Dissertation Type: Research

DEPARTMENT OF COMPUTER SCIENCE

Great Ape Behaviour Recognition using Deep Learning

Faizaan Sakib

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Master of Engineering in the Faculty of Engineering.

Friday 29th May, 2020

Declaration

This dissertation is submitted to the University of Bristol in accordance with the requirements of the degree of MEng in the Faculty of Engineering. It has not been submitted for any other degree or diploma of any examining body. Except where specifically acknowledged, it is all the work of the Author.

Faizaan Sakib, Friday 29th May, 2020

Ethics Statement

All data involved in this work is gathered prior to the start of the project. As such, no ethical review was required.

Faizaan Sakib, Friday 29th May, 2020

Contents

List of Figures

List of Tables

List of Algorithms

List of Listings

Executive Summary

Whilst species detection [56] and individual identification [46] of great apes have been attempted with success, research in automated behaviour recognition remains a human-centric task [50, 24, 19]. In response, this project successfully introduces a novel behaviour recognition model specifically for great apes, detecting a wide-ranging set of core behaviours. Among others, they include acts of sitting, walking, climbing and interactions with the camera. As per the domain-specific requirements of performing this task in the wild, the model is capable of simultaneously evaluating multiple great apes, each dynamically displaying behaviours, for a given video. More than 180,000 video frames from the PanAfrican [67] jungle footage archive used in [56] were further annotated with identification numbers and behaviour labels, amounting to a comprehensive great ape dataset for the purposes of this project, and beyond.

The model utilises a deep two-stream architecture, incorporating ResNet-18 CNNs to effectively capture properties based on both the appearance and motion of visual behaviours. After an extreme level of class imbalance was identified within the dataset to be a performance bottleneck, the model underwent a substantial transformation. The introduction of balanced sampling and the focal loss directly addressed the dominance of the majority classes. Stateful memory retaining LSTMs also proved to be vital, leveraging the inherent sequential temporality of video data. These techniques, along with an extensive set of experiments and ablation studies, crucially facilitated the discovery of a locally optimised final model for this task, vastly suppressing the issue at hand. The final cross-validated results obtained from 500 videos show that it is possible to achieve an accuracy of **73.52%** when classifying 9 behaviours.

The work in this thesis falls in the class of animal biometric systems [25], led by the ecological requirements to extract behaviours from 10,000+ hours of footage. Such a system could vastly reduce the manual efforts field studies. The outcome of this project establishes the potential of deep learning for great ape behaviour recognition, motivating its involvement in future research in this area.

Summary of Achievements

- I have spent ∼200 hours labelling every ape seen across 500 trap camera videos, equating to ∼180K frames. Some of this time was also spent improving the existing bounding box annotations.
- I wrote ∼800 lines of code in Python across a number of scripts to speed up the labelling of the dataset to an extent. They also include scripts for editing, interpolating and merging bounding boxes to improve the existing annotations.
- I have trained the YOLO [42] object detector on the provided dataset, resulting in a stand-alone lightweight great ape detector and classifier. It performs with an average precision of **81.27%**. The component is ready to be integrated into a real world biometrics system which can perform the detection, classification and behaviour recognition of great apes.
- I have researched and learned about numerous methods and concepts of deep learning, including but not limited to: two-stream action recognition, class imbalance, LSTMs, statistical model evaluation.
- I wrote ∼2000 lines of code in Python with the deep learning framework PyTorch. It includes the implementation of the behaviour recognition model, the dataloader which extracts samples from the dataset and an evaluator which produces the final output videos with drawn labels. I have ensured that the code is highly compatible and configurable for ease of use by external parties.
- I have logged and carried out extensive analysis and evaluation across different implementations of the behaviour recognition model. Through optimisations and experiments that include ablation studies and cross-validation, the final model performs at an accuracy of **73.52%** with an average class accuracy of **42.33%**.

Supporting Technologies

A number of third-party software technologies were used to facilitate the implementation of this project. They are listed below, and will be referred to throughout this thesis.

- I used PyTorch [70] to implement the deep learning model, including the pipeline involving data loading, training and evaluation.
- I used parts of the OpenCV [68] computer vision library. It was used to compute optical flow images of the dataset. It was also used for image and video processing of the dataset. This involved tasks such as video splitting, stitching and drawing.
- I used TensorBoard [64] to formally log the performance metrics of the models. The logs were then subsequently used to track a model's performance through visualisations.
- I used Kornia [7] for integrating their PyTorch implementation of focal loss [29] into my model.
- I used the `ElementTree` XML API for parsing and manipulating the annotations of the dataset.
- I used Amazon S3 [60] to automatically upload the final output of videos to cloud storage. The `boto3` SDK [61] is used to enable access and management with S3 from within Python.
- I used Git [62] to version control the code of the project. The project repository is hosted on GitHub [63].

All computation in relation to this project, including model training and evaluation, is performed on the University of Bristol's supercomputer BlueCrystal 4 [71]. This allowed me to access NVIDIA P100 GPUs with up to 128GB of RAM. This enabled GPU-based acceleration which heavily sped up the training of the neural networks involved in this project. This also provided me with 530GB of disk memory which was sufficient for storing the data and the saved weights of the models.

Notation and Acronyms

ANN	:	Artificial Neural Network
API	:	Application Program Interface
AWS	:	Amazon Web Services
BPTT	:	Backpropagation Through Time
CNN	:	Convolutional Neural Network
FLOPs	:	Floating-point Operations per Second
FPS	:	Frames Per Second
GPU	:	Graphics Processing Unit
LSTM	:	Long Short-Term Memory
RGB	:	Red Green Blue colour model
RNN	:	Recurrent Neural Network
SGD	:	Stochastic Gradient Descent
XML	:	Extensible Markup Language
YOLO	:	You Only Look Once (object detection system) [40]
ϕ	:	Activation function
α, γ	:	Focal loss parameters
$h^{(t)}, c^{(t)}$	:	Hidden state, cell state of an LSTM cell at time-step t
$H \times W \times C$	:	Input dimensions of a CNN (height, width, number of channels)
η	:	Learning rate
σ	:	Sigmoid activation function

Acknowledgements

I would like to express my deep gratitude for my supervisor Dr. Tilo Burghardt for his invaluable guidance throughout this project, and over the course of my degree. His continuous presence, especially when difficulties arose, was reassuring and greatly appreciated.

I would also like to thank Xinyu Yang. This project would not have been possible without his efforts of labelling the great ape dataset.

I wish to thank my family and friends who have supported me, especially during such trying times. Special thanks to Ainsley for his willingness to lend an ear for feedback. Finally, I want to thank Georgia for her constant love and support, putting up with me even when I sometimes prioritised this project over her.

Chapter 1

Contextual Background

1.1 Introduction

The task of action recognition involves the automated identification of activity seen in video. It can be seen as a developmental step from image classification, where instead of inferring a class of an object from a single image, it is instead performed over a sequence of consecutive video frames. Deep learning, which is a type of machine learning, has proved to be effective in performing these tasks. With the arrival of big data, and more specifically ImageNet [6] in 2009, the use of deep learning in image classification has seen a meteoric rise in interest, from both academia and industry. Famously, AlexNet [23] used a deep Convolutional Neural Network (CNN), which outperformed any other existing approach in classifying ImageNet at the time. CNNs also do not require hand-engineered filters in order to learn features. This combination of superior performance and automatic visual feature extraction meant deep learning techniques became a mainstay in image classification, and consequently, action recognition. Nevertheless, this is still a rapidly evolving area of research and its potential is yet to be fully realised.

There is an increased complexity that comes with working with videos, due to the higher computation required and the introduction of motion. This has meant that action recognition is to some extent behind the proven capabilities in image classification. Yet, there have been great advancements in the area, one of them being the "two-stream" CNN method introduced by Simonyan and Zisserman [47]. This method is able to capture both the "spatial" and the "temporal" (motion) context across frames unlike typical CNNs. Findings such as these are helping action recognition to reach the heights of image classification.

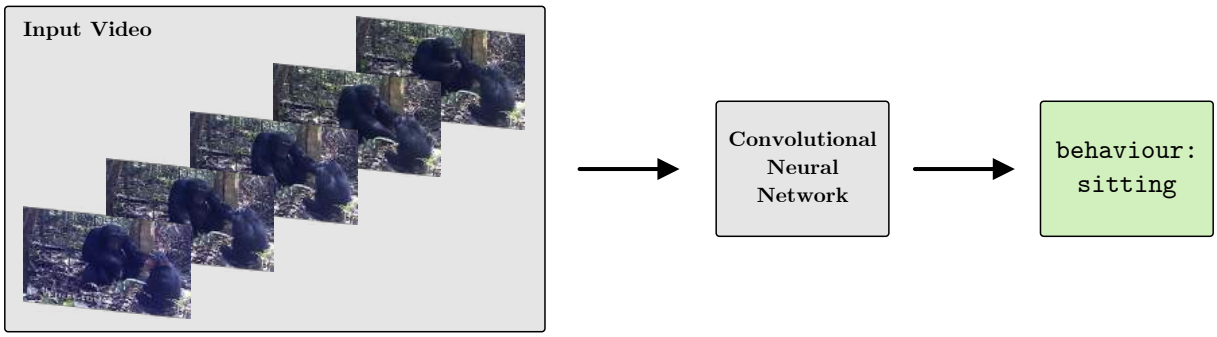


Figure 1.1: A high-level visualisation of **action recognition**. A CNN learns underlying features on the data it has trained on in order to make inferences of the subject's behaviour.

Although the search for an optimal action recognition model continues, with the primary focus being set on humans, it is exciting to see how well the task can be performed in different contexts. One such context is the use of CNNs to perform action recognition of great apes. Their physical humanoid structure means they have a distinct set of appearances across actions, similar to humans. This means that methods of action recognition currently in use could translate well to great apes. As such, the intent of the project is to use a deep CNN to perform behaviour (action) recognition exhibited by great apes found in jungle trap camera footage. Essentially, the CNN would be able to recognise what a great ape is doing (e.g.

walking, standing) in a video and be able to classify it as such. The application of such a network could automate the processing of camera footage involving great apes. This could vastly reduce the manual effort required from ecologists when conducting field experiments. In order to discover the optimal model for this task, this project will implement and experiment with different approaches, evaluating their overall effectiveness and performance.

The rest of this chapter takes a further look into the points raised above. This includes the motivations and objectives of the project, while also identifying the key challenges that are to be expected.

1.2 Motivations

Great apes are a family of primates made up of eight species. They are commonly seen as the extant species that holds the closest genetic relationship to humans, stemming from our shared ancestry. As such, continued studies in great apes are crucial as they could help provide us with an insight into their cognition, and in turn, a better understanding of our own evolutionary path. Meanwhile, with the majority of great apes now being classified as "critically endangered" [65], there is also an urgent need to prevent their extinction. Deep learning has the potential to play a key role in substantially enhancing efforts of research and conservation of great apes.

1.2.1 Animal Biometrics

A key area of improvement in this domain would be to introduce a non-invasive computational system that obtains great ape "biometrics" [25] from gathered data, removing the need for manual data analysis. The foundations of such a system have already been introduced, accomplishing automated detection [56] and identification [46] of great apes, using deep learning techniques. They are further discussed in Section 1.3. With the addition of automated behaviour recognition, this system could have a significant impact on the effectiveness of ecologists' work. It can allow them to have a better insight into great apes through fine-grained behaviour monitoring. Such information can help them to track populations, and fight against some of the main causes of great ape endangerment, like poaching or human encroachment. For example, ecologists could be alerted in the event of a great ape running as it could suggest that they are fleeing from some presence of danger.

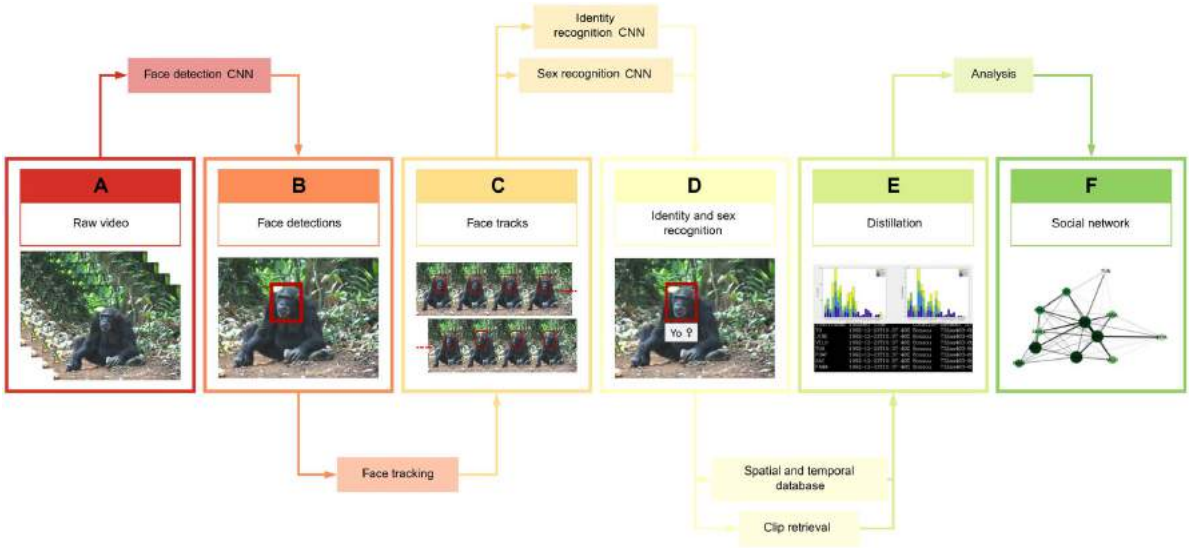


Figure 1.2: **Animal biometrics system**: An end-to-end pipeline outlining the processes that form the chimpanzee identification biometric system in [46]. Raw video input is leveraged to perform chimpanzee detection, identification, sex classification and ultimately a social network graph. Image taken from original paper.

Currently, there is a vast amount of sensory data that is being documented in the wild. To process all of this manually would take too long, affecting the duration of vital field studies that depend on the data. By automating the processing of the data, it would save ecologists a significant amount of time. This could crucially allow them to increase their study sizes for more ambitious experiments. It can also help to increase publication rate and contribution to academia. Furthermore, this approach can produce

standardised results by removing human bias. This can lead to better comparisons across results and accurate experiments.

Questions of ethical issues have been raised with regards to field experiments involving great apes. For example, the use of invasive techniques, such as "habituation", where researchers make the apes acclimatise to them through food provisioning, can be seen as an unwanted interference of a great ape's growth and development [10]. The mere presence of humans may cause stress among the apes. This is undesirable in itself and in terms of how it may affect the behaviour of apes that are being observed. A non-invasive system can address these issues, taking into consideration the great apes' health and as a result, producing more reliable results.

1.2.2 PanAfrican Dataset

The PanAfrican [67] jungle trap camera dataset (from this point on referred to as "dataset") is made up of videos situated in central Africa. Among them, there are two species of great apes that feature: gorillas and chimpanzees. The videos span across multiple locations, with footage captured during both day and night. A subset of 500 videos of this dataset were manually annotated for training a CNN which detects great apes in challenging jungle footage [56]. The annotations consist of bounding box coordinates that encode the location of great apes that are present across the videos.

Beyond documenting the presence of great apes, the current value of the dataset can be extended by adding annotations for other features found across the videos. An enhanced set of annotations can make it possible to train CNNs to learn new features found within the same data. Thus, the addition of behaviour annotations will be required so that there will ground truth examples that a CNN can learn from. This will make it possible for this project move towards achieving its goal of producing a deep great ape behaviour recognition system that utilises deep learning techniques. Appendix A contains examples of the raw video footage and annotations found in the dataset.

Figure 1.3: **Annotations of PanAfrican great ape dataset**: Examples illustrating manually annotated bounding box coordinates for the PanAfrican dataset, used in [56]. The dataset contains 500 videos with annotations stating every ape's species and their position on each frame.

1.3 Related Work

As stated in Section 1.1, there is no current work involving the behaviour recognition of great apes, as this task has largely been applied to human actions so far. There are however instances of deep learning being used to address problems related to this area of ecological research.

The potential of integrating deep learning computer vision methods into ecological processes was indicated by the work of Brust et al. [4]. Existing techniques such as the YOLO [40] object detector and AlexNet [23] were used, creating a system that could detect and identify individual great apes. The model produced a facial detection rate of 90.8% Average Precision (AP), with a Top 5 accuracy of 80.3% for identification. These findings showed that current ecological processes can be improved by automating processes involving manual labour.

Xinyu et al. [56] produced a robust video object detection model specifically for great apes, which aimed to address multiple challenges present in jungle camera trap footage, such as partial occlusions and difficult lighting conditions. The PanAfrican [67] video dataset with location annotations of great apes was created for the purposes of training the detection model, before being passed on for use in this

project. Capturing spatial and temporal contexts across videos, the model obtained a high 90.81% mean AP, proving it to be capable of assisting the processing of trap camera footage.

Schofield at al.[46] utilised a deep CNN, which was trained on 10 million images, to create a fully automated pipeline for the facial detection, recognition and tracking of wild chimpanzees in video footage. This lays the foundations of a biometric system for chimpanzees. An overall accuracy of 92.5% for individual identification and 96.2% for sex recognition was achieved. This work further demonstrates the vast potential of such a system. For example, by leveraging the identification of co-occurring chimpanzees in the data, it is possible to form social network graphs which outline relationships and communities amongst the great apes within the dataset. This is an instance of automation that can greatly reduce the time and effort needed to evaluate ecological data, while making discoveries previously not possible.

1.4 Aims and Objectives

The aim of this project is to further augment the biometric profile of great apes that can currently be obtained from video footage. In particular, this work addresses the task of behaviour recognition of great apes using a deep two-stream ResNet-based Convolutional Neural Network (CNN) which can capture properties based on both the appearance and the motion of visual behaviours, given multiple consecutive video frames. The PanAfrican [67] dataset used in [56] will be expanded with the addition of behaviour labels to facilitate the training of the CNN. The objectives of the project are as follows:

- Label 500 videos of the PanAfrican [67] jungle trap camera footage dataset, as per the project's requirements.
- Explore the effectiveness of deep learning techniques used to perform automatic behaviour recognition of wild great apes found in camera trap footage.
- Implement a deep learning model, which is configurable and performs appropriate logs to allow for efficient model testing and monitoring.
- Conduct extensive experiments and ablation studies to evaluate model performance across implementations, enabling the discovery of the optimal model for performing great ape behaviour recognition.

1.4.1 Challenges

Labelling the Dataset

The start of this project will involve labelling the dataset of jungle trap camera footage, which would subsequently be used to train the behaviour recognition model. The primary challenge is the scale of the task. There are 500 videos in the dataset, amounting to around 180,000 frames. There are annotations for every frame. Each frame is capable of containing from 0 to many apes present, where every ape needs to be individually labelled. Thus, there will be a substantial amount of time that will be needed to complete the labelling.

Prior to the labelling, the behaviours that the model will attempt to classify need to be defined. It has to be ensured that the majority of behaviours shown by great apes within this dataset, if not all, can be appropriately categorised in any one of the behaviour classes. Considering that the videos will be labelled done frame-by-frame, there may be some frames where the behaviours exhibited by the apes are ambiguous. These issues will need to be identified and handled, since inconsistencies in the labelling may propagate into the behaviour recognition model causing undesired impact.

Sampling

Once the labelling is complete, the dataset will need to be appropriately split into samples that the behaviour recognition model can then train on. However, the videos of this dataset do not trivially translate to data samples. Each video consists of many frames, and each frame may have multiple apes. There needs to be a clear definition of what constitutes a sample, which may depend on a number of variables. Following this, an algorithm needs to be written that can extract all possible samples for each ape that appears across all the videos, while abiding by the requisites. This must ensure that potential samples are not missed and samples are not incorrectly used for training. The sampling strategy that is eventually used will need to be effective and consistent for training the model.

Class Imbalance

A common pitfall for machine learning networks is the imbalanced distribution of training samples across each of the classes, a problem known as class imbalance. This can lead to disproportionate performance for certain classes which it has not had as much exposure to. In this dataset, some behaviours are more common than others. For example, there are more instances of a great ape walking in the dataset, than they are climbing a tree. However, this may be due to the fact that great apes in reality do spend more time doing some things more than others. So, it can be said that the imbalance of the dataset is simply representative of the natural frequencies at which each of the behaviours occur. One way to solve this would be to gather more footage of great apes that exhibit the less represented behaviours. But since the collection of more data is not in the scope of this project, one of the main challenges will be to ensure that the model can perform well in classifying all of the behaviours.

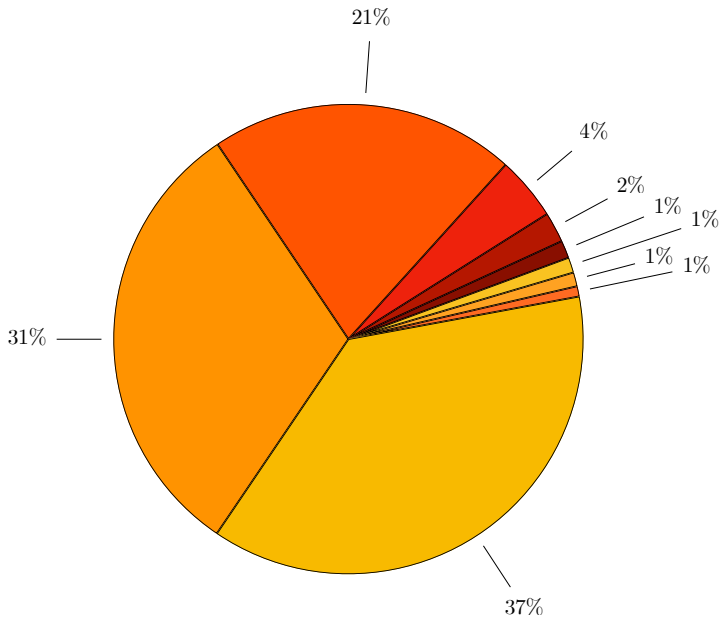


Figure 1.4: **Class imbalance**: A pie chart displaying the unequal distribution of behaviour annotations across the classes defined. This leads to an imbalance in the amount of data that a deep learning model can train on for each class, known to cause uneven performance. The classes that make up of the dataset are discussed and defined in Section 3.1.1.

Chapter 2

Technical Background

This section acts as a comprehensive view of the technical content that is considered as a prerequisite for the execution and understanding of the project.

2.1 Deep Learning

Machine learning is the use of algorithms and statistical analysis to find patterns in large amounts of data. Deep learning is a subset of machine learning, specifically utilising Artificial Neural Networks (ANNs). The design of ANNs are inspired by the biological neural network of the human brain [31], which has resulted in a type of machine learning that is more capable than other techniques in this area.

2.1.1 Supervised Learning

Supervised learning is a category of machine learning where an approximation function $f(x)$ is learned from training on labelled data samples that are used as "examples". The function is consequently used to map unseen input to the correct output. The great ape dataset will need to be appropriately labelled to facilitate the task of supervised learning in order to infer behaviour.

With the use of supervised learning, the behaviour recognition model aims to perform classification, which is where input is categorised as a "class". In this case, classes correspond to behaviours of great apes. The existence of multiple behaviours implies that this is a multi-class classification task.

2.2 Artificial Neural Network (ANN)

Introduced in 1943 [31], ANNs are classifiers in the form of multi-layered feed-forward networks. They take input in their initial layer, followed by hidden layers made up of "neurons" or "hidden units".

Each neuron can take in several inputs x, to produce an output that can be fed to neurons in the following layer. Connections to onward units are assigned a weight w, which act as coefficients that can either amplify or dampen the input. These weights are consequently updated over the course of training, towards achieving a better mapping of the approximation function. The output is computed as follows:

$$y = \phi \left(\sum_{i=1}^{N} x_i w_i + b \right) \tag{2.1}$$

involving the summation of the product of x_i and w_i from 1 to N. A bias b is added as a constant, typically used as prior knowledge to help the network fit the given data. Finally, an activation function ϕ is used to introduce bounded non-linearity to the otherwise linear behaviour of a neuron. This transformation of the neuron's output makes ANNs better suited to learning the underlying complexities of input data. ReLU [33] is a commonly used activation function, which is defined as $\max(0, x)$ for input x.

The classification of the input culminates in the final layer of the ANN, the output layer. This is often preceded with the use of the Softmax activation function [23], normalising the incoming values to a

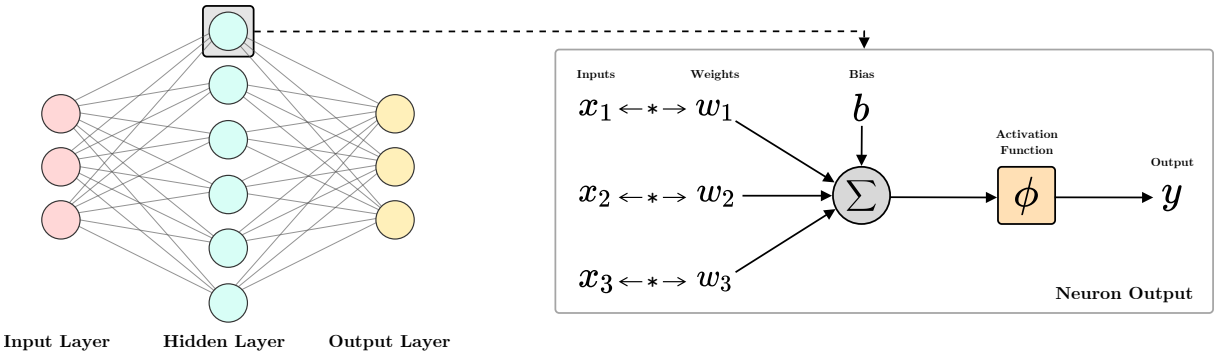


Figure 2.1: **Artificial Neural Network** displaying its neurons and the weight connections between the layers (left). The calculation of output y of a neuron in the hidden layer (right). The sequence of operations can be seen, involving the summation of inputs x and weights w from the input layer, followed by an activation function.

probability distribution over the number of classes C, as a vector of size C. The highest value in this vector corresponds to the class that is predicted for the input sample.

2.3 Loss

The difference between a network's predicted output and the actual output is known as the "loss". This is a measure of error used for evaluating the performance of a network, where the greater its value, the more suboptimal it is. The way the loss is calculated depends on the loss function that is used. There are a number of such functions, that may give different values of loss for the same predictions. Therefore, the performance of the network heavily relies on the loss function that is used.

2.3.1 Optimisation

Naturally, the goal is to minimise this loss to improve the performance of the network. This is done with the use of an optimisation algorithm, such as Stochastic Gradient Descent (SGD) [20], which treats the loss function J, as the objective function to minimise.

Initially, the network computes the gradient of the loss function with respect to the its weights. The gradient is efficiently computed using the chain rule, starting from the end of the network, iterating backwards one layer at a time until the start. This is known as backpropagation [44]. The network then updates the weights in the opposite direction of the calculated gradient. This can be seen in Equation 2.2, where the current weights $\mathbf{W}^k$ is subtracted by the gradient to obtain the new weights $\mathbf{W}^{k+1}$. This process is repeated until the weights converge to a local minima of the loss function, representing the "learning" of a neural network.

$$\mathbf{W}^{(k+1)} = \mathbf{W}^{(k)} - \frac{\partial}{\partial \mathbf{W}^{(k)}} J(\mathbf{W}) \tag{2.2}$$

2.3.2 Learning Rate

The learning rate η is a scalar hyperparameter (a parameter which is set before a model begins to train) which is multiplied with the gradient term in Equation 2.2. This effectively determines the magnitude of every update to the weights. The optimal learning rate varies across problems. A learning rate too small would heavily lengthen the time taken to find the minima. Conversely, if it is too high, it may cause instability in the loss function, unable to converge as it overshoots the minima.

2.3.3 Loss Functions

As stated above, the choice of loss function impacts the learning of a neural network, and subsequently its performance. The following loss functions are considered relevant to this project.

Cross-Entropy The cross-entropy loss function [26] is used for tasks where a sample can belong to only one class. In particular, the categorical cross-entropy loss function is used for multi-class classification,

relevant to the task of this project. The overall loss is calculated as a sum of separate losses for each class label per observation:

$$\mathrm{CE} = -\sum_{c=1}^{M} y_{o,c} \log\left(p_{o,c}\right) \tag{2.3}$$

where M is the number of classes, y is a binary value indicating if class c is the correct classification for observation o and p is the predicted probability for class c for observation o. Predicted probabilities that diverge from the ground truth will cause an exponential rise in loss.

Weighted Cross-Entropy The cross-entropy loss function can be extended, by using weighting factors α_c for each class c, to re-scale the loss it computes for each class. This is a technique used for addressing class imbalance (see Section 1.4.1, 2.10). The weights are determined by the frequency of the class within the dataset, calculated as such:

$$\alpha_c = \frac{\max\left(\text{samples}\right)}{\text{samples}_c} \tag{2.4}$$

where $\max\left(\text{samples}\right)$ is the highest number of samples that a class has. samples_c is the number of samples that class c has. Because of this, the loss will be scaled to 1 for the class with $\max\left(\text{samples}\right)$. Meanwhile, every other class would have a factor of greater than 1, where the lower the number of samples, the greater the value of its loss. This allows minority classes to instead dominate the loss, while majority classes contribute less.

Focal Loss A modified version of the cross-entropy loss function, which also attempts to address the class imbalance problem. Focal loss ensures that the value of the loss is dictated more by the classes that are being incorrectly predicted, than correctly predicted.

$$\mathrm{FL} = -\alpha_t \left(1 - p_\mathrm{t}\right)^\gamma \log\left(p_\mathrm{t}\right) \tag{2.5}$$

Equation 2.5 includes a scaling factor $\alpha_t \left(1 - p_\mathrm{t}\right)^\gamma$. Meanwhile, the focusing parameter γ determines the level of down-weighting of class' contribution to the loss according to how well they have been learned. A high γ would down-weight the importance of well-classified classes much more aggressively. In contrast a low γ would limit the suppression of well-classified classes to a smaller extent. As such, the focal loss is equivalent to the cross-entropy loss function when $\gamma = 0$. This can be seen in Figure 2.2. For multi-class classification, α acts as a coefficient that can be used to enlarge, or shrink the loss produced.

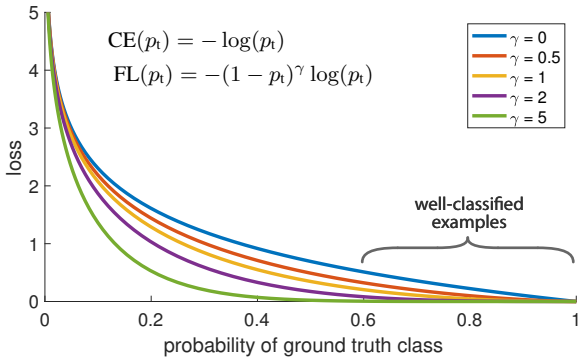


Figure 2.2: **Focal loss**: The effects of γ on the calculation of the loss. The higher γ is, the more aggressive it is in suppressing the contribution of well-classified samples to the loss. Image taken from the original paper. [29]

By calculating the loss this way, it allows the network to focus on training on the tougher classes, which are less represented in the dataset, instead of classes that are already well-learned.

The great ape dataset, although large in size, suffers from class imbalance, as stated in Section 1.4.1. If necessary, the use of alternate loss functions such as weighted cross entropy and focal loss can help to improve the network's performance on behaviours that are not well represented and classified.

2.4 Convolutional Neural Network (CNN)

A CNN [28] is a special type of neural network. Central to its implementation is the convolutional layer. Given an input of data, these layers are responsible for performing convolution using filters. The output of this operation is a "feature map", which can encode patterns and features found in the data.

Unlike other machine learning techniques, CNNs are capable and well-suited for processing 2D and 3D data. By representing an image as a matrix of pixel values, or a video as images with an additional dimension across time, it is possible to train CNNs on visual data. It uses filters that can span over a subset, or a neighbourhood, of the input, instead of just a single element. This successfully reduces the data into a form that is easier to process, while capturing contextual features to perform good predictions.

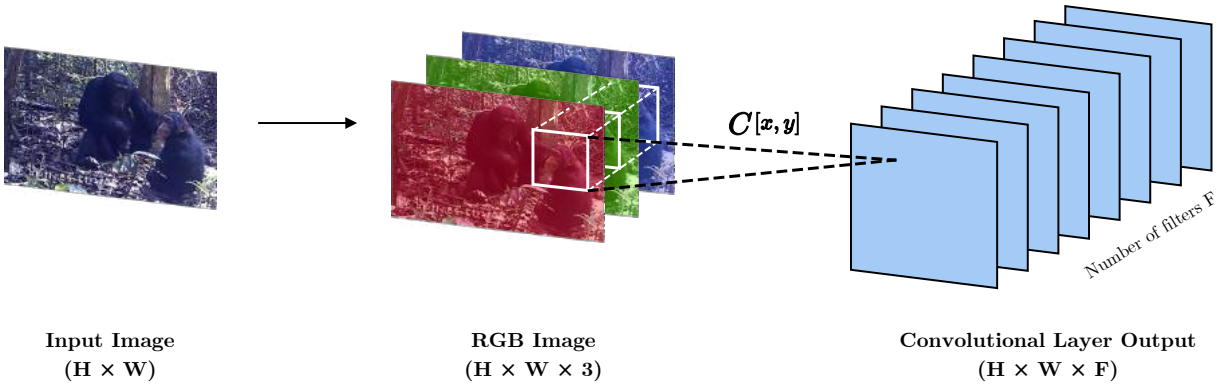


Figure 2.3: **Convolutional Neural Network**: Visualising a convolutional layer with convolution operation C being applied to an RGB image as input of size $H \times W$. C is performed with F different filters, resulting in an output of size $H \times W \times F$.

Figure 2.3 shows the convolution operation C being performed on the RGB version of the input image to obtain its feature map. The diagram shows the use of just one convolutional layer, where in practice, there tends to be a number of layers within a network. This enables the network to capture high-level features such as shapes and colours in the initial layers, before capturing specialised low-level features at the tail end of the network. The convolution, C can be shown as:

$$C[x, y] = \sum_{u} \sum_{v} A[x + u, y + v] \cdot B[u, v] \tag{2.6}$$

where the output at coordinates (x, y) is the sum of the dot product between a filter of $(n \times n)$ values B, and the $(n \times n)$ subset of the input A, that is underneath the filter.

In order to perform classification, the output of the CNN is flattened into a 1D vector. This is then fed into a fully connected neural network, which is an ANN where every neuron is connected to all other neurons outside of its layer. This acts as a classifier for the CNN. Its final layer would consist of a number of nodes equal to the number of ape behaviours that are aimed to be classified. The value found in each of these nodes would represent the probability of the input containing the corresponding behaviours.

Typically, hand-crafted filters (e.g. Sobel filter [49] used for edge detection) would be required to extract certain features from data. However, CNNs eliminate the need for manually identifying such filters. They are able to automatically identify filters, which are subsequently trained based on the data it sees. Furthermore, a CNN can learn multiple filters in parallel for a given input.

These key characteristics have led to CNNs being very effective in computer vision tasks, further motivating its use for this project. With the great ape dataset only consisting of video footage, the use of a CNN is appropriate due to its compatibility with visual data. Its ability to extract features quickly and in high numbers are also desirable for behaviour recognition. Such a task would require complex low-level filters specific to great apes, which if hand-crafted, would take significantly longer to produce.

2.4.1 ResNet

ResNet [12] is a "very deep" CNN architecture, popular for image classification. The network is made up of stacks of multiple layers, each representing a "residual block". Within these blocks, ResNet introduced "skip connections", where the input x of a block, is added element-wise to the output of the last layer of the block, before ReLU is applied. This ensures that the information of the block's input is not altered by any of the intermediate convolutional layers. Thus, the network instead learns the residual mapping of $f(x) + x$, where x becomes the identity. The network is now capable of learning the identity function.

If the layers within the residual block improve the network's overall performance, the change in its weights are retained. However, if the identity x remains the optimal mapping, then the weights of the layers within the block can be set to zero, ignoring the residual function $f(x)$. In summary, skip connections effectively allow layers to be appropriately skipped, preserving the network's weights from any performance degrading transformations.

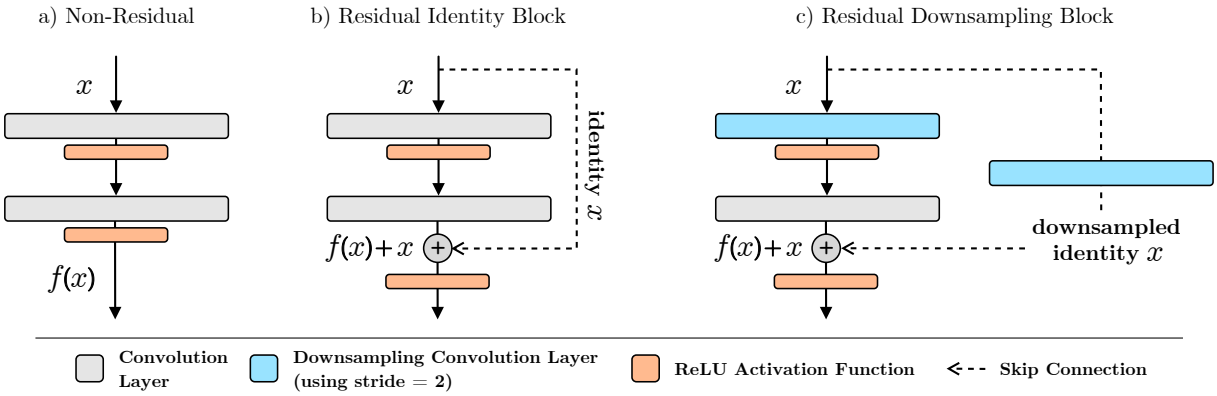


Figure 2.4: Diagram showing **a)** a normal, non-residual configuration of consecutive CNN layers. In comparison, the **ResNet residual blocks** are displayed. **b)** The **identity** block shows the use of the skip connection, portrayed as dashed arrows, to perform $f(x) + x$ before ReLU at the end of the block. **c)** The **downsampling** block illustrates the use of a convolutional layer in parallel to the block to reduce the size of x as required.

ResNets are made up of two types of residual blocks. Its primary form, the "identity" block, operates as described as above. However, the initial layer of some blocks within the network changes the size of the input. In ResNet-18, this consists of a doubling in the number of filters, along with the halving of its spatial dimensions with the use of a stride of 2. The reduction in input to form a compressed feature map is a natural progression seen in CNNs. To ensure that the $f(x) + x$ operation can still be performed, the identity x will also need to be reduced in size to match the dimensions of $f(x)$. This is done with a linear projection in the form of a 1×1 convolutional layer, downsampling x as part of the skip connection. The layer, similar to that of the initial layer of $f(x)$, has a stride of 2 and the same number of filters. This block is known as the "downsampling" block, and can be seen in Figure 2.4c.

ResNet has a number of variants which differ by depth, ranging from 18 to 152 layers. Before ResNet, an increase in the number of layers used in a CNN would not necessarily improve performance. A common pitfall for very deep CNNs would be the vanishing gradient problem [15]. This is when the partial derivatives of the error function with respect to the current weights become so small that it prevents the network to continue training. ResNet's residual approach resolves this, making it possible for CNNs to be made up of a much higher number of layers than before without loss of performance.

The task of behaviour recognition is considered a complex domain, involving four-dimensional input taking time into account. The combination of ResNet's high level of depth and its residual characteristics make it a desirable choice for the model's architecture.

2.5 Transfer Learning

Transfer learning [38] is a method where the weights of a network which is trained to perform a particular task, can be used to aid the learning of a network to perform another task in a related domain. The weights of a network that are "pre-trained" on a large general dataset, such as ImageNet [6] can be used as a starting point for other networks. Such pre-trained models may already have certain filters to extract

features such as edges and shapes learnt in its initial layers. From this point, there are two ways to train the network towards the desired domain.

Feature Extraction This involves adding a fully connected classifier on top of a pre-trained network. Only the classifier is trained from scratch, while the layers of the pre-trained network are "frozen". This ensures that backpropagation only occurs up until the start of the classifier, thus retaining the weights of the pre-trained model.

Fine-Tuning A pre-trained network is used, including a classifier on top, with the entire network being trained as normal. This allows to "fine-tune" the model to make it better suited to the new domain.

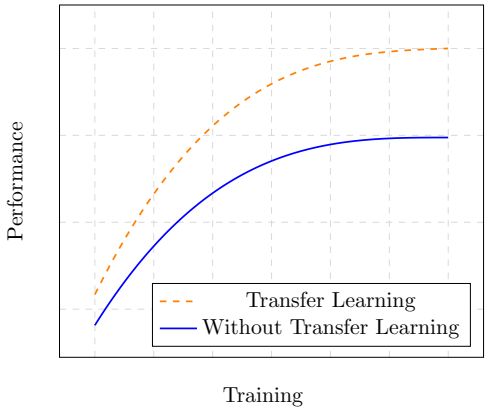


Figure 2.5: **Transfer learning**: A comparison between the expected performance of a transfer learning model and a model trained from scratch. Note the higher level of performance of the transfer learning model at the start of training. The steeper slope indicates that its rate of improvement during training is also higher. These aspects lead to an overall higher asymptote in regards to final performance for the model trained with transfer learning.

Although the great ape dataset is more specialised relative to the images found in ImageNet [6], transfer learning would still be beneficial compared to training from scratch. As later discussed in Section 2.15, there are publicly available pre-trained models for popular architectures such as ResNet which can be utilised for this project. However, the use of a pre-trained model will impose some constraint to the architecture that can be used. This is due to the fact that the pre-trained weights have a one-to-one mapping to the architecture of the network they were originally trained for.

2.6 Overfitting

Overfitting is considered to be a long-standing challenge in machine learning [1]. The issue arises when a model is more complex (e.g. high number of layers/units/parameters) than necessary [11]. When non-linear underlying patterns are to be found within the data, the model attempts to "overfit" the data with a complex approximation function, catering to every training sample it is exposed to. Naturally, this leads to good performance on the training set, but the model is unable to generalise when evaluating unseen data. As such, Hawkins states that "a model overfits if it is more complex than another model that fits equally well" [11].

The existence of overfitting can be identified from examining a model's metrics, such as loss and accuracy. When a model's validation metrics deteriorate, while its training metrics improve, it shows that the model is optimising towards fitting the training set specifically, and thus overfitting. Reducing the complexity of the model, or increasing the size of the training set can help address this issue. However, these approaches may not always be feasible. This project aims to use an existing architecture with pre-trained weights (see Section 2.5), constraining the ability to modify it extensively. Also, the great ape dataset is not being considered for expansion. Instead, several alternative techniques can be utilised.

Dropout A simple way to prevent neural networks from overfitting [51]. The idea is to randomly drop units and their connections within the network during training, with some probability p. This prevents the network from relying too much on just several units assigned with very large weights. By forcing the network to spread its weights to other units, it is able to be more efficient with less complexity.

L_2 Regularisation This involves the use of a penalty term, which encourages the sum of the squares of the network's weight parameters to remain small, and hence its complexity [36]. This term is incorporated into the loss function, so that both the model's loss and its weights are simultaneously minimised:

$$\text{Loss} = \mathcal{L}(y, \hat{y}) + \lambda \sum_i \sum_j w_{ij}^2 \tag{2.7}$$

$\mathcal{L}$ is the loss value obtained between the true value y, and the predicted $\hat{y}$ from the loss function in use. The second term corresponds to the L_2 regularisation. λ is a hyperparameter which acts as a coefficient to tune the scale of regularisation. Squaring the weights ensures that the larger the weight, the more strongly it is penalised.

Data Augmentation The use of data augmentation can help bolster the dataset without the need of collecting more data. It is common practice in deep learning, often leading to better model performance due to the increase in training samples [23, 48, 12]. This is done by randomly modifying data samples resulting in artificial unseen data. Examples of modifications include brightness/contrast adjustments, rotation, image flipping among others. Using data augmentation to increase the number of samples for classes that are less represented may be a valid countermeasure to the class imbalance in the great ape dataset.

2.7 Object Detection

Object detection is a computer vision task which deals with identifying the locality of objects in images or videos. A system that classifies great ape behaviour can only function based on two assumptions; there is a great ape present within the given input, and that the system knows where they are. The foundations of such a system therefore must have the ability to detect apes within the input space, upon which behaviour recognition can be performed.

2.7.1 YOLO

YOLO (You Only Look Once) [40] is a CNN capable of performing object detection in real-time. The network requires only one forward pass to evaluate an image, akin to an "only look once" approach as suggested by the name. This results in YOLO being much faster than its counterparts, while also retaining a high accuracy. With subsequent iterations [41, 42, 3], YOLO proves to be a state-of-the-art. It is publicly available [59] and can be trained on custom datasets to detect any desired object.

As such, it would be a suitable backbone for locating great apes within videos. Beyond detection, YOLO would also be able to classify the detected ape (e.g. gorilla or chimpanzee). Since YOLO is primarily an object detector for images, some work would have to be done in order to apply it to videos. The behaviour recognition model can then evaluate the apes that are detected by YOLO. Its fast performance can help to limit the overhead in pre-processing the given input.

2.8 Action Recognition

Action recognition is the task of identifying an activity (e.g. walking, swimming) present within a video. Generally, efforts in this area are trained and tested on videos in which only a singular action is present, while also being in the primary focus of the frame of the video. This is indicated by the datasets commonly used for action recognition [50, 19, 24]. In comparison, jungle trap camera footage, apart from its static viewpoint, does not inherit these controlled properties. The great ape dataset consists of videos where over its duration there may be one of, or a combination, of the following:

- more than one ape present
- multiple apes each exhibiting different behaviours
- a particular ape transitioning between, and exhibiting, multiple behaviours

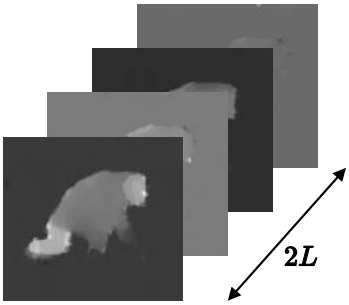
This introduces complexity to the problem, as a video simultaneously containing a sitting ape and a walking ape, cannot be sufficiently classified using a single label. It would be possible to train a network

with the videos as is, and perform multi-label labelling, where the network predicts the existence of all behaviours for a given video. However, this project aims for a higher level of precision, where the detection of behaviour can be assigned to the specific ape present within a particular frame of the video.

2.8.1 Two-Stream Method

The integral challenge for an action recognition model is to capture the spatio-temporal context across a video. A nuanced observation of an object, taking into account both its appearance and motion is desirable. The Two-Stream method introduced by Feichtenhofer et al. [47] addresses this. It essentially utilises two separate CNNs, seen as "streams". The "spatial" stream captures dependencies based on the appearance of an action. For example, this stream would focus on what a chimpanzee looks like when it is sitting (e.g. legs crossed). Meanwhile, the "temporal" stream captures motion information. This could consist of the swinging movement of a chimpanzee's arms and legs when they are walking. The different behaviours of great apes vary both in terms of appearance and the movement needed to carry them out. Therefore, using a two-stream method, that independently trains on two different types of features, before bringing them together, allows the network to have a more comprehensive view of the behaviour taking place.

(a) **Spatial**: An RGB data sample.



(b) **Temporal**: An optical flow stack sample of size $2L$.

Figure 2.6: **Two-stream input**: Examples of input data samples for the spatial and temporal streams.

Since the two streams are responsible for different features, their input also differs. The spatial input is a single frame of size $H \times W \times C$, where $C = 3$ for each of the RGB channels. The temporal stream is exposed to a stack of optical flow images. Optical flow encodes the motion of objects found between consecutive frames in a video [17]. Suggested in [47], the TV-L1 optical flow [39] algorithm is used. For every consecutive pair of frames in a video, two greyscale images are obtained, encoding flow in both the horizontal and vertical directions. The flow for each direction is alternatively stacked across L consecutive video frames [47]. This results in an input of size $H \times W \times 2L$, where L is the number of consecutive frames considered, while $2L$ takes into account both directions of the flow for each frame.

Fusion Methods

The act of "fusion" in deep learning refers to the combining of multiple outputs produced by separate networks. This is used for further training or to produce a final classification. Feichtenhofer et al. [47, 9] explored two effective methods methods of fusing the spatial and temporal streams.

Late Fusion The initial fusion method used averages the final prediction layer outputs of both streams [47]. The resulting output are the prediction values for the whole model.

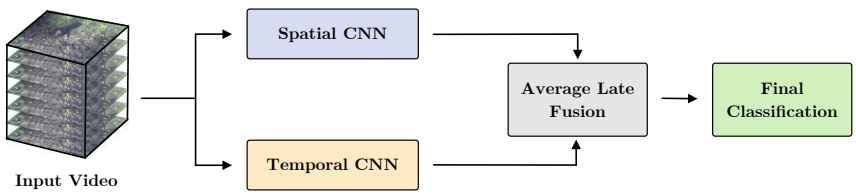


Figure 2.7: A diagram of a two-stream network using **late fusion**. The output values of the spatial and temporal streams are averaged to obtain the final classification values.

Convolutional Fusion In [9], Feichtenhofer et al. propose an alternative fusion method, which occurs at a convolutional layer earlier within the network. To do this, the feature maps output from the two streams (x^a, x^b), are stacked at the same coordinates (i, j) across feature channels d:

$$y^{\text{cat}}_{i,j,2d} = x^a_{i,j,d} \qquad y^{\text{cat}}_{i,j,2d+1} = x^b_{i,j,d} \tag{2.8}$$

where $\mathbf{y} \in \mathbb{R}^{H \times W \times 2D}$ and $\mathbf{x^a}, \mathbf{x^b} \in \mathbb{R}^{H \times W \times D}$. This stacked feature map is then passed through a 3D convolutional layer, where the filters are able to learn the correlations within the spatial and temporal dependencies of the data. The dimensionality of the output is then reduced to a 1D vector which can then be passed through a fulyl connected network to perform classification. In comparison to late fusion, this approach is able to integrate the two streams in advance for better performance.

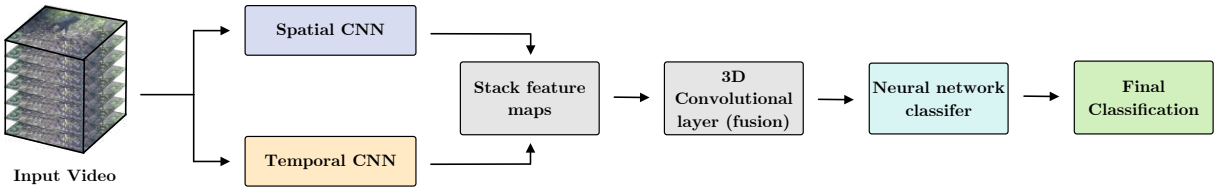


Figure 2.8: A diagram of a two-stream network using **convolutional fusion**. Feature maps obtained from both streams are stacked as shown in Equation 2.8. This is passed through a 3D convolution layer, followed by a fully connected classifier, to obtain the final classification. This approach fuses spatial and temporal dependencies earlier and does so in a less naive way than late fusion.

2.9 Long Short-Term Memory (LSTM)

Long Short Term Memory (LSTM) [16] networks are an extension of the Recurrent Neural Network (RNN) [32], which itself is a class of neural network designed to capture sequential dependencies in data using temporal characteristics when learning. LSTMs are capable of capturing long-term dependencies, an area in which RNNs tend to fail.

A sequence in the context of behaviour recognition would be represented by a number of consecutive video frames. LSTMs are "unrolled" to process the sequence in T time-steps, where the t^{th} frame in the sequence is considered to be the input $x^{(t)}$ for time-step t. For every t, the network receives from the previous time-step input $x^{(t-1)}$ and a hidden state $h^{(t-1)}$ to produce the output for the following time-step. The sustained use and evolution of the hidden state makes it possible for the LSTM to build up a sequential memory where each output depends on the event before it, and every event before that.

LSTMs additionally store information in a cell state $c^{(t)}$, adjacent to the recurrent network itself. The make up of its information is modified via analog gates that open and close. They are the external input gate $i^{(t)}$, forget gate $f^{(t)}$ and the output gate $o^{(t)}$. The gates allow the LSTM to learn and retain its memory, while also being able to forget as required. $o^{(t)}$ for time-step t is computed with the non-linear Sigmoid [35] activation function σ as follows:

$$\boldsymbol{o}^{(t)} = \sigma\left(\boldsymbol{U}^o \boldsymbol{x}^{(t)} + W^o \boldsymbol{h}^{(t-1)} + \boldsymbol{b}^o\right) \tag{2.9}$$

where matrices $\boldsymbol{U^o}$ and $\boldsymbol{W^o}$ contain the weights for the input and recurrent hidden state, respectively. The bias parameter b is added. This is then used to compute the aforementioned hidden state $h^{(t)}$:

$$\boldsymbol{h}^{(t)} = \tanh\left(c^{(t)}\right) \circ o^{(t)} \tag{2.10}$$

The *tanh* [30] activation function is applied to $c^{(t)}$ before being multiplied with $o^{(t)}$. $h^{(t)}$ is subsequently passed on to compute $h^{(t+1)}$. A further breakdown of the computation of every dependency of $h^{(t)}$ can be seen in Figure 2.9. It is the integration of the cell state $c^{(t)}$ dictated by the learnable gates that allow LSTMs to retain long-term dependencies.

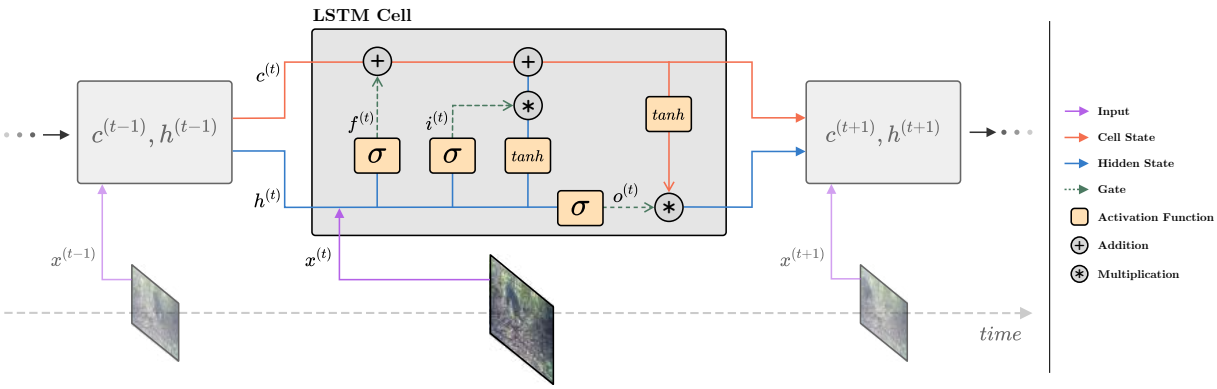


Figure 2.9: An **LSTM cell** computing the cell state c and hidden state h for time-step t, while showing the compositions of the three gates $f^{(t)}$, $i^{(t)}$, $o^{(t)}$. Input $x^{(t)}$ is added to $h^{(t-1)}$. Note that the blue line is shown to be $h^{(t)}$, but at the point of the addition, it has not yet been transformed in any way. Thus, it is still considered as hidden state $h^{(t-1)}$ of the previous time-step. The LSTM cell can be seen in relation to the rest of the sequence, where states c and h are retained and passed across every time-step of the sequence.

LSTMs depend on a variation of the backpropagation algorithm, called Backpropagation Through Time (BPTT) [54]. A forward pass of the unrolled LSTM is performed sequentially, followed by backpropagation to calculate derivatives across the time steps. Parallelisation is not possible due to its sequential characteristics. Thus, computing the gradient for LSTMs is naturally memory and time intensive.

The effectiveness of LSTMs for action recognition has been previously demonstrated [52]. Its method for obtaining latent representations of temporality alternate to the Two-Stream method [47] may prove useful. Furthermore, it allows both streams to consider multiple frames as input, instead of just the temporal stream.

2.10 Class Imbalance

Class imbalance refers to the unequal distribution of data representing each class. Most of the data may belong to only one, or several, classes. They are known as majority classes. This leaves a disproportionately lower volume of data for the remaining classes, named minority classes. This leads to networks being overwhelmed by the majority classes, unable to satisfactorily perform on minority classes.

It appears that the great ape dataset suffers from class imbalance. As mentioned in Section 1.4, some behaviours occur far more often than others. Japkowicz concluded a model's sensitivity to class imbalance "increases with the complexity of the domain" [18]. Considering the scale of the input and the two-stream architecture, the effects of the class imbalance may significantly challenge its performance.

2.11 Sampling

To effectively train and evaluate the model, the dataset is split into three mutually exclusive subsets, which are sampled for different purposes.

1. **Training set**: Used for training the model. Includes the majority of the dataset.
2. **Validation set**: Used for periodic evaluation of the model during, or after, training. It allows for evaluation of the model on unseen data, helping to identify the model's performance. It also makes it possible to track the impact of adjustments made to the model.
3. **Test set**: Only used for the evaluation of the final optimal model.

The subsets of data are randomly allocated. However, the smaller validation and test sets may have to be further curated, to ensure they contain samples that represent all the possible classes.

2.11.1 Oversampling

Oversampling involves duplicating samples from minority classes when training, to make up for class imbalance. The result of this is that the model is trained on the same number of samples for each of the classes. Studies have shown that it is very effective in dealing with class imbalance [18, 14]. However, since the minority classes are matched to the size of the majority classes with duplication, the model

trains on the same subset of data many times. The downside to this is that the model may overfit the training data that it has seen for the minority classes.

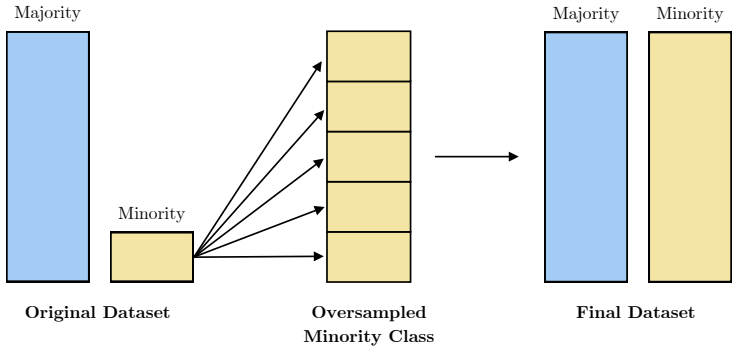


Figure 2.10: **Oversampling** is shown to be done with the duplication of the minority class until it is of equal size to the majority class. This is seen as an effective technique to counter class imbalance.

2.12 Dataset

As mentioned in Section 1.2.2, the great ape dataset that forms the basis of this project includes existing annotations created as part of [56]. There are existing annotations for each great ape in every frame across the entire dataset. They include labels of:

- **Species:** The type of great ape (e.g. chimpanzee)
- **Bounding box coordinates:** A pair of coordinates forming the localisation of the great ape, inside a rectangle within the frame. They are in the form (x, y, x_{max}, y_{max}) where (x, y) represents the top left corner of the bounding box, and (x_{max}, y_{max}) represents the bottom right corner.

These annotations make it possible to classify, and spatially locate the great apes within the videos. This will be useful for obtaining appropriate samples for the behaviour recognition model to train on. This is also sufficient information to train an object detector for great apes, using YOLO [40]. However, there are no existing annotations relevant for performing behaviour recognition of great apes. Therefore, the following annotations need to be added to the dataset:

- **Behaviour:** The activity (e.g. walking) being exhibited by the great ape. This will only be made up of one behaviour, as simultaneous behaviours will not be taken into account by the model.
- **Identification number (ID):** A unique integer to identify each great ape within a video. IDs can be assigned by order of appearance, starting from 0 and incrementing from then onwards.

The behaviour is the "target" that the model will use to train. Its difference from the model's predictions will be used to minimise the loss function (see Section 2.3) to improve its performance.

Section 2.8.1 outlines the use of a two-stream model where the temporal stream is exposed to stacks of optical flow images across consecutive video frames. The stack of images must contain the same great ape throughout, so that the motion of only that ape is analysed without disruption. The existing annotations contain no information that indicates which ape a certain bounding box belongs to, within a video. A human is able visually assign each of the boxes to the great apes present, as they are constrained to the outline of their assigned box. However, a computer is unable to do this (without additional information) and so cannot automate the extraction of a great ape across consecutive video frames, with the guarantee that it is the same great ape throughout. For this reason, IDs must be added to the annotations to create "tracklets", where each bounding box is associated to a great ape within the frame. This will allow for correct automated extraction of optical flow stacks of apes, as needed for the temporal stream.

2.13 Statistical Metrics

The following metrics are used to examine and evaluate the performance of the model in this project. They are be applied as appropriate, to all three subsets of the data (outlined in Section 2.11).

Top1 Accuracy The overall proportion of predictions a model has correctly made.

Top3 Accuracy A Top3 accuracy is used as a looser metric. Similar to Top1 accuracy, but instead a prediction has to be in the top 3 probability values for it to be correct. The bound of 3 is used as it appropriately reflects the number of classes that the dataset consists of.

Class Average Accuracy The mean average of the Top1 accuracies for the individual classes, valuing a model's performance across all classes. Inconsistencies between the Top-1 accuracy and the class average accuracy may point to the model's inabilities to classify some behaviours as well as others.

Loss The loss, as explained in Section 2.3 and 2.6, is an indicator of a model's performance.

2.14 Experiments

The following experiments are carried out to gain a better understanding of the final optimal model.

Ablation Study This involves identifying key components and parameters of the model, which are then removed or reduced. The model is evaluated to measure the influence of the modifications upon its overall performance. This helps to discover aspects of the model that are indispensable for its performance, and conversely, of those that are not.

Cross-Validation k-fold cross-validation [22] is a technique used for thoroughly testing a model's performance on unseen data. The dataset is split into k mutually exclusive "folds". The model is trained k times. In each iteration, a different fold is assigned as the test set for evaluation, while using the other $k-1$ folds for training purposes. Cross validation can help to measure a model's generalisability, while also pointing out any signs of instability. A model's performance with regards to a certain class may heavily depend on the samples that it has seen for that class. Thus, cross-validation can minimise the presence of selection bias [13], which arises due to poor non-representative sampling.

2.15 PyTorch

PyTorch [70] is an open-source machine learning framework for Python. `torchvision` is a part of PyTorch [70]. `torchvision.models` contains the definitions of popular models that are used in computer vision. Pre-trained weights of the models can also be obtained for the purposes of transfer learning.

PyTorch defines an n-dimensional array class named `Tensor`, which can be stored and operated on within the GPU for faster performance. PyTorch also supports NVIDIA CUDA [37]/CUDNN [5], providing GPU-accelerated libraries specifically for neural networks. This is desirable for a project of this nature, where large amounts of visual data will need to be processed. The acceleration from GPU resources significantly speeds up model training and evaluation.

PyTorch makes it possible to conform any dataset, to be used for training models. They need to inherit the PyTorch `Dataset` class [70]. This is used to index all of the samples in the dataset. The `Dataloader` [70] class can then be used to represent the custom dataset as an iterable. This allows the loading of data to the model to be heavily customised, with features such as batch loading, custom samplers and more. PyTorch also parallelises the process of data loading, significantly reducing overhead during training.

```
class GreatApeDataset(torch.utils.data.Dataset):
  def __init__(self, args):
      # Initialise the dataset of samples into iterable list
      # Initialise parameters and paths to data on disk
  def __len__(self):
      # Return the number of samples available in the dataset
  def __getitem__(self, i):
      # Given an index i < self.__len__(), find the i^th sample in the dataset.
      # Pre-process then return the sample, along with its ground truth label
```

Listing 1: **Custom dataset**: The abstract functions to be implemented to inherit the PyTorch `Dataset` class. This allows the dataset to make use of optimised data loading, batch sampling and custom samplers.

Chapter 3

Project Execution

This chapter explains the methodology underpinning the outcomes of this project. The project naturally splits into a number of components that are to be focused on. Their completion is key to achieving the desired project goals. The components and their purpose within the overall project are illustrated in Figure 3.1. Components targeting major aspects of the project are grouped together, each forming a section within this chapter holding detailed descriptions of how they were accomplished.

3.1 Data

The project begins by examining the great ape dataset, which is what the behaviour recognition model ultimately depends on for its training and evaluation. There is a total of 500 videos of jungle trap camera footage. Across the dataset, there are two types of great apes that can be seen: chimpanzees and gorillas. Each video is 15 seconds long, playing at a speed of 24 frames per second (FPS). This amounts to 360 frames per video. The resulting total number of frames within the dataset is ∼180,000 frames. There is an equal number of `XML` files, holding annotations, as discussed in Section 2.12, for the corresponding frame. The annotation files are named in the format `<video>_<frame_no>.xml`, where `frame_no` represents the i^{th} frame which appears in `video`. The 500 videos in the dataset are split into three subsets. The training set is made up of 400 videos. There are 25 videos in the validation set, while the remaining 75 are reserved for the test set.

3.1.1 Class Definition

Initially, the data is assessed to ensure that there is a thorough understanding of the scope of the dataset. Aspects such as the range of behaviours that can be observed, and the frequencies at which they occur, are considered. This helps to define the individual behaviours the model should attempt to classify.

It is important to note that the data is labelled on a frame-by-frame basis. Each frame in a video is 0.042 ($1 \div 24$ FPS) seconds long. This means that the representation of the apes' behaviours are fine-grained. There is a very small amount of movement that is expected to occur within the duration of one frame. With this, it is easier to determine an ape's behaviour in some contexts compared to others. For example, the decision to label an ape as "walking" at frame f is straightforward if the frame falls halfway through the overall duration that it was walking. At this point in time, there is more likely to be contextual information of the ape appearing to walk before and after f to help determine this. However, at the event at which the ape stops to stand still, the surrounding contextual information is not enough to pinpoint the exact frame where the transition between the behaviours take place. So, it must be objectively clear how the appearance of an ape, and its movement, establishes its behaviour, in order to remove any ambiguity found within these "transitional behaviours".

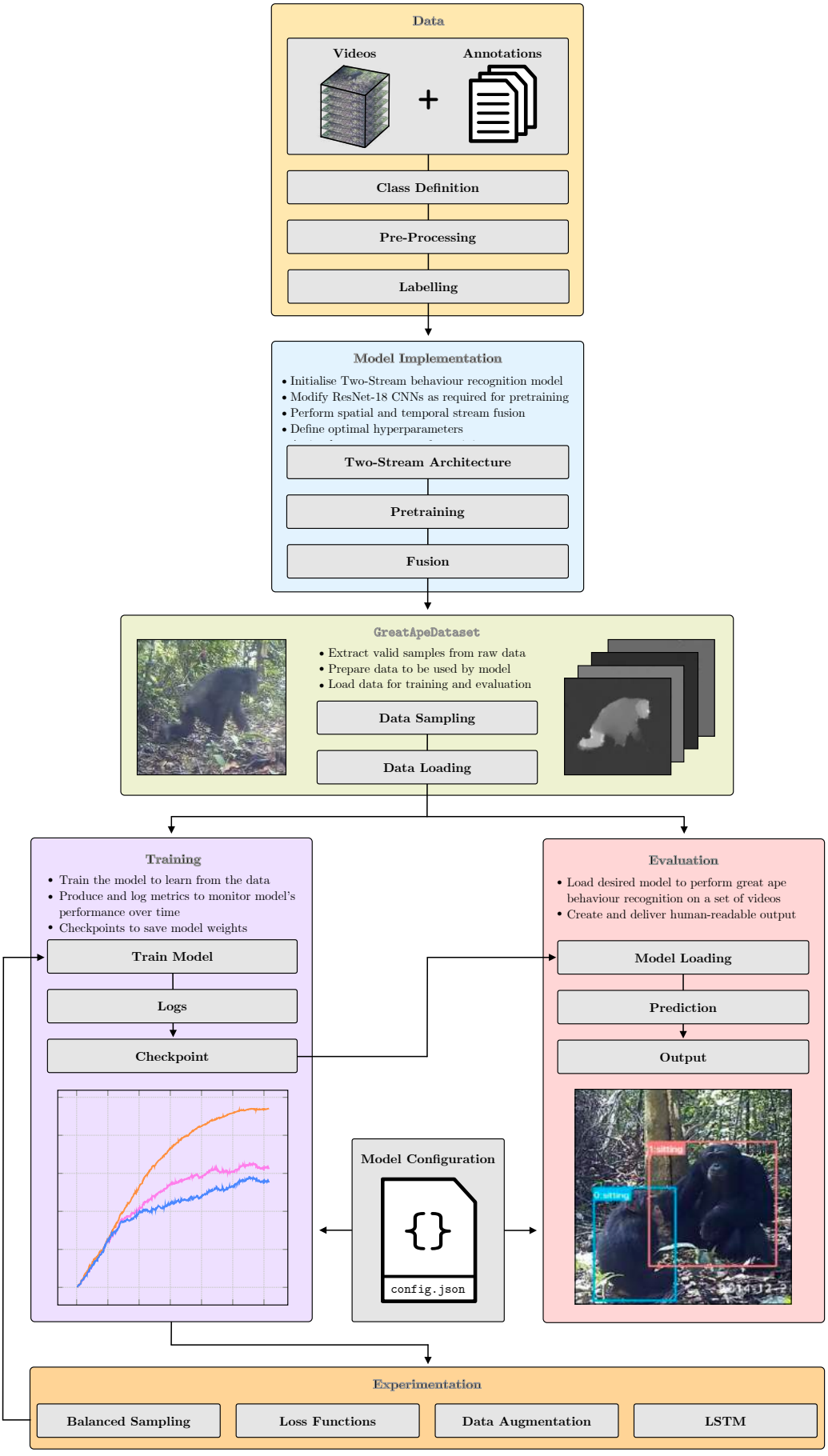


Figure 3.1: **Project diagram** illustrating the main sections of the project and their individual components. Arrows indicate relations and the interoperability between the components. Each coloured box is associated with a section in this chapter of the same name.

By considering the issues above, all behaviours seen within the data can be categorised as correctly and consistently as possible. This is crucial since poorly labelled data may have a negative impact on the model itself. Therefore, a robust framework is created, specifying each viable behaviour class, while thoroughly outlining the visual requirements for an ape to be classified as such. Admittedly, this may not remove all error in labelling, especially in the edge cases of transitional behaviours. Nevertheless, it is a practical attempt to mitigate ambiguity. It also provides a guideline for labelling additional data in the future, for this project or for external parties and purposes.

Figure 3.2: **Transitional behaviours** displaying the type of ambiguity found in the data. Apes are commonly caught for a small number of frames transitioning between behaviours, but not exhibiting either behaviour concretely. The apes in these examples are seen standing still for some amount of time, before beginning to walk (left), lowering to sit down (centre) and preparing to climb up a tree (right).

After assessing the data, a total of nine core behaviour classes are committed to. The following shows the framework defining each class, followed by their necessary visual indications (class name shown as labelled within the annotations):

- `camera_interaction` : The ape is interacting with the camera.
  - The ape is physically interacting with the camera with their hands.
  - The ape may also instead stare at the camera from a very close distance. If so, they must be close enough so that their entire body is unable fit into the frame of the camera.
- `climbing_down` : The ape is making a vertical descent.
  - The ape is climbing down the trunk of a tree. The behaviour begins as soon as their vertical movement begins.
  - The ape may also instead climb down surfaces or tree branches with a movement similar to walking. The behaviour begins as soon as one of their limbs no longer makes contact with the surface they are currently on, thus initiating their descent.
- `climbing_up` : The ape is making a vertical ascent.
  - Same conditions as `climbing_down`, but the ape is seen to be moving upwards instead.
- `hanging` : The ape is stationary while hanging off a tree or its branches.
  - The behaviour applies for as long as the ape is stationary while clutching on to the surface.
  - The ape is considered stationary as long as none of their limbs are used to elevate or lower their position.
- `running` : The ape is running.
  - The ape's movement is significantly faster than that of when it is walking.
  - Great apes tend to have a strong swinging motion when running. They plant their hands into the ground in front as anchors, before swinging their legs to thrust them forward with speed.
- `sitting` : The ape is sitting on the ground, or any other surface.
  - As soon as the ape's bottom makes contact with the surface they begin sitting.
  - In contrast, as soon as their bottom leaves the surface, they are no longer sitting and the transition to the next behaviour has occurred.

- `sitting_on_back` : The ape is sitting or holding on to the back of another ape.
  - The ape is commonly a child, on top of an adult ape.
  - Typically the ape will not come off of the adult ape. If they do, the behaviour ends as soon as they make contact with the surface they are trying to reach.
- `standing` : The ape is stationary while standing.
  - Great apes most commonly stand on all four of their limbs. This class also represents the rare occasion they stand only with their feet.
  - When all four limbs are planted and the ape remains stationary, they begin standing.
  - The movement of an ape's limb towards a change in their position indicates an end to the behaviour, and the start of the next.
- `walking` : The ape is walking at a normal pace.
  - The exact end of an ape exhibiting walking behaviour is indicated by the start of another behaviour (e.g. all four limbs are planted when beginning to stand).

There are certain "auxiliary" behaviours, namely eating and the act of scavenging, that are usually done while other behaviours are being exhibited (e.g. an ape could eat while sitting or walking). However, these are not considered as behaviour classes as the model defined for this project aims to only classify one behaviour at any one time. This can however be seen as future work (see Section 5.1).

3.1.2 Preprocessing

A number of tasks need to be completed to make sure the data is made ready for labelling, and eventually training a model. This is known as "preprocessing" the data.

With the annotations of ~180,000 video frames available, the dataset is large. An attempt to manually process and label each frame would take an unreasonably long time. To reduce this time spent, a number of scripts are written to help automate some aspects of the tasks involved. The main area where time can be saved, is by having the scripts apply the same single operation across multiple frames. As such, each script is responsible for a particular operation. They take in a number of parameters specific to the operation. Among them are the indexes of the first and the last frame for which the operation needs to be performed on, including all the frames in between. This can be done as the annotations are ordered and numbered with their frame indexes. Thus, the script can then quickly perform an operation for a subset of the video.

The scripts are written in `Python`. In order to access and modify the `XML` annotation files, the `ElementTree XML` API [69] is used. It allows to parse and store `XML` files as hierarchical data structures that can be read and manipulated as required.

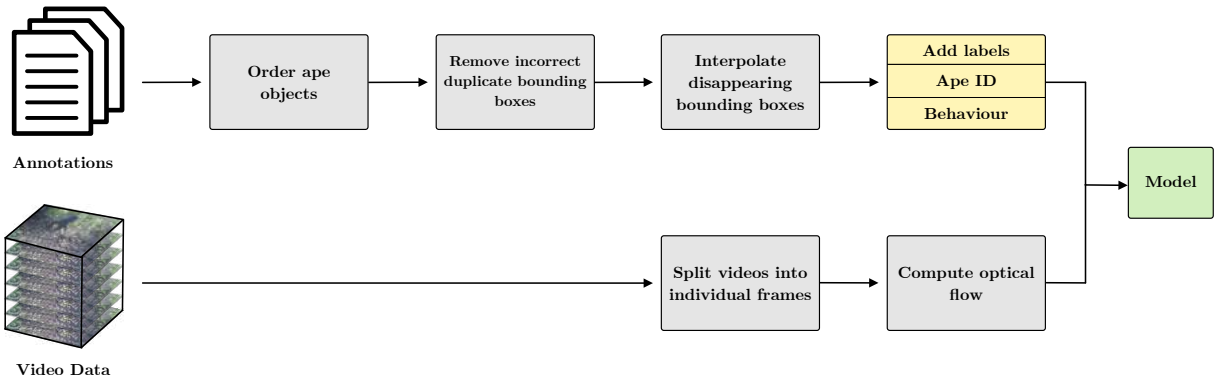


Figure 3.3: **Data preparation**: A diagram showing the numerous preprocessing tasks required to prepare the data for labelling, and subsequently for use by the behaviour recognition model. Note that the dataset consists of two types of data: raw video footage and annotations for each frame. Details of each task are described in Section 3.1.2 and 3.1.3.

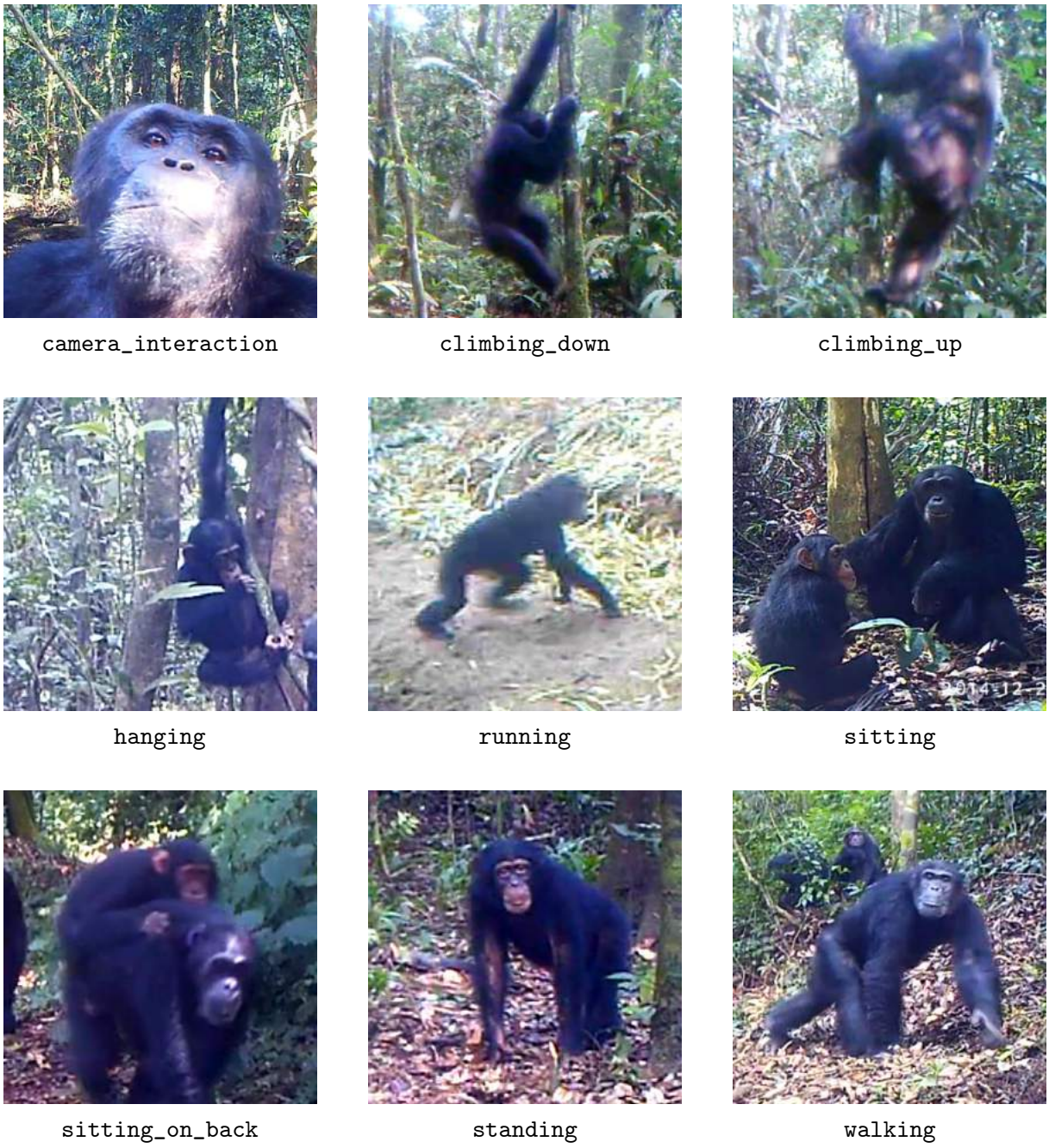


Figure 3.4: **Behaviour class definition**: Examples of great apes exhibiting each of the nine core behaviours defined in Section 3.1.1. These behaviours were seen to occur most often in the dataset and have been categorised as such.

Ape Ordering: An `XML` annotation file may include a number of apes. Each ape is represented by an "ape object", holding its species class, and bounding box coordinates. To be able to modify the ape object, their position (index) within the list of ape objects needs to be known. Discussed in Section 2.12, the existing annotations do not continuously keep track of which ape object is associated with which ape in the video. This is because there is no order to the appearance of the ape objects in the annotations. For example, in a frame f, there may be two apes present, with their objects, `ape1` and `ape2`, appearing in that order. In frame $f+1$, a new ape `ape3` appears. The `XML` annotation for this frame could then become `ape1`, `ape3` and `ape2`, in that, or any other order. Thus, the ordering, and the indexes for each ape, is not maintained. Furthermore, this issue may arise even if there are no new apes that are introduced in the video, where the order of the existing ape objects change in between frames.

This issue with the existing annotations needs to be addressed, because an ordering of some nature is required to be able to perform automated operations on a particular ape across multiple frames. If a

behaviour label needs to be assigned to `ape1` from frames f to $f + k$, it needs to be guaranteed that the index for this ape is maintained all the way through between these frames. The best way to order the ape objects is to utilise the information they already hold. Therefore, the x value of the bounding box is used to order the ape objects in the annotations. This essentially means the ape objects are ordered in terms of the bounding box positions from left to right. However, it is important to note that this does not fully resolve the issue. Due to the movement of apes, where one bounding box overtakes another in either direction, the order of the ape objects would change. The changing order, and indexes, of the ape objects need to be carefully observed to avoid incorrect labelling.

Duplicate bounding boxes: In some cases, the annotations show that there are multiple ape objects attributed to a single ape, with more than one bounding box placed over the same area of the video. The boxes mostly overlap, but do not have the same exact coordinates. These are mistakes found in the annotations. Addressing this by simply removing all boxes apart from one, could lead to loss of precision or being left with an incorrect bounding box. The number of bounding boxes for an ape needs to be reduced to one, while taking into account the information of the duplicate boxes. Instances of multiple bounding boxes tend to have a significant Intersect over Union (IoU), which describes how similar two boxes are by taking into account the value of the area in which they overlap. In this case, a common way to derive one bounding box out of n separate boxes, is to take the average of the coordinates.

$$\text{box} = \frac{\sum_i^n (x_i, y_i, x_{\max\ i}, y_{\max\ i})}{n} \tag{3.1}$$

Interpolation: There are also instances where the bounding box for an ape disappears before reappearing, even when the ape is in clear sight. This is undesirable, as it leads to an incomplete dataset where potential samples for use by the model are lost. Fortunately, the disappearance of bounding boxes typically last for frames within one order of magnitude. Therefore, it is possible to confidently interpolate the coordinates of the bounding box, from where they first disappear, to when they return.

$$(x, y, x_{\max}, y_{\max})_{\text{diff}} = (x, y, x_{\max}, y_{\max})_{\text{last}} - (x, y, x_{\max}, y_{\max})_{first} \tag{3.2}$$

$$\text{box}_i = (x, y, x_{\max}, y_{\max})_{\text{first}} + \frac{i}{n}(x, y, x_{\max}, y_{\max})_{\text{diff}} \tag{3.3}$$

The difference *diff* is calculated between the coordinates of the bounding box at the first frame and the last frame of the interpolation. These frames correspond to when the box is seen last, and when it reappears again, respectively. To get the coordinates of the bounding box at the i^{th} frame, the bounding box at the start of the interpolation is incremented by *diff* with a factor of $\frac{i}{n}$, where n is the total number of frames that the box is being interpolated across. This effectively divides the positional movement of the bounding box from the first frame of interpolation until the last into n steps.

Video Processing: The data consists of videos, where its annotations are for each frame. The model processes the data as images. Therefore, the videos need to be split into still frames. To do this, `OpenCV`'s `VideoCapture` function is used to obtain every frame for every video. The frames are then saved to disk as RGB `JPEG` images, using `imwrite`. The frames are named in coordination with the annotations, using the same naming format.

The requirement of optical flow images for capturing the temporal information of the data is stated in Section 2.8.1. Although it is possible to compute optical flow in real-time, it may prove to be a bottleneck, since the model may have a very high throughput of data during training. To minimise the overhead in computation and time, the optical flow for the dataset is instead pre-computed and saved to disk. The `optflow` package in `OpenCV` is used to compute the optical flow images. Consecutive pairs of video frames are given as input to the function `createOptFlow_DualTVL1()`, producing TV-L1 [39] optical flow images in the horizontal and vertical directions. They are then normalised to values $[0, 255]$ to convert to greyscale, as suggested in [47]. Since the optical flow is calculated for pairs of images, a video with n frames results in $n - 1$ optical flow images. To ensure that the size of the spatial and temporal data is kept equal, the optical flow is "padded". This means the final optical flow image is simply duplicated and is marked as the n^{th} frame of the video.

Figure 3.5: **Duplicate bounding boxes**: An instance of incorrect bounding box coordinates found in the existing annotations (left). They are corrected by averaging the coordinates across the multiple bounding boxes (right).

3.1.3 Labelling

Following the preprocessing above, the data is ready to be labelled. This involves the labelling of behaviours and identification numbers as mentioned in Section 2.12. Several scripts are written to facilitate the labelling. To carry out an operation, the relevant script can be run with the desired parameters:

```
$ python add_behaviour.py [filename] [start frame] [end frame] [ape index] [behaviour]
$ python add_id.py [filename] [start frame] [end frame] [ape index] [id]
```

`filename` refers to the video of which the annotations need to be modified. `start frame` is the frame index at which the operation will begin, performing on every frame until the frame specified by `end frame`. `ape index` signifies the particular ape object that will be operated on. Finally, `behaviour` and `id` are the actual labels that will be applied to the ape object in the operation.

The framework introduced in Section 3.1.1 is thoroughly adhered to, ensuring consistency in labelling. It is worth noting that labelling such a large amount of data such as this, makes it susceptible to human error. Validation is put in place where possible to minimise the error. This includes checking for existing labels in the annotation, ensuring correctly spelled behaviour classes, among other measures. In the case that an error is made, and is subsequently noticed, scripts are created to edit or remove incorrect labels. They are implemented similar to the label addition scripts, performing operations quickly across multiple frames, to minimise the time taken to fix errors.

The final size of the dataset amounts to ~220 `GB`. This includes the videos, their individual frames, optical flow and the annotations. The directory structure of the data is shown in Appendix B. An example of a pre-processed and labelled `XML` annotation file can be seen in Appendix A.

3.2 Model Implementation

This section explains how the initial great ape behaviour recognition model is implemented.

3.2.1 Two-Stream Architecture

A two-stream network, as described in Section 2.8.1, is proposed for the implementation of a behaviour recognition model for great apes. The model consists of two CNNs, taking into account the spatial and temporal dependencies separately [47].

The two networks are based on the 18-layer variant of the ResNet [12] architecture, named ResNet-18. This is in contrast to the use of the VGG [48] architecture in [47, 9]. At the time of writing, the use of ResNet over VGG is motivated by a number of factors. ResNet is shown to consistently outperform VGG [12] while having signficantly lower time and memory requirements. ResNet's residual properties, along with its use of fewer channels enables it to have lower complexity. ResNet-18 requires only 1.8 billion floating point operations per second (FLOPs), as opposed to VGG-19 with 19.6 billion FLOPs. This efficiency is gained without the expense of performance. ResNet-18 is used as it is the variant with the closest number of layers to the 16-layer variant of VGG used by Simonyan et al [9].

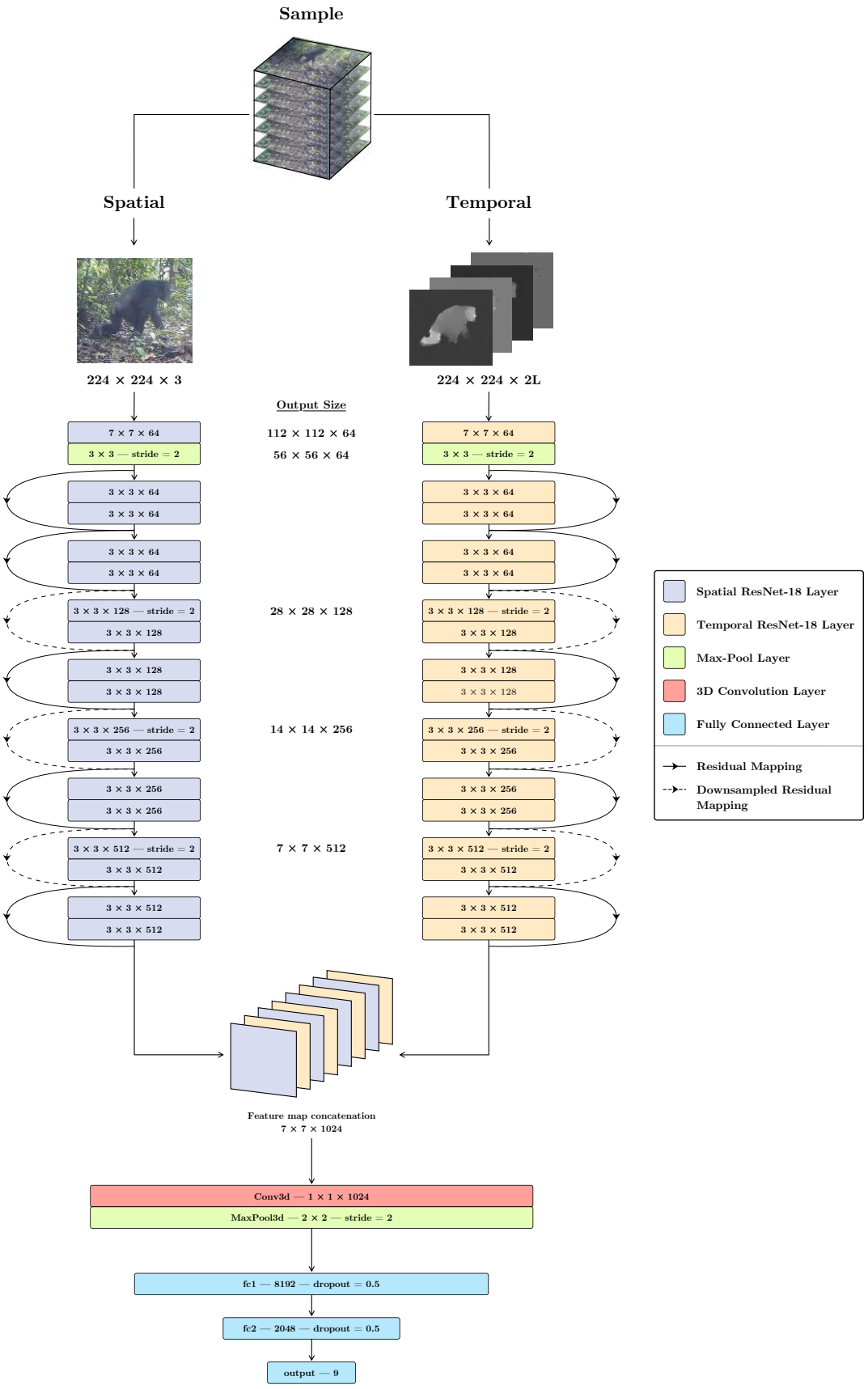


Figure 3.6: **Two-stream great ape behaviour recognition model architecture**. The spatial stream is shown on the left which takes in a single RGB still image. The temporal stream uses a stack of $2L$ optical flow images for L consecutive frames, as shown on the right. The streams are modified ResNet-18 CNNs pretrained on ImageNet. The feature map output of the streams are stacked and passed through a 3D convolutional layer, combining the spatial and temporal dependencies together. The fused output is finally passed through a fully connected network for classification.

Early experiments specific to great ape behaviour recognition confirms ResNet-18's superior performance to VGG-16 and VGG-19. Further experiments showed the ResNet variants with more than 18 layers performed worse than ResNet-18. Although ResNet's residual layers are able to prevent accuracy decreasing parameter updates, they can still be prone to overfitting, specifically form models of greater depth have higher complexity with more parameters. The lesser performance of the more complex variants of ResNet can be attributed to the insufficient data available for them to learn generalised parameters.

3.2.2 Pretraining

The spatial ResNet-18 network is pretrained on ImageNet [6] using weights obtained from Pytorch's `torch.utils.model_zoo` to make use of transfer learning. It is kept unfrozen, training the pretrained weights as normal. The temporal network is also pretrained. Although the data that the temporal network trains on are greyscale optical flow images, which is different to the RGB data found in ImageNet [6], the pre-training is still expected to be beneficial. It provides a good basis to start training from, with fundamental features such as shape extraction still required by the temporal stream.

Recall from Section 2.8.1 that the spatial and temporal streams both have different input sizes of $H \times W \times 3$ and $H \times W \times 2L$ respectively, where L is the number of consecutive frames in the temporal stack. The implementation of ResNet-18 uses a default input size of $224 \times 224 \times 3$. The spatial stream is compatible with the default ResNet-18 architecture. However, to ensure compatibility with the temporal stream, its ResNet-18 network needs to be modified. In particular, the number of channels of the first convolutional layer is changed to $2L$. Furthermore, the default fully connected classifiers at the end of both streams output a total of 1000 values corresponding to the number of classes in ImageNet [6]. These classifiers need to be modified to output nine values, as per the number of behaviour classes that exist in the great ape dataset, as defined in Section 3.1.1.

An issue arises due to the change in input channels of the first convolutional layer in the temporal ResNet-18 stream. One of the constraints of using pretrained weights is that the network they are mapped to must exactly match the network that the weights were originally trained for. In this case, the pretrained weights come from the original ResNet-18 network. However, some modification of the application of the weights themselves can help to circumvent this issue. The modifications proposed for the network only exist at the input and the output layers of the network. The mapping of weights is not be affected by the output layer which is modified to produce a nine value vector. This is because the output layer does not have any weights assigned to them, as their values are dependent on the weights across the network before it. On the other hand, ResNet-18's first convolutional layer takes in 3 channels as input. Each channel has a pretrained weight assigned to it. In contrast, the temporal ResNet-18 has $2L$ channels at the input layer. Therefore, the pretrained weights of the 3 channels need to be appropriately distributed across $2L$ channels. This is done by summing the pretrained weights across the 3 channels of the input layer. It is then divided by 3 to obtain the mean average of the weights. Finally, this average weight is assigned to each of the $2L$ channels in the first convolutional layer. This approach is invariant to the number of channels that the network uses. Although some precision is lost in the weights of the input layer, the network retains the value from being able to utilise the vast majority of the pretrained weights. During training, the weights of the input layer can still be fine-tuned as necessary.

3.2.3 Fusion

Late Fusion

The initial method used to fuse the two networks together is late fusion. The spatial and temporal networks are initialised separately, with their own loss functions. Both are trained and validated together in step using the relevant data as their input. When obtaining the output of the two networks, they are used to optimise only their own loss functions. To compute the final classification output, the two output vectors of size 9 from both streams are averaged.

Convolutional Fusion

Convolutional fusion is used to capture spatial and temporal inter-dependencies earlier in the network. The two ResNet-18-based networks are placed within the same model optimising a single loss function. The output of the two networks need to be prepared accordingly so that they can be fused.

The networks are further modified by removing the final two layers; an average-pool layer and a fully connected classifier. They are not required as they reduce the dimensionality into a 1D final classification vector. To perform convolutional fusion, the feature map from both networks need to be preserved so that they can be combined and convolved over together. The new shortened networks produce an output of $7 \times 7 \times 512$. The output of the spatial and temporal networks, which are of same size, can be stacked at the same coordinates as described in Section 2.8.1, to obtain a feature map of size $7 \times 7 \times 1024$.

```
def forward(self, spatial_data, temporal_data)
    # Compute output from spatial and temporal ResNet-18 streams
    x1 = self.spatial(spatial_data)
    x2 = self.temporal(temporal_data)

    # Stack feature maps to form fused output
    y = torch.cat((x1, x2), dim=1)
    for i in range(x1.size(1)):
        y[:, (2 * i), :, :] = x1[:, i, :, :]
        y[:, (2 * i + 1), :, :] = x2[:, i, :, :]

    # Pass output to 3D convolution layer, followed by fully connected classifier
    y = y.view(y.size(0), 1024, 1, 7, 7)
    cnn_out = self.layer1(y)
    cnn_out = cnn_out.view(cnn_out.size(0), -1)
    out = self.fc(cnn_out)
    return out
```

Listing 2: **Forward pass** implemented for the two-stream model. It involves computing the output from the ResNet-18 streams, which are then stacked as described in 2.8.1. This is then passed through a 3D convolutional layer to perform fusion, before finally performing classification using a fully connected network.

The stacked feature map is then sent through a 3D convolutional layer with an input of 1024 channels and an output of 512 channels. A kernel size of 1 is used, along with padding and stride of 1. Bias is also enabled. This performs convolutions over the alternating spatial and temporal channels, fusing them together. ReLU is applied to the output. Then a 3D max pool layer with a kernel size and stride of size 2 is applied. The output from the max pool layer signifies the end of the fusion, and it is then flattened to a 1D vector of size 8192.

Finally, a fully connected network is used to perform classification. A total of three layers are used. The first layer has an input size of 8192 and an output size of 2048. The second layer then has an output size of 512. The final layer produces a 1D vector of size 9, with each value representing the probability of a behaviour. ReLU is performed after the first and second fully connected layers. A dropout with probability $p = 0.5$ is also applied after the first and second layers.

3.2.4 Hyperparameters

The model uses the cross entropy loss function. SGD with a momentum of 0.9 is used for optimising. The learning rate is set to 0.001. One of PyTorch's learning rate scheduler `ReduceLROnPlateau` [70] is used to dynamically adapt the learning rate depending on the model's performance during training. It acts to reduce the learning rate when training stagnates by keeping track of a metric. The metric used is the loss. The scheduler is given the parameters `mode='min'` since the loss is a minimisation objective function; `factor=0.1` which is the multiplication factor used to reduce the learning rate; `patience=5` which is the number of epochs with no improvement that is required for the learning rate to be reduced. A batch size of 32 is used.

3.3 `GreatApeDataset`

The videos and annotations in the dataset need to be used specifically for the task of behaviour recognition. They need to be appropriately sampled and loaded to the network for use during training and evaluation. For this, the `GreatApeDataset` class is initialised, inheriting the PyTorch `Dataset` class.

3.3.1 Data Sampling

In order to obtain any i^{th} sample from the dataset, there needs to be a list defining every sample that exists in the dataset. When an instance of `GreatApeDataset` is created, `initialise_dataset()` is called which searches through every annotation of the dataset to produce and store a Python `dict` hashmap containing the samples. It is worth noting that a "sample" in this context is not the actual data, but instead an identifier of where the particular item of data is. The hashmap is categorised by the names of the videos. For each video, there is a list of samples. A sample is made up of three items: the behaviour label for the ape, `behaviour`; the index of the ape object within the annotation, `ape_id`; the index of the frame at which the sample begins, `start_frame`.

Due to the complexity of the data and the use of two different networks, as discussed in Section 2.8, there are several aspects to consider in terms of what actually constitutes a sample. During its search, `initialise_dataset()` ensures that the samples chosen are valid by meeting certain criteria. There are three main parameters that are observed: `temporal_stack`, `sample_interval` and `behaviour_duration_threshold`. The parameter `temporal_stack` is the number of consecutive frames that are considered for a particular temporal sample. Note that the actual number of images in the stack is double this value due to the optical flow being calculated for both directions (see Section 2.8.1). A sample has to have the same behaviour throughout the length of the `temporal_stack` for it to be valid. `sample_interval` refers to the number of frames between each sample. Finally, the parameter `behaviour_duration_threshold` is the number of consecutive frames that an ape has to exhibit a behaviour for it to qualify for sampling.

The particular problem this parameter aims to address are the instances where an ape does not display its behaviour for a sufficient period of time. In more sustained displays of behaviour, there may be more prominent characteristics in appearance and movement. By having the threshold, the data is filtered to obtain quality samples where desired and less ambiguous characteristics are likely to be displayed. This allows the model to better train and learn the behaviours. Taking these parameters into account together, `initialise_dataset()` searches and filters samples accordingly. The algorithm is shown in detail in Algorithm 3.1.

Every sample found is composed of both a spatial and a temporal data sample. This is further explained in Section 3.3.2. This ensures that for every sample, both streams are exposed to the same part of the dataset. This also means that each stream trains on an equal number of samples.

The `GreatApeDataset` instance is initialised with a number of other parameters. It takes a `mode` which signals whether or not the data it represents is for training, which changes the way samples are initialised, as later discussed. The names of the behaviour classes is provided as an alphabetically ordered list, named `classes`, which is used when data is being loaded. Paths to the data are provided so that they can be loaded. Data transforms for the spatial and temporal data are also passed as parameters.

The following sections outline the differences of how each data split is sampled. They involve setting parameters, as outlined in Section 3.3, to obtain appropriate data samples for the model to use.

Training Samples

As mentioned in Section 3.3, there is a need for high quality samples that the model can learn from. Thus, `behaviour_duration_threshold` is chosen to be a value of 72 frames, which equates to 3 seconds. This means all samples must be obtained from an instance where the behaviour has been displayed for at least 72 frames. This value is used as a confident estimate capable of filtering out instances of sporadic behaviours that are not useful for training the model. The size of the `temporal_stack` is 5, leading to a total temporal stack size of 10 optical flow images when accounting for each direction. The `sample_interval` is 10 frames long.

Validation/Test Samples

When the model is evaluated on the test dataset, it is expected to perform in conditions similar to how it should in the "real world". This is also the case for the validation dataset, even though it is evaluated during the training process. This gives an insight into how the model would do in normal circumstances even while it is learning. The real world scenario for this model is that it should take a video as input and perform behaviour recognition throughout.

Algorithm 3.1: `initialise_dataset()`

```
Result: samples – a hash table of all samples extracted from the dataset ordered by video
input: dataset, behaviour_duration_threshold, temporal_stack, sample_interval
set samples = new hash table
Traverse every frame for every ape in every video
for each video in dataset do
    set apes = number of apes in video (each indexed by an ID number)
    set no_of_frames = total number of frames in video
    for each ape in apes do
        set current_frame = 1
        while current_frame < no_of_frames do
            if ape not in current_frame then
                current_frame += 1
                continue
            end
            set valid_frames = 1
            set behaviour = the behaviour label of ape in current_frame
            Keep track of the duration of the behaviour
            while ape's behaviour in (current_frame + valid_frames) = behaviour do
                valid_frames += 1
            end
            Check if the duration of the behaviour meets the threshold
            if valid_frames < behaviour_duration_threshold then
                current_frame += valid_frames
                continue
            end
            set sample_frame = current_frame
            set last_valid_frame = current_frame + valid_frames
            Obtain samples from within valid sequence at the rate of the given interval
            while sample_frame < last_valid_frame do
                if temporal_stack frames exist after sample_frame to form optical flow stack
                 then
                    sample = ape, behaviour, current_frame
                    samples.insert(video, sample)
                    sample_frame += sample_interval
                end
            end
            current_frame = last_valid_frame
        end
    end
end
```

Therefore, unlike the filtering of the data for training, all possible samples are used from the data. `GreatApeDataset` uses a modified version of `initialise_dataset()` which does not take into account the `behaviour_duration_threshold`. The `temporal_stack` is of 5 consecutive frames, to match the number of channels that the model takes as input. The `sample_interval` is 5 frames long, so that no frames are skipped. Since all data is effectively sampled, there is no guarantee that for a consecutive number of frames, `temporal_stack`, the behaviour displayed is the same. This is resolved by taking the mode (most frequent) behaviour across the frames to be used as ground truth that is presented to the model.

It is worth noting that there may be instances at the end of a video where there are not enough following frames left (< 5) to make up a full temporal stack for the model to evaluate. These frames are ultimately dropped. Skipping such a few number of frames would cause minimal impact on the overall evaluation of a video. Whereas an on-the-fly attempt to configure the model to accept a varying number of input channels may have undesired impact as it does not correspond to how the model is trained.

The initial randomly allocated splits of the dataset proved to be not suitable for training, as there were

classes that were not well represented by the validation set. In particular, there were two classes for which there were zero samples: `camera_interaction` and `sitting_on_back`; the class `hanging` only had a total of 7 samples. To address this, lists of the videos in which these classes exist, are obtained. Then, videos for each class are randomly selected to be moved to the validation set. Meanwhile, the same number of videos from the validation set is moved to the training set. The videos were taken from classes `sitting` and `walking`, of which there are many samples in the validation set. It is important to strike a balance when modifying the splits, ensuring that classes are well represented in the validation set for evaluation, while not losing too much data for the purposes of training.

Type	camera_interaction	climbing_down	climbing_up	hanging	running	sitting	sitting_on_back	standing	walking	**Total**
Training	101	39	197	683	72	5787	161	2568	3493	**13101**
Validation	30	16	20	93	7	1195	63	498	979	**2901**
Test	100	32	74	156	92	1751	24	1899	2161	**6289**

Table 3.1: **Sample count by class** obtained for each dataset split given the sampling configurations specified in Section 3.3.1 where `temporal_stack`=5, `sample_interval`=10 and `behaviour_duration_threshold`=72.

3.3.2 Data Loading

During the execution of the model, the PyTorch `Dataloader` class handles the fetching of the data samples through `GreatApeDataset`'s function, `__getitem__(index)`. It uses the index to traverse the hashmap of samples that are ordered by video to get the i^{th} sample. Once the sample is found, the actual data is then fetched. The spatial data is found in the RGB frame of the video at the end of the temporal stack, where the frame index would be the sample's `start_frame`, offset by `temporal_stack`. The image is loaded from disk using `Pillow`, Python's image library. However, the entire image is not wanted as the behaviour only corresponds to one ape. So, the coordinates of the ape's bounding box are retrieved from the annotations using the `ape_id` recorded in the sample. They are used to crop the image to extract the ape belonging to the sample. The cropped image is resized to size 224×224, followed by RGB normalisation, as required by the ResNet-18 spatial network.

The model expects to train both streams in step, so the temporal stack is also created in the same function. A `Tensor` of size $224 \times 224 \times 2L$ is initialised, where L is `temporal_stack`. For every two channels in $2L$, the vertical and horizontal optical flow of a frame is assigned, stacking them across consecutive frames as described in Section 2.8.1. Similar to the spatial data, the frames are cropped around the ape, followed by greyscale normalisation. Finally, `__getitem__(index)` returns a tuple of three items. This consists of the spatial data, temporal data, and the behaviour label that is used by the model to verify its predictions. The label is a value between $[0, 8]$ representing the index at which the corresponding behaviour appears in the list `classes`.

3.4 Project Management

Over the course of the project, good coding practices are maintained where possible. This includes good documentation, use of explanatory comments, appropriate modularisation of code, to name a few. This is to ensure compatibility and accessibility of the resulting work beyond the time frame of the project. The project structure is kept as close as possible to the project design defined in Figure 3.1. The directory structure of the dataset is also well defined so that external parties can adapt to the use of their own data with ease. It can be seen in Appendix B along with execution instructions required to train or evaluate the model.

3.4.1 Model Configuration – `config.json`

Even after its implementation is complete, deep learning models tend to be highly customisable. There may be a number of assignable hyperparameters for the model to control its training. Other parameters are involved too, such as the ones discussed in relation to sampling the dataset (e.g. `temporal_stack` or the paths to the data on disk). Typically a model such as this written in Python is accessed through the OS command-line interface. With the use of `argparse`, "flags", such as `--learning-rate 0.1`, can be added to a command to pass custom arguments for variables. However, with the existence of many configurable parameters, this becomes slow and inefficient. It also becomes increasingly difficult to distinguish the configuration of the model between the many number of experiments that are performed.

To resolve this, a `JSON` configuration file is used, named `config.json` by default. This file contains all of the settings for the model that a user may want to adjust. `config_parser.py` uses the package `jsonargparse`, to define a schema for `config.json` to identify and obtain the arguments stored within it. The parser carries out validation to avoid any errors caused by an incorrect configuration. All parameters are validated by type and any provided paths are checked for existence. Moreover, the parser imposes values for certain variables depending on what the model is being used for (e.g. set the batch size to 1 if evaluating). A name is provided for the model to `config.json`. Logging and model checkpointing to disk is done under this name. A copy of `config.json` is saved within the model's log folder for later viewing if necessary. This streamlines the task of tracking the models and their configurations. An example `config.json` is shown in Listing 6 found in Appendix B.

Version Control

Git is used for project management and version control. The repository, along with execution instructions, is hosted on GitHub at https://github.com/fznsakib/great-ape-behaviour-detector. Although an individual project, Git is heavily used for implementation versioning. Branches are used for model experimentation. This is useful as modifications to a model may not perform well, in which case the project can be reverted back to where needed. Releases are used to indicate and publicly deliver source code at milestones. GitHub issues allowed for tasks to be created and tracked. The issues were categorised in terms of the type of issue and its urgency (e.g. bug, experimentation, urgent). This ensured that the project was consistently focused and on track.

3.5 Training – `train.py`

The script `train.py` is responsible for preparing the process of training the model. Its job is to parse the configuration file `config.json`, and use the variables within to initialise instances of `GreatApeDataset` (for both training and validation), the two-stream model and the `SummaryWriter`, which is used for logging. These instances are then used to call the constructor of the `Trainer` class defined in `trainer.py`. The function `train()` is called on the `Trainer`, conducting the training of the network.

The network is trained for the given number of epochs, where each epoch represents the entire dataset being trained on once. The dataset is split up into batches for training. At the start of each batch, the data is received through the `DataLoader` for `GreatApeDataset`. The data, which includes the spatial data, temporal data and behaviour labels, is moved to the GPU for performance acceleration. The spatial and temporal data is then used as input to compute the model's current forward pass. The output produced by the forward pass comes from the classifier discussed in Section 3.2.3, representing the probability predictions made for each of the behaviour classes. They are also known as "logits". These predictions, along with the ground truth labels are used to compute the model's loss as such: `loss = model.criterion(logits, labels)`. To signify the end of training for a batch, backpropagation is performed, updating the weights of the model.

After every epoch of training, the model is validated, where it makes predictions on the validation set. The validation process is very similar to training, apart from two aspects. Firstly, there is no backpropagation involved. There is no need to update the weights of the model as it is simply being evaluated. Secondly, every prediction, along with the true labels need to be collected. This allows performance metrics to be calculated for the model. Following this, the model returns to training.

3.5.1 Logs

The metrics are computed during both training and validation. They include the Top-1, Top-3 and class average accuracies, along with the loss. The accuracy of each individual class is also calculated. They are all logged using TensorBoard [64]. A `SummaryWriter` is initialised in `train.py`, where it is assigned to a directory to write the logs to. Each model has logs written to in their own directory. Using the function `add_scalar`, the metric values can be saved, while also keeping track of the number of epochs and other runtime metrics. The logs automatically produce visualisations of the metrics as time series graphs over the course of training. They can be viewed on the TensorBoard dashboard.

3.5.2 Checkpoint

There is a need to save the model's weights so that they can be used at a later point. This process is called "checkpointing". A checkpoint of the model is saved, using `torch.save()`, at the end of every epoch after its validation. In addition, the highest Top-1 accuracy of the model is kept track of during training. Using this, there are two separate checkpoints of the model that exist. One being the most recent checkpoint, and the other being the "best" model with the highest Top-1 accuracy. A `load_checkpoint` function is implemented to load models later for evaluation. This can also be used to load an existing model to continue its training for any given number of epochs.

3.6 Evaluation – `test.py`

When evaluating the model on the test set, `test.py` is called. Similar to `train.py` it initialises the `GreatApeDataset` and then loads the checkpoint for the model being evaluated. The model evaluates the data samples similar to how it is done during validation, resulting in an array of predictions. The accuracy metrics of the evaluation are computed and returned.

Following this, the second main function of `test.py` is to produce human-readable output from the results. The task of behaviour recognition of great apes is inherently visual. The aim is to augment the existing detection/classification of a great ape, with its behaviour. The model and its system, if used by ecologists, has to be intuitive in its use. This includes the resulting output. Thus, the final output of the model produces modified versions of the input videos, on which bounding boxes are drawn on. The boxes show the ape's position on the video, along with the model's prediction of its behaviour. This allows a person, of any technical background, to be able to understand and analyse the model's predictions.

To produce the final output, `test.py` copies every frame from the test set over to the specified directory found in `config.json`. Then, the predictions, along with the samples' information as produced by `GreatApeDataset` (see Section 3.3) are used to draw the bounding boxes. For every sample, the bounding box is drawn at where it starts, and a proceeding number of consecutive frames, determined by the size of the `temporal_stack` used by the model. The coordinates of each box corresponds to the ape with index `ape_id`. `OpenCV`'s `cv2.rectangle` function is used to draw the box. The `ImageFont` library in `Pillow` is used to produce text for the behaviour annotation. The text size is obtained for every frame. This is used to draw a filled rectangle of a slightly larger size to provide background to the annotation text. It changes its size adapting to the text if necessary. Each ape's bounding box is of a different colour so that it is easy to distinguish apes within the video. Furthermore, the bounding box is displayed as red if the prediction for the ape on a particular frame is incorrect. Both the prediction and the true label is shown when this occurs. These modified frames are saved to disk. The individual frames for each video are stitched together using `cv2.VideoWriter`. Examples of the final output can be seen in Figure 3.7. It is important to note that when evaluating the test set, the bounding box coordinates can be drawn on as they already exist in the dataset's annotations. To evaluate any data outside of this project's dataset, the videos must be annotated with the coordinates in the same format, as shown in Appendix B.

A normalised confusion matrix is also produced, showing the performance of the model across every class at a glance. It provides a visual way of understanding where the model performs best and where it does not. The values of the confusion matrix is computed using `scikit-learn`'s `metrics` package, passing the predictions and labels as arguments to the `confusion_matrix` function. Python's visualisation library `matplotlib` is used to initialise a grid and populate it with the values from the matrix.

The final output of the videos and the confusion matrix are stored in the given output directory, on the system that the model was evaluated on. There may be cases where the model is executed on a remote system due to computation requirements, similar to the use of BlueCrystal4 [71] for this project. For automating the process of acquiring and viewing the output from the remote system, the cloud storage service Amazon S3 from Amazon Web Services [60] (AWS) is utilised. Amazon S3 is made up of "buckets" in which data can be stored and retrieved from. Given that the user has the required AWS credentials configured on their system, and that they provide the name of their S3 bucket in `config.json`, `test.py` attempts to upload the output to the user's specified S3 bucket. Before uploading, the output is compressed using Python's `ZipFile` package. The library `boto3` is used to interface with the S3 bucket as a client. Once uploaded, `boto3's` function `generate_presigned_url()` is used to obtain a download link for the compressed output. This link is returned to the user on the command line interface. This allows for easy access to the output and preserves it in cloud storage for future use.

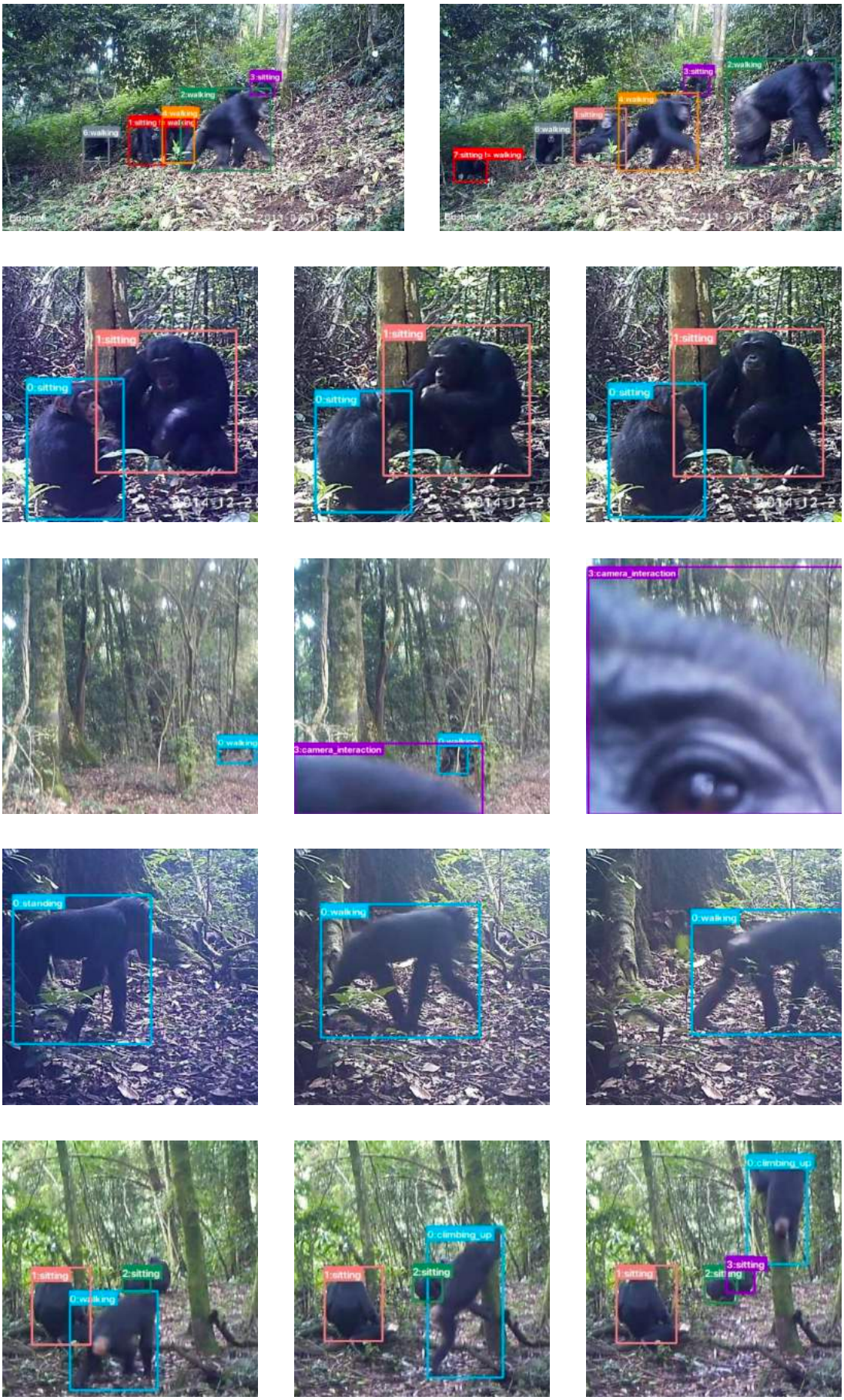


Figure 3.7: **Final output** consisting of drawn bounding boxes with behaviour classification. The ape's ID and behaviour is shown on the top left corner of each box. A red box signals an incorrect prediction made by the model. Each row represents a sequence progressing from left to right.

3.7 Experimentation: Tackling Class Imbalance

This section aims to document the works of the project following the formation of the initial two-stream great ape behaviour recognition model.

When evaluating the number of samples used in training shown in Section 3.3.1, it can be seen that there is a disproportional spread of samples between the classes. This shows that the dataset suffers from class imbalance. Based on its known effects, it would be a reasonable hypothesis to claim that the classes which are made up of a higher number of samples (majority classes) are better represented and will therefore perform better than classes which have a relatively lower number of samples (minority classes).

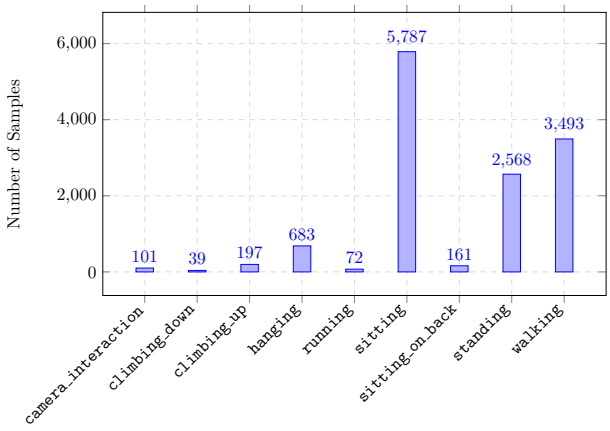


Figure 3.8: **Class imbalance**: A bar chart showing the number of samples available in the training set for each class. The majority classes `sitting`, `standing` and `walking` are better represented than the rest of the classes by a significant margin. This indicates the presence, and the level, of class imbalance in the data.

The validity of this hypothesis is demonstrated by the performance of the two-stream behaviour recognition model. The confusion matrix displayed in Figure 3.9 is made up of the predictions the model makes on the test set. The figure shows that there are a few classes that clearly dominate the behaviours that are predicted. They include the behaviour classes `sitting` and `walking`. Having the highest number of samples, they also have the highest class accuracies, with 74% and 89% respectively. Minority classes suffer from low accuracy. Predictions in such cases are instead made in favour of incorrect majority classes. This shows the severity of the impact caused by the data's class imbalance. As such, the next primary focus is to tackle this problem in an attempt to improve the current model.

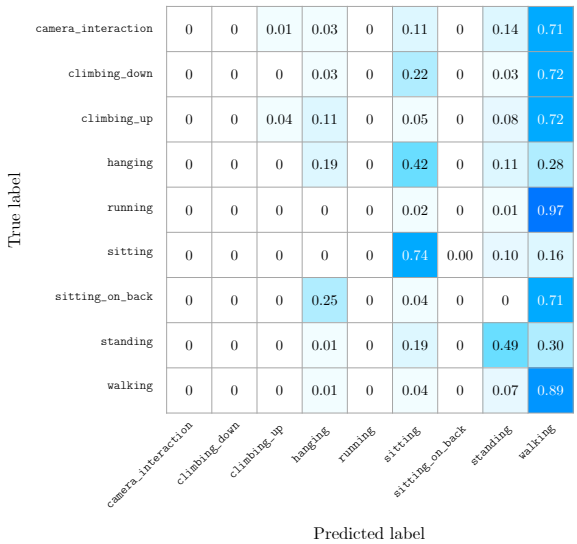


Figure 3.9: **Effects of class imbalance**: A confusion matrix produced from the model's predictions on the test split of the dataset. The diagonal represents the accuracies for each class. The matrix shows that minority classes suffer heavily in performance, where most predictions are incorrectly asserted as `walking`.

3.7.1 Balanced Sampling

Balanced sampling is a method where every training batch that is produced consists of an equal number of samples for each class. This is achieved by oversampling the minority classes, as discussed previously in Section 2.11.1. By exposing the model to every class equally, the aim is that the model's predictions is not biased towards certain classes.

To implement the sampling strategy, the class `BalancedBatchSampler` is created in `sampler.py`, inheriting PyTorch's `Sampler` class. This allows `BalancedBatchSampler` to be integrated with and used by the `GreatApeDataset DataLoader`. In the sampler's initialisation function, `GreatApeDataset` is iterated through, aggregating the indices for each class to make up its own `dataset`. It keeps track of `max_class_samples`, which is the highest number of samples for any one class. This is used as the target for which all other classes are oversampled to.

```
class BalancedBatchSampler(torch.utils.data.sampler.Sampler):
    def __init__(self, dataset, labels):
        self.labels = labels
        self.dataset = dict()
        self.max_class_samples = 0

        # Save every index for every class
        for index in range(0, len(dataset)):
            label = self._get_label(dataset, index)
            if label not in self.dataset:
                self.dataset[label] = list()
            self.dataset[label].append(index)

        # Find highest number of samples for a class
        for label_class in self.dataset.keys():
            if len(self.dataset[label_class]) > self.max_class_samples:
                self.max_class_samples = len(self.dataset[label_class])

        # Oversample the classes with fewer elements than max_class_samples
        for label in self.dataset:
        while len(self.dataset[label]) < self.max_class_samples:
            self.dataset[label].append(random.choice(self.dataset[label]))

        self.keys = list(self.dataset.keys())
        self.currentkey = 0
        self.indices = [-1]*len(self.keys)
```

Listing 3: **Initialisation of** `BalancedBatchSampler`: It obtains a representation of all the samples by its class. This is then used to oversample the classes as required. This ensures that every batch can be made up of an equal number of samples from each class despite the existing class imbalance.

The oversampling is achieved by iterating through each class of the `BalancedBatchSampler`'s `dataset`. A `while` loop is used to populate each class up to a number of `max_class_samples` samples, by duplicating samples from the same class. The sample to be duplicated is chosen at random with the use of `random.choice(dataset[label])`. The `__iter__()` function keeps track of the number of times each class has been sampled. This number is also used as the current index of each class, allowing the sampler to access and iterate through the samples. The indices are reset when every class has been sampled `max_class_samples` times, signalling the end of an epoch.

Following the implementation of `BalancedBatchSampler`, the batch size is increased from 32 to 36. This ensures that every batch is made up of exactly 4 samples of each of the 9 classes. The `max_class_samples` for the model, using the data sampling parameters set as stated in Section 3.3.1, would be derived from the `sitting` behaviour class with 5731 samples. With balanced sampling, every class consists of the same number of samples as `sitting`. Epochs no longer describe one pass of training across the real dataset, but instead for this balanced dataset that has been synthesised. This naturally results in much longer epochs, where, as explained earlier, the model trains on many duplicate samples.

3.7.2 Loss Functions

Weighted Cross-Entropy

The PyTorch implementation `torch.nn.CrossEntropyLoss()` takes a parameter `weights` as a 1D `Tensor`. By providing it with weights for each class, it acts as a weighted cross-entropy loss function. The weights are calculated as shown in Section 2.3.3. As a result, the loss for the behaviour class `sitting` is scaled to 1 as it has the higher number of classes. Loss of all other classes use a coefficient greater than 1.

Focal Loss

The focal loss [29] is used as an alternate to the cross entropy loss function to address the issue of class imbalance.

A PyTorch implementation provided by the computer vision library Kornia [7] is used. It inherits the class `nn.Module` with the functions `__init__()` and `forward()`, making it compatible for use with the two-stream model through PyTorch. When initialised, the focal loss takes in the parameters α and γ. The focal loss and its parameters can be specified for use through `config.json`.

The loss is computed in the function `forward()`. It takes in two arguments: `input` and `target`. The argument `input` is the output of logits produced by the forward pass of the model. The `target` is the true behaviour label of the sample. The focal loss is essentially an extension of the cross entropy loss. So, the cross entropy $log(p_t)$ is obtained first, using PyTorch's `log_softmax(input)`. The focal loss is then computed as shown in Equation 2.5 in Section 2.3.3.

3.7.3 Data Augmentation

Data augmentation is implemented to help synthetically increase the size of the dataset. PyTorch's computer vision library `torchvision` comes with a number of functions in `transforms` that can be applied to image data. The augmentations applied to the RGB spatial data include `ColorJitter` (changes the brightness, contrast, hue and saturation), `RandomHorizontalFlip`, and `RandomRotation`. Each augmentation is set to have an independent probability of 0.5 to be applied. Therefore, any spatial sample may have all, some or none of the augmentations applied.

Meanwhile, the temporal data is made up of optical flow images. Their pixel values have flow information encoded in them, representing the intensity and direction of motion. For example, rotating the image would render the values incorrect by some degree. Therefore, applying any image augmentation technique, including the ones used for the spatial data, alters the actual representation of flow within the image. Data augmentations on the optical flow images may have an undesirable impact on the temporal stream's learning, as different samples would have varying representations of flow depending on how the data has been modified. A possible approach to augment the temporal data would be to modify the original spatial RGB images across the temporal stack, and then compute their flow. However, the model aims to use pre-computed flow in training to minimise overhead. As such, the use of data augmentation is not considered for the temporal stream.

3.7.4 LSTM

LSTM networks are implemented as an alternative method of capturing temporality within the data, for both the spatial and temporal streams.

Two-Stream ResNet + LSTM

Given that LSTMs have high requirements of memory and computation due to the use of recurrent feedback loops to retain information, it is not feasible to process images as input. As such, the two-stream ResNet-based CNN architecture of the model remains to extract feature maps of reduced complexity from the data. Two LSTM networks are utilised, one for each stream. ResNet-18 networks are extract a feature map of size 512 for every time-step in the sequence for both types of data. Doing so, it provides the LSTMs with a latent representation of each video frame. It is worth noting that even with the use of a 1D feature map as input, the memory required to train the entire model is significantly increased with the introduction of the LSTM networks. To address memory allocation errors, batch size is reduced to 9. This also ensures that each batch still equally represents each class.

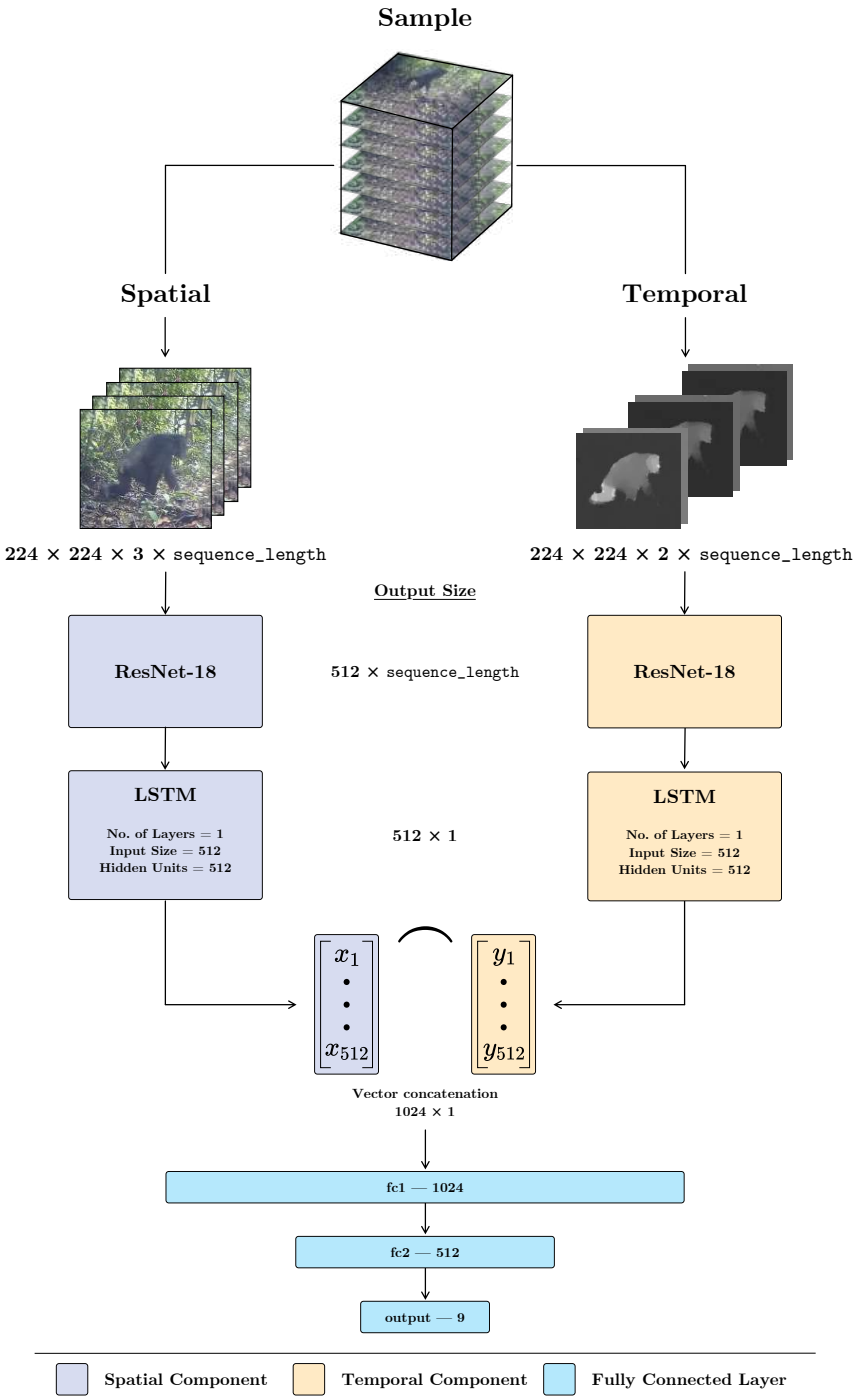


Figure 3.10: **Two-Stream ResNet-18 + LSTM great ape behaviour recognition model architecture**. The input for both streams now form a sequence. An LSTM network is appended to each of the ResNet-18 streams. The vector output from both LSTM networks are fused using concatenation.

The output from the ResNet networks are passed on to the corresponding spatial or temporal LSTM. The LSTM networks are initialised with PyTorch's `LSTM` constructor. The input size is 512 and the number of hidden units is 512. Only one LSTM layer is used. The LSTM networks each output a feature vector of size 512, which are concatenated to make a vector of size 1024. This is passed into a fully connected network of two layers. The first layer takes in input size 1024 and produces an output size 512. Finally, the second layer produces the classification vector of size 9.

Sequence Input

To train an LSTM, it must receive data in the correct form. The samples have to represent a sequence of data. This applies for both the spatial and temporal data. The size of the sequence `sequence_length`

corresponds to the number of consecutive frames that it is made up of. Spatial data samples consist of `sequence_length` consecutive RGB frames stacked together, making up a `Tensor` of size $H \times W \times 3 \times$ `sequence_length`. Unlike previously, the spatial stream is now exposed to several frames at a time, which may allow the model to learn temporal dependencies found solely in the appearance of great apes.

The temporal data is also sampled differently. Instead of using temporal stacks, where flow is fused across the 3rd dimension (channels), the goal is to represent the flow as a sequence. Recall each for each frame, there is a total of two optical flow images calculated, both in the horizontal and vertical direction. Therefore, for each frame of the video, the two flow images are stacked in the 3rd dimension forming a `Tensor` of size $H \times W \times 2$. Each of these stacked `Tensors` represent a time-step in the sequence. These optical flow stacks are then themselves stacked for a consecutive number of frames. This results in a `Tensor` of size $H \times W \times 2 \times$ `sequence_length`. Due to this, the temporal ResNet network requires an input channel size of 2 instead.

```
    def forward(self, spatial_data, temporal_data):
      # Spatial forward pass
      # Initialise hidden state and cell states
      h0 = torch.zeros(self.lstm_layers, spatial_data.size(0),
      self.hidden_size).requires_grad_().to(self.device)
      c0 = torch.zeros(self.lstm_layers, spatial_data.size(0),
      self.hidden_size).requires_grad_().to(self.device)

      # Reduce dimensionality of input data for ResNet-18
      batch_size, seq_length, c, h, w = spatial_data.shape
      spatial_data = spatial_data.view(batch_size * seq_length, c, h, w)
      spatial_out = self.spatial(spatial_data)

      # Revert data back to sequence for LSTM
      spatial_out = spatial_out.view(batch_size, seq_length, -1)
      spatial_out, (hn, cn) = self.lstm(spatial_out, (h0.detach(), c0.detach()))
      spatial_out = spatial_out[:, -1, :]

      # Do the same to perform the temporal forward pass...

      # Concatenate spatial and temporal features
      fused_out = torch.cat((spatial_out, temporal_out), dim=1)
      # Fully connected classifier
      fused_out = F.relu(fused_out)
      fused_out = self.fc1(fused_out)
      return self.fc2(fused_out)
```

Listing 4: **Forward pass for the ResNet + LSTM network:** The dimensions of the sequential input needs to be adjusted depending on which network is it passed to. Note that code for only the spatial stream is included for visual compactness.

Forward Pass

The forward pass includes the execution of both spatial and temporal streams, before propagating through the fully connected classifier. Firstly, the cell state `c0` and the hidden state `h0` for the two LSTM networks are initialised. They are `Tensors` of size `lstm_layers` $\times$ `batch_size` $\times$ `hidden_size` filled with zeroes. The cell state is the memory of the LSTM cell, whereas the hidden state is the output of the cell. These states are used by the LSTM network to carry information over time-steps.

In order to obtain the feature map of the data, it is passed through the ResNet-18 network. However, the initial input data has a size of $H \times W \times C \times$ `batch_size` $\times$ `sequence_length`. ResNet-18 requires a 4D `Tensor` as its input. To address this, the data is restructured, flattening the `Tensor` across the dimensions `batch_size` and `sequence_length` using `torch.view()`. This modifies the data into being 4D, with each sequence representing one sample within the batch, where the ResNet instead sees a batch size of `batch_size` $\times$ `sequence_length`. The feature map of the data can then be computed through ResNet-18.

The output is then returned to its original 5D structure in preparation for the LSTM networks.

The LSTM takes the feature map and states `h0` and `c0` as arguments. The states are detached so that truncated backpropagation through time (BPTT) applies. This ensures that the network does not backpropagate all the way to start of training. Instead it backpropagates only as far as the start of the batch. For the purposes of classification, only the hidden state of the last time-step is required from the output. As the whole output is in the form of `batch_size` $\times$ `sequence_length` $\times$ `hidden_size`, it is obtained with `output[:,-1,:]`. The hidden state output for both LSTM networks are concatenated, before being passed through the fully connected classifier to obtain the final output.

Hyperparameters

With the addition of the LSTM networks, the behaviour recognition model was found to perform optimally with the change of certain hyperparameters. The focal loss is used, with parameters $\alpha = 1.00$ and $\gamma = 1.00$. The learning rate is set to 0.0001. L_2 regularisation of 0.01 is used. Balanced sampling is also introduced. To gain benefit from the LSTM networks, the model is exposed to longer sequences of input. As such, the `sequence_length`, previously named `temporal_stack`, is set to 20, along with `sample_interval`.

Chapter 4

Results & Evaluation

This chapter presents the results of the great ape behaviour recognition model. Its performance is evaluated across multiple model implementations and configurations. The chapter aims to express how motivations, formed by the evaluation of the results, helped to guide the direction of this project. The chapter concludes with the discovery of the optimal model, followed by its further analysis and reporting of the final results.

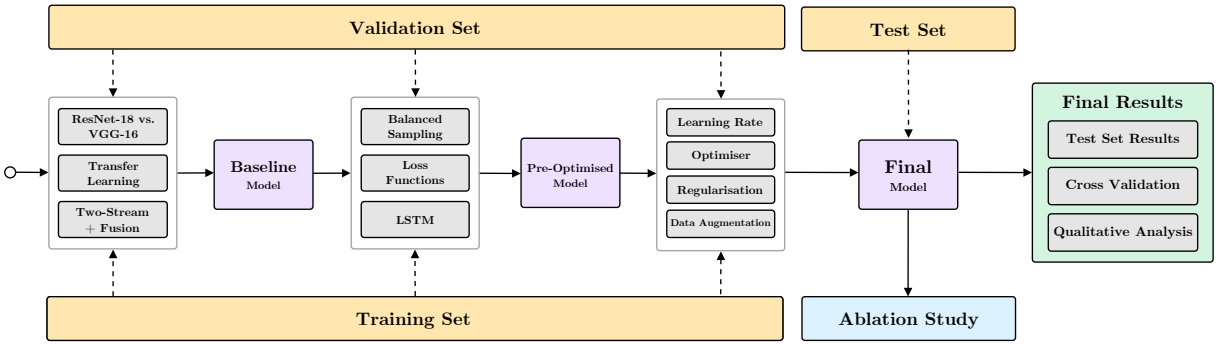


Figure 4.1: **Evaluation plan**: A flow chart visualising the the various model implementations which are explored towards the discovery of the final model and its results. The orange boxes resemble the splits of the dataset used for obtaining results. The purple boxes indicate model implementations that were definitive over the duration of the project.

4.1 Initial Results

The following iterations of experimentation helped towards establishing the initial "baseline" version of the great ape behaviour recognition model. Their results were used to gain an understanding of the fundamental components involved in the model, and their impact on performance. All results of models prior to the final model are based solely on their evaluation of the validation set.

The main components of the two-stream behaviour recognition model, the spatial CNN and the temporal CNN, were initially both implemented separately. Each of the network were independently analysed, based on their performance on their respective data. This was to ensure that neither networks would act as bottlenecks to the overall performance when integrated into the same model.

4.1.1 VGG-16 vs. ResNet-18

Two network architectures, VGG-16 [48] and Resnet-18 [12] were considered when forming the backbone of the two-stream model. After investigation into their performance training on spatial and temporal data, they both accomplished learning from the provided data to some extent. Apart from the spatial VGG-16, all other models obtain a Top1 accuracy of **40%** or higher. This, paired with their significantly high values of Top3 accuracy above **85%**, shows that the models are extracting relevant features towards making good inferences of great ape behaviour. However, every model suffers from a low class average

accuracy. This is an expected consequence of class imbalance, but these results show the high severity of impact caused by the data.

	Network	Top1	Top3	Class Average	Parameters
Spatial	VGG-16	29.82%	**92.11%**	12.12%	135.31M
	ResNet-18	**40.74%**	85.18%	**12.77%**	**11.18M**
Temporal	VGG-16	47.26%	**92.07%**	12.72%	135.32M
	ResNet-18	**49.57%**	88.35%	**15.18%**	**11.2M**

Table 4.1: **VGG-16 and ResNet-18 CNN architectures** incorporated into individual spatial and temporal streams. ResNet-18 performs better with the use of only ∼**8.7%** of the number of parameters used by VGG-16. **Yellow rows indicate optimal configuration**.

The results indicate that ResNet-18 is more favourable out of the two networks. Its Top1 and class average accuracies are higher than VGG-16. The spatial ResNet-18 has a Top1 accuracy which is **11%** higher than when compared to VGG-16. This may have been a result of the greater depth and residual characteristics of ResNet-18 [12], allowing it to perform better in the specific application of great ape behaviour recognition. The number of parameters used by ResNet-18 was only ∼**8.7%** of what was used by VGG-16, saving ∼630MB of memory. This correctly reflects the much lower requirement of computation from ResNet-18 compared to VGG-16 (1.8B vs. 19.6B FLOPs) as discussed in Section 3.2.1. ResNet-18's Top3 accuracy is **85.18%** for spatial and **88.35%** for temporal, compared to **92.11%** and **92.07%** for VGG-16. Although ResNet-18 falls notably short in this case, its Top3 accuracy is still considered to be at a high level, while also maintaining good performance margins over VGG-16 in other areas.

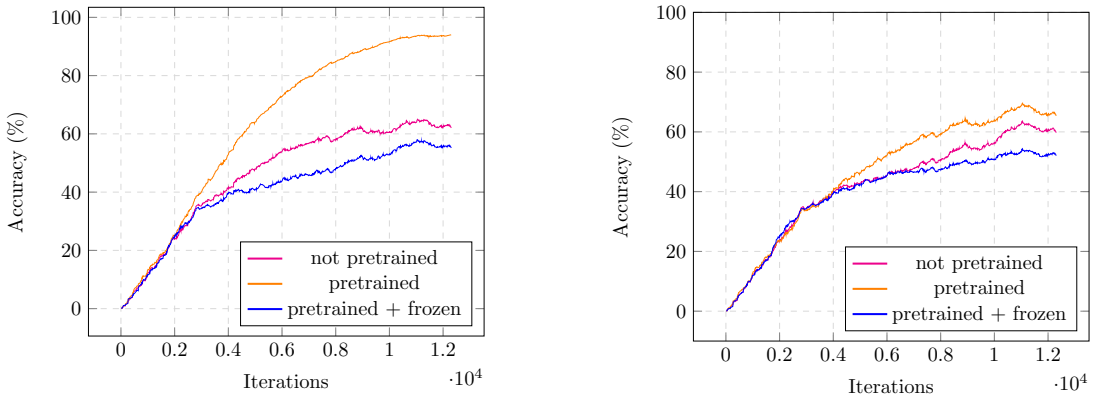
When evaluating across the two streams, a clear discrepancy can be seen in performance. This provides an insight into the difference of effectiveness between training the CNNs on still RGB frames and stacks of optical flow frames. The temporal stream, consistent between both types of networks, produces better results overall. Although this shows that optical flow leads to better performance for behaviour recognition, it is important to note that the temporal stream is exposed to multiple consecutive frames at a time. Meanwhile, a single frame is presented to the spatial stream. Therefore, it must be considered that the temporal stream's performance may in fact also be positively influenced by the higher volume of data, and not just the nature of the data, that it was trained on.

4.1.2 Transfer Learning

Models for both ResNet-18 streams were modified to make use of pre-trained weights. The impact of transfer learning on the task of great ape behaviour was examined. The first implementation used the pre-trained weights and continued fine-tuning on the dataset as normal, updating all of the model's parameters. The second type of model utilised feature extraction. The layers of the pre-trained ResNet-18 component were frozen, appended with a two layer fully connected classifier for deriving predictions.

The findings show that transfer learning is highly beneficial, specifically when fine-tuning is performed. The training curves in Figure 2.5 show that the fine-tuning model converges to a much greater accuracy than the non-pre-trained model. Meanwhile, in both cases, the pre-trained model with frozen layers for feature extraction performs worse than the non-pre-trained model. It began training with a similar rate of increase in accuracy, but fails to improve after some time. This shows that the domain of this problem deviates from the target problem of the pre-trained weights, ImageNet [6]. The network needs to backpropagate through the ResNet-18 CNN and update its parameters to better learn from the dataset. As such, fine-tuning leads to better performance as it achieves the balance of making use of the pre-trained features as a foundation before improving itself specifically towards this task.

The difference in impact of the pre-trained weights between the two streams highlights the importance of the target task when attempting transfer learning. It can be seen in Figure 4.2 that the performance gains of the pre-trained spatial model is much higher than that of the pre-trained temporal model. This supports the claim made in Chapter 3.2.1, stating that the temporal stream may benefit less from pretraining than the spatial stream. This is due to the differences of optical flow compared to the data found in ImageNet [6]. Though, the pretraining of the temporal stream improves its performance to some extent. This is in agreement with the findings of Yosinski et al. [57], stating that the "transferability of features decreases as the distance between the base task and target task increases" and that the use of pretraining proves to be better than random initialisation of weights, regardless of dissimilarity of the



(a) Spatial

(b) Temporal

Figure 4.2: **Effects of transfer learning**: Top1 accuracy curves displaying the impact of pretraining on both streams. The impact of pretraining on the spatial stream appear to be much greater than on the temporal stream. This corresponds to the fact that the spatial stream is exposed to data that is much more similar to that of ImageNet [6], which is what the pre-trained weights were trained on. Values have been smoothed for visual purposes.

target task. The method of fine-tuning with pre-trained weights was proven to be greatly beneficial and would be used for proceeding implementations.

	Model	Top1	Top3	Class Average
Spatial	Not Pre-trained	35.84%	85.18%	13.53%
	Pre-trained	**49.82%**	86.25%	**16.90%**
	Pre-trained + Frozen	39.53%	**90.45%**	12.73%
Temporal	Not Pre-trained	49.16%	91.93%	13.87%
	Pre-trained	**55.26%**	90.83%	**17.69%**
	Pre-trained + Frozen	54.12%	**92.11%**	17.46%
+ ResNet-18				

Table 4.2: **Transfer learning**: The use of pre-trained weights for fine-tuning (pre-trained) and feature extraction (pre-trained + frozen) is considered. Fine-tuning results in performance gains, especilly for the spatial stream.

4.1.3 Fusion

The next step was to fuse the spatial and temporal streams together to form a two-stream behaviour recognition model. Results were obtained for the model utilising two different approaches; late fusion and convolutional fusion. They were compared to the performances of the individual pre-trained CNNs.

Model	Top1	Top3	Class Average
Spatial	49.82%	86.25%	16.90%
Temporal	55.26%	**90.83%**	17.69%
Late Fusion	57.39%	86.21%	18.47%
Convolutional Fusion	**61.39%**	86.34%	**21.60%**
+ fine-tuning pre-trained (spatial and temporal) ResNet-18			

Table 4.3: **Two-stream fusion**: Both methods of fusion perform better than individual streams. Convolutional fusion results in the best model so far with a Top1 accuracy of **61.39%**.

The advantage of fusing the two streams together to comprehensively capture the spatio-temporal dependencies can be seen in the results. Both average and convolutional fusion show improvement compared to the individual streams, with increased Top1 and class average accuracies. They obtain Top1 accuracies of **57.39%** and **61.39%** respectively. The class average accuracy, while increased, remains at a fairly low level. The disproportionate increase of the Top1 accuracy compared to the class average accuracy shows that fusion had allowed the model to strengthen particularly on the majority classes, but not on the minority classes. However, there was some decrease in Top3 accuracy with the introduction of fusion, especially comparing to the temporal stream. This indicates that a subset of the data of certain behaviours may be better represented by optical flow only.

Convolutional fusion performs better than late fusion. Although late fusion takes both streams into account, it does so with a simple average of classification scores at the end. Convolutional fusion captures

the dependencies of both streams by performing 3D convolution over spatial and temporal feature maps, followed by a fully connected classifier. This allows the model to cross-pollinate information between the streams at an earlier stage. The loss is calculated for the entire model, being minimised taking into account both streams together. These crucial differences in the approach of convolutional fusion reflect its superior performance in comparison to average fusion. These results are consistent with the outcome of the original paper experimenting with two-stream fusion [9].

4.2 Baseline Model

As a result of the findings so far, the two-stream approach using convolutional fusion, with pre-trained ResNet-18 networks that are fine-tuned, is adopted as the baseline model. It had the best overall performance among the initial models and acts as a good foundation to improve upon. Table 4.4 summarises the configuration and hyperparameters involved for reproducing the baseline implementation of the two-stream great ape behaviour recognition model, along with its results from evaluating the validation set. However, the model still has key weaknesses that need to be addressed.

Initial Great Ape Behaviour Recognition Model		
Model	Architecture	Two-Stream [47]
	CNN	ResNet-18
	Pre-trained	Yes (both streams)
	Fusion	Convolutional
	Optimisation	SGD with Momentum = 0.9
	Loss	Cross Entropy
	Learning Rate	0.001
	Batch Size	32
Data Sampling	`temporal_stack`	5
	`sample_interval`	10
	`behaviour_duration_threshold`	72

Validation Set Results		
Top1	Top3	Class Average
61.39%	**86.34%**	**21.60%**

Table 4.4: **Baseline model summary**: Table of model configurations and hyperparameters of the baseline behaviour recognition model (top). The model's results for the validation set is also shown (bottom).

The class average accuracy is disproportionately low when compared to the other metrics. The model's good performance for majority classes is the main driving force of the Top1 accuracy, but its poor performance for minority classes results in the low class average accuracy. This trend is common for both the training data and the validation data, due to a property that they share: class imbalance. This is supported by the number of samples that belong to each class, previously shown in Table 3.1. Moreover, the distribution itself is similar across each set of data, in the sense that they have the same majority classes and minority classes between them. This is an indication that some behaviours of great apes occur much more commonly than others (e.g. many more occurrences of walking than climbing). As such, it can be said that the class imbalance may be a representation of how great apes behave. However, this observation is based on the assumption that this dataset is fully representative of great apes. A number of factors such as trap camera location, time of recording, among others, may lead to different findings upon further data gathered.

Examining the confusion matrix of the model's evaluation gives a better understanding of how it is making predictions. The three highest class accuracies belong to `standing`, `sitting` and `walking`, at **49%**, **67%** and **82%** respectively. These are the same classes which, by some margin, have the highest number of samples among the nine behaviours. The other six classes are considered minority classes, of which only `climbing_up` has a non-zero accuracy. Thus, it is fair to say that the nature of the class imbalance in the dataset resembles the predictions that are made by the model.

On further inspection, the matrix shows that minority classes heavily tend to be mistaken for the majority classes. The extremely high occurrence of `sitting`, `standing` and `walking` samples compared to other classes means that they dominate the learning, and consequentially, the predictions made by the model. It can be further said that minority classes are usually labelled as a majority class behaviour that

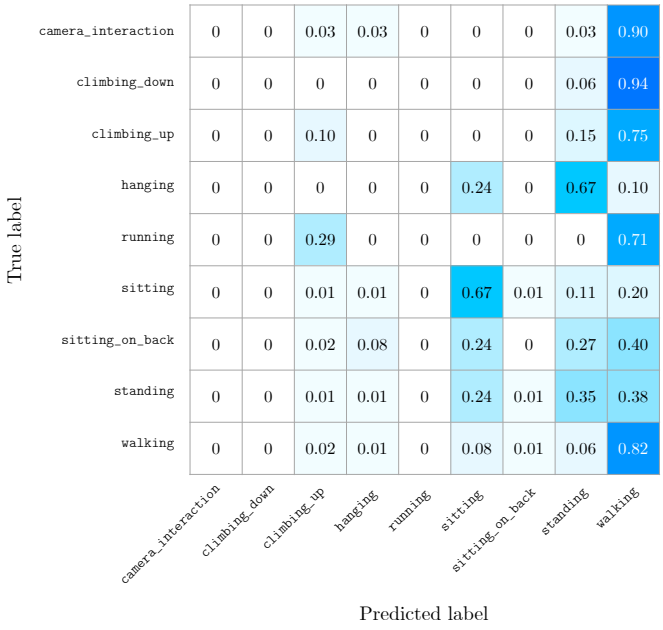


Figure 4.3: **The confusion matrix of baseline model's evaluation of the validation set.** Note that the diagonal of the matrix resembles the accuracies for each class. The weak colours of the diagonal show that the model is in most cases unable to predict correctly. It can be seen that the majority of predictions are wrongly classified as `walking`.

has a similar motion or appearance. Most samples of classes `camera_interaction`, `climbing_down`, `climbing_up` and `running` are incorrectly predicted as `walking`. These minority classes have aspects of motion in their behaviours, leading the model to label them as the only majority class which involves motion, `walking`. Likewise, `hanging`, representing a mostly stationary behaviour, is often incorrectly predicted as the majority classes, `sitting` and `standing` which also display a similar lack of movement. It was anticipated that the climbing behaviours would be well predicted as they specifically involve vertical motion, much different to the other classes. Failure to do so indicates that class imbalance may be causing negative impact to the training of the model.

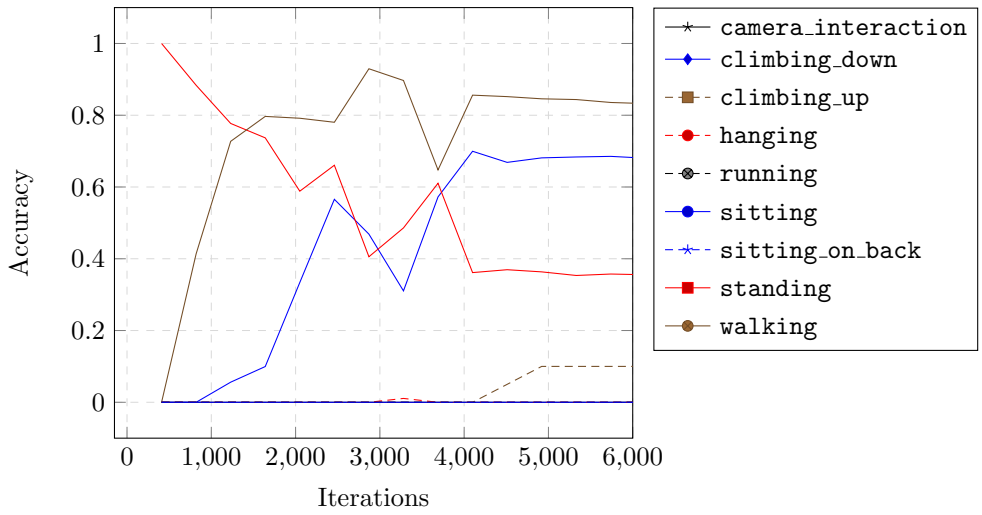


Figure 4.4: **Individual class accuracy for the validation set** plotted over the duration of training. Apart from the three majority classes, there is sign of minimal performance throughout training.

The performance of the model for each class on the validation set during training can be seen in Figure 4.4. It shows that multiple classes do not obtain an accuracy of more than **0%** at any point over the course of training. Along with the confusion matrix, this means the model is heavily biased towards majority classes. This graph particularly shows, that the model's disproportionately large exposure to the majority classes, such as `walking` and `sitting`, is causing it to suppress the learning of the other

classes. While it trains on many instances of some classes, there is not much opportunity for the model to optimise its loss function for the remaining minority classes.

In contrast, the model's weaknesses as described above can be seen as good potential. As determined from the confusion matrix, its proneness to incorrectly classify minority classes as majority classes that are similar, shows that the model is, to an extent, successfully capturing spatio-temporal context found in the data. This claim is also supported by the high Top3 accuracies that have been achieved so far. The model is doing well to narrow down the possible behaviours in terms of their spatial and temporal properties. However, the final prediction is being driven towards the majority classes that it has had a much higher level of exposure to. As such, the next step is to address the issue of class imbalance in an attempt to improve the baseline model.

4.3 Balanced Sampling

The first measure taken to address class imbalance was to introduce balanced sampling. With the model being trained on an equal number of samples per class, it was expected that its training would be more representative of all nine classes, resulting in predictions less dominated by the majority classes.

The results show that the use of balanced sampling improves the performance of the model in all aspects. This is true for the two-stream model, and when evaluating each stream independently. The outcome is that the two-stream model achieves its best metrics so far, with a Top1 accuracy of **68.77%**, Top3 accuracy of **93.17%** and a class average accuracy of **28.28%**.

Model	Sampling	Top1	Top3	Class Average
Spatial	Unbalanced	49.82%	86.25%	16.90%
	Balanced	**63.63%**	**89.87%**	**37.35%**
Temporal	Unbalanced	55.26%	90.83%	17.69%
	Balanced	**64.36%**	**91.73%**	**29.46%**
Two-Stream	Unbalanced	61.39%	86.34%	21.60%
	Balanced	**68.77%**	**93.17%**	**28.28%**
+ baseline two-stream ResNet-18				

Table 4.5: **Balanced sampling**: Notable increase in performance is seen for the individual streams when compared to normal unbalanced sampling. The two-stream model obtains a Top1 accuracy of **68.77%** and a class average accuracy of **28.28%**.

When using regular sampling, there is a total of 5 classes that suffer from having **0%** accuracy. However, balanced sampling reduces this to 3 classes. Although `climbing_up` loses all accuracy from **10%** to **0%**, the classes `camera_interaction`, `hanging` and `sitting_on_back` gain significant accuracy from being unidentifiable before. This proves that balanced sampling can, to some extent, counteract against the suppression of majority classes as seen in earlier models. The model now has equal amount of opportunity to train on the minority classes, which translates to more correct predictions of those classes. Accuracy of `sitting` and `walking` display considerable improvements too, showing that majority classes also benefit from this method of sampling. Balanced sampling is wholly beneficial for the model, and as such, is incorporated in all following implementations.

Sampling	camera_interaction	climbing_down	climbing_up	hanging	running	sitting	sitting_on_back	standing	walking
Unbalanced	0.00%	0.00%	**10.00%**	0.00%	0.00%	67.11%	0.00%	**35.14%**	82.12%
Balanced	**23.33%**	0.00%	0.00%	**21.51%**	0.00%	**79.00%**	**9.52%**	30.92%	**90.19%**

Table 4.6: **Unbalanced vs. balanced sampling class accuracies**: The model performs favourably in a number of both minority and majority classes.

Admittedly, the problem of class imbalance has not been fully resolved with the use of balanced sampling. The class average accuracy remains relatively low, derivative of the unsatisfactory performance of the minority classes. It is important to note that the minority classes are being inflated via duplication. In essence, the class imbalance found in the dataset is being addressed indirectly, as the problem fundamentally remains. The model's training on samples, repeated many times, can lead to its overfitting of the minority classes. This means the model may be unable to identify instances of minority class behaviours that deviate from the ones seen in training. With this in consideration, it is understandable that the minority classes do not yet achieve desirable levels of performance. It would also be worth examining the effectiveness of techniques that reduce overfitting.

4.4 Loss Functions

Two loss functions, the weighted cross entropy [26] and the focal loss [29] were used to assess their ability to address the class imbalance of the dataset. They were implemented as part of the two-stream model using balanced sampling.

Both loss functions fulfil their function of reducing the effects of class imbalance, producing an increase in the class average accuracy. Focal loss, for every configuration used, results in an improvement in this metric. However, this comes at the expense of the overall Top1 accuracy.

Particularly for this dataset, since the class imbalance is so high, the weighted cross entropy suffers due to its own approach. It uses the technique of inversely weighting the classes according to their frequencies. Since the weighting given to the minority classes is accordingly very high, this leads to a situation where the majority classes are overwhelmed by the inversed importance given to the minority classes towards the loss. The significant decrease in performance for majority classes `sitting` and `walking`, while improving in 4 minority classes, reinforces this. As a result, the weighted cross entropy loss, while improving the class average accuracy, loses ∼**17%** of Top1 accuracy. The weighted cross entropy loss function is more likely to be suited to instances where the class imbalance is not as extreme.

Loss	α	γ	Top1	Top3	Class Average
Cross Entropy	-	-	**68.77%**	93.17%	28.28%
Weighted Cross Entropy	-	-	51.64%	92.24%	31.42%
Focal	0.50	0.50	62.46%	91.55%	29.33%
	0.25	1.00	65.29%	91.14%	28.98%
	0.25	2.00	62.84%	89.93%	32.58%
	0.25	5.00	62.87%	93.86%	33.91%
	1.00	1.00	67.32%	**94.90%**	**39.08%**
	1.00	2.00	67.87%	91.52%	33.68%
	1.00	5.00	64.39%	93.04%	34.86%
+ baseline two-stream ResNet-18 + balanced sampling					

Table 4.7: **Weighted cross entropy and focal loss**: The alternative loss functions are able to improve on the class average accuracy. Focal loss with parameters $\alpha = 1.00$ and $\gamma = 1.00$ performs best, with a class average accuracy of **39.08%**.

Unlike the weighted cross entropy loss, the focal loss is able to down-weigh the importance of classes according to how well they have been learned [29]. The first four results of the focal loss seen in Table 4.7 use recommended values of α and γ suggested in the original paper [29], in which $\alpha = 0.25$ and $\gamma = 2.00$ is claimed to be optimal. These configurations mostly show decline in Top1 and Top3 accuracies. However, it can be seen that for $\alpha = 0.25$, higher values of $\gamma = 2.00, 5.00$ produces a higher class average accuracy than the weighted cross entropy loss, at **32.58%** and **33.91%** respectively. Therefore, it is successfully suppressing the learning of majority classes in favour of less well-classified minority classes, similar to the weighted cross entropy.

It would be justified to use an increased α constant to consequently increase the loss value produced across all classes. This would help alleviate some of the aggressiveness of the down-weighting of majority classes. Values of 0.50 and 1.00 are used for α with the same values of γ. The higher Top1 accuracies, while also gaining class average accuracies indicate that the higher α values help to achieve what is desired. Note that $\gamma = 5.00$ still shows poor performance in Top1 accuracy. This shows that this value of γ remains too aggressive, where even middle-level well-classified samples ($0.4 < p < 0.6$) are contributing very little to the loss. Therefore, it can be said that the class imbalance does not lead to a case where there are only very well-classified samples for majority classes, and the opposite for minority classes. Instead the model's performance falls across a spectrum for the classes, where there is correlation, to some level, of performance of a class with its frequency.

The use of $\alpha = 1.00$ and $\gamma = 1.00$ produces the best Top3 accuracy and class average accuracy with focal loss. It does so while retaining a similar Top1 accuracy of **67.32%** when compared to the use of cross entropy loss. The parameters for focal loss provide a good balance between the importance given to poorly classified minority samples, and the well classified majority samples.

The breakdown of class accuracies clearly shows that focal loss, and weighted cross entropy, is able to give importance to minority classes. Both loss functions are yielded non-zero accuracies for 8 of the 9 classes, compared to 6 out of 9 classes when using the normal cross entropy loss. Focal loss is able to make significant gains in performance for minority classes such as `camera_interaction` and `sitting_on_back`.

Sampling	camera_interaction	climbing_down	climbing_up	hanging	running	sitting	sitting_on_back	standing	walking
Cross Entropy	23.33%	0.00%	0.00%	**21.51%**	0.00%	**79.00%**	9.52%	30.92%	**90.19%**
Weighted Cross Entropy	30.00%	0.00%	**5.00%**	11.83%	**28.57%**	40.25%	41.27%	**45.78%**	80.08%
Focal	**50.00%**	0.00%	**5.00%**	19.35%	**28.57%**	71.21%	**52.38%**	36.14%	89.07%

Table 4.8: **Class accuracies of loss functions**: Focal loss is able to improve on minority classes, while mostly retaining performance, unlike the weighted cross entropy loss.

Although there is some decrease in its accuracy for the majority classes, it is not to the extent which weighted cross entropy suffers in this aspect. This is due to the focal loss' ability to adapt importance given to classes based on how well they are classified, in contrast to the weighted cross entropy's static approach. The results of the class accuracies shows the need for thoroughly testing the parameters for the focal loss as a slight difference in value can significantly impact the overall performance.

The results provide good motivation to use focal loss as part of an attempt to reduce the effects of class imbalance on the model's training. Accordingly, it is used with parameters $\alpha = 1.00$ and $\gamma = 1.00$ in the following implementations.

4.5 LSTM

LSTM networks were introduced to the model as an alternative method in capturing temporality found in the data. Two approaches were implemented and experimented with. The first approach used just one LSTM network. Here, the output of the ResNet-18 streams were stacked, which was then passed into the LSTM followed by a fully connected classifier. In addition, a method involving two LSTM networks is used, where there was one LSTM network at the end of each ResNet-18 stream. The output of each LSTM networks were concatenated, and passed in as input to a fully connected classifier.

LSTM Parameters			Top1	Top3	Class Average
Components	Layers	Dropout			
1 (pre-LSTM fusion)	1	0	55.84%	83.13%	17.80%
2	1	0	65.63%	93.93%	**36.49%**
2	2	0	**66.63%**	89.38%	28.43%
2	2	0.2	64.32%	92.86%	31.31%
2	2	0.5	17.17%	60.53%	10.66%
+ baseline two-stream ResNet-18 + balanced sampling + focal loss					

Table 4.9: **Two-stream model with LSTM networks**: The use of 1 LSTM network signifies the fusion of the two streams occurring before being passed through to the LSTM. When 2 LSTMs are used, there is one at the end of each ResNet-18 CNN. The output from both LSTM networks are fused by concatenation before being passed to a fully connected classifier. 2 LSTM networks with 1 layer performs most similar to the two-stream model without LSTM.

The use of 1 LSTM network yields poor results. It is unable to capture spatio-temporal dependencies at a level comparable to preceding models. One possible cause for this is the LSTM's input. The stacked output from the two ResNet-18 streams alternate between spatial and temporal feature maps. This causes the data to lose its original sequence-like structure. In particular, the sequence now represents one frame of a video with two time-steps. Furthermore, the input for each time-step of the sequence is derived from a different source of data compared to the previous time-step (i.e. at time-step t, the LSTM sees a spatial feature map, at time-step $t + 1$, it sees a temporal feature map, and so on).

When 2 LSTM networks are used, they are both exposed to sequences specific to the nature of their stream. Consequently, the results show better performance. Using 1 LSTM layer results in better class-wise performance, while 2 LSTM layers is marginally better in Top1 Accuracy. With multiple LSTM layers, dropout is considered. With a probability of 0.5, the use of dropout is detrimental. Meanwhile a lower dropout of 0.2 is of the same nature, to a lesser effect. Deducing from this, the effectiveness of dropout within LSTM networks is questionable. The poor results may indicate that the dropout of levels similar to what is used for CNNs and fully connected networks, may not be applicable to LSTM networks. This finding corresponds with research that states the non-transferability of dropout as motivation for exploring alternative methods of regularisation for RNNs, extending to LSTM networks [2, 58].

Performance varies among the different models. The use of 2 LSTM components with 1 layer using no dropout has the most balanced performance out of the LSTM implementations, retaining good levels for

all three metrics. However, there is no sign that they outperform previous non-LSTM models.

4.5.1 Sequence Length

LSTM networks are characteristically encoders of sequence. As such, it is also worth considering the sequence itself. In this case, the main parameter of the sequence is its length, seen as the number of frames that it is made up of. So far, a temporal stack size of 5 has been used for two-stream non-LSTM networks. To allow for comparison, the results in Table 4.9 come from LSTM two-stream networks that also use a sequence length of 5. The use of sequences of such short duration currently contrasts with the ability in LSTMs to capture long-term dependencies. Therefore training the LSTM networks with longer sequences may give an better insight for how well they perform in this task.

The non-LSTM two-stream model is also trained with increased temporal stack sizes. It is important to note that for the non-LSTM model, a greater temporal stack only affects the size of the temporal stream's input (spatial stream only takes one frame at a time). For the LSTM model, the size of input for both streams is impacted by the change in sequence length. It is trained using 1 LSTM layer with no dropout.

Temporal Stack Size	Top1	Top3	Class Average
5	67.32%	**94.90%**	**39.08%**
10	68.77%	94.04%	37.03%
20	**70.89%**	92.87%	35.57%
30	67.49%	92.78%	28.22%
+ baseline two-stream ResNet-18 + balanced sampling + focal loss			

Sequence Length	Top1	Top3	Class Average
5	65.63%	93.93%	36.49%
10	70.32%	93.61%	42.38%
20	**74.82%**	**95.49%**	**49.52%**
30	74.04%	94.58%	43.65%
+ baseline two-stream ResNet-18 + balanced sampling + focal loss + LSTM			

Table 4.10: **Experimentation with input size**: Results of a longer temporal stack/sequence are compared for the two-stream model with and without LSTM. With a sequence length of 20, the LSTM outperforms all configurations of the non-LSTM model.

Results are obtained for sequence/temporal stack lengths of 10, 20 and 30 frames, in addition to the existing results for the length of 5 frames. Without an LSTM involved, the model's Top1 accuracy peaks at **70.89%** when a temporal stack size of 20 frames is used. Otherwise, the model fails to improve when a greater size of temporal stack is used. With a larger temporal stack, the data is being considered over longer periods. This is expected to help the model by giving it more opportunity to capture the temporal dependencies of a behaviour. Conversely, this leads to a great decrease in the volume of data the spatial stream uses for training. Recall that for every temporal stack, the final frame's RGB image is used as the spatial stream's input. So with a larger temporal stack, the sampling interval is implicitly increased, resulting in far fewer images seen by the spatial stream. Specifically for a temporal stack size of 20, the spatial stream is exposed to only $\frac{1}{4}$ of the samples compared to when using a stack size of 5.

In contrast, the LSTM two-stream model shows that it is able to leverage the increased volume of data, allowing the LSTMs to capture spatio-temporal context to an extent that it could not do with the constraint of a 5 frame sequence. This is apparent in the overall improvements in performance when an increased sequence length is used, especially for 10 and 20 frames. For a sequence length of 20 frames, the model achieves the highest levels in each metric in this project.. It is able to gain Top1 accuracy of **74.82%** along with the class average accuracy of **49.52%**, suggesting that it is not solely focusing more on the majority classes.

It is worth noting that a higher sequence length leads to a reduction in the overall number of samples available in the dataset, where multiple samples are now pooled into one. This means that there are fewer samples to evaluate. Hence, it is important to ensure that the improved metrics with longer sequences in question are not inflated and consequentially misleading. However, at a sequence length of 20 frames, each sample is only 0.83 seconds (20 frames / 24 FPS) long. Supported by observations during the labelling of the data, it can be said the vast majority, if not all, of the behaviours displayed by great apes

exceed this duration. Considering this, along with its notable performance gains, the use of LSTMs with a sequence length of 20 frames appears to be a viable direction to take in terms of model design.

This marks the end of results relating to the exploration in model implementation. The remainder of this chapter commits to the best performing model discovered, for further optimisation and experimentation. The model in consideration is the two-stream LSTM model utilising focal loss and balanced sampling.

Pre-Optimisation Great Ape Behaviour Recognition Model		
Model	Architecture	Two-Stream [47] w/ LSTM
	CNN	ResNet-18
	Pre-trained	Yes (both streams)
	LSTM	Both streams. 1 layer with 512 hidden units
	Fusion	LSTM output concatenation
	Optimisation	SGD with Momentum = 0.9
	Loss	Focal Loss ($\alpha = 1.00, \gamma = 1.00$)
	Learning Rate	0.001
	Batch Size	9
Data Sampling	Method	Balanced
	`sequence_length`	20
	`sample_interval`	20
	`behaviour_duration_threshold`	72

Validation Set Results		
Top1	Top3	Class Average
74.82%	**95.49%**	**49.52%**

Table 4.11: **Pre-optimisation model summary**: Table of model configurations and hyperparameters of the model that is set to be optimised (top). Its results for the validation set is also shown (bottom). **Blue cells indicate modifications made since the baseline model.**

4.6 Optimisations

This section documents the adjustments made to the two-stream LSTM model, as specified in Table 4.11, in an attempt to achieve further improvement. Several aspects of the model were considered, and the impact of the adjustments were assessed. The attempted optimisations were performed sequentially, where if one does result in a significant improvement of the model, it is incorporated into its configuration. Any following adjustments are then executed while including the improving optimisation.

Admittedly, this approach of optimisation is not as exhaustive as a grid search, where the model is trained for every parameter combination possible [27]. However, each model takes a considerable amount of time to train due to the high levels of data and complexity involved. Thus, a grid search is not a feasible option given the time constraints of this project.

At this point, the model is able to perform behaviour recognition on the validation set with a good level of performance. Furthermore, its proficiency in predicting both majority and minority classes have seen significant improvements when compared to the initial baseline implementation. The main concern for the model is its tendency to overfit. The validation loss it produces sees an increase in early stages of training, before stabilising at a high level relative to the training loss. Some discrepancy is expected between the losses, but a large margin indicates that the model is overfitting to the training data. As such, the following optimisations are considered with a focus to address this issue.

4.6.1 Learning Rate

At first, the learning rate η is adjusted to examine its effects on the training of the model. It is one of the key hyperparameters that can dictate a model's performance. Across all implementations so far, a generally well-performing value of 0.001 has been used. For experimentation, the learning rate is adjusted on the logarithmic scale.

All other variants of the learning rate led to a decrease in the model's performance in terms of the accuracy metrics. But more importantly, the results show that a lower learning rate is able to reduce the validation loss over the course of training. With $\eta = 0.0001$, the loss is at **0.8267** compared to **1.027** when $\eta = 0.001$ is used. Although the latter shows better accuracies when evaluating the validation set,

η	Top1	Top3	Class Average	Loss
0.00009	67.69%	92.43%	36.95%	1.0512
0.0001	70.31%	**95.58%**	48.03%	**0.8267**
0.00011	66.23%	92.43%	42.67%	1.1451
0.0005	71.62%	93.74%	37.12%	0.9581
0.001 (Default)	**74.82%**	95.49%	**49.52%**	1.0270
0.01	70.01%	91.56%	28.31%	1.4070
0.1	17.61%	61.14%	11.11%	1.8550
+ pre-optimisation two-stream model				

Table 4.12: **Learning rate**: A value of $\eta = 0.0001$ is able to reduce the loss to minimise the risk of overfitting, while performing favourably in terms of accuracy.

its tendency to overfit may be damaging to its performance on unseen data. A lower loss indicates that the model produces less error across the distribution of the output probabilities and would be able to generalise better on unseen data. Such a model is more confident in the predictions that it makes. Thus, it is vital that the loss is kept minimal, while also taking into account the accuracy. $\eta = 0.0001$ is crucially able to minimise the loss, while also retaining the levels of accuracies to an acceptable degree. Smaller adjustments above and below $\eta = 0.0001$ were made to find a more optimal learning rate. However, they failed to show any improvements.

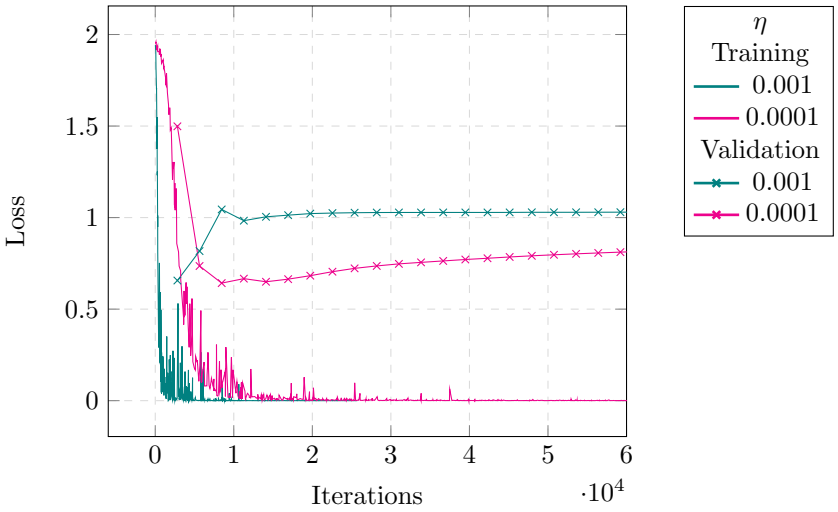


Figure 4.5: **Loss curves for training and validation using $\eta = 0.001$ and 0.0001.** The use of the smaller learning rate is able to reduce the validation loss significantly. Unlike $\eta = 0.001$, it does not attempt to converge too quickly to a sub-optimal solution. However, there is some sign of increasing overfitting when using $\eta = 0.0001$ in later stages of training.

When examining the curves on Figure 4.5, the effects of the learning rate can be seen. The rapid decline in the training loss, with no decrease in the validation loss at any point, shows that $\eta = 0.001$ converges too quickly to a sub-optimal configuration. This is different to when a $\eta = 0.0001$, where the loss for both sets of data falls gradually. Note that there is still a slow increase of validation loss after a minimum is reached, showing that there is still some overfitting taking place. Further optimisations will aim to address this. Based on the findings above, the model will now use a learning rate of 0.0001.

In hindsight, the choice to tune the learning rate at the very end of the period involving implementation may have been a misjudgement. Its ability to impact the model's ability to reach an optimal minima to great effect was underestimated. It may have been beneficial to experiment with learning rates during the implementation and experimentation of the different techniques used for the model. This would have given a better insight into the effectiveness of each technique on the model's performance.

4.6.2 Optimisers

The momentum used with the SGD optimiser was adjusted next. For this project, a momentum of 0.9 has been used. Values of both lower and higher momentum are used, where performance clearly deteriorates. It is unable to navigate towards a good minima with these settings.

The adaptive optimisation algorithm Adam [21] was also tested. It is able to maintain individual learning rates for each network parameter, which are adapted as the training progresses. The results show that it performs considerably worse than when SGD is used with the optimal momentum value of 0.9. This is in contrast to statements made in regards to Adam's ability of faster convergence, and that it is the best overall choice [45] among the many optimisers available. Alternatively, Wilson et al. [55] questions claims made about the superiority of adaptive gradient methods such as Adam. Their experiments show, especially for CNNs, that SGD outperforms Adam by a significant margin. It is also stated that adaptive methods are particularly popular for problems that do not involve optimisation [55], such as the training of GANs [43]. This does not apply to this task of behaviour recognition, which requires the minimisation of loss within the search space. It is difficult to pinpoint why Adam does not perform desirably in this context, however it is acceptable that every problem is unique and that there is no definitive choice among optimisers that will guarantee the best possible performance. In this particular case, SGD with momentum remains best suited.

Optimiser	Momentum	Top1	Top3	Class Average
SGD	0.5	66.96%	94.18%	38.46%
SGD	0.7	**70.74%**	94.32%	41.42%
SGD (Default)	0.9	70.31%	**95.58%**	**48.03%**
SGD	0.95	70.14%	94.67%	42.03%
Adam	-	64.19%	92.04%	29.32%
+ pre-optimisation two-stream model + $\eta = 0.0001$				

Table 4.13: **Optimisers:** The default use of SGD with momentum=0.9 remains best performing when comparing to the use of other momentum values and the Adam optimiser.

4.6.3 L_2 Regularisation

L_2 regularisation was applied in an attempt to reduce the overfitting of the model. The results show that its addition succeeds in doing so. It is able to maintain the complexity of the model leading to a significant reduction in its loss. Furthermore, L_2 regularisation resolves the issue of the slow increase of loss over time, discussed in Section 4.6.1 and illustrated in Figure 4.5. Experiments showed that a regularisation value of 0.01 worked best. It produced a loss of **0.6335** and a Top1 accuracy up to **73.07%**. The increased generalisability of the model has allowed it to score a class average accuracy of **54.13%**, the highest obtained on the validation set in this project. Thus, the model will now use an L_2 regularisation of 0.01.

L_2 Regularisation	Top1	Top3	Class Average	Loss
0 (Default)	70.31%	**95.58%**	48.03%	0.8267
0.001	72.05%	94.47%	53.39%	0.7816
0.01	**73.07%**	94.03%	**54.13%**	**0.6335**
0.05	69.14%	91.12%	33.79%	0.8663
0.1	40.76%	79.33%	18.09%	1.9020
+ pre-optimisation two-stream model + $\eta = 0.0001$				

Table 4.14: **L_2 regularisation**: A value of 0.01 is able to significantly reduce the loss of the model while increasing the Top1 and class average accuracies.

4.6.4 Data Augmentation

With the presence of class imbalance in the dataset, there is motivation to increase its size, in order to better represent the classes within it. Data augmentation was used as a method to supplement the existing dataset with modified variants of the samples. The types of augmentation used is outlined in Section 3.7.3.

It is also explained in Section 3.7.3 that the use of data augmentation is not appropriate for the temporal stream, due to the nature of optical flow. Nevertheless, this assumption was tested. The temporal stream was introduced with augmentation, where the optical flow was simply flipped horizontally. In theory, this should cause inconsistency in the representation of flow in the x direction and hence, negatively impacting the training of the model. For example, training on an optical flow image of an ape moving rightwards, which is horizontally flipped to make it look like it is moving left, would not translate well into instances where the model has to evaluate an optical flow image with an ape naturally moving left. The colours between the natural optical flow, and the flipped optical flow, would be inverted.

The model was trained for instances using only spatial augmentation, only temporal augmentation and finally, augmentation for both streams. They were applied with a probability of 0.25 and 0.5, representing the chance of any given sample to be augmented.

Stream w/ Augmentation	Probability	Top1	Top3	Class Average
None (Default)	-	73.07%	94.03%	**54.13%**
Spatial	0.25	**73.59%**	93.61%	43.90%
Temporal	0.25	71.49%	92.16%	42.66%
Spatial + Temporal	0.25	72.57%	93.01%	35.59%
Spatial	0.5	73.45%	**94.20%**	44.39%
Temporal	0.5	69.29%	91.85%	44.83%
Spatial + Temporal	0.5	73.11%	93.12%	43.13%
+ pre-optimisation two-stream model + $\eta = 0.0001$ + $L_2 = 0.01$				

Table 4.15: **Data augmentation**: When applied to both streams independently and together, with varying probability, the results show no conclusive sign of improvement.

Overall, data augmentation fails to show conclusive signs of improvement to the model. When using only spatial augmentation, a slight increase in the Top1 accuracy is seen, but the significant cost to the class average accuracy is undesirable. Meanwhile, the assumption about the incompatibility of augmentation of optical flow seems to hold true. The model suffers in all three metrics when only temporal augmentation is used, with an even greater impact seen as the probability increases to 0.5. The results solidify that this inconsistency hinders performance. When utilising data augmentation for both streams, a similar pattern of deterioration, especially of per class performance, is seen.

These results show a level of inter-dependence between the streams. Both streams are expected to see a consistent view of the data in both formats, RGB and optical flow. The use of augmentation can introduce these unwanted inconsistencies leading to lesser performance. This incompatibility is highlighted by the lack of data augmentation seen in the original paper [47].

4.7 Final Model

Final Great Ape Behaviour Recognition Model		
Model	Architecture	Two-Stream [47] w/ LSTM
	CNN	ResNet-18
	Pre-trained	Yes (both streams)
	LSTM	Both streams. 1 layer with 512 hidden units
	Fusion	LSTM output concatenation
	Optimisation	SGD with Momentum = 0.9
	Loss	Focal Loss ($\alpha = 1.00, \gamma = 1.00$)
	Learning Rate	0.0001
	L_2 Regularisation	0.01
	Batch Size	9
Data Sampling	Method	Balanced
	`sequence_length`	20
	`sample_interval`	20
	`behaviour_duration_threshold`	72

Validation Set Results		
Top1	Top3	Class Average
73.07%	**94.03%**	**54.13%**

Table 4.16: **Final model summary**: Table of the optimised configurations and hyperparameters of the final model (top). Its results for the validation set is also shown (bottom). **Blue cells indicate modifications made since the pre-optimised model.**

In conclusion to the adjustments made, the model now incorporates a learning rate of 0.0001 and an L_2 regularisation of 0.01. This establishes the final model. It performs at a Top1 accuracy of **73.07%**, Top3 accuracy of **95.58%** and a class average accuracy of **54.13%** when evaluating the validation set. Table 4.16 outlines the key configurations of the final behaviour recognition model.

4.8 Ablation Study

An ablation study of the final model was performed. A number of components of the model considered to be integral were identified and then consequently removed or reduced. The effects of these modifications on the performance of the model demonstrate their necessity. If there is no resulting decrease in performance, the modification will be put in place. In doing so, the model's complexity is kept only to a level that is required.

The following modifications were performed for the study:

1. The residual blocks of the ResNet-18 networks are removed.
2. Blocks within the ResNet-18 networks are frozen (each block is made up of 4 convolutional layers)
   (a) The 1^{st} block is frozen.
   (b) The 1^{st} and 2^{nd} blocks are frozen.
3. The number of hidden units in the LSTM networks are reduced from 512 to 256 each.
4. `behaviour_duration_threshold` (see Section 3.3.1) used for sampling the dataset is reduced to:
   (a) 48
   (b) 24

Ablation Study of Final Model			
Modification	Top1	Top3	Class Average
None (Optimal)	**73.07%**	94.03%	**54.13%**
1	63.46%	91.27%	27.03%
2(a)	66.45%	93.42%	42.40%
2(b)	65.14%	91.99%	41.30%
3	67.29%	92.58%	35.32%
4(a)	67.25%	**96.07%**	39.82%
4(b)	69.14%	91.70%	41.33%

Table 4.17: **Ablation Study results**: All attempted modifications of the final model result in deterioration in performance. Modification numbers map to the list stated in this section.

The results show that the key component of ResNet-18, the residual block, is crucial for its training. The residual connections exist to allow the network skip layers, as mentioned in Section 2.4.1. Without them, the network is unable to avoid updating the weights in instances where it degrades the model's performance. Thus, the model is unable to perform as effectively without residual blocks.

Unlike the fully frozen pre-trained ResNet-18 network trained in Section 4.1.2, the study examines the impact of freezing only the initial layers. As such, the first block is frozen, making up a quarter of the network. In addition, the first and second blocks are frozen, which is half of the network. The results reinstate the fact that the task of great ape behaviour recognition is highly specific, where fine-tuning the whole model is the more appropriate approach. This is further supported by the results showing that freezing the first two blocks performs worse than freezing only the first block.

The effects of reducing the number of hidden units within the LSTM is examined. Using LSTM networks with many hidden units leads to a large amount of GPU memory consumption, as they retain learned memory over the course of training. Thus, any reduction in this area is welcome. However, this is not possible without a significant decrease in performance. When attempting to reduce the number of hidden units from 512 to 256, the class average accuracy of **35.32%** resembles what was seen for the two-stream model without LSTM networks. This highlights the complexity of the task, specifically the input with a sequence length of 20 frames, which seems to require a greater number of hidden units in order to effectively capture spatio-temporal dependencies within the feature maps produced by the ResNet-18 networks.

Section 3.3.1 defines the key parameters used to specify the type of sample that is valid for use. Of which, the `behaviour_duration_threshold` is set to a value of 72 frames (3 seconds). The assumption was that constraining all samples to be derived from longer periods of behaviour would be more reliable for the network to train on. A counterargument is that by loosening the criteria for obtaining samples from the

dataset, it will lead to an increase in the number of samples that the network can train and evaluate on. This may consequently result in better performance. The model was tested with thresholds of 48 and 24 frames, from which an extra 394 and 588 samples were obtained respectively. These changes show a noticeable effect to the model's performance. The Top1 accuracy is reduced, and class average accuracy for both thresholds in largely impacted. Typically, more data is beneficial to the training of a model. However, in this case, the extra data gained may be of poor quality due to reasons mentioned above. On further inspection, the classes that gain the highest number of samples due to a smaller behaviour duration threshold are the majority classes, `standing` and `walking`. This gives reason to believe that the issue of class imbalance is exacerbated, corresponding to the large decrease in the class average accuracies for both thresholds used.

4.9 Final Results

This section conclusively evaluates the performance of the final model beyond the validation set.

4.9.1 Test Results

The test set containing 75 videos of data was evaluated using the optimal model. As this subset of the data has remained unseen to the model, it would be a good indicator of how generalisable, and applicable, its configuration is in a real world context. It obtains a Top1 accuracy of **79.25%**, Top3 accuracy of **95.23%** and a class average accuracy of **56.65%**, as shown on Table 4.18. The model has been able to maintain, and even surpass the performance seen for the validation set, improving on all three accuracy metrics. This suggests that the model has been able to successfully train on the training set without overfitting, as its performance is able to stay consistent with the introduction of unseen data.

Test Set Results (Not Cross-Validated)		
Top1	Top3	Class Average
79.26%	**95.33%**	**56.65%**

Table 4.18: **Final model evaluation of the unseen test set.** Levels of accuracy surpass those seen from evaluation of the validation set, indicating successful generalisation.

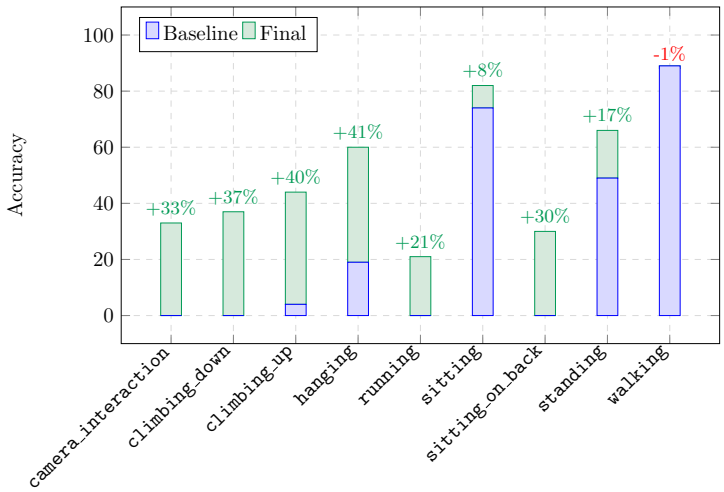


Figure 4.6: **Comparisons of class accuracies between the baseline model and the final model's evaluation of the test set.** The minority classes see significant gains in accuracy. Similar trends are seen for the majority classes too, but to a smaller extent.

The baseline model established in Section 4.2 was also used to evaluate the test set. This allowed for comparison and to assess the effectiveness of the numerous improvements made towards achieving the optimal model. The change in the class accuracies can be seen in Figure 4.6. There is clear indication that the issue of class imbalance has been alleviated to some degree. The minority classes have made significant gains in accuracy. Although the focus was on better representation of the minority classes, there were also notable improvements also seen for the majority classes. Naturally, these improvements were of smaller significance when compared to those of the minority classes.

Additionally, the confusion matrix for the final model's evaluation of the test set can be seen in Figure 4.7. When compared to the baseline model's evaluation of the test set shown in Figure 3.9 (validation set in Figure 4.3), there is a much stronger indication of the model's class-wise effectiveness, visualised by the increased consistency across the diagonal. The final set is able to achieve non-zero accuracy for all classes, compared to just five by the baseline model. Although there is still some dispersion among the predictions, it can be seen that the overwhelming tendency of the model making predictions of the majority classes has largely subsided. This shows that the techniques used to address class imbalance has largely been successfully, counteracting the suppression of minority classes during training and consequently in evaluation. `running` presents the anomaly to this, with 95% of its predictions being attributed to `walking`. Both classes are considered to be similar, with a key difference being the speed at which the ape moves. A possibility of this pattern may due to the constraints of the TV-L1 optical flow used. With the bounded the representation of optical flow in grayscale, it may be failing to capture the contrast in speed of the two behaviours, resulting in the model's weakness in identifying running.

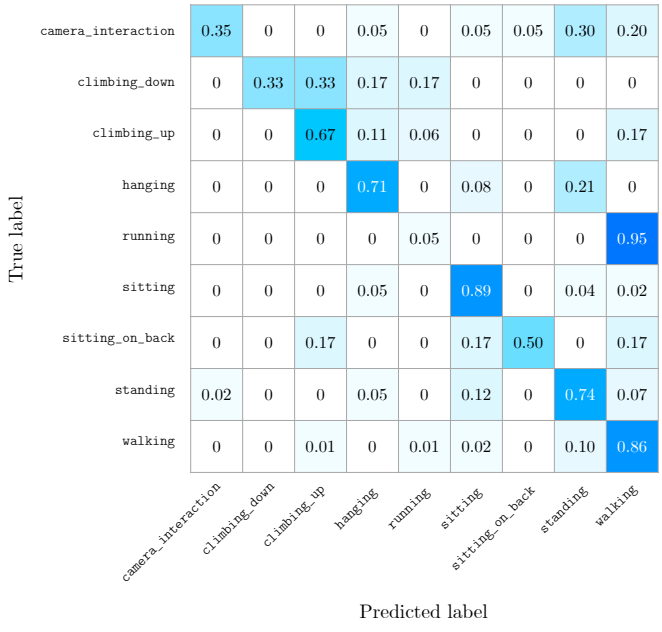


Figure 4.7: **The confusion matrix of final model's evaluation of the test set.** In comparison to the baseline model's confusion matrix in Figure 3.9, the final model is able to much better represent the classes overall. It also shows a reduced level of "confusion" where minority classes are incorrectly predicted to be majority classes.

4.9.2 Cross-Validation Results

To further test the model, cross-validation was carried out. The dataset was split up into 4 folds. For each fold, the training set consisted of 375 videos, while the test set contained 125 videos. The test set was evaluated for each fold at the end of training. The average of these metrics were calculated and were established as the final results of the optimal model, as shown in Table 4.20. The final individual class accuracies are also calculated as averages across the models. They are displayed in Table 4.21.

Fold	Top1	Top3	Class Average
1	75.74%	92.85%	47.67%
2	76.94%	95.18%	42.83%
3	70.70%	93.88%	40.48%
4	70.69%	94.38%	38.34%

Table 4.19: **4-fold cross-validation results**: Instability of the model can be seen to exist, with differing performance across the folds.

Final results of the cross-validation show that the model performs at a Top1 accuracy of **73.52%**, Top3 accuracy of **94.07%** and a class average accuracy of **42.33%**.

When compared to the initial test results of the model, the cross-validation results show a notable decrease in performance for all three metrics. The final Top1 accuracy produced is **5.74%** lower than the initial test set. The class average accuracy suffers more, with a decrease of **14.32%**. Although performance

remains at a satisfactory level for different splits of the data, there is a level of instability of the model indicated by its varying levels of success. The Top1 accuracy has a range of **6.25%**, while the class average accuracy has a range of **9.33%**. Each fold is unable to reach the class average performance of the test results by a significant margin. This contrast between the 4-fold cross-validation results and the test set results indicates that the model's performance is somewhat dependent on the data it trains on. Thus, the success in generalisability seen from its evaluation of the test set should not be interpreted as a true and complete view of its performance.

Final Cross Validated Results		
Top1	Top3	Class Average
73.52%	**94.07%**	**42.33%**

Table 4.20: **Final cross validated results of the optimal great ape behaviour recognition model.** The metrics are obtained from averaging the results of the 4-fold cross-validation.

The final class accuracies coincide with results seen throughout the project, with majority classes performing better than minority classes. `climbing_up` and `hanging` perform particularly well as minority classes. Meanwhile, there were four instances of behaviours classified with zero accuracy. They involved classes `camera_interaction` and `climbing_down`. Overall, this suggests that the issue of class imbalance and its effects on performance persists across different splits of the data.

Final Cross Validated Class Accuracy									
Class	camera_interaction	climbing_down	climbing_up	hanging	running	sitting	sitting_on_back	standing	walking
Accuracy	**32.13%**	**27.52%**	**40.25%**	**50.57%**	**18.76%**	**80.71%**	**12.15%**	**62.16%**	**87.55%**

Table 4.21: **Final per class accuracies obtained from the cross-validation results**.

4.9.3 Qualitative Analysis

The visual output of the final model specified in Table 4.20 was further inspected to get a better understanding of how it makes its predictions. A number of re-occurring instances were identified where the model fails to predict correctly. These patterns may suggest areas of improvement for future work.

When examining the behaviour classes found within the dataset in Section 3.1.1, a possible challenge for the model was identified in the form of transitional behaviours. Recall that they were recognised to be periods where an ape is in the process of ending its current behaviour and beginning to display the next. They were a source of ambiguity, even to the human eye. A number of instances showed that this problem had transpired in the predictions of the model.

Figure 4.8 shows two examples of where the model fails to predict transitional behaviours. In these scenarios the model is able to correctly identify the ape's behaviour displayed before and after the transition, but struggles during the transition itself. Within this period, the incorrect predictions made can vary greatly. For example, the top row in Figure 4.8 shows that the model starts to predict the second behaviour, `sitting`, early. In contrast, the bottom row shows that the model continues to predict the first behaviour, `standing`, late into the second behaviour. There are other instances where multiple incorrect predictions, different to both the first and second behaviours, are made within the short period of transition. This issue is certainly a weakness of the model. On the contrary, it must be noted that there are many instances where the model is able to correctly predict behaviours across transitions. After this issue was identified prior to labelling, great caution was taken to ensure unambiguous labelling with the establishment of a framework, detailed in Section 3.1.1. These findings show that the consistent labelling may have helped dealing with transitional behaviours to an extent, but the problem still remains.

The behaviour `running` remains the worst performing class at **18.76%**. In line with the findings from the confusion matrix of the baseline model shown in Figure 4.3, it continues to often be incorrectly predicted as `walking`. Stated first in Section 4.2, their similarity in appearance, along with the limited ability of optical flow to encode speed, causes the model to perform poorly in this case. Figure 4.9 shows an instance where the running of the ape is initially predicted correctly, before switching its prediction to `walking`. It is important to note that `walking` has the second highest number of samples, while `running` is made up of the lowest. The project has so far aimed to address the general class imbalance found in the dataset. But here, there may be a case of extreme pairwise class imbalance, where one class is being specifically dominated by another. The similarities between `walking` and `running` are substantial, but there are not enough samples of the latter for the model to distinguish between them.

Figure 4.8: **Incorrect predictions made during transitional behaviours**: The sequences progress in time from left to right. The top row shows an instance where the model starts to predict the subsequent behaviour `sitting` early. The bottom row shows the model predicts the subsequent behaviour `walking` too late.

Figure 4.9: **Poor performance on `running`**: A sequence where the model is unable to consistently predict an ape running. It begins with correct predictions, but at a later stage diverges to predict the ape as `walking`.

The model has a tendency to classify samples of apes as `climbing_down`, `climbing_up` or `hanging`, when there are branches visible. These behaviours are naturally associated with trees, and therefore appear in many of the samples of those behaviours that the model trains on. This is then reflected in its predictions. An example of this is shown on Figure 4.10 (left).

Figure 4.10: **Other observed weaknesses**: A `standing` ape predicted to be `hanging` **(left)**. This is attributed to the tree branches within the frame, which occur often in samples of `hanging`. The model incorrectly predicts an ape as `walking` **(right)**. Movement of the ape's arms while standing may be causing confusion for the model.

Another common trend involves the situation when an ape is standing stationary in the same place, but they show movement in their arms. Even though their feet are planted, they sometimes use their arms to dig or pick objects up from the ground. This movement, while the rest of their body remains still,

closely resembles the initial motions of walking. Understandably, in such instances, as shown in Figure 4.10 (right), the model predicts that the ape is starting to walk, when it is not.

4.9.4 YOLO

The direction towards an end-to-end great ape biometrics system was also explored. Its potential use for behaviour recognition in the wild would depend on the detection of great apes. Hence, YOLO was trained on the dataset to detect and classify the apes, to form the foundations of such a system.

Class	True Positives	False Positives	Average Precision
Chimpanzee	11453	3372	77.87%
Gorilla	14262	1407	84.68%
Overall	**25715**	**4779**	**81.27%**

Table 4.22: **YOLO ape detection and classification results:** The model was trained on the bounding box annotations in the great ape dataset.

Results of the YOLO network evaluating the test set show that it achieves a mean average precision (mAP) of **81.27%**. Its detection of gorillas is better performing than for chimpanzees. Among **25715** true positives, there is also a significant number of false positives of **4779** that occur. Nevertheless, given that YOLO is a fast and generalised object detector/classifier, it is able to perform at a good level and would be a suitable starting point if integrated into a behaviour recognition system.

Chapter 5

Conclusion

So far, the research of action recognition has remained human-centric [47, 53]. In contrast, this project attempted to explore and evaluate the effectiveness of deep learning techniques in performing this task in a new context, involving great apes. As a result, for the first time, a model capable of automating the recognition of great ape behaviours in the wild has been established. The nature of the problem, and the dataset, meant that the solution was required to be capable of multi-subject, multi-behaviour recognition across any given video. This is unlike what is seen in the majority of existing work, which usually involves the recognition of static behaviours displayed by an individual across the duration of the video [19, 24, 50]. These key differences necessitated a novel domain-specific approach to effectively perform behaviour recognition of great apes.

The project presents the results of the best performing model, made possible through exhaustive experimentation and optimisation carried out across various implementations. The outcome of a 4-fold cross validation procedure shows that the final model is capable of classifying nine core behaviours among gorillas and chimpanzees, at a Top1 accuracy of **73.52%** with a class average accuracy of **42.33%**.

Final Cross Validated Results		
Top1	Top3	Class Average
73.52%	**94.07%**	**42.33%**

Table 5.1: **Final cross-validated results of the great ape behaviour recognition model.** The model uses a two-stream ResNet-18 architecture in conjunction with LSTM networks. It also takes advantage of balanced sampling and the focal loss in order to suppress the effects of class imbalance in the dataset.

The contributions of this project are evaluated with respect to the objectives first defined in Section 1.4. To facilitate the learning and evaluation of the eventual behaviour recognition model, 500 videos of jungle footage, amounting to 180,000+ frames, were labelled with behaviours and identification numbers. This resulted in an extensive great ape dataset documenting multiple features, that will prove to be useful in research beyond the completion of this project.

The initial solution of the project utilised custom ResNet-18 networks leveraging the potential of transfer learning. Inspired by the two-stream method [47], it was able to capture the spatial and temporal dependencies found in the raw data to make predictions of behaviour. However, the evaluation of the model demonstrated it to be sub-optimal. Following this, detailed investigation into its performance, and the underlying characteristics of the dataset, revealed an overwhelming presence of class imbalance, severely impacting the model's ability to learn.

A broad approach was taken to suppress this issue, exploring three key areas of the model: method of sampling, loss function, and its architecture. Balanced sampling [14] was used to expose the model to every class equally during training. The focal loss [29] allowed poor performing minority classes to be assigned with more importance during training. Furthermore, the architecture was modified to incorporate LSTM networks as an alternative method of representing sequential motion found in the data. By rigorous analysis of numerous training runs and ablation studies, the cause and effect of these techniques on the performance of the model were examined. This helped towards discovering the locally optimised configuration and parameterisation of the final model, specific to the task of great ape behaviour recognition. All motivations guiding the direction of the model's evolution were stated in

Chapter 4. However, the issue of class imbalance can be seen to persist in the results of the model's cross validation. Section 5.1 outlines a number of possible directions that can be explored beyond this project to further address this issue.

The project also sets the direction towards a complete monitoring system, training a YOLO model to perform at an average precision of **81.27%**. This component is ready to be integrated into an end-to-end system to perform the detection, species classification, and behaviour recognition of great apes.

The accomplishment of the project's objectives indicate that the overarching goal has been met: to explore the effectiveness of deep learning techniques in performing automatic behaviour recognition of wild great apes. The potential of deep learning for the conservation and research of animals continues to be demonstrated. The findings of this project, along with the augmented dataset, will be of great aid to further research in this area, of great apes and of other species. To conclude, we are one step closer towards a comprehensive biometrics system to truly automate the monitoring of great apes.

5.1 Future Work

A vast improvement in the performance of the model has been achieved over the course of the project. The extreme class imbalance that hindered the baseline model has been addressed to a significant extent. However, though the final cross-validation results show good performance in isolation, the inconsistencies in performance between the folds show that the model suffers from some instability. Therefore, there is undoubtedly room for improvement, relating to its reliability, accuracy, and the expansion of its functionality. With this in mind, propositions of future direction for the project are stated below.

5.1.1 Expanding the Dataset

The most straightforward method of improving the model's performance would involve further gathering and labelling of great ape footage. Although it is trivial, its importance cannot be dismissed, especially for a dataset that suffers from severe class imbalance. By focusing on gathering data specific to the minority classes, the impact of the imbalance can be reduced. This would relieve the model's dependency on oversampling, in turn reducing its overfitting on duplicated samples of minority classes.

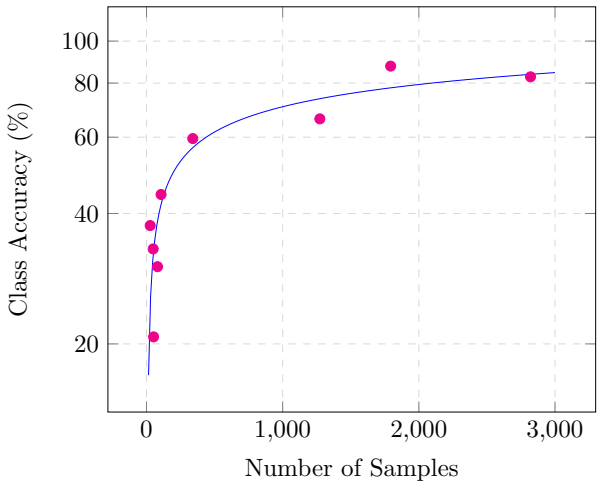


Figure 5.1: **Logarithmic correlation between the class accuracies and the number of samples that are available.** This relationship can be leveraged to determine the number of additional samples required to most efficiently improve performance.

Analysis of the optimal model's individual class accuracies in relation to the number of samples they consist of (given a sequence length of 20), showed that they share a logarithmic correlation. Illustrated in Figure 5.1, the curve can be seen as an approximation of the growth in performance that can be expected as the sample count for a class increases. In addition, the diminishing returns of this relationship can also be seen. This can be leveraged to estimate of the number of samples that need to be collected to achieve a certain level of performance. To begin with, gathering a further $200 - 400$ samples for the extreme minority classes would help to gain large margins of performance. In later stages, a target of ~ 2000 samples per class could be set, potentially establishing a class average accuracy of $\sim 80\%$.

Considering that the results for the YOLO network (see Section 4.9.4) achieved a high true positive rate, it could be used as a way of automatically obtaining bounding box coordinates of great apes in jungle trap camera footage. They can then be supplemented with behaviour annotations and consequently used for training the behaviour recognition model. It is important to note that the performance of YOLO is error-prone with a false positive rate of 18.73%, and as such it would not be appropriate for fully automating the identification of apes in raw data. However, it could be considered as part of a human-in-the-loop approach to gathering data, where results from the YOLO detector are manually examined and prepared, removing any incorrect samples found.

5.1.2 Transitional Behaviours

When examining the behaviour classes found within the dataset in Section 3.1.1, a possible challenge for the model was identified in the form of transitional behaviours. Qualitative evaluation in Section 4.9.3 showed that this problem had appeared in the final model. These behaviours tend to be of very short duration, lasting up to only a few frames. The final model's relatively coarse-grained approach to process temporality, in sequences that are 20 frames long, may struggle to pick up the nuances of a short transitional period between behaviours. Exploration into alternative methods of learning temporal dependencies may be useful in addressing this issue.

This project examined different aspects of sampling the data, such as the sequence/stack size (see Section 4.5.1) and the behaviour duration threshold (see Section 4.8), and its effects on the model's performance. However, the potential of temporal speed/rate (i.e. the interval between frames within the sequence/stack) was not explored.

SlowFast [8] networks make use of a two-stream structure similar to the work of Simonyan et al. [47], which also formed the basis of this project. The key characteristic of SlowFast is that it utilises different temporal speeds, taking into account the temporality of the data in two separate "pathways". The Slow pathway samples the data with a low temporal speed aiming to capture spatial dependencies, while the Fast pathway uses a high temporal speed designed to capture fast-changing motion. The network fuses the two pathways multiple times with the use of lateral connections. Furthermore, the spatial capacity of the Fast pathway is weakened, by reducing the input resolution and removing colour information [8]. This allows it to further focus on the temporal features of the video. SlowFast networks produced state-of-the-art accuracy on a number of video benchmarks.

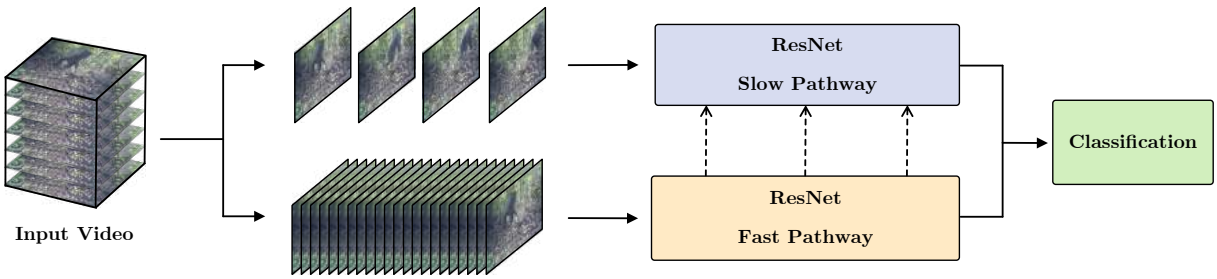


Figure 5.2: A high-level diagram of a ResNet-based **SlowFast** [8] network. Similar to the Two-Stream approach [47], it utilises two pathways to extract spatio-temporal dependencies in the data. The Slow pathway uses a low temporal rate to obtain spatial features, while the Fast pathway samples at a high temporal rate, focusing on fast-moving temporality. The dashed arrows represent lateral connections from the Fast pathway fusing into the Slow pathway.

When considering in particular, the issue of transitional behaviours, the differing temporal speeds may increase the model's proficiency as follows: Long, sustained periods of action, where the behaviour of the ape is visually clear, would be picked up by the Slow pathway. Meanwhile, the Fast pathway is able to constantly sample at a high temporal speed, ensuring that any rapid motion, for instance a sudden change in behaviour, is detected. Learning behaviours in a combination of speeds may thus provide a broader temporal context for the model to make better, more confident predictions in periods of transition.

Similar to the behaviour recognition model, it uses a ResNet backbone. As such, the techniques used in SlowFast can be integrated into the existing model. Interestingly, SlowFast networks are able to drop the use of optical flow, while achieving better performance. If this also translates to great ape behaviour recognition, it would be beneficial. It allows the model to be increasingly lightweight, removing the time and memory intensive prerequisite of computing optical flow from the raw data.

5.1.3 Auxiliary Behaviours

First mentioned in Section 3.1.1, the dataset contains a considerable number of instances where an ape is performing a secondary behaviour while already displaying a core behaviour. These "auxiliary" behaviours are currently not attempted to be classified. The two auxiliary behaviours in the dataset were identified as eating and scavenging. This transforms the current behaviour recognition problem into a multi-class, multi-label classification problem. As before, there are multiple behaviours that a sample can be classified as. But, there is now the possibility of two of behaviours occurring simultaneously.

A good starting point would be to use the Binary Cross Entropy (BCE) loss [34]. This effectively creates a set of binary classification tasks of C classes, where each behaviour class either exists, or does not, in a sample. The target for the model to learn from is provided to BCE as a multi-hot vector, where all classes applicable to the sample are set to 1. The current model makes use of the multi-class variant of focal loss. This can be changed to its original version which uses BCE [29], ensuring the effects of class imbalance is remain suppressed.

Compared to the CE loss used for multi-class single-label classification, BCE computes loss that is independent for each class. This means a sample being identified as one behaviour class, should not influence the decision for another class. Although this makes it suited for multi-label classification, it may introduce undesired error of class predictions where multiple behaviours are incorrectly inferred for a single behaviour.

5.1.4 Pose Coordinates

The model currently uses still RGB frames, along with the optical flow of videos, to derive information about an ape in order to make an inference. A key question is whether there is more to be gained from the data that is already available. One possible answer to this is the use of the ape's skeletal pose. The pose of an ape would include the locations of integral body parts, including their head, hands and feet, encoded as coordinates on a video. The motivation for its use stems from the fact that an ape's pose is dependent on the behaviour they exhibit, and vice versa. As such, inspecting the positioning of parts of an ape's body, relative to the rest of their pose, may help to predict their behaviour.

For example, an ape which is climbing a tree, or hanging off of it, is expected to have its hands near to adjacent, or above, its head. This would be in stark contrast to when the ape is on the ground, where all of its limbs are below its head. Characteristics such as these can supplement the features that the model can already extract from data to make better predictions.

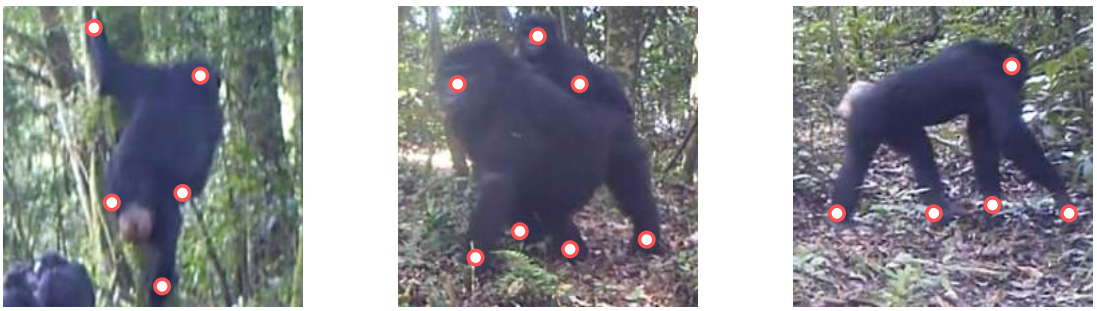


Figure 5.3: A visualisation of **pose coordinates** superimposed on the ape's body. The coordinates correspond to their head, hands and feet. By encoding them into sequences, they can be used for training a fully connected or LSTM network in order to learn dependencies of body movement to the behaviour the ape is exhibiting.

The coordinates of the ape's pose can be further leveraged if inspected temporally. This would be similar to how the two-stream model is exposed to the data as sequences. This could be vital in correctly classifying behaviours with similar temporal characteristics. One such instance is when the model has to distinguish between an ape walking and running. At the moment, optical flow fails to encode the difference in speed of these behaviours. By tracking the coordinates of the ape's limbs, the higher rate of change expected when an ape is running may help the model to differentiate between the two behaviours.

In implementation, the pose coordinates could be trained using a fully connected network. However, with evidence from the two-stream model, the use of an LSTM network may be more appropriate with its ability to train on sequences through time-steps. The input for the LSTM would be of size $[x \times y \times p \times T]$,

where (x, y) resemble the pose coordinates, for a body part p. They are stacked over a total of T time-steps, which signifies the length of the sequence. Its effectiveness, when used with either a fully connected network and an LSTM, would be assessed independently to the ResNet-18 two-stream model. The better performing variant would be integrated to the current model, as its own "stream", where the output of the pose network would be fused with the output of the two-stream network by concatenation. Following this would be the existing fully connected classifier with an expanded input size, fusing features obtained from all three streams to make a prediction.

Naturally, for this to work, a significant effort would be required to manually label the skeletal pose coordinates of the apes. The open-source polygonal annotation tool labelme [66] would be an appropriate choice for performing this particular labelling task, due to its support for coordinate point annotations and compatibility with video.

The benefits of this would not be constrained only to the improvement of the behaviour recognition model. If considering this work even further, the completion of the labelling would form a dataset with fully annotated great ape skeletal pose coordinates. This source could then be utilised to also explore ways to infer the ape's skeletal pose itself. As such, this direction has great potential in serving the project's wider ambition towards a comprehensive great ape biometric system [25] that can automate the monitoring of great apes.

Bibliography

[1] Eric B Baum and David Haussler. What size net gives valid generalization? In *Advances in neural information processing systems*, pages 81–90, 1989.

[2] Justin Bayer, Christian Osendorfer, Daniela Korhammer, Nutan Chen, Sebastian Urban, and Patrick van der Smagt. On fast dropout and its applicability to recurrent networks. *arXiv preprint arXiv:1311.0701*, 2013.

[3] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection, 2020.

[4] Clemens-Alexander Brust, Tilo Burghardt, Milou Groenenberg, Christoph Kading, Hjalmar S Kuhl, Marie L Manguette, and Joachim Denzler. Towards automated visual monitoring of individual gorillas in the wild. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 2820–2830, 2017.

[5] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014.

[6] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, June 2009.

[7] D. Ponsa E. Rublee E. Riba, D. Mishkin and G. Bradski. Kornia: an open source differentiable computer vision library for pytorch. In *Winter Conference on Applications of Computer Vision*, 2020.

[8] Christoph Feichtenhofer, Haoqi Fan, Jitendra Malik, and Kaiming He. Slowfast networks for video recognition. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 6202–6211, 2019.

[9] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. Convolutional two-stream network fusion for video action recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1933–1941, 2016.

[10] Lori Gruen, Amy Fultz, and Jill Pruetz. Ethical Issues in African Great Ape Field Studies. *ILAR Journal*, 54(1):24–32, 04 2013.

[11] Douglas M Hawkins. The problem of overfitting. *Journal of chemical information and computer sciences*, 44(1):1–12, 2004.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[13] James J Heckman. Sample selection bias as a specification error (with an application to the estimation of labor supply functions). Technical report, National Bureau of Economic Research, 1977.

[14] Julio Hernandez, Jesús Ariel Carrasco-Ochoa, and José Francisco Martínez-Trinidad. An empirical study of oversampling and undersampling for instance selection methods on imbalance datasets. In *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, pages 262–269. Springer Berlin Heidelberg, 2013.

[15] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

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

[17] Berthold KP Horn and Brian G Schunck. Determining optical flow. In *Techniques and Applications of Image Understanding*, volume 281, pages 319–331. International Society for Optics and Photonics, 1981.

[18] Nathalie Japkowicz. The class imbalance problem: Significance and strategies. In *Proc. of the Int'l Conf. on Artificial Intelligence*. Citeseer, 2000.

[19] Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, et al. The kinetics human action video dataset. *arXiv preprint arXiv:1705.06950*, 2017.

[20] J. Kiefer and J. Wolfowitz. Stochastic estimation of the maximum of a regression function. *Ann. Math. Statist.*, 23(3):462–466, 09 1952.

[21] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[22] Ron Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the International Joint Conferences on Artificial Intelligence*, 1995.

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

[24] Hildegard Kuehne, Hueihan Jhuang, Estíbaliz Garrote, Tomaso Poggio, and Thomas Serre. Hmdb: a large video database for human motion recognition. In *2011 International Conference on Computer Vision*, pages 2556–2563. IEEE, 2011.

[25] Hjalmar S Kühl and Tilo Burghardt. Animal biometrics: quantifying and detecting phenotypic appearance. *Trends in ecology & evolution*, 28(7):432–441, 2013.

[26] S. Kullback and R. A. Leibler. On information and sufficiency. *Ann. Math. Statist.*, 22(1):79–86, 03 1951.

[27] Hugo Larochelle, Dumitru Erhan, Aaron Courville, James Bergstra, and Yoshua Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *Proceedings of the 24th international conference on Machine learning*, pages 473–480, 2007.

[28] Steve Lawrence, C Lee Giles, Ah Chung Tsoi, and Andrew D Back. Face recognition: A convolutional neural-network approach. *IEEE transactions on neural networks*, 8(1):98–113, 1997.

[29] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[30] Amanda C Mathias and Paulo C Rech. Hopfield neural network: the hyperbolic tangent and the piecewise-linear activation functions. *Neural Networks*, 34:42–45, 2012.

[31] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[32] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*, 2010.

[33] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[34] Jinseok Nam, Jungi Kim, Eneldo Loza Mencía, Iryna Gurevych, and Johannes Fürnkranz. Large-scale multi-label text classification—revisiting neural networks. In *Joint european conference on machine learning and knowledge discovery in databases*, pages 437–452. Springer, 2014.

[35] Sridhar Narayan. The generalized sigmoid activation function: competitive supervised learning. *Information sciences*, 99(1-2):69–82, 1997.

[36] Andrew Y Ng. Feature selection, l 1 vs. l 2 regularization, and rotational invariance. In *Proceedings of the twenty-first international conference on Machine learning*, page 78, 2004.

[37] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with cuda. *Queue*, 6(2):40–53, 2008.

[38] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

[39] Javier Sánchez Pérez, Enric Meinhardt-Llopis, and Gabriele Facciolo. Tv-l1 optical flow estimation. *Image Processing On Line*, 2013:137–150, 2013.

[40] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[41] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[42] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv*, 2018.

[43] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative adversarial text to image synthesis. *arXiv preprint arXiv:1605.05396*, 2016.

[44] Raúl Rojas. *The Backpropagation Algorithm*, pages 149–182. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996.

[45] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[46] Daniel Schofield, Arsha Nagrani, Andrew Zisserman, Misato Hayashi, Tetsuro Matsuzawa, Dora Biro, and Susana Carvalho. Chimpanzee face recognition from videos in the wild using deep learning. *Science advances*, 5(9):eaaw0736, 2019.

[47] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In *Advances in neural information processing systems*, pages 568–576, 2014.

[48] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[49] Irwin Sobel. An isotropic 3x3 image gradient operator. *Presentation at Stanford A.I. Project 1968*, 02 2014.

[50] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012.

[51] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[52] Amin Ullah, Jamil Ahmad, Khan Muhammad, Muhammad Sajjad, and Sung Wook Baik. Action recognition in video sequences using deep bi-directional lstm with cnn features. *IEEE Access*, 6:1155–1166, 2017.

[53] Limin Wang, Yuanjun Xiong, Zhe Wang, Yu Qiao, Dahua Lin, Xiaoou Tang, and Luc Van Gool. Temporal segment networks: Towards good practices for deep action recognition. In *European conference on computer vision*, pages 20–36. Springer, 2016.

[54] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[55] Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning. In *Advances in Neural Information Processing Systems*, pages 4148–4158, 2017.

[56] Xinyu Yang, Majid Mirmehdi, and Tilo Burghardt. Great ape detection in challenging jungle camera trap footage via attention-based spatial and temporal feature blending. *IEEE*, 2019.

[57] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.

[58] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

[59] Alexey Bochkovskiy. YOLOv3 for Windows and Linux. https://github.com/AlexeyAB/darknet.

[60] Amazon. Amazon Web Services Home Page. https://aws.amazon.com/.

[61] Amazon. Boto3 Documentation. https://boto3.amazonaws.com/v1/documentation/api/latest/index.html.

[62] Git. Git Version Control Home Page. https://git-scm.com.

[63] GitHub. GitHub Home Page. https://github.com.

[64] Tensorflow Google. TensorBoard: TensorFlow's visualization toolkit. https://www.tensorflow.org/tensorboard.

[65] International Union for Conservation of Nature (IUCN). Red List of Threatened Species, 2020. https://www.iucnredlist.org.

[66] Kentaro Wada. labelme: Image Polygonal Annotation with Python. https://github.com/wkentaro/labelme, 2016.

[67] Max Planck Institute for Evolutionary Anthropology. Pan African Programme: The Cultured Chimpanzee. http://panafrican.eva.mpg.de/index.php.

[68] OpenCV. The OpenCV Reference Manual. https://docs.opencv.org/master/.

[69] Python Software Foundation. The ElementTree XML API Documentation. https://docs.python.org/2/library/xml.etree.elementtree.html.

[70] PyTorch. PyTorch Documentation. https://pytorch.org/docs/.

[71] University of Bristol Advanced Research Computing Centre. BlueCrystal Phase 4. https://www.acrc.bris.ac.uk/acrc/phase4.htm.

Appendix A

Dataset

The following illustrates the two main components of the dataset: great ape trap camera video footage and their corresponding annotations.

A.1 Annotations

The annotations of the dataset are in `XML` format. There are ∼180,000 files, where each file matches up to specific frame in a video. An example can be seen below:

```
<annotation>
    <videoname>adXBSoAug0</videoname>
    <frameid>4</frameid>
    <size>
        <height>404</height>
        <width>720</width>
        <depth>3</depth>
    </size>
    <object>
        <id>0</id>
        <category>Great Ape</category>
        <name>chimpanzee</name>
        <bndbox>
            <xmin>675</xmin>
            <ymin>28</ymin>
            <xmax>716</xmax>
            <ymax>105</ymax>
        </bndbox>
        <activity>climbing_down</activity>
    </object>
</annotation>
```

Listing 5: `XML` **annotation**: An example showing the labels found in an annotation specific to a frame.

An annotation is specified for a frame using `frameid` for video `videoname`. Every instance of an ape is stated as `object`, where `id` is its identification number. `name` classifies the species of the great ape. `bndbox` contains the coordinates of the bounding box of the ape. Finally, `activity` refers to the behaviour of the ape displayed in the frame.

A.2 Videos

There are a total of 500 videos. They are 360 frames long, playing at 24 FPS. The dataset consists of footage taken in daylight, during the night and in night vision.

Figure A.1: **Great ape trap camera footage**: Frames extracted from videos in the dataset.

Appendix B

Execution Instructions

This appendix aims to detail all prerequisites and instructions necessary to perform the training and evaluation of the great ape behaviour recognition model implemented in this project.

The structure of the project shown as `great-ape-behaviour-detector`, and with everything else shown as `local` can be seen in Figure B.1 below. The training and evaluation of the model is executed through `train.py` and `test.py` respectively. `config.json` allows for model customisation and settings paths to the data. Code for the project directory `great-ape-behaviour-detector` is also available on GitHub at https://github.com/fznsakib/great-ape-behaviour-detector.

`local` consists of the dataset of videos and annotations. The videos need to be extracted into RGB and optical flow frames (see Appendix B.1.2) and placed in their specific directories. `local` also contains the various output from the model, including logs, model checkpoints and final prediction output. `splits` includes text files of the names of videos involved in each of the three sets of data. `classes.txt` holds the list of behaviour classes in alphabetical order, as required by the model.

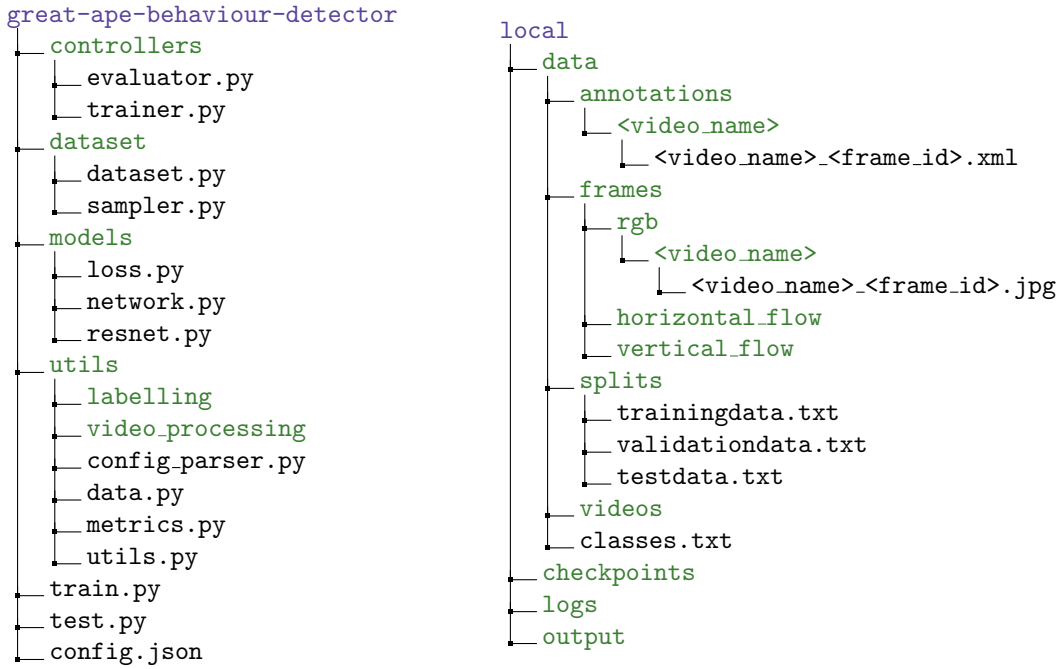


Figure B.1: **Project directory trees**: The green items represent directories, black items are files. `great-ape-behaviour-detector` is the directory hosting the project. `local` holds all items that are used for input to the model (the dataset) and any output produced by the model. As the name suggests, all files in `local` are kept on disk, hence they are not part of the GitHub repository.

B.1 Pre-Requisites

In order to run the model, the following must be completed to ensure all dependencies are met.

B.1.1 Installation

Ensure all Python dependencies are installed by doing the following:

1. Create a new Python virtual environment:

   ```
   python3 -m venv <env_name>
   ```

2. Activate virtual environment:

   ```
   source <env_name>/bin/activate
   ```

3. Install project dependencies:

   ```
   pip install -r requirements.txt
   ```

When execution is complete, deactivate virtual environment:

```
deactivate
```

B.1.2 Data Preparation

The model trains and evaluates on raw video as individual frames. It requires videos to be split up in RGB frames, and also pre-computed as optical flow images (for both horizontal and vertical directions). Scripts are available as part of the project to perform these tasks.

The videos are split as such:

1. Ensure there is a `data` directory in which there is a directory named `videos` holding all raw videos that will be used.

2. Run video to frames converting script:

   ```
   python utils/video_processing/video_to_frames.py
   ```

3. The following prompt will be displayed:

   ```
   Provide path to data directory:
   ```

   Type in the absolute path to the `data` directory.

4. Another prompt will be displayed:

   ```
   Provide path to txt file with names of videos to convert to RGB frames:
   ```

   Type in the absolute path to the text file that holds the names of the videos to be converted. If one does not exist, create one.

The optical flow is computed using the script `utils/video_processing/optical_flow.py`, following the same instructions as above.

Ensure that there are text files containing the names of videos that are to be used for their purpose by the model. For training, the files `trainingdata.txt` and `validationdata.txt` are required. When evaluating, the file `testdata.txt` needs to be created, listing all videos that are to be predicted upon.

B.2 `config.json`

The configuration of the model can be specified using `config.json`. It holds all parameters relevant to the training and evaluation of the model, along with paths to the required data.

For training, `name` refers to the identifier which the run will be saved and logged as. For evaluation, it is the name of the model to be loaded.

mode can either be `train` or `test`, depending on whether the model is being trained or evaluated, respectively. `resume` allows training to carry on from where it ended. `best` ensures that the model

with the best Top1 accuracy is loaded. `bucket` refers to the AWS S3 bucket name which final output is uploaded to a the end of evaluation. This can be kept empty if the uploading of the output is not desired.

`paths` specifies the different directories, presented in Figure B.1, required for the execution of the model. `frequencies` determines the number of epochs that must pass for a certain action to occur.

All other parameters are related to the model architecture and sampling of data. Their functions and relevance to the model can be found throughout Chapter 3.

```
{
    "name": "twostream_LSTM",
    "mode": "train",
    "resume": false,
    "best" : false,
    "log": true,
    "bucket": "output_bucket",
    "cnn": "resnet18",
    "hyperparameters": {
        "epochs": 50,
        "learning_rate": 0.0001,
        "sgd_momentum": 0.9,
        "regularisation": 0.01,
        "dropout": 0
    },
    "loss": {
        "function" : "focal",
        "cross_entropy": {
            "weighted" : false
        },
        "focal": {
            "alpha": 1,
            "gamma": 1
        }
    },
    "lstm" : {
        "layers": 1,
        "hidden_units": 512,
        "dropout": 0
    },
    "dataset": {
        "sample_interval": 20,
        "sequence_length": 20,
        "activity_duration_threshold": 72
    },
    "dataloader": {
        "batch_size": 9,
        "shuffle": false,
        "worker_count": 1,
        "balanced_sampler": true
    },
    "augmentation": {
        "probability": 0.2,
        "spatial": {
            "colour_jitter": false,
            "horizontal_flip": true,
            "rotation": false
        },
        "temporal": {
```

```
            "horizontal_flip": true
        }
    },
    "paths": {
        "annotations": "local/data/annotations",
        "checkpoints": "local/checkpoints",
        "classes": "local/data/classes.txt",
        "frames": "local/data/frames",
        "logs": "local/logs",
        "output": "local/output",
        "splits": "local/data/splits"
    },
    "frequencies": {
        "log": 1,
        "print": 1,
        "validation": 1
    }
}
```

Listing 6: `config.json`: An example configuration file showing all the different model hyperparameters and execution parameters that can be specified.

B.3 Running the Model

Once the data and `config.json` are prepared, the model can be run.

To train the model, simply run:

```
python train.py --config config.json
```

The model will output metrics to the command-line. Additionally, the metrics are logged to disk in a directory by the model `name` provided, at the `log` path. Model weights are saved at the `checkpoints` path, also by its `name`.

To evaluate the model, simply run:

```
python test.py --config config.json
```

The model will output status updates and the results of the evaluation on the command-line. The final output is stored at the `output` path, by the model `name`. It is also uploaded to the user's AWS S3 `bucket` if provided, returning a download link once complete.