by 1s that in most cases is an error. The proposed solution consists in post-process the predictions by using mathematical morphology techniques, in particular the *Closing* operation which recovers the original objects closing holes and joining structures.

- Moving to the evaluation metrics, this work has not considered the computational time spent in the training, evaluation and prediction stages. Usually, there are limitations in this aspect so considering the computational time as another metric is an important decision that could affect the selection of the best model concluded in this work.
- Finally, considering the possible contribution of this work, it would be useful to test the models in public datasets. The evaluation of the models presented in this work on public datasets, also used in the current literature, would allow to compare the results obtained with the ones obtained by other researchers.

Appendix A

Video Dataset

A.1 Videos

Video Names	Set
HD_ChR2_480A	Train
HD_ChR2_586A	Train
HD_YFP_443A	Train
HD_YFP_463A	Train
WT_ChR2_400A	Train
WT_ChR2_425A	Train
WT_YFP_154A	Train
WT_YFP_435A	Train
WT_YFP_535A	Train
WT_YFP_602A	Train
WT_YFP_792	Train
HD_YFP_037A	Test
WT_ChR2_087A	Test
WT_ChR2_635A	Test
WT_ChR2_654A	Test
WT_YFP_741A	Test

TABLE A.1: The names of the videos provided and the set they belong after the split

Bibliography

Abadi, Martín et al. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. URL: https://www.tensorflow.org/.

Bohnslav, James P et al. (2020). "DeepEthogram: a machine learning pipeline for supervised behavior classification from raw pixels". In: *bioRxiv*.

Deng, Jia et al. (2009). "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee, pp. 248–255.

Efros, Alexei A. et al. (2003). "Recognizing Action at a Distance". In: *IEEE International Conference on Computer Vision*. Nice, France, pp. 726–733.

He, Kaiming et al. (2015). *Deep Residual Learning for Image Recognition*. DOI: 10.48550/ARXIV.1512.03385. URL: https://arxiv.org/abs/1512.03385.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long Short-Term Memory". In: *Neural Comput.* 9.8, 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL: https://doi.org/10.1162/neco.1997.9.8.1735.

Kopaczka, Marcin et al. (2019). "Assessment of Laboratory Mouse Activity in Video Recordings Using Deep Learning Methods". In: *2019 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pp. 3673–3676. DOI: 10.1109/EMBC.2019.8857807.

Kuehne, H. et al. (2011). "HMDB: A large video database for human motion recognition". In: *2011 International Conference on Computer Vision*, pp. 2556–2563. DOI: 10.1109/ICCV.2011.6126543.

Lea, Colin et al. (2016). *Temporal Convolutional Networks: A Unified Approach to Action Segmentation*. DOI: 10.48550/ARXIV.1608.08242. URL: https://arxiv.org/abs/1608.08242.

Nath, Tanmay et al. (2018). "Using DeepLabCut for 3D markerless pose estimation across species and behaviors". In: *bioRxiv*. DOI: 10.1101/476531. eprint: https://www.biorxiv.org/content/early/2018/11/24/476531.full.pdf. URL: https://www.biorxiv.org/content/early/2018/11/24/476531.

Neimark, Daniel et al. (2021). *Video Transformer Network*. DOI: 10.48550/ARXIV.2102.00719. URL: https://arxiv.org/abs/2102.00719.

Ngoc Giang, Nguyen et al. (Jan. 2019). "Applying Deep Learning Models to Mouse Behavior Recognition". In: *Journal of Biomedical Science and Engineering* 12, pp. 183–196. DOI: 10.4236/jbise.2019.122012.

Remy, Philippe (2020). *Temporal Convolutional Networks for Keras*. https://github.com/philipperemy/keras-tcn.

Scherer, Moritz et al. (June 2020). *TinyRadarNN: Combining Spatial and Temporal Convolutional Neural Networks for Embedded Gesture Recognition with Short Range Radars*.

Simonyan, Karen and Andrew Zisserman (2014). *Two-Stream Convolutional Networks for Action Recognition in Videos*. DOI: 10.48550/ARXIV.1406.2199. URL: https://arxiv.org/abs/1406.2199.

Soomro, Khurram, Amir Roshan Zamir, and Mubarak Shah (2012). *UCF101: A Dataset of 101 Human Actions Classes From Videos in The Wild*. DOI: 10.48550/ARXIV.1212.0402. URL: https://arxiv.org/abs/1212.0402.

Szegedy, Christian et al. (2016). *Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning*. DOI: 10.48550/ARXIV.1602.07261. URL: https://arxiv.org/abs/1602.07261.

van Dam, Elsbeth A., Lucas P.J.J. Noldus, and Marcel A.J. van Gerven (2020). "Deep learning improves automated rodent behavior recognition within a specific experimental setup". In: *Journal of Neuroscience Methods* 332, p. 108536. ISSN: 0165-0270. DOI: https://doi.org/10.1016/j.jneumeth.2019.108536. URL: https://www.sciencedirect.com/science/article/pii/S0165027019303930.

Vaswani, Ashish et al. (2017). *Attention Is All You Need*. arXiv: 1706.03762 [cs.CL].

Zhang, Zhenchuan, Yingchun Yang, and Zhaohui Wu (2019). "Social Behavior Recognition in Mouse Video Using Agent Embedding and LSTM Modelling". In: *Pattern Recognition and Computer Vision: Second Chinese Conference, PRCV 2019, Xi'an, China, November 8–11, 2019, Proceedings, Part II*. Xi'an, China: Springer-Verlag, 530–541. ISBN: 978-3-030-31722-5. DOI: 10.1007/978-3-030-31723-2_45. URL: https://doi.org/10.1007/978-3-030-31723-2_45.