UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE FÍSICA

Deep Learning for Multi-Animal Tracking

Madalena de Oliveira Santos Lourenço Valente

Mestrado Integrado em Engenharia Biomédica e Biofísica
Perfil em Engenharia Clínica e Instrumentação Médica

Dissertação orientada por:
Dr. Gonzalo de Polavieja
Prof. Raquel Conceição

2021

Acknowledgements

For any students reading this and starting their own Master Thesis, I just have one thing to say: **DO NOT PANIC!**

With the completion of my thesis project I want to thank Raquel Conceição, for being a great professor and having the patience to guide the lost student I sometimes am. Thank you to Gonzalo de Polavieja, for accepting me in his lab and giving me an interesting project to work in and learn various skills. Thank you to Fran for helping me when sometimes I did not know what something was.

A special thank you to Paco, for always teaching me, for helping me when I was lost in a conversation, for laughing when I made my ”what is happening” face, and for being very patient with me. You are a great person to work with, and I am very lucky to have been able to do it!

Thank you to my family and my friends, for supporting me and always listening to me when I rant about my work. Daniel, you suffered the most, but you were always there to give me advice. Andreia, Joana, Patricia, Catarina and Pedro, most of you are in the same boat as me, and were a great help to talk to, besides being there for the past 5 years. Thank you all!

And now that this stage of my life is over, I have only one last thing to say: ”So long and thanks for all the fish”!

Resumo

Na área de estudos sobre comportamentos de grupos/coletivas de animais, a possibilidade de podermos identificar e ”seguir” (*tracking*) os indivíduos durante as suas interações é uma mais valia para o estudo dos seus comportamentos.

Existem diversas técnicas que permitem este *tracking* de animais, nos seus habitats naturais ou em ambiente laboratorial. Nomeadamente, podem ser usados sensores conectados ao corpo do animal, ou uma técnica mais usada em animais de tamanho reduzido e em ambientes experimentais - *tracking* baseado em sistemas de imagens.

O *idtracker.ai* é um sistema de *tracking* que usa vídeos para detetar e fazer o *tracking* dos animais. Neste caso, o sistema é composto por duas redes neuronais convolucionais. A primeira rede neuronal classifica as *frames*/imagens do vídeo como um indivíduo ou como o cruzamento/contacto de pelo menos dois indivíduos - *crossing detector*. A segunda rede neuronal faz a classificação de cada imagem, de modo a identificar a qual dos indivíduos/classe esta pertence - *identification network*.

As imagens usadas para treinar a segunda rede neuronal são as mesmas que foram classificadas na primeira rede como imagens de um único indivíduo. Isto leva a que, por vezes, o número de imagens existentes por indivíduo/classe seja reduzido, quando são utilizados vídeos com maior densidade de animais, sendo que existem mais imagens classificadas como cruzamentos. Quando estamos perante esta situação, o tempo de *tracking* do sistema pode demorar até 24 horas. Para um sistema como o *idtracker.ai*, um dia é considerado um tempo demasiado longo para este treino.

O objetivo deste projeto é corrigir o problema referido anteriormente, tornando a *identification network* mais rápida. Com o intuito de arranjar uma nova estratégia de *machine learning*, para tornar o treino da *identification network* mais rápido, mantendo uma precisão equiparável à do *idtracker.ai* numa situação óptima (98%), realizaram-se testes utilizando três estratégias distintas.

A primeira estratégia utilizada consiste em usar redes neurais previamente treinadas (*transfer learning*) e voltar a treiná-las com as mesmas imagens utilizadas com o *idtracker.ai*. Deste modo, resolvemos o problema da existência de poucas imagens para o treino. Os modelos utilizados para esta experiência tinham arquiteturas diferentes do modelo de *idtracker.ai*, sendo que todos tinham mais camadas (*layers*) que o de *idtracker.ai*. De modo a não termos um aumento de tempo, devido ao facto de os modelos serem maiores, apenas utilizámos parte das camadas de cada um dos modelos, para que estes ficassem com um tamanho semelhante ao do modelo de *idtracker.ai*.

Para a segunda estratégia foi utilizado um método de *ensemble*, isto é, treinando várias vezes a rede neuronal do *idtracker.ai*, guardamos vários modelos com precisões diferentes na identificação dos indivíduos. Com estes modelos fizemos *ensembles* de número variado (3, 5 e 10) de modelos, de modo a obter uma precisão mais elevada na identificação das imagens. Ou seja, ao considerarmos as predições de vários modelos para uma imagem, a probabilidade de certeza aumenta. Neste caso, focamo-nos em melhorar a precisão na identificação das imagens, para o caso em que temos poucas imagens para treinar a *identification network*, dando menos importância ao tempo de treino.

Na última estratégia tomada, exploramos um método diferente de treinar a rede neuronal do *idtracker.ai*. Para isto, não utilizamos apenas imagens com *labels* (identificadas com a classe a que pertencem, denominadas de imagens com *multi-class labels* nesta experiência), como nos métodos de treino das experiências anteriores e do método implementado em *idtracker.ai*. Neste caso, para além disto, utilizamos a comparação entre pares de imagens (*pairwise-labels*). Desta forma, treinamos o modelo de *idtracker.ai* com 30 imagens por classe com *multi-class labels*, mudando o número de imagens por classe sem estas *labels*. O que significa que nestes testes, o que mudamos foi o número de *pairwise-labels* com que treinamos a rede neuronal. O objetivo desta técnica é mostrar como podemos aumentar o número de imagens para o treino da rede neuronal de *idtracker.ai*, mesmo sem ter um número elevado de imagens identificadas (*multi-class labeled*), o que leva a um aumento da precisão na identificação de imagens.

Com a primeira experiência conseguimos uma melhor precisão na identificação de imagens, com maior parte dos modelos utilizados, exceto com um (*Resnet18*). A precisão é maior quando comparada com a obtida com o modelo de *idtracker.ai*, especialmente nos testes em que treinamos os modelos com o menor número de imagens (30 imagens por classe). No entanto, isto é à custa de um tempo de treino mais demorado que com o modelo de *idtracker.ai*.

A segunda experiência mostrou que o tempo de treino também aumentou. Sendo que, neste caso, era esperado, tendo em conta que o que estamos a fazer é treinar o modelo de *idtracker.ai* e juntar várias predições, o que significa que o tempo que consideramos é o tempo de treino dos vários modelos que estamos a usar no *ensemble*. No entanto, o aumento na precisão da identificação de imagens não é considerável, para além de que não chega a equiparar o melhor modelo de *transfer learning*, e o tempo de treino é maior ou equiparável a esse modelo.

Na última experiência observamos o maior aumento de todas as experiências, em relação à precisão da identificação de imagens, quando comparado com o modelo e modo de treino do algoritmo de *idtracker.ai*. O aumento do tempo de treino obtido é esperado, pois o número de imagens utilizado para treinar o modelo é superior ao utilizado com o método original de *idtracker.ai*. O número de imagens com *multi-class labels* é o mesmo para o novo método de treino (*pairwise comparisons*) e para o método original de *idtracker.ai*, mas com a adição das imagens com *pairwise-labels* conseguimos ver o aumento da precisão de identificação.

Após testarmos estes três métodos de *machine learning* podemos concluir que a última hipótese foi a que proporcionou melhores resultados. Ao treinarmos a rede neuronal com mais imagens, que não só aquelas que têm *multi-class labels*, aumentamos a informação que o modelo tem sobre as imagens. Esta informação adicional, que vem na forma de comparação entre pares de imagens, permite que o modelo de *idtracker.ai* seja treinado de um modo mais eficaz, para que posteriormente seja usado na classificação de imagens. Observamos isto na precisão conseguida com este método: uma média de 94% (treinando o modelo com 3000 imagens por classe, sendo que apenas 30 imagens por classe apresentavam *multi-class labels*). Este resultado aproxima-se daquilo que o algoritmo do *idtracker.ai* consegue obter quando é treinado com 3000 imagens por classe (sendo todas imagens com *multi-class labels*) - 98% - e é superior ao resultado obtido quando este é treinado com apenas 30 imagens por classe com m*ulti-class labels* - 56%.

Um dos objetivos desta dissertação era também diminuir o tempo de treino da *identification network*. Este último método não se apresenta mais rápido, devido ao maior número de imagens utilizadas. No entanto, acreditamos que utilizando este método para treinar esta rede possibilitamos uma melhor aprendizagem e maior precisão, mesmo com um acréscimo de tempo.

Em conclusão, o método de p*airwise comparison* (comparação de pares) é uma mais valia no treino desta rede neuronal, tomando partido de um maior número de imagens não utilizadas anteriormente.

Com futuros testes em vídeos poderemos observar se esta tendência se mantém, de modo a melhorarmos o funcionamento do algoritmo de *idtracker.ai*.

Palavras-chave: *tracking* baseado em imagens; *transfer learning*; aprendizagem em *ensemble*; identificação de pares.

Abstract

In collective behaviour studies, the use of multi-animal tracking systems is extremely valuable. To be able to identify and track each individual in a group helps in the study and understanding of their behaviour in the collective. For this, researchers can use tracking systems, which can use sensors to detect the individuals; or they can use image-based tracking, with or without the need to mark the individuals. idtracker.ai is a state-of-the-art multi-animal image-based tracking system that uses convolutional neural networks to identify each of the individuals in a video. In videos with a higher density of individuals, idtracker.ai cannot extract enough frames of the single individuals (few frames ≈ 30) and the training of the identification network is slower. With the idea of decreasing this training time, here we propose to test three different machine learning methods. The first method is to use Transfer Learning models, with the expectation that the training can be done with few data. The second method is to use the ensemble method to join the results of various models of the idtracker.ai identification network, and thus decrease the variability of classification. Finally, the third method is to use not only multi-class labels but also pairwise-labels to increase the amount of information the network has available for training. The three methods are compared to the idtracker.ai model in terms of image classification accuracy and training time. Transfer learning and ensemble improved the accuracy of classification, but failed to reduce the time of training of the identification network. The pairwise method increased accuracy and time of training was comparable to the one of idtracker.ai. More specifically, by training the identification network with multi-class labeled images and pairwise-labeled images, the information the network can have from few images leads to an average classification accuracy of 94% (for 3000 images per class with 30 multi-labeled images per class). This is comparable to idtracker.ai trained with 3000 multi-labeled images (per class) - 98% accuracy, and is better than when idtracker.ai is trained with 30 multi-labeled images - 56% accuracy.

Key words: image-based tracking; transfer learning; ensemble learning; pairwise-labels.

Contents

List of Figures

List of Tables

List of Abbreviations

AdaGrad Adaptive Gradient Algorithm.

Adam Adaptive Moment Estimation.

ANN Artificial Neural Network.

CARP Champalimaud Animal Recognition Project.

CE Cross-Entropy.

CNN Convolutional Neural Network.

KL Kullback-Liebler.

MCL Meta Classification Likelihood.

PE Processing Elements.

ReLU Rectified Linear Units.

ResNets Residual Neural Network.

RNN Recurrent Neural Network.

ROI Region of Interest.

SGD Stochastic Gradient Descent.

Chapter 1

Introduction

For the study of collective behaviour multi-animal tracking systems are useful for tracking and identification. An example of a state-of-the-art system is the idtracker.ai [1] algorithm [1]. The focus of this dissertation is on experimenting with different machine learning strategies to improve the performance of idtracker.ai. In this chapter, we introduce the field of collective behaviour and the problem which we aim to solve. This is followed by the objectives of this project and an outline of the dissertation.

1.1 Context and Motivation

The study of collective behaviour is relevant for areas such as Economics, Control Theory, Social Sciences [2], Biology and Neuroscience. This field aims to understand the interaction between individuals and the patterns that can be observed in collectives. The behaviour of each individual in relation to their environment and other individuals in the group is what determines ecological interactions, which have implications in the organization of populations and ecosystems [3]. The results of investigating behavioural patterns and underlying rules can be used for biological and robotic research [4].

In order to quantitatively assess animal collectives, it is helpful to track each individual in a group, because it allows for the gathering of precise information about the position of each individual in the collective at any given time. This tracking of individuals can be done following different strategies, for example, using GPS trackers and other sensors, or simply tracking individuals in a video [3, 5–7].

One of the main difficulties in multi-animal video tracking is to keep the identity of the animals after they touch or cross with other animals. To solve this problem, the de Polavieja Lab, at the Champalimaud Foundation, developed two software packages to track animal collectives in videos under laboratory conditions, idTracker [8] and idtracker.ai [1]. idtracker.ai is an improvement over idTracker, as it can track and identify up to 100 animals, 10 times more than the previous software. idtracker.ai uses artificial neural networks for the tracking and identification of the individuals. It uses two artificial neural networks: the first to detect if the animals are touching/crossing (*crossing detector*), and the second to identify each animal individually (*identification network*). idtracker.ai trains its networks with data it gathers automatically from the video. This algorithm has a major limitation: in videos with higher density of individuals or where animals are too close together, idtracker.ai cannot get enough information from the video to efficiently train the identification network. This leads to the tracking being too slow or not accurate enough.

[1] Available at `https://idtrackerai.readthedocs.io/en/latest/`

This project will focus on improving the training efficiency of the identification network, especially for the cases where idtracker.ai cannot gather enough data to train the identification network.

1.2 Objectives

To improve the training of the identification network we will explore various machine learning techniques and evaluate which ones achieve a faster training and a better accuracy. The goal of this is the improvement of the algorithm to allow successful tracking in more complex videos (e.g. with higher density of individuals). In particular, we will study 3 distinct strategies. Firstly, to overcome the problem of having few data, we will use transfer learning models [9]. Secondly, we will try to decrease the noise in the predictions of the algorithm by joining various predictions of the models with an ensemble method [10]. Finally, we will increase the amount of data used during the training of the model, by using siemese-like architectures [11, 12] to take advantage of pairwise constraints between the identity of the animals, e.g. two animals in the same frame cannot have the same identity, and using that information to improve classification accuracy. These strategies will be explained further in the dissertation.

1.3 Outline

This dissertation is divided as follows: Chapter 2, we present the review of the State-of-the-art, focusing on image-based tracking systems and in the idtracker.ai algorithm. In Chapter 3, we introduce the main concepts of deep learning and neural networks, focusing on convolutional neural networks and the three machine learning techniques used in this project. In Chapter 4, we specify the dataset used and the parameters of the performed experiments. In Chapter 5, we present and discuss the results of the project itself. Finally in Chapter 6, we show the conclusions.

Chapter 2

Collective behaviour and image-based tracking systems

In this chapter, we present a brief explanation of what multi-animal tracking is and why it is useful for the study of collective behaviour (Section 2.1). After that, we describe different types of image-based tracking systems (Section 2.2). Lastly, we explain how the identification network of idtracker.ai works and its limitations (Section 2.3).

2.1 Collective behaviour and animal tracking

To understand ecological systems, it is important to understand the individual and collective behaviour of the animals within the environment and between each other. Quantification of behaviour started with direct observation, but this had some significant constraints: it only worked for small collectives, it ended up being somewhat subjective, and there was not an exact record of the events. It was also harder to consider every interaction of all individuals [3].

Tracking each animal in the group is considered an easy way of quantifying behaviour. This approach allows observation in the individual's environment, or in laboratory conditions. At the same time, we can present external stimuli to the animals and see the change in their behaviour, while registering all their interactions and reactions to stimuli [13]. This can be done by marking the individual with bio-loggers (or animal-mounted sensors). Some examples are: GPS, accelerometers, magnetometers, pressure sensors and acoustic recorders. These can record data continuously and directly at the location of the animal, and without the need to maintain an identification process. With these on-board sensors it is easy to monitor animal position, movement and behaviour, independently of where the studies are done. The trackers (bio-loggers) can record data with high sampling rate that allows an almost complete reconstruction of an animal's 3D-path [14, 15]. Unfortunately, the need to attach the sensors to each of the animals in the studies can be a disadvantage. The fact that the animals need to carry these sensors can be a limitation, as it can lead to changes in their behaviour. Besides, not all animals can carry a bio-logger, since some are too small [14].

A less intrusive way of studying animal behaviour is to avoid attaching any object to the animal, while avoiding changing their behaviour. This can be achieved using object tracking in videos. An advantage of this unmarked tracking method is that it can track a greater number of individuals and allow for a higher spatial-temporal resolution of behavioural data [3]. Multi-object tracking in video has two main steps: detection of the object of interest in movement, and tracking it from frame to frame. This is useful

for: human detection based on movement, automated surveillance, video indexing, traffic monitoring, vehicle navigation (for smart cars), etc. The main features used are: color, edges, optical flow, texture, spatial-temporal features, and multiple feature fusion [16, 17].

In image-based tracking, several software packages can be used, some of which are mentioned in the following section (Section 2.2). There are examples of tracking systems that can track in real-time and/or recognize individuals across image sequences [8]. Being able to track in real time is very useful because it removes the need to store large amounts of data [18]. Researchers can also change environmental conditions/interact with the individuals while tracking [13]. These external stimuli can be applied by means of virtual reality and robotics [19, 20].

An advantage of image-based tracking is the ability of having detailed information about the behaviour of the individuals: how they move, with whom they interact, what sensory information is available to them, and how the environment and internal drivers influence their behaviour. One of the things that makes image-based tracking hard is the fact that animals can change their shape, which makes their identification over time harder [21]. Also, they can move rapidly, which requires videos/images with high spatial-temporal resolution. Even then, interactions between animals result in occlusions/crossings between them, making their individual identification and tracking harder to complete [22, 23]. One other limitation regarding this type of tracking is that it is not as useful for large animals, as they could not be tracked over their entire environment. Also, tracking in complex physical habitats or in dense populations is something difficult to accomplish [3].

For human detection, an advantageous tool is using gradient features. These use the shape, contour or silhouette of the person to be detected. Some tracking methods are based on online learning, in which the algorithm continuously learns the characteristics/features it uses to track the objects/individuals, without the need to save all these data. This can be useful when tracking objects with varying appearance. An example are generative methods, these learn the appearance of an object, and this learning can be updated online, to adapt as the object shifts/changes. One other method is the discriminative method. In this method, a classifier is trained and updated (incrementally over time) online to distinguish between the object and the background. This last method can also be called tracking-by-detection [17].

2.2 Image-based tracking systems

There are some examples of image-based trackers, each with different capabilities. However, most cannot track a large number of individuals at the same time. For example, the systems from [18, 23–28] can only track between 2 and 7 individuals at a time. Others have the limitation of only being able to detect animals of certain shapes, rodent-like [23, 24], fly-shaped [25], worm-like [29], zebrafish-like [28]. Other systems can be used more freely, working for differently shaped animals, such as elliptically-shaped [22, 27] or even flexible systems [3, 26, 29, 30], which can be used for various animal species.

Some systems are mostly for commercial use and, unlike idtracker.ai, are not free, for example [3]:

- EthoVision XT can track up to 16 individuals of flexible shape, and the individuality of each animal is maintained through comparison of different sizes (animals of same size need to be marked, so that to maintain their identity the software compares markings). This method of maintaining identity is not robust and can lead to identity changes and propagation of errors.

- GroupScan tracks up to 100 individuals of no specific shape, but does not maintain the identity of each individual through an occlusion.

- PhenoTracker can track up to 50 individuals, is flexible to any shape, while maintaining the individuals' identity, similarly to EthoVision.

An example of an open-source online tracking software is xyTracker. This software is used for detection and tracking of individual animals in a group by online-learning the identity of each individual, using appearance features. The tracking is possible in real-time and the system allows the use of a stimulus (available in the software). The software uses OpenCV libraries and has a MATLAB-based interface. It was developed mainly for zebrafish, but it can be used with other animals as long as they have cylindrical to oval shapes. xyTracker detects blobs (individuals) and an ellipse is fitted to the shape of the blob. The following blob features are extracted, namely the ellipse's two half-axis lengths, orientation and its center, and pixel values. After this, each individual is assigned to a track/identity [13]. Additionally, xyTracker has a recognition system for the individual animals, to ensure the detections are correctly assigned to their tracks. This uses a classifier (Gaussian mixture model) that gives a probability of whether the feature belongs to one of the animals. The classifier maintains a model of each animal (their appearance), and is updated continuously to learn a better model over time. In the assignment of tracks, the system can change them while analysing the subsequent images when crossings occur, which can lead to some errors in the assignment. The study in [13] mentions the tracker was tested for only 5 individuals.

Sridhar *et. al* [7] presented an image-based automated tracking, used for single or multi-object tracking. Tracktor is a python-based program (uses the OpenCV library), which uses adaptive thresholding to automatically detect the object in the video. It separates the different objects using k-means clustering, and maintains the identities of the objects between frames by using the Hungarian algorithm (the reader is referred to [31] for more information on this algorithm). This algorithm cannot maintain identities through occlusions or crossings of the objects/individuals. In [7], the videos tracked had between 1 and 8 individuals. Another tracking software that is not able to resolve identity trough crossings is MARGO [6]. MARGO is an object-tracking platform, which is MATLAB-based, that can be used in real-time tracking. It detects objects by background subtractions and by fitting the identified blobs in a region of interest (ROI).

Ctrax is another algorithm which can detect up to 50 individuals. The animals need to have an elliptical shape for Ctrax to work. The animals are first detected and then associated to an identity of a previous frame. The detection of individuals is based on background subtraction: classifying pixels as belonging to the individual or not (foreground and background, respectively). The pixels classified as foreground are then fitted to one or more ellipses depending on if the system recognizes the connected components as one individual or multiple individuals. Connected components can be merged or separated, to guarantee a more precise detection of each individual. The labeling of individuals is done by comparing previous frames and using a constant velocity model, which allows for the prediction of the positions of the individuals in the current frame, using the previous ones, then comparing the predicted positions with the observed ones, minimizing the error. However, considering the way this system assigns identities, error propagation can happen as the identities are not maintained through multiple frames, but are always obtained by comparing to previous frames [3, 22].

Similar to idtracker.ai, DeepLabCut [32] and LEAP [33] are two tracking systems that use deep neural networks to acquire data from videos to quantitatively study animal behaviour. However, these tracking systems focus on tracking animal postures and their multi-animal mode does not use CNNs for the identification of the animals. DeepLabCut uses the network DeeperCut [34], a deep convolutional neural network, that contains a pre-trained deep Residual Neural Network (ResNets) with deconvolutional lay-

ers as their output, instead of a classification layer. The deconvolutional layers are used to up-sample the information and produce spatial probability densities for each body part it is detecting [32]. LEAP uses a simpler network: a fully convolutional architecture, composed of 15 layers of repeated convolutional and pooling layers. This architecture improves training and prediction performance. The network uses a single image as input, and produces a set of confidence maps (probability distributions) as outputs. These confidence maps describe the location of each body part of the input image [33].

A tracking software that manages to achieve very similar results, in terms of accuracy, to idtracker.ai, is the TRex [35]. The video can be recorded using a task-specif tool of the software, TGrabs, which is also used for the pre-processing of already existing video files. This software processes the videos in the following way: firstly it segments the video, the individuals are detected in the frames and cropped out. After that, the tracking generates the trajectories of the individuals. This can be accomplished by using information about the individuals' movement and speed. This approach is fast, but can lead to errors in maintaining identities. To solve this, TRex can use another option in the tracking stage, employing a machine learning approach (automatic identity correction). The convolutional neural network used is similar to the idtracker.ai identification network. However, the processing time is improved when compared with idtracker.ai's current algorithm, as TRex minimizes the variance landscape of the problem and covers all poses and lighting-conditions an individual can be in, in addition to reducing training time by changing the training process (samples used for training and using different stopping-criteria). The last stage of TRex is data visualization and exploration [35].

2.3 idtracker.ai

idtracker.ai [1] is an algorithm and software used for identifying, tracking and extracting trajectories from videos of animal collectives of up to 100 individuals. This software is species-agnostic. The algorithm uses two convolutional neural networks: one to distinguish when animals are separated from when they are crossing or touching; and another for the identification of each individual, when they are separated. A schematic of the algorithm is represented in Figure 2.1.

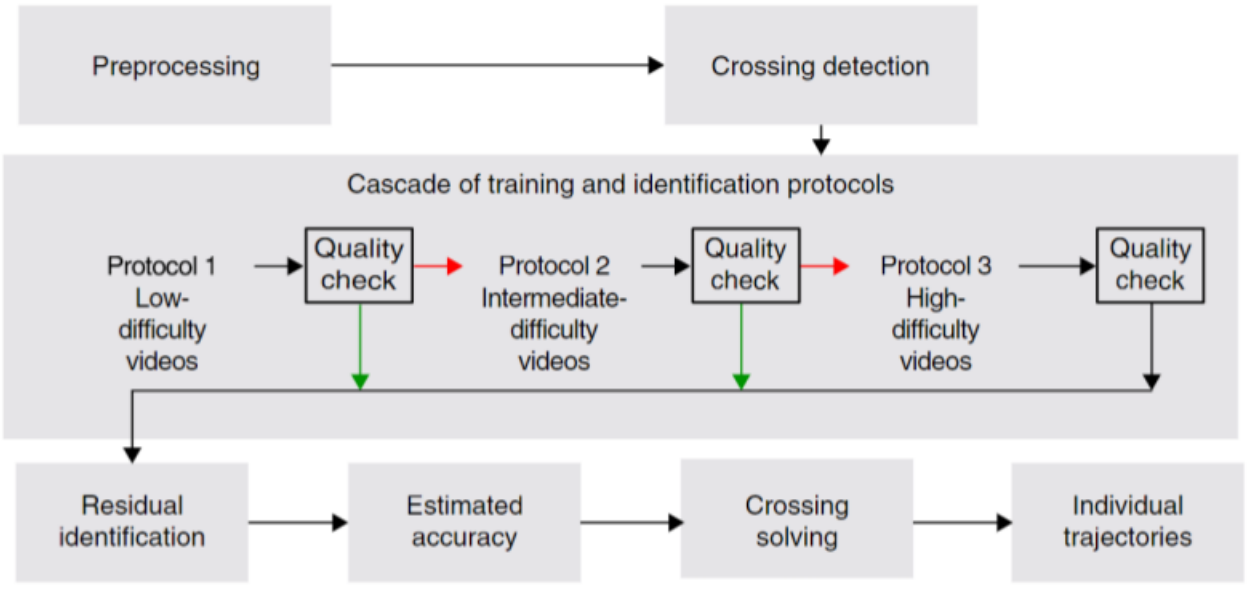


Figure 2.1: Schematic of idtracker.ai algorithm. Adapted from [1].

Firstly, the images from the video are pre-processed. This module corresponds to the extractions of 'blobs', which are areas of the video frames that correspond to single animals or crossings/touching of multiples animals; the blobs are then oriented according to the axis of maximum elongation of each blob.

The first network, the crossing detector, is trained by a set of images, labeled as individuals or crossings by a set of heuristics. These heuristics depend on the number of animals in the video, for this reason there is a need to know how many animals there are in a video. Besides this, it is essential that there are frames in the video where all animals are separated from each other. If these two criteria are not met, the network cannot be trained and subsequently cannot classify the individuals or crossings in the images.

After the images are separated into single individuals or crossing of individuals, the software constructs individual fragments, i.e. an interval of the video/consecutive frames of the same animal. By joining the individual fragments of all the individuals, idtracker.ai creates a global fragment. In this global fragment we have the interval of time/frames in which the software can identify all the individuals in the video. A global fragment is not uniform, i.e. each individual fragment does not necessarily have the same number of frames. We can observe the representation of a global fragment in Figure 2.2.

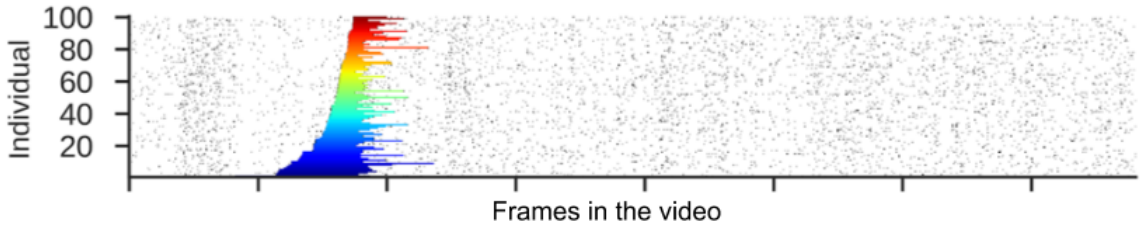


Figure 2.2: Representation of a global fragment. In all the frames from the video, idtracker.ai is able to identify intervals of time (i.e set of frames) where there are no crossings between animals, which means idtracker.ai can identify all of the animals in the video. In this representation the global fragment corresponds to the colored frames, in which each color is an individual fragment. The black dots correspond to crossings between the animals. Adapted from [1].

The second network, or identification network, uses a global fragment as an initial training dataset. Because a video can have any number of global fragments, the software chooses the best one, i.e. the one with the most variability of images for each class, for the initial training of the network. In order to achieve a good training of the identification network by using many images along the video, the training is performed using one of three protocols. The protocol used depends on the complexity of the video. For simple videos, protocol 1 is enough to train the network, whilst for more complex videos, protocol 3 will start.

In protocol 1, before starting the training of the network, the images of the best global fragment are labeled from 1 to N, (or 0 to $N-1$), with N being the number of animals in the video. These are the assigned identities the individuals will maintain throughout the tracking process. After this, idtracker.ai trains the network and assigns labels to the individual fragments that were not used for the training. The newly identified individual fragments pass through an evaluation to check how accurate their classification is. This evaluation checks how accurate the classification of each individual fragment is in relation to all the individual fragments in the global fragment.

If a global fragment passes this quality check, it is labeled as a high-quality global fragment. After all global fragments pass through this evaluation, idtracker.ai computes the percentage of identified images in the video. If this value is at least 99.95%, the cascade of protocols is interrupted. If not it starts protocol 2.

Protocol 2 retrains the network using the additional high-quality global fragments from protocol 1. Next, the network assigns labels to the remaining global fragments. This protocol repeats these steps of training the network and classifying the individual and global fragments until at least 99.5% of the images in global fragments pass the identity check explained above or there are no more acceptable global or individual fragments. After this the accumulation process stops and if at least 90% of images are identified, the cascade of protocols stops. If this is not accomplished, protocol 3 starts.

In protocol 3, idtracker.ai first pre-trains the convolutional part of the network with most of the global fragments in the video. While training the network with a global fragment, idtracker.ai assigns arbitrary labels to each individual fragment. Every time the network finished training with a global fragment, classification layers of the network are re-initialized. The training of the convolutional layers with global fragments stops when 95% of the images in global fragments have been used. After this, protocol 3 works similarly to protocol 2, where the network is trained and accumulates high-quality global fragments. Except that in the case of protocol 3, the parameters of the convolutional layers learnt during pre-training are freezed, which means only the classification layers (fully-connected layers) of the network are trained now. To understand the division of the network, Figure 2.3 represents the identification network and the division hereby mentioned.

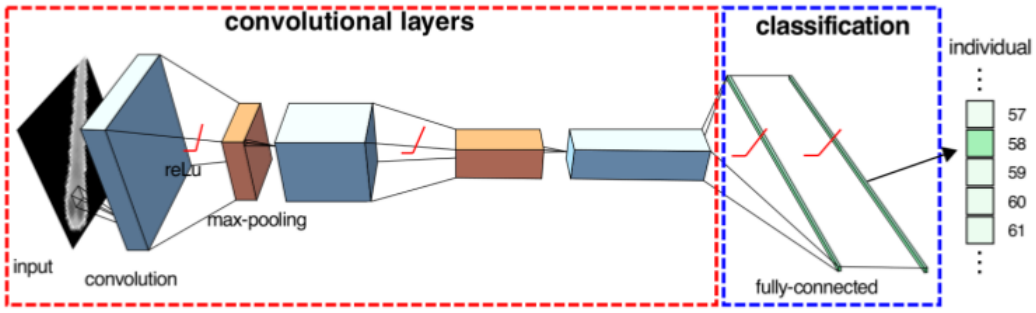


Figure 2.3: Schematic of the identification network of idtracker.ai's algorithm. Here represented is an image of a fish passing first through the convolutional layers, and then through the fully-connected (classification) layers, where it is classified as the corresponding individual. Adapted from [1].

idtracker.ai can take from around 20 minutes to 7 hours to complete the tracking and classification of each video, depending on the number of animals. These correspond to videos where idtracker.ai only needs to go into protocol 2. If the training requires protocol 3 to start, then this process can take between 4 hours to a day to track the video. This depends on the number of individuals and the number of frames per individual/size of the global and individual fragments [1].

In this project the goal is to prevent idtracker.ai of starting protocol 3. We want to make the training of the identification network more efficient for videos that require this protocol. The accuracy of idtracker.ai for 3000 images per class is between 95% and 100%, as we can see in the graph of Figure 2.4. Usually global fragments do not have 3000 images per class. What Romero *et. al* [1] did was to train the identification network using a dataset of individual fish images (used in the experiments of this work and presented in Section 4.1), in order to test the identification network. This is how they created the plot in Figure 2.4, which shows the single image identification accuracy for different numbers of individulas to identify. We are going to use the results in Figure 2.4 as reference for the experiments in the project. What we want to achieve is a similar accuracy as the one shown in Figure 2.4, but for fewer images (≈ 30 images), which would be an approximation of a small global fragment.

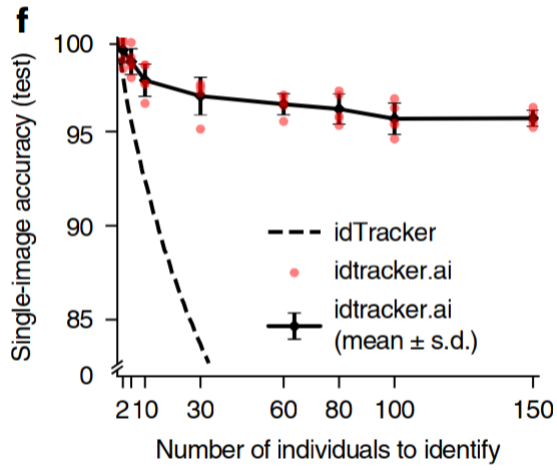


Figure 2.4: Single-image identification accuracy of idtracker.ai for 3000 images per class. This also shows the comparison with idTracker. Adapted from [1].

Chapter 3

Deep Learning and Neural Network Training Strategies

This chapter presents an explanation of what Neural Networks are (Section 3.1), the basics of NNs training and some types of NNs, with a special emphasis on Convolutional Neural Networks. After this, we explain Transfer Learning (Section 3.2), the Ensemble method (Section 3.3), and Pairwise Comparisons (Section 3.4). These deep learning methods are the ones used in the experimental part of this project.

3.1 Neural Networks

Artificial Neural Networks (ANNs) mimic a biological nervous system and the way the human brain processes information through simplified mathematical models, in order to solve computational problems. ANNs run parallel processing of information/signals to solve computational tasks. The basis of these networks starts with artificial neurons (also referred to as nodes or processing elements - PE). These are organized in layers, and connected with different weights, between layers. Each PE has weighted inputs, a transfer (activation) function and an output. The input weights change according to the learning algorithm chosen. The general architecture of an ANN consists of 3 main elements: input, hidden and output layers, which we can see in Figure 3.1. The hidden layer(s) can be more than one, depending on the data being analysed and what kind of architecture is required for a given network [36, 37].

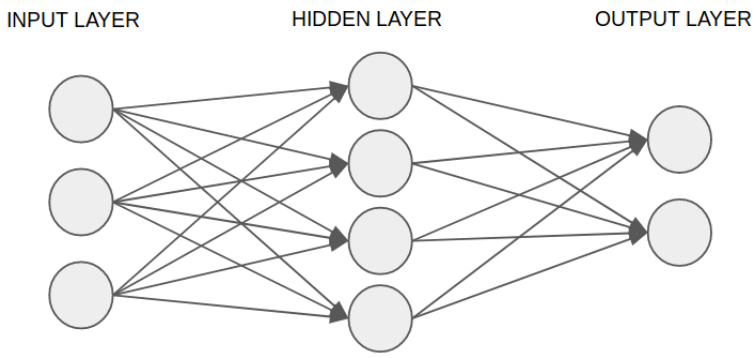


Figure 3.1: Architecture of a simple ANN. It is composed of 3 layers: the input, hidden and output layers.

We can train Neural Networks by using different types of learning methods [37]:

- Supervised - we feed the network with the input and the matching output for each data point during training (uses labeled data).
- Unsupervised - it is the opposite of supervised learning. There is no prior information on categories for the input data, so the network must learn the best way to categorize it (uses unlabeled data).
- Semi-supervised - the majority of the data does not have labels [12].
- Reinforcement - similarly to unsupervised learning, the network is trained without labeled data, and must learn the best way to solve the problem. It employs trial and error to reach a solution that maximizes the reward for that problem.

3.1.1 Neural Network Training

For a network to learn, it is important to have the right number of hidden layers and parameters. Most importantly, the number of parameters will influence how well the network learns. Using too many parameters will make the network overfit the data, in this situation, the network will perform almost perfectly (very high accuracy) on the training data, but with new data it will not have a good accuracy. The opposite can also occur, with too few parameters there might be underfitting, which also means that the network is not well-trained [36, 37].

While training a NN, its final output passes through an error function (or cost/loss function), which is used to evaluate the accuracy of the network. In classification problems, one generally used function is the Cross-Entropy function (used in idtracker.ai). After this, and in order to minimize this loss function, we use an optimizer algorithm to update the values of the weights in the network [38, 39]. Some examples of optimizer algorithms are: Stochastic Gradient Descent (SGD), Adaptive Moment (Adam) Estimation, and Adaptive Gradient Algorithm (AdaGrad) [40]. To update the weights, the optimizer needs some parameters: the learning rate, which can be a constant parameter, or be updated while the network is being trained; the rate at which the learning rate is updated; and the momentum, which helps the convergence to a minimum to be faster. The learning rate controls the step at which the model converges to its local minimum. A pass of the training data through the network for the update of the weights is called an epoch. Usually NNs are trained for more than one epoch. Each epoch can be divided in mini-batches, which means the training set does not pass the network all at once, but in subsets of the data [38, 39].

3.1.2 Types of Neural Networks

There are various types of ANNs, the simplest one is the Multilayer Perceptron, which consists of a layered feed-forward topology, where the network uses the inputs for a biased weighted sum and these inputs pass through an activation function, to produce the output [36].

Another example of ANNs is the Recurrent NN (RNN), which can be used with temporal data because it stores the last output in its memory. This means that, besides the input it also considers last instances to predict the current ones [41]. Some examples of RNN are the Hopfield Network and the Simple Recurrent Network [36].

One other type of ANN is the autoencoder. It comprises an encoder and a decoder. The network first encodes the input into a hidden representation and then decodes it, making it the same as the input. The

information gathered by the coding of the input can be used by being fed to a classifier. Because of this, autoencoders can also be used for image classification [42, 43].

The Convolutional Neural Network (CNN) is another popular ANN type, which we can find in image processing. This is the architecture that is used in the idtracker.ai model, and in most of the other models used in this dissertation. Because of this, the CNN is explained in more detail in the next section.

3.1.2.1 Convolutional Neural Network

The architecture of a CNN is a bit different from the ones described above, because it uses some different layers: the convolutional and pooling layers. The CNN ends with fully-connected layers [36, 44, 45]. In the convolutional layer(s), the NN extracts features from the images and creates convolutional/feature maps, by passing a set of filters (or convolutional kernels) through images, as observed in Figure 3.2, the passing of the filter depends on the stride value (parameter of the convolutional layer), if the stride is 1 the filter moves one pixel at a time [36, 45].

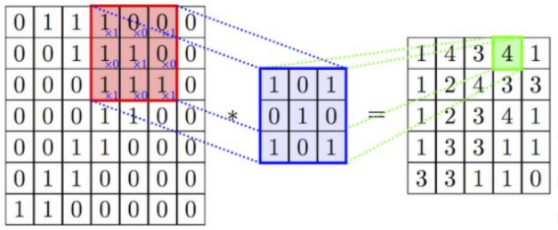


Figure 3.2: Image adapted from [45]. It represents the convolutional process. The kernel/filter (blue matrix) passes through the first matrix, and creates a feature map. The first matrix can be an image or another feature map. In this case, the size of the kernel/filter used is 3x3.

Between consecutive convolutional layers we insert pooling layers, which apply a downsampling operation, to reduce the size of the convolutional maps. Some types of pooling layers are the average pooling, the L2-norm pooling and the max pooling. The latter is the most used because it converges faster and leads to a better generalization [45, 46]. The output from these layers is the input to the fully-connected layer(s), which works like a Multilayer Perceptron [36, 44]. The general architecture of a CNN model is represented in Figure 3.3.

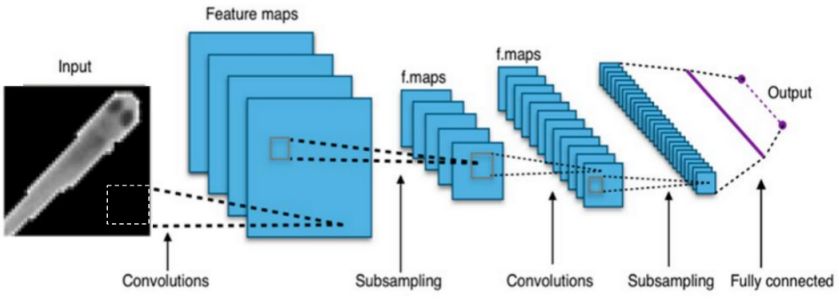


Figure 3.3: Figure adapted from [45]. Represents the general architecture of a CNN model, when applied to images. The image passes through convolutional layers, and in between, passes through subsampling, i.e. pooling layers, creating feature maps. Subsequently, the feature maps pass through the fully-connected layers and these give the output, which can be the classification of the image.

After the convolutional layers, most CNN models have an activation function, which calculates the output of a node. The most used activation function is the ReLU (Rectified Linear Units), which is the

one used in the network of idtracker.ai. Other activation functions used are: Sigmoid, Hyperbolic and Tanh. The ReLU function is described in equation 3.1, where x is a node [1, 39, 45].

$$f(x) = max(0,x) \tag{3.1}$$

A layer that some CNN models can have is the batch normalization layer. This layer normalizes the input to the subsequent layers, adjusting the activation of the nodes, by correcting them to be zero-mean and of unit standard deviation. These are usually used when training very deep NNs [47].

In some cases, after the last layer of the network, there is an activation function, used as a classification layer, which transforms the output of the last fully-connected layer into the class probability of each instance. The most common classification layer is the softmax activation function, which is the one used in the model of idtracker.ai [1, 38, 39].

3.2 Transfer Learning

Gathering enough data to train machine learning models is not always easy. In some situations the amount of images and labelling needed is impossible or too expensive to get [9, 46]. In these situations, a way to solve that problem is to use transfer learning. This method is also useful to improve the performance of models trained on datasets that are already large enough [48].

In general, in transfer learning we train a network on a base dataset and task, and the first n layers are then transferred/copied to a target network, replacing the first n layers of the target network. The remaining layers of the target network are then randomly initialized and trained on a target dataset [48]. The transferred layers can be "frozen", which means their features will not change during training on the new task; or they can be trained with the rest of the network, to fine-tune their features. This latter option helps to generalize networks better than the ones that are only trained on a target dataset [48]. For neural networks that are trained with images, the features from the first-layers are usually similar to a Gabor filter or colour blobs, i.e. in these layers, the features learned are more abstract. These are called the general features, while the features of the last-layers are more specific [46, 48].

The transfer of features between networks is more effective if the datasets are similar; otherwise, because the features are too specific, transfer learning can be more harmful than helpful [48]. Another reason that may cause transfer learning to fail is if the domains between which we are transferring are not related or are too dissimilar. An example of this is given in the work by Rosenstein *et. al* [49], where they compare the results of no transfer and transfer from similar and dissimilar individuals/data. The experiment involved 3966 labeled examples of a meeting acceptance task. Because the labeled examples came from different individuals, the experiment divided the data points into similar and dissimilar. They show that transferring data from dissimilar individuals yields a worse classification performance than no transfer at all. When the result of using transfer learning is worse than with a network where there was no transfer, we have negative transfer [9].

To make some of the next definitions clearer, we firstly present a brief explanation of the notation used forward, taken from Pan [9].

A data domain D consists of a feature space χ and a marginal probability distribution $P(X)$ where $X = \{x_1, ..., x_n\} \in \chi$, where X is a particular learning sample. For a specific data domain, $D = \{\chi, P(X)\}$, a task consists of a label space Y and an objective predictive function $f(\cdot)$, denoted by $T = \{Y, f(\cdot)\}$. For

simplicity, we consider the case where there is only one source domain

$$D_S = \{(x_{S_1}, y_{S_1}), ..., (x_{S_n}, y_{S_n})\} \tag{3.2}$$

where $x_{S_i} \in \chi_S$ is the data instance and $y_{S_i} \in Y_S$ is the corresponding class label; and one target domain

$$D_T = \{(x_{T_1}, y_{T_1}), ..., (x_{T_n}, y_{T_n})\} \tag{3.3}$$

where the input $x_{T_i} \in \chi_T$, and $y_{T_i} \in Y_T$ is the corresponding output.

Transfer learning can be divided in three main methods. For all of them, given a source domain D_S and a learning task T_S, and given a target domain D_T and a learning task T_T, the objective of knowledge transfer is to help improve the learning of the target predictive function $f(\cdot)$ in D_T using the knowledge of D_S and T_S [9].

- **Inductive transfer learning:** In this case, the target and source tasks are different ($T_T \neq T_S$), and there has to be some labeled data in D_T.
- **Transductive transfer learning:** For this $D_S \neq D_T$ and $T_S = T_T$. During training, some unlabeled data from the target domain have to be available to attain the marginal probability for the target data.
- **Unsupervised transfer learning:** $T_S \neq T_T$ and Y_S and Y_T are not observed, i.e., there is no observed labeled data in either domain. Even if the source and target tasks are different, they need to be related.

Building datasets with enough information is difficult in many fields, for example in the medical imaging domain. But with transfer learning, models pre-trained on large image based datasets can be fine-tuned and used for medical imaging tasks [50]. One of the most used datasets to train models is the ImageNet [51], which is a well-annotated dataset of natural images, with 1.2 million images of 1000 classes. Models like AlexNet and GoogLeNet were pre-trained with ImageNet and used for problems such as thoraco-abdominal lymph node detection and interstitial lung disease classification [50]. Besides medical challenges, these pre-trained models have also been used, for example, in crack detection on pavement [46]. These classification problems have different domains, but it is clear that using pre-trained models is more useful and efficient than just training neural networks from scratch [46, 50].

For this project, we used Inductive transfer learning. By using models pre-trained on ImageNet (natural images) to classify individuals in an all fish dataset (Section 4.1), we have different domain and target tasks. We use this strategy to improve the accuracy of idtracker.ai when we have few images to train the model. A more detailed explanation of our implementation and experiment is in Chapter 4.

3.3 Ensemble Learning

The ensemble learning method uses multiple trained models and combines their predictions in order to improve the original predictive performance of a single model [10, 52]. An ensemble can be comprised of different machine learning algorithms (e.g. neural networks, decision trees, linear regression models) [10]. The predictive performance of an ensemble is usually better than that of a single model, because the combination of various models compensates for any errors each individual model might have [10].

The use of ensemble methods is useful to avoid overfitting on small datasets, by averaging the various hypotheses found by the learning algorithm to predict the data. Besides this, some learners ”get stuck”

at a local minimum, and these methods prevent this from happening [10]. Another example of when ensemble methods are preferable is when the problem we have cannot be solved by just one model. This is illustrated in Figure 3.4 from [10].

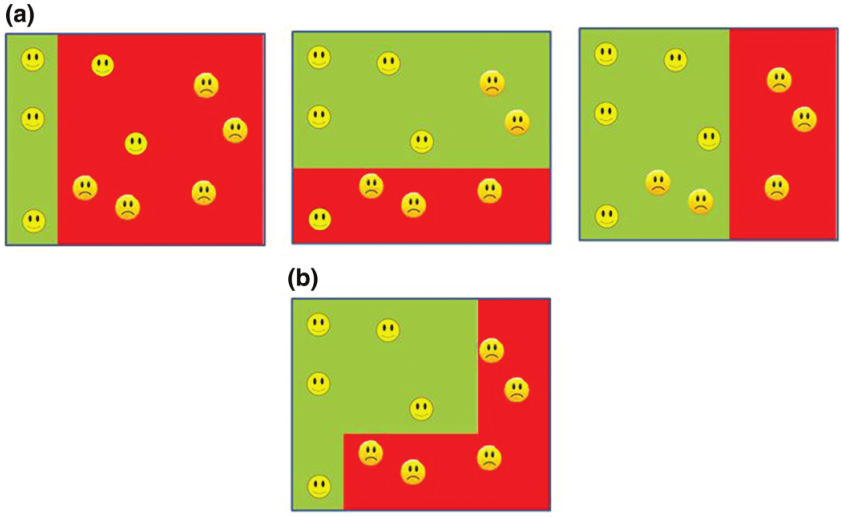


Figure 3.4: Figure adapted from [10]. In (a) we can see 3 different models that cannot represent all of the data points. In (b) we have the ensemble of the previous models, where the data are well-represented.

We can describe an ensemble model (φ) as follows:

$$\hat{y}_i = \varphi(x_i) = G(f_1, f_2, ..., f_k) \tag{3.4}$$

where x_i and y_i correspond to the input and label of the dataset, G is the aggregation function that combines the K models $f_1, f_2, ..., f_k$, in order to get a single prediction $\hat{y}_i$ [10].

Ensemble methods can have two types of output fusion methods: weighting methods or meta-learning methods. The first refers to the combination of outputs from various models, by assigning a weight to each one. The latter uses the outputs of the models as features for the meta-learner which will generate the final output [10]. One way of combining the output of the ensemble, for weighting methods, is to use majority voting, which means selecting the class that has the most votes from all of the predictions [10, 52]. This can be completed by adding the conditional probability vector from each model, and then choosing the class with the highest value in the vector, as shown in equation 3.5:

$$\hat{y} = argmax(\varphi(x_i)) \quad \text{where} \quad \varphi(x_i) = \sum_{k=1}^{K} f_k(x_i). \tag{3.5}$$

Besides this, we can assign a weight to each model, according to their predictive performance [10]. The output fusion can also be done by an unweighted average of the predictions of the models. This average can be of the output score or of the predicted probability obtained after the softmax transformation, as we can see in the equation 3.6:

$$p_{ki} = softmax(\overrightarrow{s_k})[i] = \frac{\overrightarrow{s_k}[i]}{\sum_{j=1}^{J} exp(s_k[j])} \tag{3.6}$$

where the $\overrightarrow{s_k}$ is the output score vector for the model k, $\overrightarrow{s_k}[j]$ is the score vector corresponding to the j^{th} label, and p_{ki} is the predicted probability for model k and label i. The softmax function (or normalized

exponential function) generates a K-dimensional vector of real values in the range (0,1). This corresponds to the normalization of the output of the network into a probability distribution [39]. It may be better to average after applying a softmax function, because if the models in the ensemble are different, their output scores may be in different scales of magnitude, and so there is less probability of error [52].

In the case of meta-learning, one method is called staking, where the chosen models from the ensemble are the more reliable ones. In this method we create a meta-dataset, of the same size as the original dataset, where the inputs are the outputs of the ensemble models [10].

Ensemble methods can also be divided into dependent or independent: if the output of a model affects the construction of the next one and so on, or if each model is independently constructed and trained from each other, respectively [10].

Some specific ensemble methods are:

- **AdaBoost:** this is a type of boosting algorithm that aims to improve the predictive capacity of weak learners [52]. This specific method focuses on instances that were initially misclassified by a model. Each instance of the training set has a specific weight assigned to it, which determines the focus given to it. All weights are the same at the first iteration, changing afterwards if the instance is misclassified or not. This means that if the instances are misclassified, the weight increases and if they are not, the opposite happens. Additionally, each model has an assigned weight based on their general predictive performance [10].

- **Bagging:** this is an independent method. Each model is trained using a random subsample of the original dataset, and the output of the ensemble is obtained through majority voting of the predictions of the model [10]. This method manages to reduce the variance of single learners [52].

- **Random forests:** it uses multiple independent decision trees, randomized by training each one on random samples of instances, and it selects the best splitting from a random subset of features. The fact that each tree is uncorrelated with the others allows for a good predictive performance [10]. Another method derived from this one is the **Extremely randomized trees** method, which also randomizes the splitting features cut-points when training. In this case, all decision trees are trained with the same training set [10].

In general, using an unweighted average, or majority voting, is successful in improving the predictive capacity when compared to single models [52]. It is usually better to have ensembles of different models to add more variability [10, 52]. When compared to other deep learning methods, as Monte Carlo Dropout and geometric approaches [53], ensemble methods have a better performance. A different use of ensembles appears in Tao [54], where they presented an algorithm that analyzes the features from the models, differentiating between the useful features and those which are due to overfitting. This method successfully improved the test accuracy of the ensemble network in comparison to each individual model.

The goal of the experiment based on the ensemble method (Section 4.3) was to minimize noise in the predictions of the model. In this project we use two ensemble strategies on various trained models of the same CNN (idtracker.ai model). One strategy consists of applying majority voting by using each prediction from the models. The second strategy consists of an unweighted average applied by averaging the softmax values from each model. We refer to this second strategy as 'softmax averaging'. This experiment and its implementation are explained in Chapter 4.

3.4 Pairwise Comparisons

Most classification problems are solved using multi-class labels for each sample in the dataset, as we explained above. However, there are different approaches, such as the Siamese Architecture, which has two networks with shared weights that extract features from the inputs of each network. These features are then compared to each other to analyze the similarity between the two inputs. The output is a value of similarity between the two inputs [55, 56]. This output gives a pairwise-label/pairwise comparison, which corresponds to determining if the pair of images corresponds to the same class or a different one. This type of network is widely used in face recognition, for example in companies to identify employees [57], or as it was used in [55] for detection of signature forgeries. We can find some examples of image matching or object tracking problems which apply Siamese networks in [57] and [56].

Training classifier networks using Siamese Architectures typically improves their performance. In the works of Hsu *et al.* [11, 12], the architecture of both described classifiers applied pairwise-labels to help in clustering and classification problems. In [11], Hsu *et al.* use pairwise-labels, that are fed later to the KL-divergence (Kullback-Liebler) layer. Usually, when using standard Siamese architectures it requires using second stage clustering or classification algorithms, such as K-means, to cluster or classify. What Hsu *et al.* [11] did was to use the pairwise-labels and the multi-class labels together to perform clustering or classification. By using this architecture, instead of instantiating the Siamese architecture and feeding the same data to the network multiple times (see Figure 3.5), the constraints are only present in the cost layer of the network. In this way, it is possible to obtain the clusters of the classes in a more efficient way.

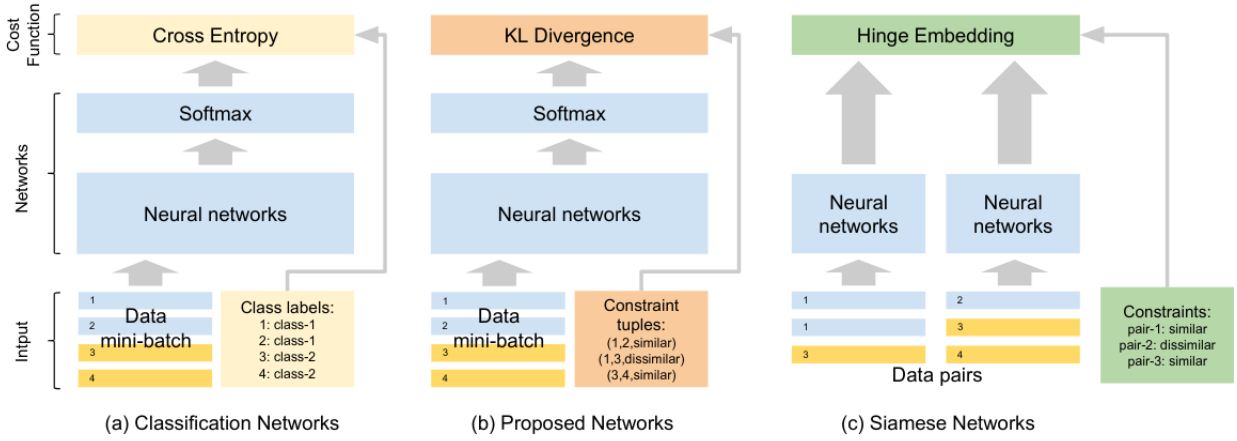


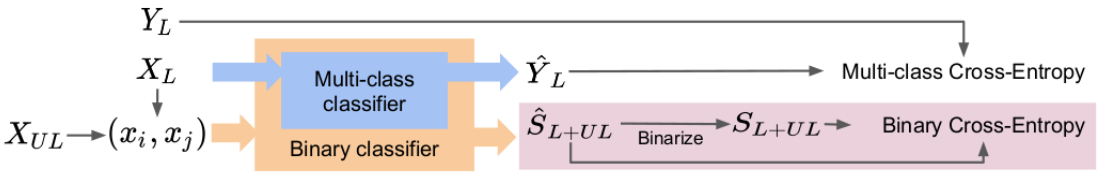
Figure 3.5: Figure adapted from [11]. Here, we compare the different architectures: (a) typical classification network, (c) siamese network, and (b) the proposed architecture of Hsu *et al.* [11]. The proposed network uses the pairwise constraints only in its last layer.

In [12], pairwise-labels are used in a semi-supervised learning architecture. This is used for a multi-classification problem, where not all the labels are available. They use pairwise-labels to create a pseudo-similarity between the two inputs. Firstly, they presented a new loss that they define as the Meta Classification Likelihood (MCL), which corresponds to:

$$L_{meta} = -\sum_{i,j} s_{ij} log \hat{s}_{ij} + (1 - s_{ij}) log(1 - \hat{s}_{ij}) \tag{3.7}$$

where s_{ij} is the similarity score between the pair (i, j) and $\hat{s}_{ij}$ is the predicted similarity, for the same pair.

This MCL loss, or Pseudo-MCL corresponds to a binary cross-entropy loss. This method is represented in the Figure 3.6, where we have a dataset D, with labeled and unlabeled portions, $D_L = (X_L, Y_L)$ and $D_{UL} = X_{UL}$, respectively. What Hsu *et al.* [12] do here is to create a pseudo-similarity S_{L+UL} by binarizing the predicted similarity $\hat{S}_{L+UL}$ at probability 0.5. And then combining the multi-class cross-entropy and the binary cross-entropy losses [12]. From the results showed in [12], this method is an improvement on the state-of-the-art methods, compared in Hsu *et al.*'s study.



(c) Pseudo-MCL for semi-supervised learning

Figure 3.6: Figure adapted from [12]. Schematic for the Pseudo-MCL training paradigm used for semi-supervised learning. X_L corresponds to the labeled data, with corresponding labels Y_L, and X_{UL} represents the unlabeled data. (x_i, x_j) are the pairs used as input for the binary classifier. $\hat{S}$ is the predicted pairwise similarity, and S is the pseudo-similarity after being binarized. $\hat{Y}_L$ corresponds to the predicted labels from the multi-class classifier.

Pairwise-labels can also be useful when applied to some other types of problems/networks, for example, in multi-criteria classification problems [58], or using pairwise ranking in problems of multi-label image classification [59].

In this dissertation, we focused on Hsu *et al.*'s [12] work, by adding this Pseudo-MCL training paradigm to the training of our network. More specifically, due to the temporal and spatial coherence of the video, *i.e.* animals in the same frame are different and animals in the same fragment are the same, idtracker.ai can create pairwise-labels just by comparing the images in the "global fragments", and without the need to have more images. This increases the amount of data available for training the network. By using the pairwise similarity combined with the multi-class classifier, we expect to have an increase in classification accuracy, specifically for the cases where we have fewer labeled images. We explain the implementation of this strategy and the experiment performed in Chapter 4.

Chapter 4

Methods and Materials

This chapter presents the dataset we used for the training and testing in each different experiment, and how we selected the images for each one, Section 4.1. This project comprises 3 different experiments that compare the accuracy and the training time of the idtracker.ai model with the new strategies. The first one uses transfer learning models to train and classify the same data as the idtracker.ai model, Section 4.2. In the second experiment we create ensembles of idtracker.ai models which we compare to only using one model or a Transfer Learning model, Section 4.3. The third experiment uses pairwise-labels in the training of the idtracker.ai model in order to add more information about the images, Section 4.4. For each experiment, we specify the architecture of the neural networks used and the training hyperparameters. We performed every experiment using Python 3.7.9 and the following libraries:

- PyTorch [1]: used for the implementation of the neural networks, the training and testing of those networks and to transfer pre-trained CNN models. The sub-libraries most used from PyTorch were torch and torchvision.
- NumPy [2]: used for mathematical computations of arrays and matrices.

4.1 Dataset

The images used for this experiment are from the individual fish image dataset presented in [1]. This dataset consists of 18,000 labeled images per 184 individuals (zebrafish), extracted from 46 videos recorded in laboratory conditions [1]. The individual images were previously pre-processed and cropped as square images. These images are in grayscale and have a size of 52x52 pixels. This dataset is referred to in this project as CARP (Champalimaud Animal Recognition Project) dataset. We show in Figure 4.1 the set-up [1] used to obtain the dataset.

For all the following experiments, the images in the dataset are randomly divided into a training set and a testing set, the sizes of which are specified in each experiment. During the training of all models, at the end of each epoch, a validation is performed in order to check if the loss of the model is at its minimum, so that we prevent overfitting and to be sure that we are selecting the best model. For this, the training set is split at the same time in training images and validation images (randomly). For all experiments the validation ratio was of 10% (of the training set). The validation images are only used to stop the training (method called Early Stopping), they are not used to update the weights or the parameters of the network. In all experiments, the weights of the networks were randomly initialized.

[1] `https://pytorch.org/docs/stable/index.html`
[2] `https://numpy.org/doc/stable/`

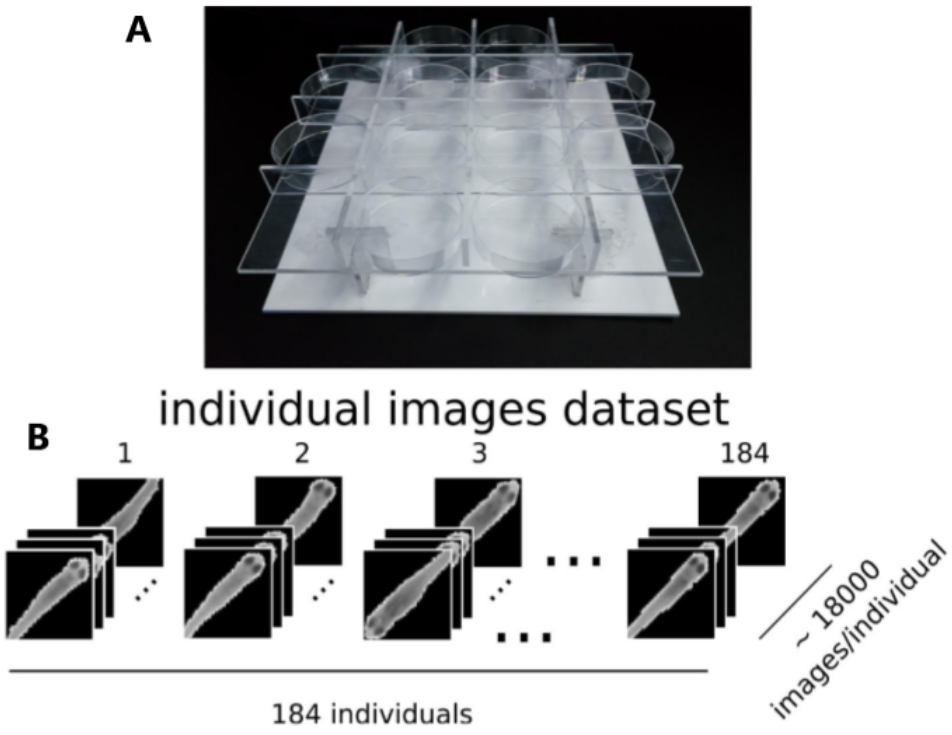


Figure 4.1: **Trainig dataset of individual images: CARP dataset.** The first panel (A) shows the holding grid used to record 184 juvenile zebrafish (TU strain, 31dpf) in separated chambers (60-mm-diameter Petri dishes). The second panel (B) shows a Summary of the individual-images dataset. Adapted from [1].

In all experiments, the accuracy of the predictions is evaluated according to the following equation [1]:

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Number\ of\ images} \tag{4.1}$$

4.2 Experiment 1: Transfer Learning Models

With the goal of improving the accuracy and training time of idtracker.ai for videos with fewer images, we performed an experiment using Transfer Learning models. We did not use full deep networks in this experiment, as the first layers are the ones with more abstract features and also for us to have a more similar size to the model of idtracker.ai, to keep the training time as small as possible. We compared the results obtained here with the ones from the model of idtracker.ai [1].

As explained in Chapter 3, transfer learning consists of using models pre-trained in a different dataset, and transferring the learned features to be used in a target dataset. More specifically, in this case, the selected models where pre-trained on ImageNet [51], and then trained and/or tested with our dataset, CARP.

4.2.1 Architecture of Neural Networks

From torchvision, the CNN models selected for this experiment were Vgg16 and Vgg16_bn [60], AlexNet [61] and ResNet18 [62]. The difference between the architecture of Vgg16 and Vgg16_bn is that the latter has batch normalization layers after the convolutional layers.

The initial idea was to only use the 3 first convolutional layers of the model, i.e. only selecting up to the 3^{rd} convolutional layer, and discard the rest of the layers from the models. This allowed mimicking

the architecture of the idtracker.ai network, and was also to prevent the training of these models to take longer than the training of the model of idtracker.ai. For some of the models, we were able to achieve this, but for others we used more convolutional layers to allow for better accuracy. The model with the more similar architecture to the one of idtracker.ai is the Vgg16 model.

The layers we use for the experiment are illustrated in Figure 4.2. Check Table A.1 and Table A.2 of Appendix A for a reference to the layers of each model that we use in this experiment. Every model had the same architecture for the fully-connected layers, illustrated in Figure 4.3, which is the same for the idtracker.ai model. Note that each of the models includes an average pooling layer between the last convolutional layer and the first fully-connected layer. This layer matches the number of output features of the last convolutional layer to the input of the first fully-connected layer.

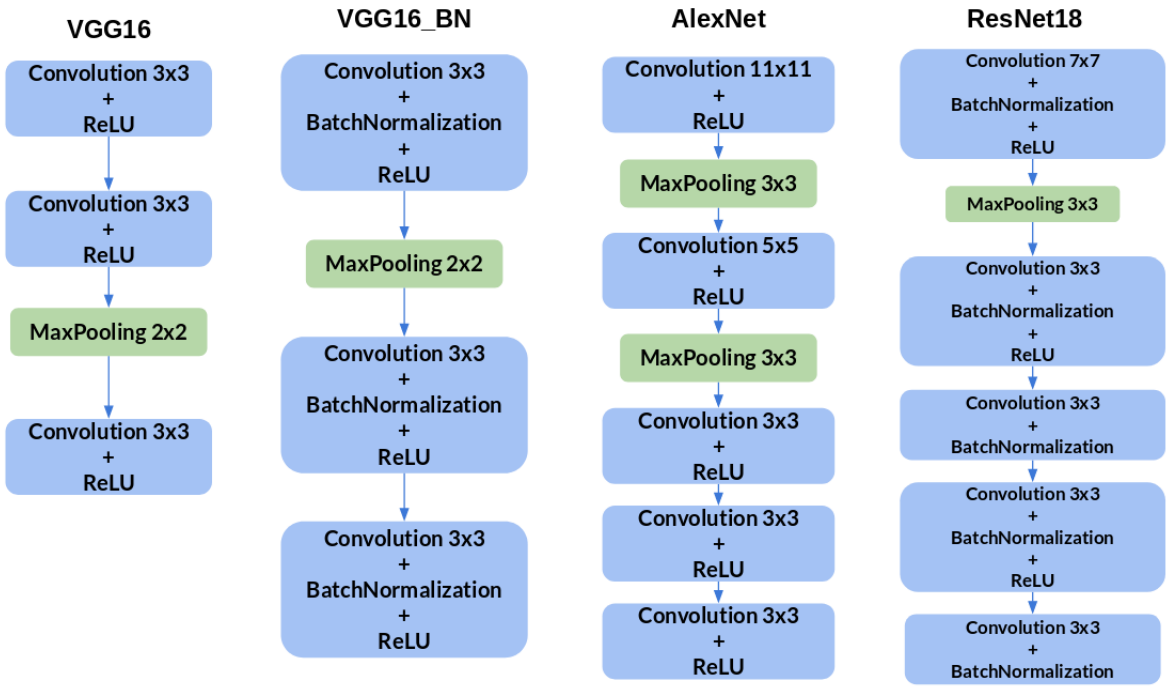


Figure 4.2: Selected convolutional layers of each pre-trained model. The models in torchvision are longer than this, their convolutional layers are described in Table A.1 and Table A.2 of Appendix A. All models have the same type of layers: convolutional layers, maxpooling layers, ReLU activation layers, and for Vgg16_bn and ResNet18 batchnormalization layers. These are all defined in Chapter 3.

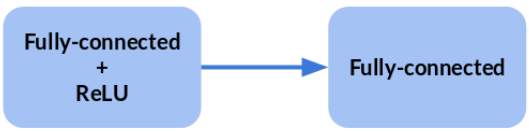


Figure 4.3: Fully-connected layers for all of the transfer learning models used in this experiment. These layers are the same as in the idtracker.ai model. They are not pre-trained like the convolutional layers.

4.2.2 Experiment Parameters

Because CARP images are grayscale, and the neural networks of these Transfer Learning models are constructed for RGB images, there was a need to perform a transformation of the images, so they could be used by these models. What we did was to repeat the number of channels of the image, so that they

have 3 instead of 1 and are compatible with the models.

As stated above, we used the following models in this experiment: Vgg16, Vgg16_bn, AlexNet and ResNet18. These models were already pre-trained, and so we divided the initial experiment in two experiments: experiment (a) using the models without training the convolutional layers (freezing them) and only training the fully-connected layers, i.e the classification part of the model; and experiment (b) fine-tuning the models, by training the convolutional layers. The goal of these was to identify in which case there were higher accuracies when classifying the images in CARP.

For both experiments, we used the optimizer SGD and the learning rate of 0.005 to match the parameters used for the idtracker.ai model. The loss function used was the Cross-Entropy Loss function (from PyTorch), a common in classification problems, which is the same loss function used for training the idtracker.ai model. The hyperparameters for both experiments are described in Table 4.1.

After experiment (a), the conclusion was that fine-tuning the models yields a better accuracy. So we performed experiment (b) with a different number of images per class for training, and compared these results with the idtracker.ai results.

Table 4.1: **Hyperparameters for Experiment 1.** Description of parameters used for training and testing with the Transfer Learning models. The number of classes refers to the number of individuals; number of images for training refers to the number of images per individual in the training set, (a) and (b) refer to the experiment in which they were used; number of images for testing refers to the number of images per individual in the testing set; validation ratio refers to the ratio of training set images used for validation during training; training and prediction batch size refer to the size of batches used for training and testing, respectively.

Parameter	**Value(s)**
Number of classes	2, 5, 10, 30, 60, 80, 100, 150
Number of images for training (a)	100
Number of images for training (b)	30, 300, 3000
Number of images for testing	1000
Validation ratio	0.1
Learning rate	0.005
Training batch size	50
Prediction batch size	100
Optimizer	SGD

Considering the multiple values for number of classes and number of images for training, there were 24 different tests for each model. In this case, a test refers to a set of conditions in Table 4.1. For example, for the first test the parameters were: number of classes: 2, and number of images for training: 30 (the rest of the parameters were always the same). For each one of the test conditions we performed 5 repetitions. A repetition of a condition consists of changing the training images, but maintaining the testing images. This is a cross-validation done to validate if the results are consistent when the network is trained with different data.

4.3 Experiment 2: Ensemble Models

By training the idtracker.ai network various times and saving a different model each time, we constructed ensembles of these models to analyse if the overall accuracy would increase by considering multiple predictions. The objective of this experiment was to improve the accuracy when we have fewer images. This of course means a longer time than training just one idtracker.ai model. So the time of training of this experiment was compared to the training times of using Transfer Learning models, to assess which strategy was the best.

The architecture of the idtracker.ai model, which we used for this experiment, is shown in Figure 4.4.

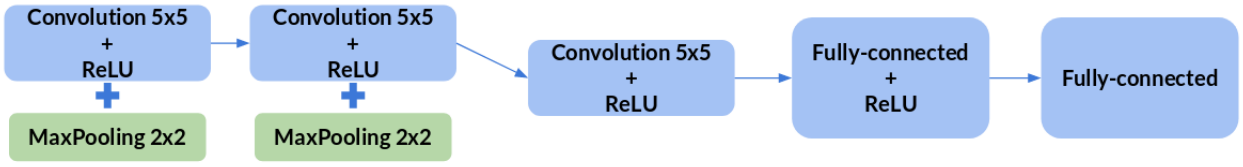


Figure 4.4: Schematic of the architecture of the idtracker.ai model.

4.3.1 Experiment Parameters

Similarly to the previous experiment, we used a SGD optimizer and a learning rate of 0.005. The loss function was again the Cross-Entropy Loss function from PyTorch. The remaining test conditions used for the training of the models are shown in Table 4.2.

For each test condition, we performed 20 repetitions, but this time, the training images used were always the same, and every model was tested using the same testing set. For each condition, we had 20 models, so we randomly chose a certain number of them (depending on the size of the ensemble) to build the ensembles. The sizes of the ensembles were of 3, 5 and 10 models, and for each ensemble size we performed 5 repetitions. At the end we have 15 ensembles for each test condition (3 sizes x 5 repetitions).

The predictions of each ensemble were computed using two different methods:

- **Majority voting:** for every image in the 'test images' we compute the prediction from each model, and we apply majority voting to all predictions, i.e. choose the prediction that happens most times. In case of a tie, we choose a random prediction from those. That value corresponds to the prediction of the ensemble;
- **Softmax averaging:** for every image in the 'test images' we compute the softmax value of each model in the ensemble, and get the average of those values. We compute the prediction of the ensemble with the average softmax value.

Table 4.2: **Hyperparameters for Experiment 2.** Description of parameters used for training and testing the models used for the ensembles. The number of classes refers to the number of individuals; number of images for training refers to the number of images per individual in the training set; number of images for testing refers to the number of images per individual in the testing set; validation ratio refers to the ratio of training set images used for validation during training; training and prediction batch size refer to the size of batches used for training and testing, respectively.

Parameter	Value(s)
Number of classes	2, 5, 10, 30, 60, 80, 100, 150
Number of images for training	30, 300
Number of images for testing	1000
Training batch size	50
Prediction batch size	1000
Optimizer	SGD
Learning rate	0.005

For each ensemble we saved the accuracy of the best and worst models, the mean accuracy of all the models in the ensemble, and the ensemble accuracy.

4.4 Experiment 3: Pairwise Comparisons

Training the network of idtracker.ai uses multi-class labeled images (multi-class labels), which correspond to the images with their corresponding class label. These images are labeled using a method called one-hot encoding. This means each image has a label vector of size equal to the number of classes. The label vector is a vector of '0's and one '1', which will correspond to the class of the corresponding image.

For this experiment we changed the training procedure so that we could also use images without multi-class labels. This new training method compares all the images in a batch, and gives them pairwise labels. These labels correspond to the comparison between two images in the batch, and informs on whether they are from the same class or not. The goal of this new training is to use the information from these pairs to improve the accuracy of classification of the network of idtracker.ai, when we have few multi-class labeled images. Additionally, we evaluate the training time considering the training of idtracker.ai without this addition. The model used in this experiment was the same represented in Figure 4.4. In Figure 4.5 we have a representation of the Pairwise method used in this experiment.

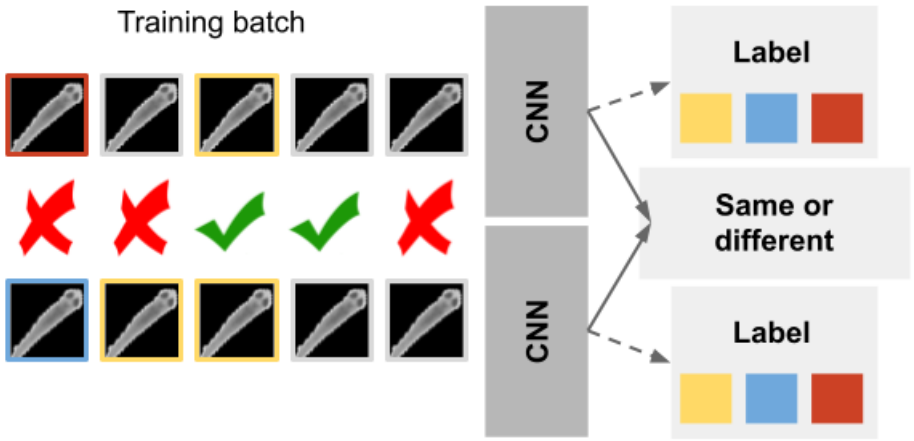


Figure 4.5: Schematic of the Pairwise method used in Experiment 3. The basis of this method is the Siamese Architecture. We use two networks (the one from idtracker.ai) and train them with multi-labeled images (the red, blue and yellow images) and with images without labels (the gray images). The images with labels are used to train the network the same way idtracker.ai trains it. The labeled and unlabeled images are all used to make pairwise comparisons, i.e we compare the images to classify them as from the same individual or from different individuals.

4.4.1 Experiment Parameters

In this experiment, besides selecting the number of images for the training and for the testing, we added a parameter which selected the number of images with multi-class labels (per class). We changed the amount of images used for training the network, but maintained the number of multi-class labeled images the same. This was to understand if training with more images, even though they do not have multi-class labels but have pairwise-labels, would improve the classification accuracy. The number of images used for training in this experiment can be a lot greater than the number used for idtracker.ai. This is because the extra images we are using do not have a multi-class label, which means it is easier to use more images with this method.

The parameters used in this experiment are in Table 4.3. Aside from adding the parameter of number of multi-class labeled images, we changed the optimizer and learning rate used for this experiment for an Adam optimizer and a learning rate of 0.001, as it yielded a better performance than the corresponding

ones used for Experiments 1 and 2.

Table 4.3: **Hyperparameters for Experiment 3.** Description of parameters used for training and testing the models used for semi-supervised testing. The number of classes refers to the number of individuals; number of multi-class labeled images refers to the amount of images per class that have a multi-class label for training and for testing; number of images for training refers to the number of images per individual in the training set; number of images for testing refers to the number of images per individual in the testing set; validation ratio refers to the ratio of training set images used for validation during training; training and prediction batch size refer to the size of batches used for training and testing, respectively.

Parameter	Value(s)
Number of classes	2, 5, 10, 30, 60, 80, 100, 150
Number of multi-class labeled images	30
Number of images for training	300, 500, 1500, 3000, 5000
Number of images for testing	1000
Training batch size	50
Prediction batch size	1000
Optimizer	Adam
Learning rate	0.001

For this experiment, we complemented the loss function used while training in the previous experiments, because the Cross-Entropy function used before does not work for images with pairwise labels. Here we used two loss functions, a Cross-Entropy (CE) loss, that only took into account the multi-class labeled images; and a Meta Classification Likelihood (MCL), that used the information from the pairwise labels. We implemented these two loss functions in python as they are described by Hsu *et al.* [12] in their code[3]. The final loss, used to check the model after validation, to check if the training can stop, is the sum of both loss functions. Because we do the division of the images per batch randomly (in this case, in PyTorch, it is random by default), some of the batches did not have multi-class labeled images during training. Because we believe that having at least some multi-class labeled images helps in the training of the network, we wanted to ensure that this happened. To resolve this, we enforce that each batch contains a certain number of multi-class labels (by using Samplers). We did this in two different ways, so we divided Experiment 3 in 2 sub-experiments, explained as follows.

4.4.2 Experiment 3a) Simple Sampler

The first Sampler separates all the images for training into multi-class labeled and pairwise labeled, and distributes the images in all batches randomly. This sampler guarantees that at least 10% of the images in a batch are multi-class labeled. For some of the test conditions, there are not enough multi-class labeled images to ensure this, so in the Sampler we used data augmentation to repeat as many multi-class labeled images as necessary, in order to have the 10% of these images in every batch.

4.4.3 Experiment 3b) Random Sampler

For the Random Sampler, the idea was to randomly choose every image for the batch. As a result, the pairs of images presented to the network in each epoch are different. Note that the number of pairwise-labels in the dataset is very large due to the amount of possible combinations. Thus, this method increases the amount of information presented to the network.

This sampler is implemented in the following way: the images for training are divided into multi-class labeled and pairwise-labeled; the images are chosen randomly and with a probability of 10% for

[3]Available at `https://github.com/GT-RIPL/L2C`

the image to be multi-class labeled. When a batch does not have multi-class labeled images, the CE loss does not contribute to the total loss of the batch.

For the experiments and tests explained before, the stopping criteria used in the training is the same of the idtracker.ai. However, for this specific experiment, we initially used the same stopping criterium, but realized that it did not work as well as it did for the rest of the experiments. Hence, for this experiment we used a different stopping criteria function, called Early Stopping [4].

Another difference in this experiment was the computer where we run the tests. At the time of this experiment, we were using a different computer; although both computers had the same specifications: Linux Mint 19.2 64-bit (quad core, Intel Core i7-4790, 32 GB RAM, TItan X GPU); and the difference in times of training are not substantial, so the results are still comparable.

[4] Available at: `https://github.com/Bjarten/early-stopping-pytorch/blob/master/pytorchtools.py`

Chapter 5

Results and Discussion

This chapter presents the results of the experiments described in Chapter 4 and a discussion of their meaning. Firstly, in Section 5.1, we show and explain the results of Experiment 1 regarding the use of Transfer Learning models in idtracker.ai. Secondly, in Section 5.2, we show the results for Experiment 2 about the use of the Ensemble Method with idtracker.ai models. Finally, in Section 5.3, we explain the results of Experiment 3 where we introduced pairwise comparisons in the training of the idtracker.ai model.

The results analysed here are of accuracy of test and the training time, for all of the experiments. The training time is measured in seconds, and the accuracy is evaluated according to equation 4.1. We compare time and accuracy because this is what we are trying to improve with this project. To have a better accuracy than the one idtracker.ai can achieve, and to reduce the time it takes for the network of idtracker.ai to train.

In all experiments we compare their results with the results from using the training strategy and model from idtracker.ai, used for single image identification [1]. A figure showing the accuracy of this strategy, for a specific set of training and testing hyperparameters, is presented in Section 2.3. In the following sections, we refer to this training strategy simply as *idtracker.ai*.

5.1 Transfer Learning Models

In Figure 5.1, we compare the accuracies of the Transfer Learning models with the accuracy of idtracker.ai. For Vgg16, the accuracy is always better than that of idtracker.ai, for all numbers of images per class. We observe a similar result for the Vgg16_bn model, which also uses a VGG network, but has the addition of batch normalization. However, for Vgg16_bn, the accuracy drops to values below the mean of idtracker.ai, for 150 individuals and 30 images per individual. In the case of AlexNet, this network is only better when training with 30 images per class, but for the other tests the accuracy is very similar to the one of idtracker.ai. The ResNet18 model has a very low accuracy when using 30 images per class while training. For 300 images there is some oscillation in the accuracy values, in some cases the accuracy is a bit better than idtracker.ai, but in others it is lower. We can explain the drop in accuracy for 60 and 80 individuals with the fact that for some repetitions (training ”runs”) the network did not converge while training, resulting in lower accuracies. This is also why the error bar for 2 individuals is bigger than most error bars, as some of the repetitions for this test condition did not converge. And for 3000 images the accuracy is slightly better than that of idtracker.ai.

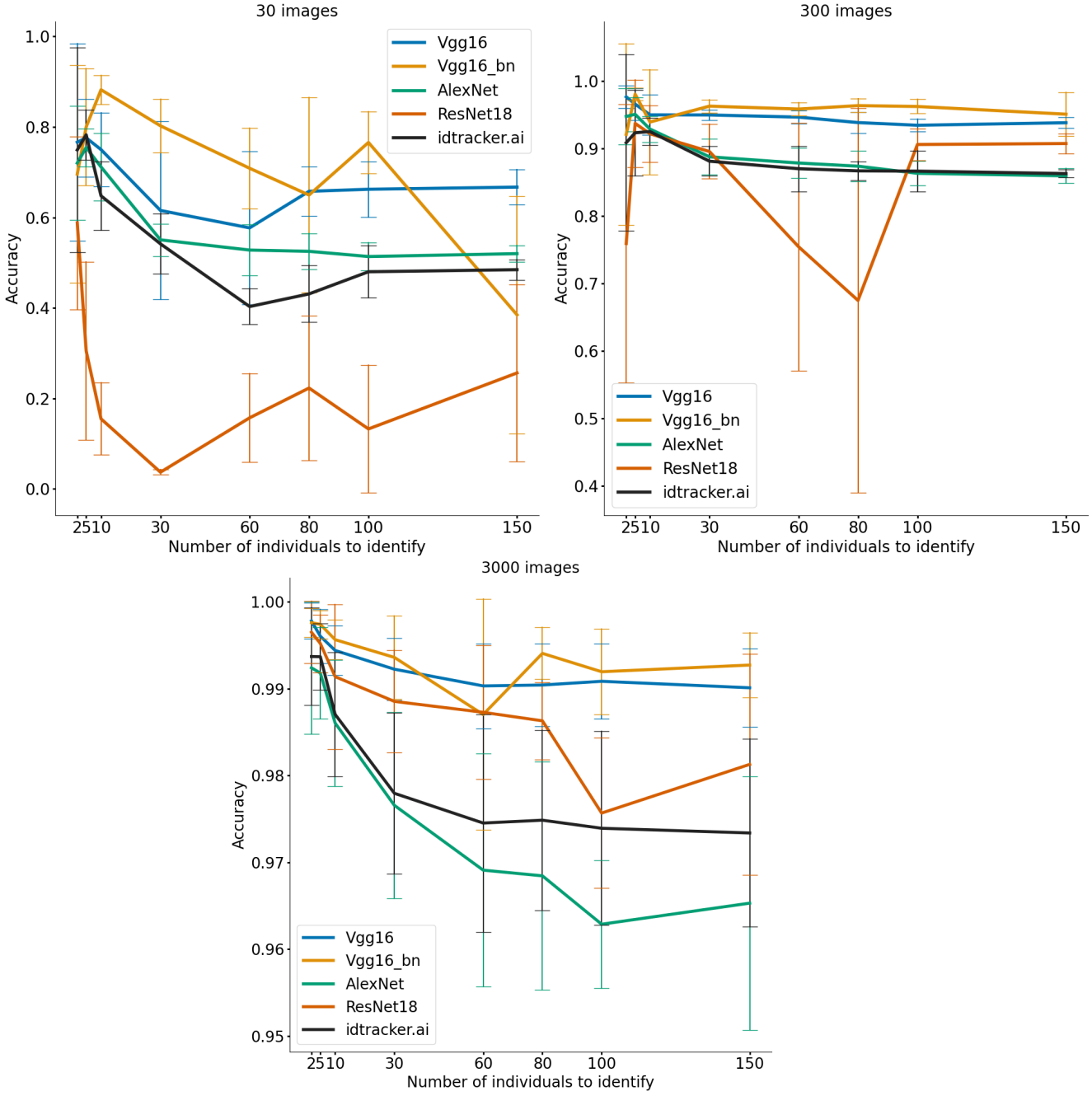


Figure 5.1: Test accuracy of the Transfer Learning models (Vgg16, Vgg16_bn, AlexNet and ResNet18), compared to the test accuracy of the idtracker.ai model, for 30, 300 and 3000 images per individual for training. The error bars are calculated as *mean* $\pm$ *standard deviation*.

The first objective of using Transfer Learning models in idtracker.ai, instead of using the idtracker.ai network, was to improve the accuracy of image classification. Specifically for the case with fewer images (≈ 30 images), where idtracker.ai does not perform as well as for 3000 images. We can see this improvement in the majority of these Transfer Learning models. In the case of the ResNet18 model, the accuracy is very low. This can be because it is a deeper neural network than the others, and the fact that we are only using the first few layers of it appears not to be enough. AlexNet is very similar to idtracker.ai, and better for fewer images, which was the goal. But with 3000 images, the model yields slightly lower accuracy than idtracker.ai, in most cases. Vgg16 and Vgg16_bn have overall better accuracies than idtracker.ai. Vgg16_bn can outperform Vgg16 for some cases, but appears to have a higher variability.

Taking the accuracy values into consideration, we can exclude the ResNet18 model. Hence we compared the training times for the other three models, in Figure 5.2. We observe that the training time is always larger for the Transfer Learning models than for the idtracker.ai model. In the case of AlexNet, it is almost the same for most cases, the bigger difference is for 150 classes and 3000 images per class. In Appendix B, we can observe the comparison of all of the Transfer Learning models' training time, in Figure B.1.

For the first goal we found at least one model that improves in accuracy over idtracker.ai in all tests, Vgg16. But in terms of reducing training time, we did not manage to find a model which was faster than the one of idtracker.ai, and had a higher accuracy. Among the best models we have, considering accuracies, we can see that the training time is different for all of them. The Vgg16 and Vgg16_bn models are the ones which achieved better accuracies, but they are also much slower than idtracker.ai, specially the Vgg16_bn model. The training time of Vgg16 is in the middle between idtracker.ai and Vgg16_bn. The training time of AlexNet is very similar to the idtracker.ai model. This can be because the models are of a similar size, and the number of epochs they take for each training is also similar.

Taking into account the accuracy values and the training times of all Transfer Learning models, Vgg16 was the one with the more significant improvement over idtracker.ai, as it seems to vary less than Vgg16_bn. The difference in training time between Vgg16 and Vgg16_bn is due to the fact that Vgg16_bn trains for more epochs than Vgg16. The training time of Vgg16 was expected to be the closest to the idtracker.ai network, because the networks' architectures are the most similar. But to reach the better accuracy values, Vgg16 trains for more epochs than idtracker.ai. Still, this model is the best among the ones tested in this experiment, when we weigh the test accuracy and the training time.

It is important to understand that by always using the same hyperparameters to train all networks we might not be using the best ones for some of them. Although changing some hyperparameters (the batch size or the learning rate, for example) might fix some of the situations where a network did not converge while training, or when the network needed more epochs during training than idtracker.ai. But for this experiment, there was not enough time in the dissertation project to evaluate such differences between networks, has we are dealing with various networks.

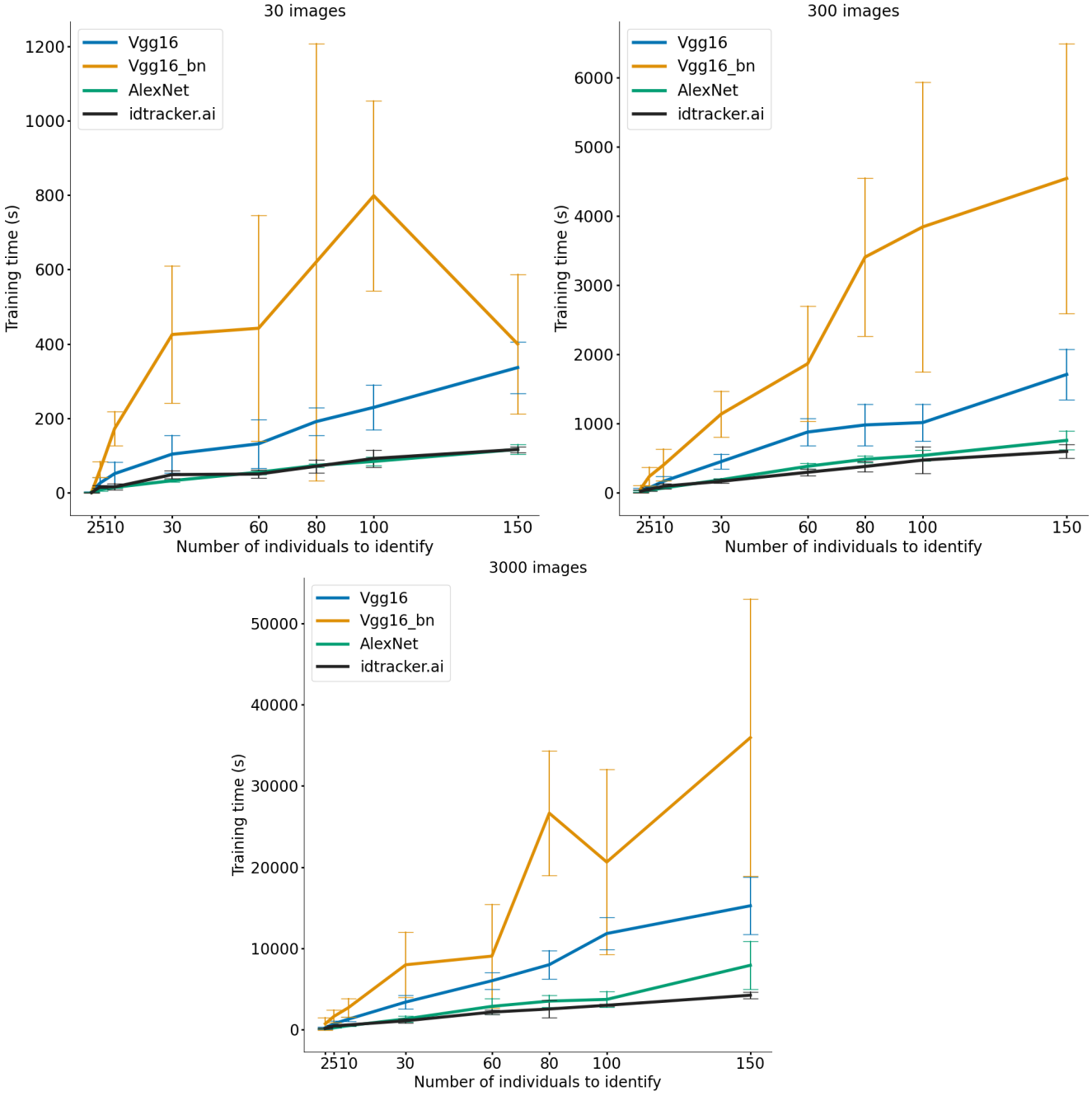


Figure 5.2: Training time of the Transfer Learning models (Vgg16, Vgg16_bn and AlexNet), compared to the training time of the idtracker.ai model, for 30, 300 and 3000 images per individual for training. The error bars are calculated as *mean* $\pm$ *standard deviation*.

5.2 Ensemble Models

From the last experiment, we see that the Vgg16 model is the best following the strategy of Transfer Learning. With this in mind, we decided to compare the results of the Vgg16 model to the results of the ensembles from this experiment.

In Figure 5.3, we can observe that the accuracy of the Vgg16 model is still better than the ensembles in most of the cases, with some variance for smaller classes. Besides that, we see that the computation of the predictions of the ensembles using the *softmax averaging* instead of *majority voting* yields more accurate results. From the results of the ensembles for 300 images (Figure 5.4), we see that the Vgg16 model's accuracy always outperforms that of the ensembles, even for ensembles of 10 models. In Figures 5.3 and 5.4, we have plotted the mean, worst and best models in the ensemble. In this experiment the models for the ensembles are of idtracker.ai, so instead of comparing to the idtracker.ai results for 30 and 300 images per class for training, as in the other cases, we made the comparison stated before. Comparing to the best model in the ensemble is, in this case, the same as comparing to the idtracker.ai results as we did for the previous experiment.

As expected, the training time of the ensembles (Figure 5.5) increases with the size of the ensemble (number of models). Comparing the training time of the ensembles to the Vgg16 model, we see that it is faster than the ensembles of size 5 and 10. For ensembles of size 3, the training time is very similar.

Using an ensemble of idtracker.ai models instead of just one model does improve the accuracy of single image classification. When we compare with the improvement achieved when using the Vgg16 model, it is not as large of an improvement. Besides this, the training time of any ensemble size is similar or superior to the Vgg16 model.

The expectation of using ensembles of idtracker.ai models was to reduce the variability of the test set accuracy between training repetitions. When we compare the accuracy of the best and the worst models in the ensembles, there is a clear difference in the results. This happens for any ensemble size and for both 30 and 300 images per class during training. This variability in results is what we tried to remove. Removing this variability was successful for both methods of getting the ensemble predictions. However, instead of making the training process faster we added more time to it.

In this experiment we only use the same type of model/architecture (idtracker.ai model) for all the ensembles. Some machine learning problems are solved with greater ease if we can use multiple models/different architectures to fit the data available. In this case, there might be the possibility that by joining the predictions of different models, besides the idtracker.ai one, will lead to a bigger improvement of the classification accuracy. However, even if this happens, the training time is probably going to be longer than desired, as we will still be training more than one model. As it stands, the ensemble method does not seems to be the solution for our problem.

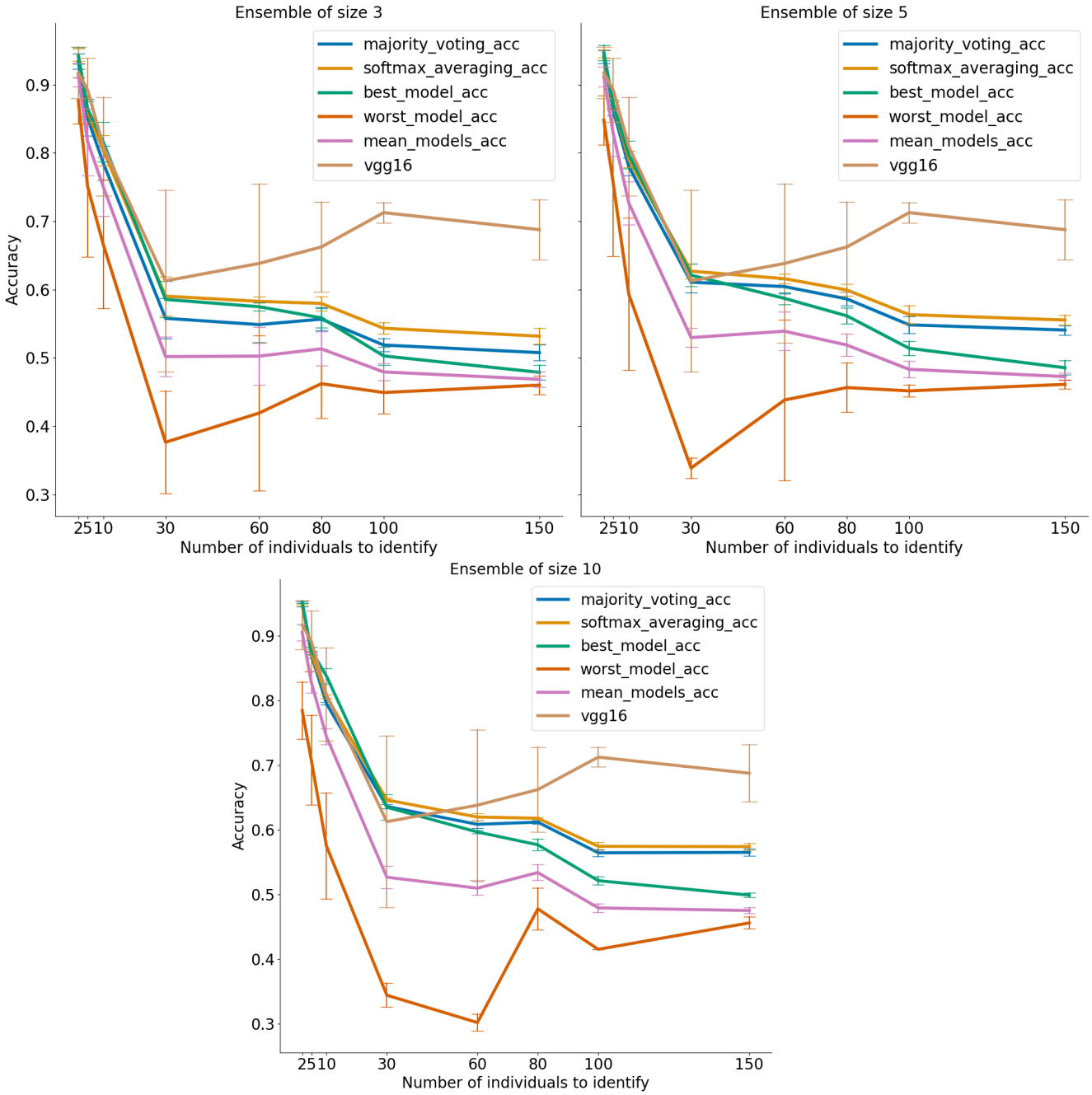


Figure 5.3: Accuracy of ensembles of size 3, 5 and 10, compared to the accuracy of Vgg16 model, for 30 images per class. Both accuracies computed (majority voting and softmax averaging) are compared to the accuracy of the best and the worst model in the ensemble and to the mean accuracy of all the models in the ensemble. The values showed are a mean of 5 repetitions for each ensemble size. The error bars are calculated as $mean \pm standard\ deviation$.

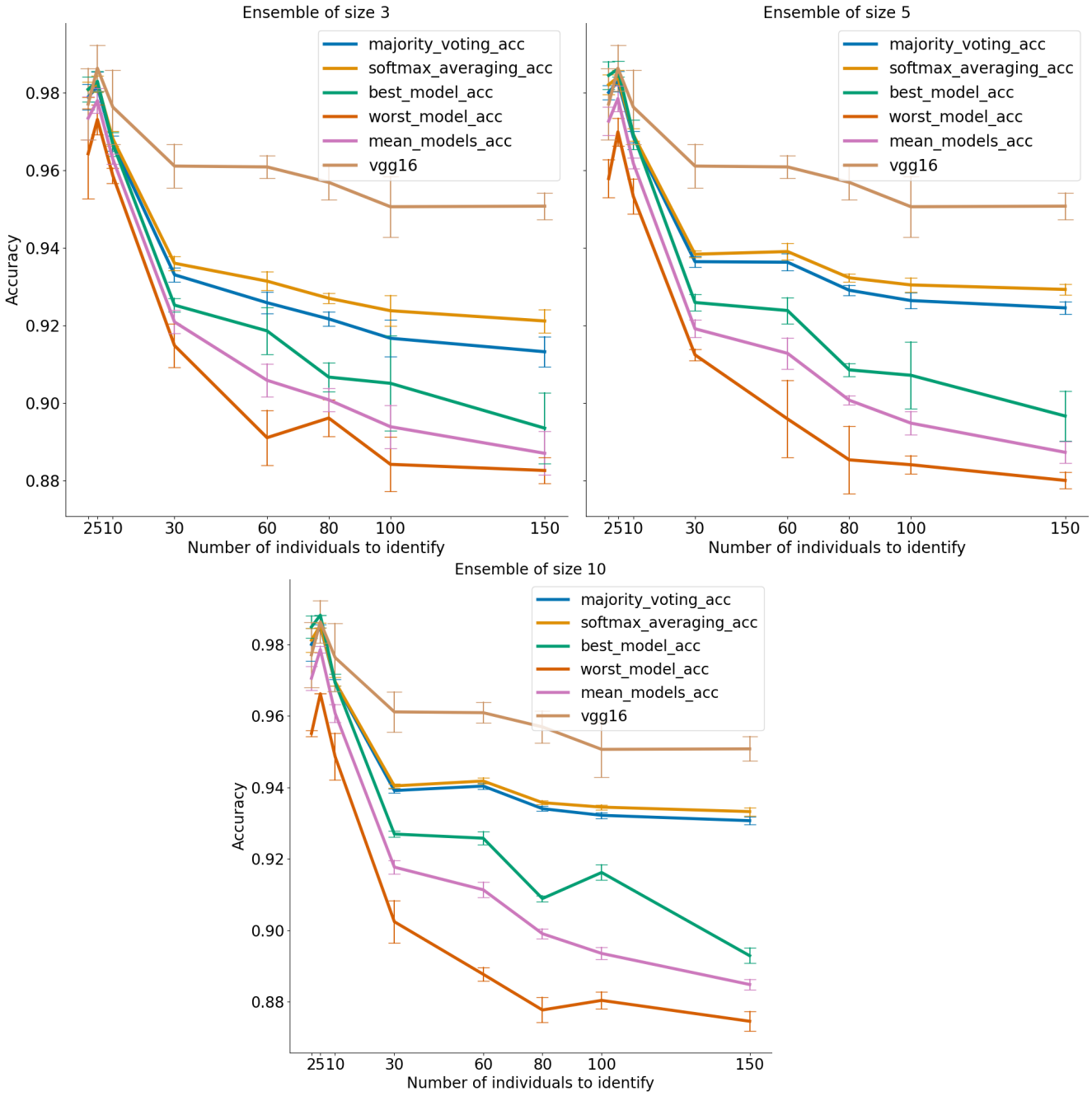


Figure 5.4: Accuracy of ensembles of size 3, 5 and 10, compared to the accuracy of Vgg16 model, for 300 images per class. Both accuracies computed (majority voting and softmax averaging) are compared to the accuracy of the best and the worst model in the ensemble and to the mean accuracy of all the models in the ensemble. The values showed are a mean of 5 repetitions for each ensemble size. The error bars are calculated as $mean \pm standard\ deviation$.

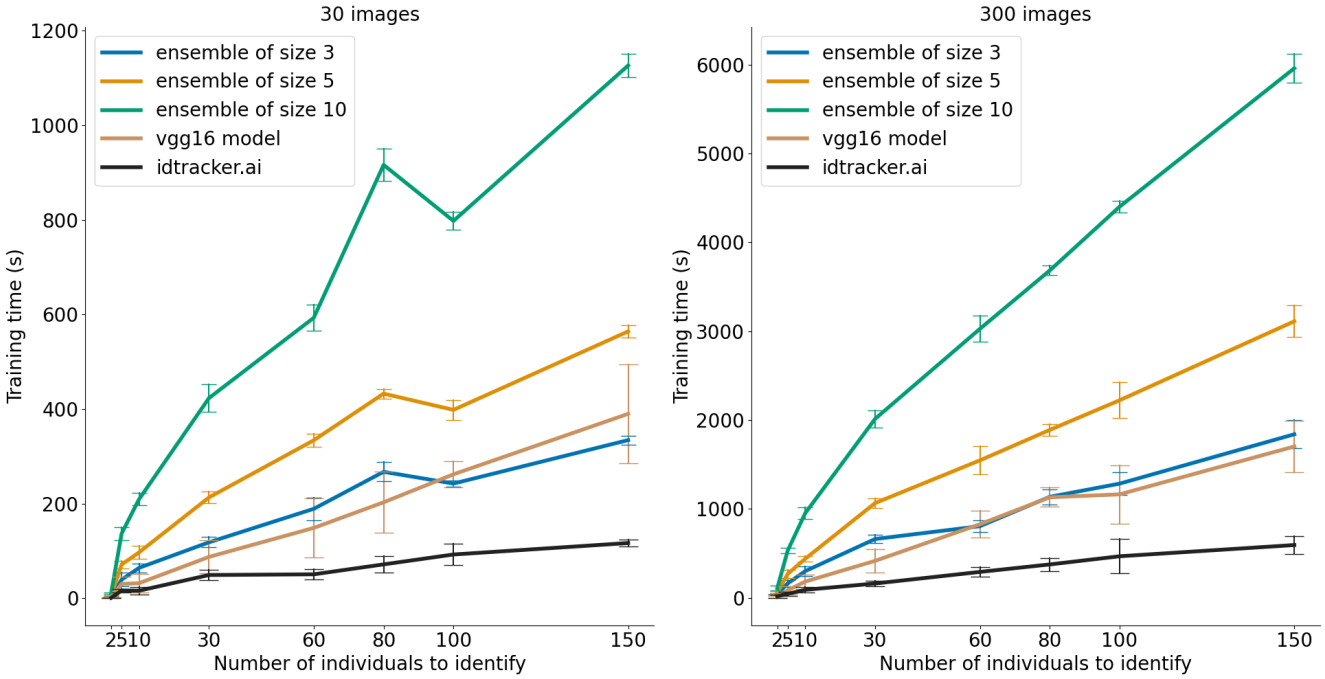


Figure 5.5: Comparison of training time for ensemble sizes of 3, 5 and 10 and with the model Vgg16, for 30 and 300 images per class, during training. The error bars are calculated as $mean \pm standard\ deviation$.

5.3 Pairwise Comparisons

Figures 5.6 and 5.7 show the comparison between the training method used in the algorithm of idtracker.ai (henceforth referred to simply as *idtracker.ai*) and the new training methods described in Section 4.4, regarding image classification accuracy. We show the results when using the two batch construction strategies and the new pairwise comparison training strategy, described in Section 4.4. We refer to both experiments as the *Simple Sampler* and the *Random Sampler*. For these experiments, we used 300 and 3000 images per class for training. All images could be used to construct pairwise labels, but 30 images per class had multi-class labels. Since the idtracker.ai training method only uses multi-class labeled images, in this case we compare against an idtracker.ai trained with only 30 images per class. As we explained in Section 4.4, we also experimented with other numbers of images per class for training. The results of all the experiments are in Figure C.1 and Figure C.3, in Appendix C.

Analysing Figure 5.6 we observe that the accuracy in image classification improves with the addition of more images per class for training. This addition results in more pairwise comparisons, i.e. more information about the individuals. For tests using 300 images per class and with 2 to 60 classes, the accuracy of the experiments using the Simple Sampler is better than the accuracy of idtracker.ai. For tests with 80 to 150 classes the accuracy drops and is worse than the accuracy of idtracker.ai. A way of improving this accuracy could be to increase the percentage of multi-class labeled images in each batch.

The variability between the repetitions of each condition is usually larger than the one we have with idtracker.ai. This variability can be due to some tests where the model did not converge while training, which results in test accuracy values lower than expected.

In Figure 5.7, we show that we also have an improvement in accuracy when we use more images for training. And that using the Random Sampler yields better results than the Simple Sampler, especially when we have more classes. The variability between test conditions is also lower than before. The accuracy of image classification is always better than the idtracker.ai results. The differences of the

Random Sampler (explained in Section 4.4.3) allowed to improve the accuracy in the situations where the Simple Sampler did not have a good performance. The bigger stability in accuracy (smaller error bars), when compared to the Simple Sampler, can be due to the higher number of different pairwise comparisons generated by the Random Sampler. Note that while with the Simple Sampler we randomize the images only once before starting the training, with the Random Sampler, we randomize the order of the images in each epoch. Because of this, in each epoch the pairs of images compared are different and as a result the network gets many more different pairwise comparisons.

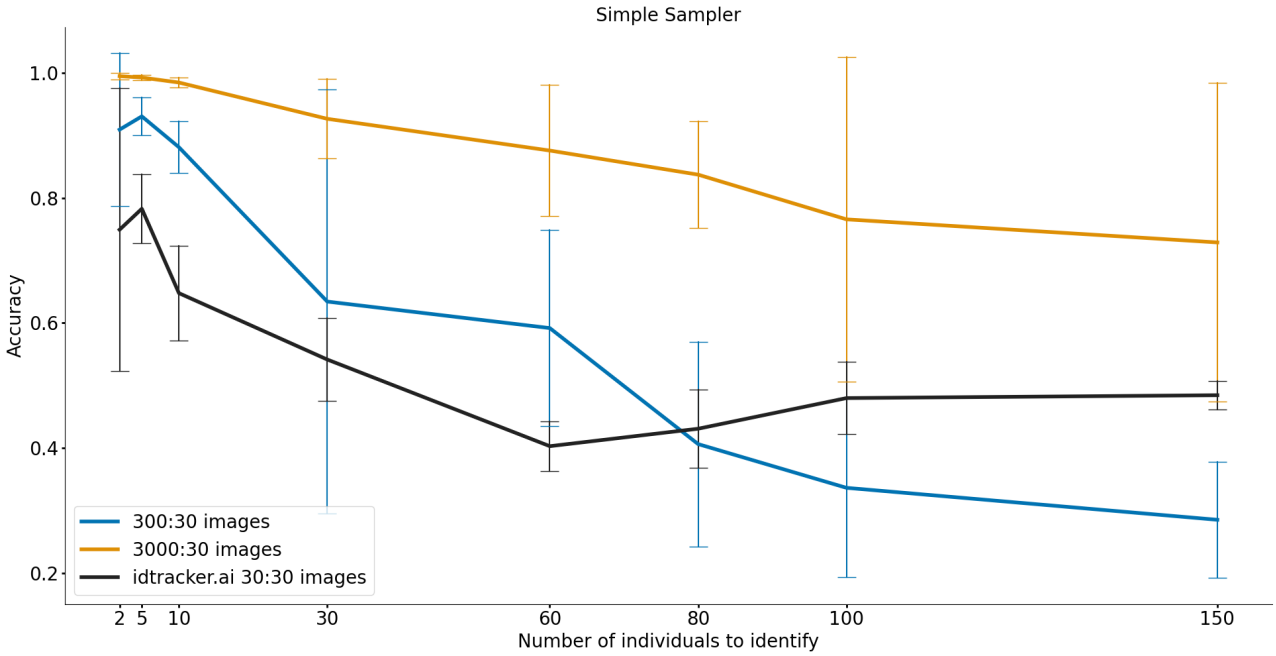


Figure 5.6: Test accuracy when using the pairwise method and the Simple Sampler described in Section 4.4.2, for 300 and 3000 images per class for training, of which 30 images (per class) are multi-labeled. These results are compared to the idtracker.ai test accuracy for 30 images per class for training. The number of images referenced in the legend correspond to *number of images per class: number of multi-class labeled images per class*. The results here represented correspond to the mean and standard deviation of 5 repetitions for every test condition. The error bars are calculated as $mean \pm standard\ deviation$. [1]

The time spent for training using these two methods is showed in Figure 5.8 and Figure 5.9. These show the times for the same cases showed above in the figures related to the test accuracy (Figure 5.6 and Figure 5.7). In Appendix C, Figures C.4 and C.6 show the results for training time for all the tests performed.

For the Simple Sampler experiment, the training time was always longer than for idtracker.ai, which was expected, as the number of images used for the training is always larger. When training with 300 images per class, the training time is not so different from the one of idtracker.ai. With 3000 images per class, the time is considerably larger, and the variability in time shows that some of the repetitions (training "runs") were very slow while others where faster (which can be observed more clearly in Appendix C in Figure C.5). Since it makes sense for the training to take longer for 3000 images per class, the cases where the training was faster might be from situations where the network did not converge when training, i.e. it stopped training before converging, which lead to the cases where the accuracy in Figure

[1] The accuracy is never above 1 for each training run/repetition. The fact that some error bars are above 1 is most likely due to tests where the model did not converge while training, resulting in lower accuracy values and higher standard deviations. But considering that for this experiment, tests where the model did not converge happened multiple times, instead of removing those tests, we decided to show them, in order to explain that this sampler is not reliable as it was computed for this dissertation. In Figure C.2 in Appendix C, we observe a representation of the accuracy in every test, in order to show what we explained in this footnote.

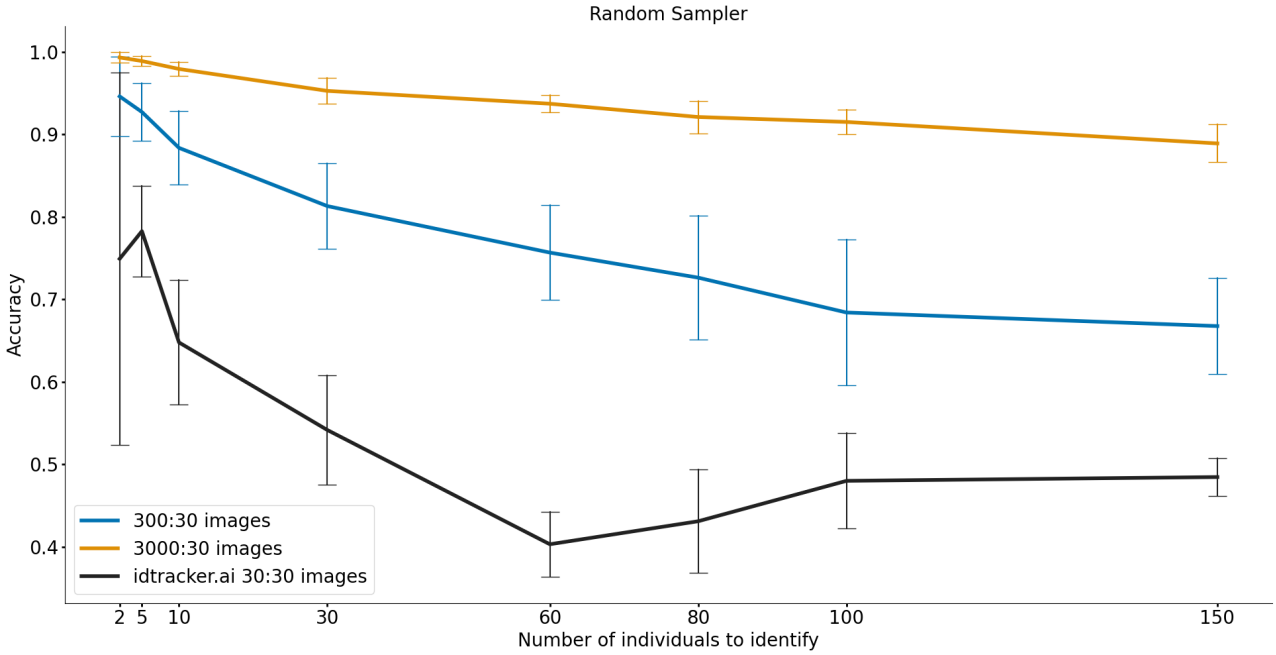


Figure 5.7: Test accuracy when using the pairwise method and the Random Sampler described in Section 4.4.3, for 300 and 3000 images per class for training, of which 30 images (per class) are multi-labeled. These results are compared to the idtracker.ai test accuracy for 30 images per class for training. The number of images referenced in the legend correspond to *number of images per class: number of multi-class labeled images per class*. The results here represented correspond to the mean and standard deviation of 5 repetitions for every test condition. The error bars are calculated as $mean \pm standard\ deviation$. In some test conditions one of the repetitions was not used here, as the value of accuracy was close to 0%, because the model was not able to learn the image features.

5.6 is lower than average. The training time of the model seems to also be related to the values of the initial loss that the model presents when training. Some repetitions of the same test conditions have an initial loss higher than others. To illustrate the way the loss converges, or not, please refer to Appendix C, Figure C.7.

In Figure 5.9, as expected, the training time for the Random Sampler experiment is also larger than that of idtracker.ai, again because of the larger number of images per class for training. The training time when using 3000 images per class varies more than when using 300 images, especially for the tests with a larger number of classes (100 and 150 classes). A possible explanation for this is the fact that in some repetitions (training ”runs”) the training converges faster (check Appendix C, Figure C.8). There are two sources of randomness, the initialization of the weights and the order of the images in the epochs. Because of this, different training repetitions have different initial conditions, which can make them converge faster. A possible way to solve the fact that for some tests the models converged faster is by changing the probability of having a multi-class label in the batch. For these tests the probability was at 10%, but maybe with a higher value we can achieve better results.

In this experiment, we achieved our goal of improving the image classification accuracy of idtracker.ai for few multi-labeled images. The added information of comparing the image pairs results in an increase of accuracy. The fact that the training time is larger is expected. This is because we are training with 10 or 100 times more images per class, with a minimum of 300 images per class. This fact explains why training times are more similar when compared to idtracker.ai trained with 3000 images (Figure 5.1) than when idtracker.ai is trained with 30 images.

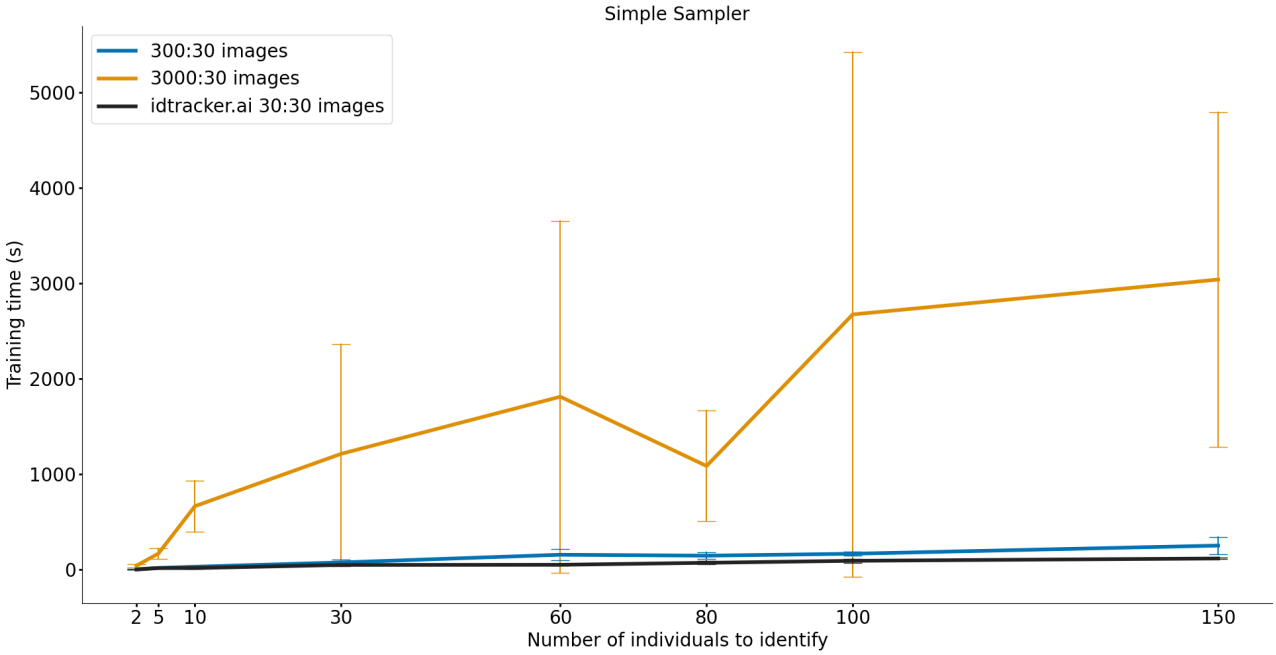


Figure 5.8: Training time when using the pairwise method and the Simple Sampler described in Section 4.4.2, for 300 and 3000 images per class for training, of which 30 images (per class) are multi-labeled. These results are compared to the idtracker.ai training time for 30 images per class for training. The number of images referenced in the legend correspond to *number of images per class: number of multi-class labeled images per class*. The results here represented correspond to the mean and standard deviation of 5 repetitions for every test condition. The error bars are calculated as $mean \pm standard\ deviation$. [2]

Even with these longer training times, the Random Sampler experiment still shows a good result, as we believe it is possible to avoid entering into protocol 3 when training the identification network of the idtracker.ai algorithm, which is the main goal. As we said before, in Section 2.3, protocol 3 is too slow, because of the pre-training of the convolutional layers of idtracker.ai's model, before the training of the full network and the accumulation of global fragments. By avoiding protocol 3, with this training method, we may maintain longer training times, but have better accuracies, allowing idtracker.ai to accumulate global fragments in a more efficient way.

[2]Similarly to what happens with the accuracy in Figure 5.6, the time of training cannot be below the 0 mark, this is just the representation of the standard deviation values that appears like this because of the odd values (faster training "runs"), corresponding to the tests where the model did not train properly.

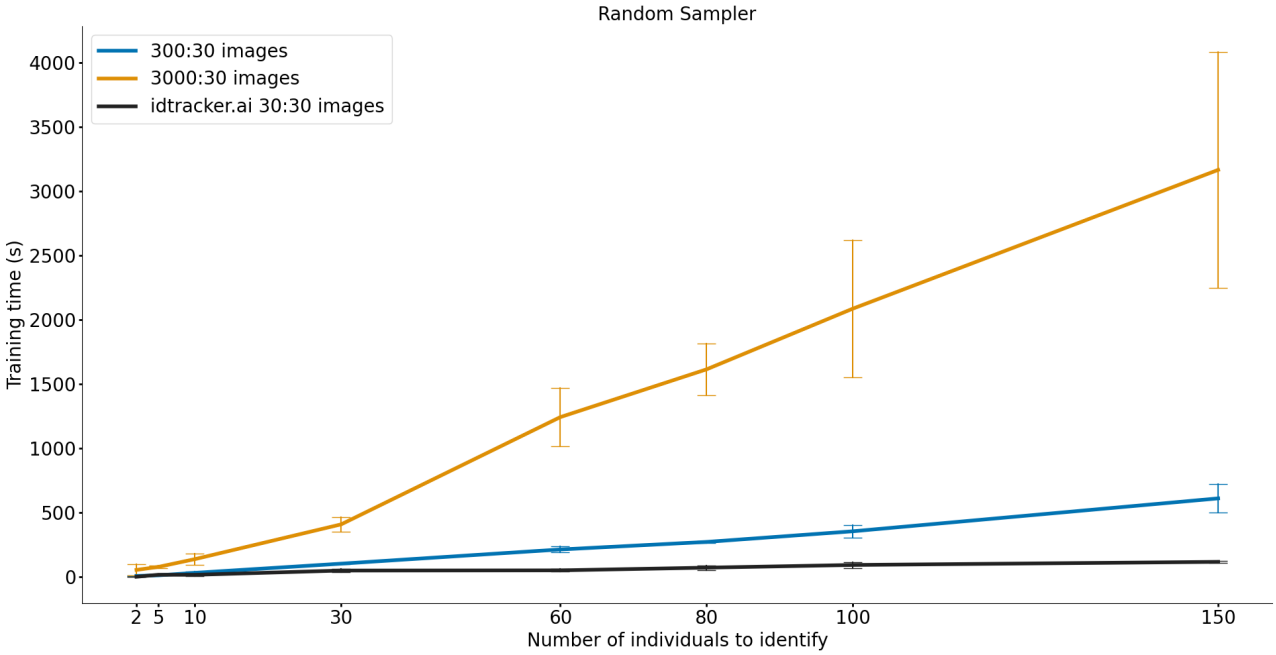


Figure 5.9: Training time when using the pairwise method and the Random Sampler described in Section 4.4.3, for 300 and 3000 images per class for training, of which 30 images (per class) are multi-labeled. These results are compared to the idtracker.ai training time for 30 images per class for training. The number of images referenced in the legend correspond to *number of images per class: number of multi-class labeled images per class*. The results here represented correspond to the mean and standard deviation of 5 repetitions for every test condition. The error bars are calculated as $mean \pm standard\ deviation$. In some test conditions one of the repetitions was not used here, as the value of accuracy was close to 0%, because the model was not able to learn the image features.

5.4 Results Summary

Our first approach, using Transfer Learning models, was intended to improve the identification network's performance. Transfer Learning allowed us to use pre-trained neural networks and fine-tune their parameters with our dataset. The main goal was to use this approach when we have a smaller dataset for training, in order to still achieve high accuracies; and to also reduce the training time of the identification network from idtracker.ai. From the models we used, the deeper network (ResNet18) did not perform well enough in terms of accuracy. The other three networks yielded good accuracy performances. Although these good accuracies came with an increase in training time, which we wanted to avoid. AlexNet performs very similarly to idtracker.ai in terms of training time and accuracy. Overall, we consider that the Vgg16 network had the best result in Experiment 1.

The Ensemble method we used by joining the predictions of multiple idtracker.ai models was not as successful as expected. The improvement in accuracy did not seem to be good enough for us to accept the necessary increasing training time. Especially when comparing this approach with the one of using the Vgg16 network to train and classify the same images. The accuracy of Vgg16 is still better than any ensemble size of idtracker.ai models. Besides, the training time of Vgg16 managed to be smaller then that of the ensembles, for most of the ensemble sizes. For this reason, the ensemble method was the least successful machine learning method.

The addition of pairwise labels generated by comparing pairs of images without multi-class labels resulted in an increase of accuracy. This strategy is particularly useful for datasets of few images with multi-class labels. This situation would correspond to videos with few or small global fragments. Both experiments using this new training method showed good performance values for only 30 multi-labeled

images per class. While for the Simple Sampler the accuracy dropped below the accuray of idtracker.ai for more than 60 classes, the Random Sampler maintained higher accuracies for all number of classes to identify (from 2 classes to 150 classes).

Figure 5.10 shows the comparison between the best accuracy results of each experiment and the idtracker.ai results, for 30 images per class for training. It also shows the results for idtracker.ai with 3000 images per class, in order to compare with the initial data we had of idtracker.ai (Figure 2.4). Figure 5.11 shows the same comparison, but for the training time results. These comparisons lead to the conclusion that the best result overall was the one for using pairwise comparisons in the training of the identification network. Regarding time, the Ensemble method or using the Vgg16 model, were faster than using pairwise comparisons (3000 images per class), but the accuracy obtained for the pairwise method is much higher, which justifies the use of a slower training method. Although we do not show it in these figures (Figures 5.10 and 5.11), using pairwise comparisons (Random Sampler method) with only 300 images per class, surpasses the results we had from the first two methods (Vgg16 model and Ensemble method), in terms of accuracy and training time (except for tests with 150 classes). The comparison of the results of Vgg16 and the Ensemble method to the Random Sampler method trained with 300 images is showed in Appendix D in Figure D.1 and Figure D.2.

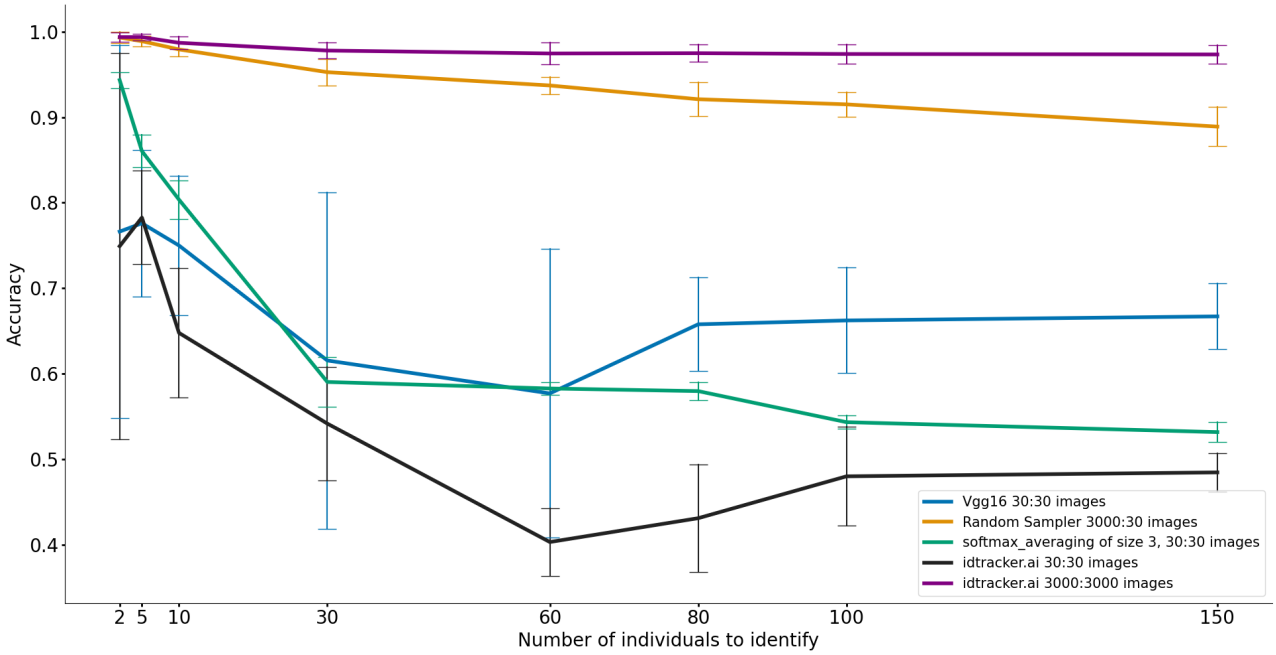


Figure 5.10: Comparison of test accuracy of the best results from all previous experiments. Here we compare the result of Vgg16 for 30 images per class; of the ensemble of size 3 for 30 images per class, more specifically the softmax averaging results; of the Random Sampler experiment for 3000 images per class (of which 30 images are multi-labeled); and of idtracker.ai for 30 and 3000 images per class. The error bars are calculated as $mean \pm standard\ deviation$. The number of images referenced in the legend correspond to *number of images per class: number of multi-class labeled images per class*.

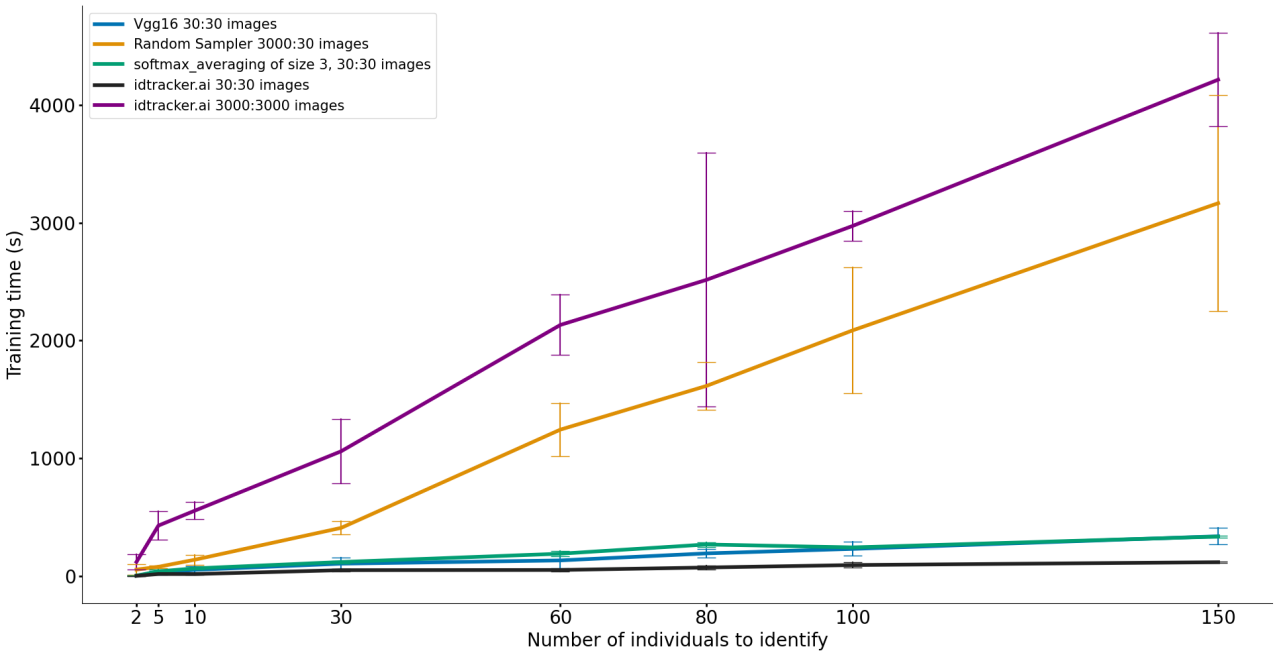


Figure 5.11: Comparison of training time of the best results from all previous experiments. Here we compare the result of Vgg16 for 30 images per class; of the ensemble of size 3 for 30 images per class, more specifically the softmax averaging results; of the Random Sampler experiment for 3000 images per class (of which 30 images are multi-labeled); and of idtracker.ai for 30 and 3000 images per class. The error bars are calculated as *mean* $\pm$ *standard deviation*. The number of images referenced in the legend correspond to *number of images per class: number of multi-class labeled images per class*.

Chapter 6

Conclusion

In this dissertation, our goal was to improve the training of the identification network of idtracker.ai. The performance of idtracker.ai could be improved, specifically in situations where it detects many crossings in a video. If a video has many crossings, the images to train the identification network are few, making the individual and global fragments small. All of this has an impact on the training of the algorithm of idtracker.ai. The accuracy of image classification and the training time in these cases is not as good as the accuracy with videos where idtracker.ai can obtain bigger global fragments.

To solve this problem, we experimented with different machine learning techniques, to understand which would be the best one for the improvement in training the identification network. First, we used Transfer Learning models to boost accuracy values when training with smaller datasets (≈ 30 images per class). Secondly, we constructed ensembles of idtracker.ai models to reduce the noise we had in our predictions and consequently improve the accuracy of a single model. And finally, we changed the training of the identification network in order to use images without multi-class labels and use pairwise labels instead. Training with fewer multi-class labeled images resulted in the improvement of the overall accuracy of the network.

Transfer Learning was successful in improving the accuracy in classification, but it showed an increase in training time in comparison to idtracker.ai's method. For other problems, where time is not an issue, this is a good solution when having small datasets for training. However, by changing some of the hyperparameters in some of the networks, we might be able to achieve a better performance of these models in terms of training time and also in accuracy.

The ensembles did not improve the accuracy over that of a single idtracker.ai model. Furthermore, the training time in these cases was obviously longer than with using just one idtracker.ai model, or one transfer learning model (Vgg16).

The last experiment, Section 4.4, using pairwise comparisons, was the one showing better accuracy. The added information that the network obtains by using the comparisons between pairs of images is enough to make the accuracy of training with 30 multi-labeled images (average of 94% for the Random Sampler experiment when training with 3000 images per class) comparable to training with 3000 multi-labeled images (average of 98% for idtracker.ai) (per class for both cases). We are still using 3000 or 5000 (Figure C.3 in Appendix C) images per class in this experiment (in the best case scenario), but only 30 are multi-labeled. Using the previous training method, from idtracker.ai and the previous experiments, the only images that could be used were the 30 multi-labeled images, which yielded lower accuracies.

The experiments performed in this project have limitations, as the images from a video would have similar statistical properties and this is not the case for our training sets of images. As mentioned before, idtracker.ai trains the identification network with global fragments. In the case of videos, the global

fragments are correlated in time. A larger correlation can mean less variability in terms of pose and illumination conditions from the animals, which can affect how well the network learns the features in the images that distinguishes the animals from each other. This might also affect the training time or the test accuracy. For our experiments, the global fragments (multi-class labeled images) we are using all have the same size, this means that we always have the same number of images per class while training. In a video, global fragments are not uniform, so we may not have the same number of images per class to train the network. Besides this, for our experiments, the multi-class labeled images can be from anywhere in the dataset, which means the variability is bigger.

Another difference from using videos is in the individual fragments. In a video, the individual fragments would be of different sizes and would have the same correlation in time that the global fragments have. For our experiment, the individual fragments are all of the same size (we always have the same number of images per class), and the images can be taken from anywhere in the dataset. This affects the experiment of the pairwise comparisons (Section 4.4) and again adds more variability to the images when comparing pairs, as they are less correlated than in videos.

Considering all of the limitations and the results, our tests give strong evidence in favor of the pairwise network improving recognition performance, in comparison to the idtracker.ai method that is currently implemented, even though this improvement will ultimately need to be checked in real videos.

6.1 Future work

In this project, we completed the experiments using the CARP dataset and we tested single-image identification accuracy. A possibility of future work is to revisit the Transfer Learning strategy and test the different networks with different hyperparameters, and subsequently test this with real videos.

Although, as the last experiment was the one with the better results, a natural future step of this work is to evaluate the pairwise comparisons training paradigm with real videos. These tests will be to check whether the improvements we observed here correspond to a similar improvement in the tracking performance of videos. In the last experiment, we are assuming global fragments constituted by same size individual fragments. In real videos, usually, global fragments have different numbers of images per class. So we need to study the impact of nonuniform global fragments when using the pairwise training method.

Besides experimenting with videos, we can also improve the Samplers used for the pairwise method. As of now, for the Simple Sampler we are ensuring that at least 10% of the images in a batch have multi-class labels. We can evaluate what happens if we increase this value. Also, for the Random Sampler, we can change its implementation, in order to ensure every batch has multi-labeled images, so that we do not have to choose the loss used for the stopping criteria of the training. We can also investigate a different type of sampler that might work better with idtracker.ai, because even the samplers we have now would have to be adapted to work with the algorithm of idtracker.ai.

References

1. Romero-Ferrero, F., Bergomi, M. G., Hinz, R. C., Heras, F. J. & de Polavieja, G. G. idtracker.ai: Tracking All Individuals in Small or Large Collectives of Unmarked Animals. *Nature Methods* **16,** 179–182 (2019).
2. Ballerini, M. *et al.* Interaction Ruling Animal Collective Behavior Depends on Topological Rather than Metric Distance: Evidence from a Field Study. *Proceedings of the National Academy of Sciences* **105,** 1232–1237 (2008).
3. Dell, A. I. *et al.* Automated image-based tracking and its application in ecology. *Trends in Ecology and Evolution* **29,** 417–428 (2014).
4. Wang, S. H., Zhao, J. W. & Chen, Y. Q. Robust tracking of fish schools using CNN for head identification. *Multimedia Tools and Applications* **76,** 23679–23697 (2017).
5. Robie, A. A., Seagraves, K. M., Egnor, S. E. R. & Branson, K. Machine vision methods for analyzing social interactions. *Journal of Experimental Biology* **220,** 25–34 (2017).
6. Werkhoven, Z., Rohrsen, C., Qin, C., Brembs, B. & de Bivort, B. MARGO (Massively Automated Real-time GUI for Object-tracking), a platform for high-throughput ethology. *PLOS ONE* **14,** 1–24 (2019).
7. Sridhar, V. H., Roche, D. G. & Gingins, S. Tracktor: Image-based automated tracking of animal movement and behaviour. *Methods in Ecology and Evolution* **10,** 815–820 (2019).
8. Pérez-Escudero, A., Vicente-Page, J., Hinz, R. C., Arganda, S. & Polavieja, G. G. D. IdTracker: Tracking individuals in a group by automatic identification of unmarked animals. *Nature Methods* **11,** 743–748 (2014).
9. Pan, S. J. & Yang, Q. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering* **22,** 1345–1359 (2010).
10. Sagi, O. & Rokach, L. Ensemble learning: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* **8** (2018).
11. Hsu, Y.-C. & Kira, Z. Neural network-based clustering using pairwise constraints. *ArXiv* (2015).
12. Hsu, Y.-C., Lv, Z., Schlosser, J., Odom, P. & Kira, Z. Multi-class Classification without Multi-class Labels. *ArXiv* (2019).
13. Rasch, M. J., Shi, A. & Ji, Z. Closing the loop: tracking and perturbing behaviour of individuals in a group in real-time. *bioRxiv* (2016).
14. Hughey, L. F., Hein, A. M., Strandburg-Peshkin, A. & Jensen, F. H. Challenges and solutions for studying collective animal behaviour in the wild. *Philosophical Transactions of the Royal Society B: Biological Sciences* **373** (2018).

15. Sherub, S., Fiedler, W., Duriez, O. & Wikelski, M. Bio-logging, new technologies to study conservation physiology on the move: a case study on annual survival of Himalayan vultures. *Journal of Comparative Physiology A: Neuroethology, Sensory, Neural, and Behavioral Physiology* **203,** 531–542 (2017).

16. Yilmaz, A., Javed, O. & Shah, M. Object tracking: A survey. *ACM Computing Surveys* **38** (2006).

17. Yang, H., Shao, L., Zheng, F., Wang, L. & Song, Z. Recent advances and trends in visual tracking: A review. *Neurocomputing* **74,** 3823–3831 (2011).

18. Gomez-Marin, A., Partoune, N., Stephens, G. J. & Louis, M. Automated tracking of animal posture and movement during exploration and sensory orientation behaviors. *PLoS ONE* **7** (2012).

19. Straw, A. D., Lee, S. & Dickinson, M. H. Visual Control of Altitude in Flying Drosophila. *Current Biology* **20,** 1550–1556 (2010).

20. Krause, J., Winfield, A. F. & Deneubourg, J.-L. Interactive robots in experimental biology. *Trends in Ecology Evolution* **26,** 369–375 (2011).

21. Branson, K. & Belongie, S. *Tracking multiple mouse contours (without too many samples)* in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)* **1** (2005), 1039–1046.

22. Branson, K., Robie, A. A., Bender, J., Perona, P. & Dickinson, M. H. High-throughput ethomics in large groups of Drosophila. *Nature Methods* **6,** 451–457 (2009).

23. Chaumont, F. D. *et al.* Computerized video analysis of social interactions in mice. *Nature Methods* **9,** 410–417 (2012).

24. Matsumoto, J. *et al.* A 3D-video-based computerized analysis of social and sexual interactions in rats. *PLoS ONE* **8** (2013).

25. Dankert, H., Wang, L., Hoopfer, E. D., Anderson, D. J. & Perona, P. Automated monitoring and analysis of social behavior in Drosophila. *Nature Methods* **6,** 297–303 (2009).

26. Straw, A. D., Branson, K., Neumann, T. R. & Dickinson, M. H. Multi-camera real-time three-dimensional tracking of multiple flying animals. *Journal of the Royal Society Interface* **8,** 395–409 (2011).

27. Ohayon, S., Avni, O., Taylor, A. L., Perona, P. & Egnor, S. E. R. Automated multi-day tracking of marked mice for the analysis of social behaviour. *Journal of Neuroscience Methods* **219,** 10–19 (2013).

28. Mirat, O., Sternberg, J. R., Severi, K. E. & Wyart, C. ZebraZoom: An automated program for high-throughput behavioral analysis and categorization. *Frontiers in Neural Circuits* (2013).

29. Swierczek, N. A., Giles, A. C., Rankin, C. H. & Kerr, R. A. High-throughput behavioral analysis in C. elegans. *Nature Methods* **8,** 592–602 (2011).

30. Feldman, A., Hybinette, M. & Balch, T. The multi-iterative closest point tracker: An online algorithm for tracking multiple interacting targets. *Journal of Field Robotics* **29,** 258–276 (2012).

31. Kuhn, H. W. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly* **2,** 83–97 (1955).

32. Mathis, A. *et al.* DeepLabCut: markerless pose estimation of user-defined body parts with deep learning. *Nature Neuroscience* **21,** 1281–1289 (2018).

33. Pereira, T. D. *et al.* Fast animal pose estimation using deep neural networks. *Nature Methods* **16,** 117–125 (2019).

34. Insafutdinov, E., Pishchulin, L., Andres, B., Andriluka, M. & Schiele, B. *DeeperCut: A Deeper, Stronger, and Faster Multi-Person Pose Estimation Model* in *Computer Vision - ECCV 2016* (Springer International Publishing, Cham, 2016), 34–50.

35. Walter, T. & Couzin, I. D. TRex, a fast multi-animal tracking 1 system with markerless 2 identiication, and 2D estimation of 3 posture and visual lelds. *bioRxiv* (2020).

36. Macukow, B. *Neural networks-state of art, brief history, basic models and architecture* in. **9842 LNCS** (Springer Verlag, 2016), 3–14.

37. Abraham, A. *Artificial Neural Networks* (eds Sydenham, P. H. & Thorn, R.) chap. 129. ISBN: 0470021438 (Wiley, 2005).

38. Hastie, T., Tibshirani, R. & Friedman, J. *Springer Series in Statistics The Elements of Statistical Learning Data Mining, Inference, and Prediction* 392–397 (Springer-Verlag New York, 2009).

39. Nogueira, K., Penatti, O. A. B. & dos Santos, J. A. Towards Better Exploiting Convolutional Neural Networks for Remote Sensing Scene Classification. *Pattern Recognition* **61,** 539–556 (2017).

40. Sun, S., Cao, Z., Zhu, H. & Zhao, J. A Survey of Optimization Methods From a Machine Learning Perspective. *IEEE Transactions on Cybernetics* **50,** 3668–3681 (2020).

41. Graves, A., Mohamed, A.-r. & Hinton, G. Speech Recognition with Deep Recurrent Neural Networks. *2013 IEEE International Conference on Acoustics, Speech and Signal Processing* (2013).

42. Luo, W., Li, J., Yang, J., Xu, W. & Zhang, J. Convolutional sparse autoencoders for image classification. *IEEE Transactions on Neural Networks and Learning Systems* **29,** 3289–3294 (2018).

43. Geng, J. *et al.* High-Resolution SAR Image Classification via Deep Convolutional Autoencoders. *IEEE Geoscience and Remote Sensing Letters* **12,** 2351–2355 (2015).

44. Jmour, N., Zayen, S. & Abdelkrim, A. *Convolutional Neural Networks for image classification* in *2018 International Conference on Advanced Systems and Electric Technologies (IC$_A$SET), Hammamet, Tunisia* (2018), 397–402. ISBN: 9781538644492.

45. Traore, B. B., Kamsu-Foguem, B. & Tangara, F. Deep convolution neural network for image recognition. *Ecological Informatics* **48,** 257–268 (Nov. 2018).

46. Gopalakrishnan, K., Khaitan, S. K., Choudhary, A. & Agrawal, A. Deep Convolutional Neural Networks with transfer learning for computer vision-based data-driven pavement distress detection. *Construction and Building Materials* **157,** 322–330 (2017).

47. Bjorck, N., Gomes, C. P., Selman, B. & Weinberger, K. Q. in *Advances in Neural Information Processing Systems 31* (eds Bengio, S. *et al.*) 7694–7705 (Curran Associates, Inc., 2018).

48. Yosinski, J., Clune, J., Bengio, Y. & Lipson, H. *How transferable are features in deep neural networks?* in *Proceedings of the 27th International Conference on Neural Information Processing Systems* (eds Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D. & Weinberger, K. Q.) **2** (2014), 3320–3328.

49. Rosenstein, M., Marx, Z. & Kaelbling, L. *To Transfer or Not To Transfer* in *Proc. Conf. Neural Information Processing Systems (NIPS '05) Workshop Inductive Transfer: 10 Years Later* (2005).

50. Shin, H. C. *et al.* Deep Convolutional Neural Networks for Computer-Aided Detection: CNN Architectures, Dataset Characteristics and Transfer Learning. *IEEE Transactions on Medical Imaging* **35,** 1285–1298 (2016).

51. Deng, J. *et al. ImageNet: A Large-Scale Hierarchical Image Database* in (Computer Vision and Pattern Recognition, IEEE Conference, 2009). ISBN: 9781424439911.

52. Ju, C., Bibaut, A. & van der Laan, M. The relative performance of ensemble methods with deep convolutional neural networks for image classification. *Journal of Applied Statistics* **45,** 2800–2818 (2018).

53. Beluch, W. H., Nürnberger, A. & Köhler, J. M. *The power of ensembles for active learning in image classification* in *2018 Conference on Computer Vision and Pattern Recognition* (IEEE, 2018).

54. Tao, S. Deep Neural Network Ensembles, 1–12 (2019).

55. Bromley, J., Guyon, I., LeCun, Y., Sackinger, E. & Shah, R. Signature Verification using a "Siamese" Time Delay Neural Network. *International Journal of Pattern Recognition and Artificial Intelligence* **7,** 25 (1993).

56. Guo, Q. *et al. Learning Dynamic Siamese Network for Visual Object Tracking* in *2017 IEEE International Conference on Computer Vision (ICCV)* (IEEE, 2017), 1781–1789.

57. Melekhov, I., Kannala, J. & Rahtu, E. *Siamese Network Features for Image Matching* in *2016 23rd International Conference on Pattern Recognition (ICPR)* (IEEE, 2016), 378–383. ISBN: 9781509048472.

58. Doumpos, M. & Zopounidis, C. A multicriteria classification approach based on pairwise comparisons. *European Journal of Operational Research* **158,** 378–389 (2004).

59. Li, Y., Song, Y. & Luo, J. *Improving Pairwise Ranking Multi-label Image Classification* in *2017 Conference on Computer Vision and Pattern Recognition* (IEEE, 2017), 1837–1845.

60. Simonyan, K. & Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. *ArXiv* (2014).

61. Krizhevsky, A. One weird trick for parallelizing convolutional neural networks. *ArXiv* (2014).

62. He, K., Zhang, X., Ren, S. & Sun, J. Deep Residual Learning for Image Recognition. *ArXiv* (2015).

Appendix A

Transfer Learning Models

Table A.1: Architecture of the convolutional layers of the Vgg16 and Vgg16_bn models, as they are implemented in torchvision (PyTorch).

	VGG16			VGG16_BN		
Layer	**Type**	**Kernel Size**	**Stride**	**Type**	**Kernel Size**	**Stride**
1	2x [Convolution + ReLU]	3x3	1x1	2x [Convolution + BatchNormalizarion + ReLU]	3x3	1x1
3	MaxPooling	2	2	MaxPooling	2	2
4	2x [Convolution + ReLU]	3x3	1x1	2x [Convolution + BatchNormalizarion + ReLU]	3x3	1x1
6	MaxPooling	2	2	MaxPooling	2	2
7	2x [Convolution + ReLU]	3x3	1x1	2x [Convolution + BatchNormalizarion + ReLU]	3x3	1x1
9	MaxPooling	2	2	MaxPooling	2	2
10	3x [Convolution + ReLU]	3x3	1x1	3x [Convolution + BatchNormalizarion + ReLU]	3x3	1x1
13	MaxPooling	2	2	MaxPooling	2	2
14	3x [Convolution + ReLU]	3x3	1x1	3x [Convolution + BatchNormalizarion + ReLU]	3x3	1x1
17	MaxPooling	2	2	MaxPooling	2	2

Table A.2: Architecture of the convolutional layers of the AlexNet and ResNet18 models, as they are implemented in torchvision (PyTorch).

	AlexNet			ResNet18			
Layer	**Type**	**Kernel Size**	**Stride**	**Type**		**Kernel Size**	**Stride**
1	Convolution + ReLU	11x11	4x4	Convolution + BatchNormalizarion + ReLU		7x7	2x2
2	MaxPooling	3	2	MaxPooling		3	2
3	Convolution + ReLU	5x5	1x1	**2x**	Convolution + BatchNormalizarion + ReLU	3x3	1x1
4	MaxPooling	3	2		Convolution + BatchNormalizarion + ReLU	3x3	1x1
5	Convolution + ReLU	3x3	1	**2x**	Convolution + BatchNormalizarion + ReLU	3x3	2x2
6	Convolution + ReLU	3x3	1		Convolution + BatchNormalizarion + ReLU	3x3	1x1
7	Convolution + ReLU	3x3	1	**2x**	Convolution + BatchNormalizarion + ReLU	3x3	2x2
8	MaxPooling	3	2		Convolution + BatchNormalizarion + ReLU	3x3	1x1
9				**2x**	Convolution + BatchNormalizarion + ReLU	3x3	2x2
10					Convolution + BatchNormalizarion + ReLU	3x3	1x1

Appendix B

Transfer Learning: Extra Results

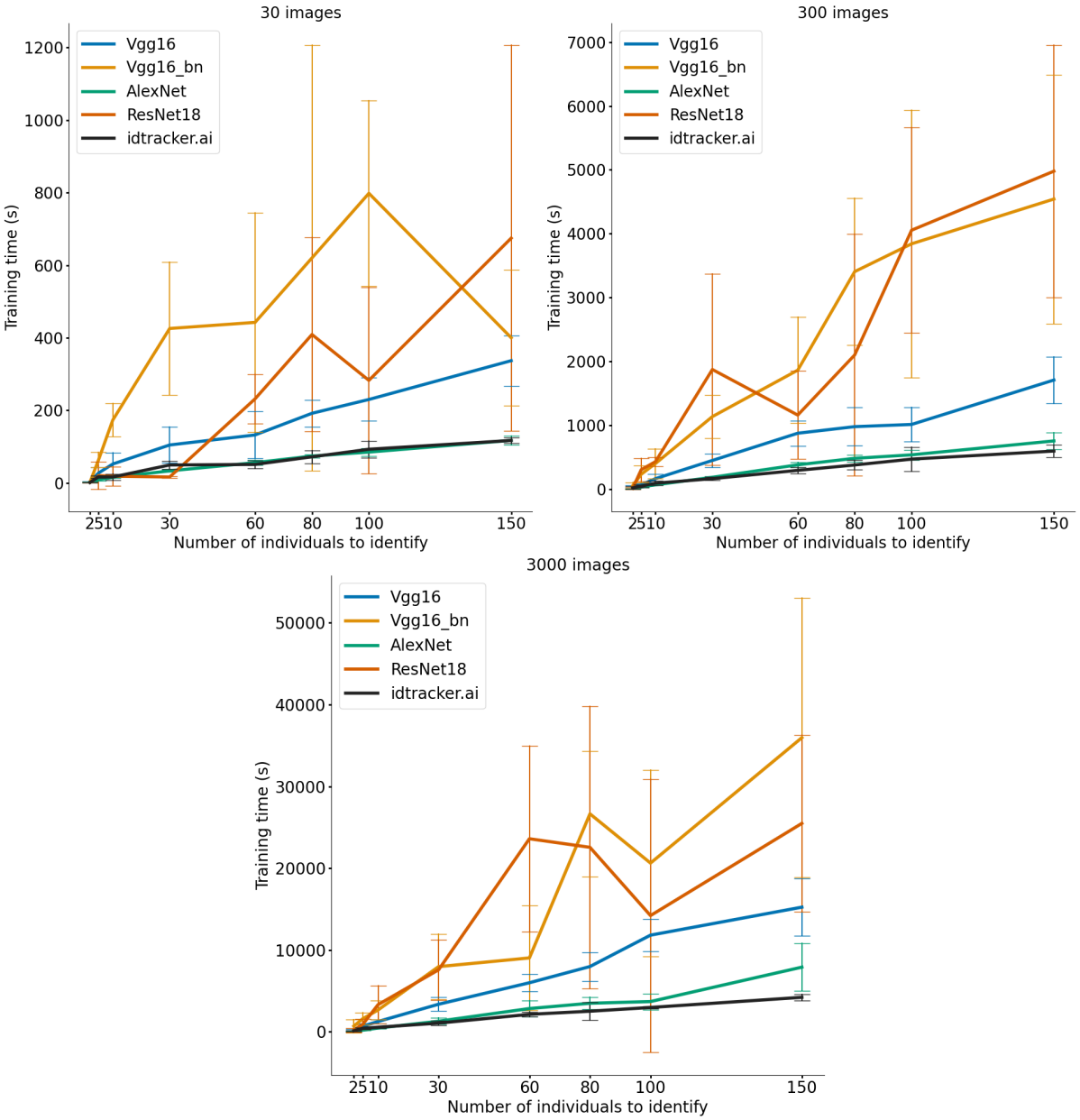


Figure B.1: Training time of the Transfer Learning models (Vgg16, Vgg16_bn, AlexNet and ResNet18), compared to the training time of the idtracker.ai model, for 30, 300 and 3000 images per individual for training. The error bars are calculated as *mean* $\pm$ *standard deviation*.

Appendix C

Pairwise Comparisons: Extra Results

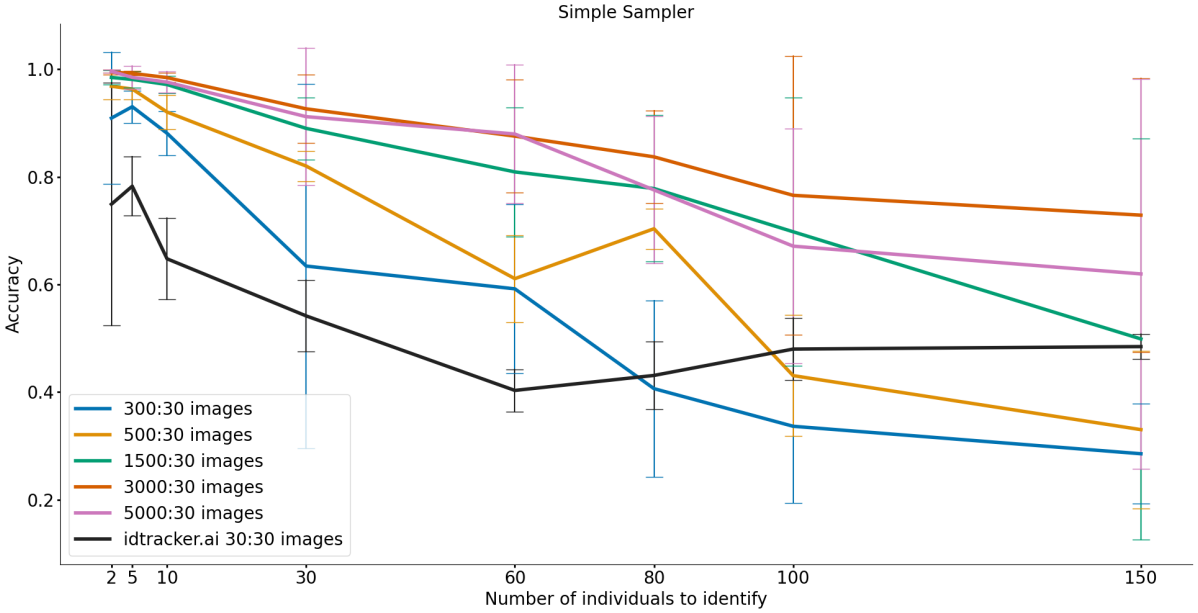


Figure C.1: Test accuracy when using the pairwise method and the Simple Sampler described in Section 4.4.2, for 300, 500, 1500, 3000 and 5000 images per class for training, of which 30 images (per class) are multi-labeled. These results are compared to the idtracker.ai test accuracy for 30 images per class for training. The number of images referenced in the legend correspond to *number of images per class: number of multi-class labeled images per class*. The results here represented correspond to the mean and standard deviation of 5 repetitions for every test condition. The error bars are calculated as $mean \pm standard\ deviation$.[1]

[1]The accuracy is never above 1 for each training run/repetition. The fact that some error bars are above 1 is most likely due to tests where the model did not converge while training, resulting in lower accuracy values. But considering that for this experiment, tests where the model did not converge happened multiple times, instead of removing those tests, we decided to show them, in order to explain that this sampler is not reliable as it is was computed for this dissertation.

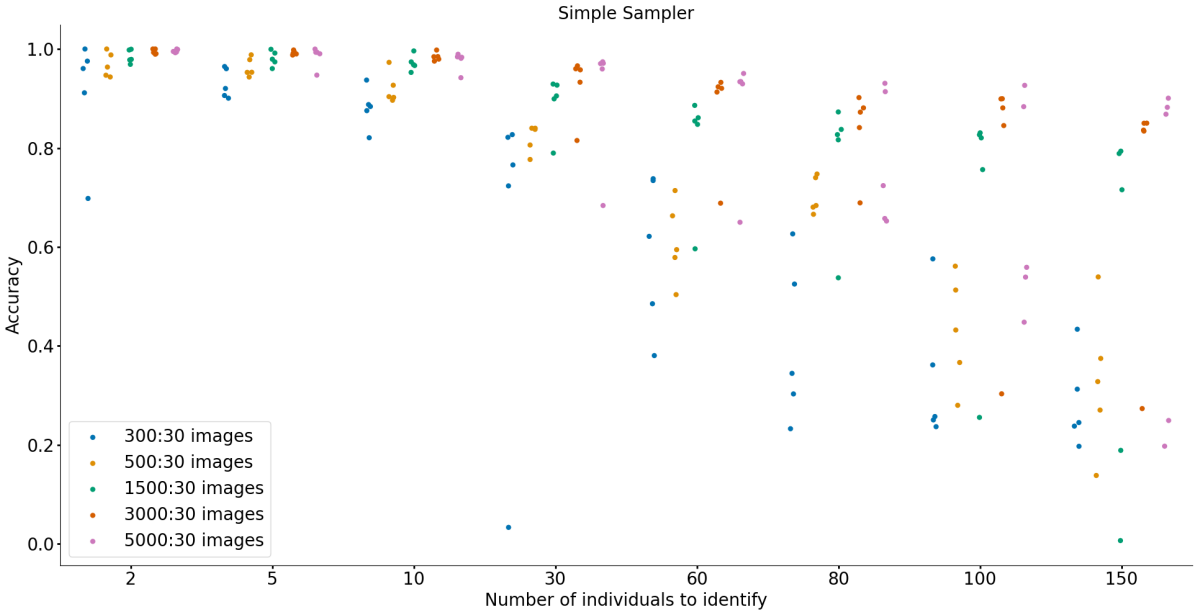


Figure C.2: Test accuracy when using the pairwise method and the Simple Sampler described in Section 4.4.2, for 300, 500, 1500, 3000 and 5000 images per class for training, of which 30 images (per class) are multi-labeled. The number of images referenced in the legend correspond to *number of images per class: number of multi-class labeled images per class*. In this graph, we show each of the five repetitions for each test condition (the dots). With this figure, it is possible to confirm previous statements regarding the accuracy values of the tests. The accuracy of a test is never above 1, but some of the repetitions of some test conditions have clearly lower accuracies than other (comparison between dots of the same color for the same number of individuals to identify).

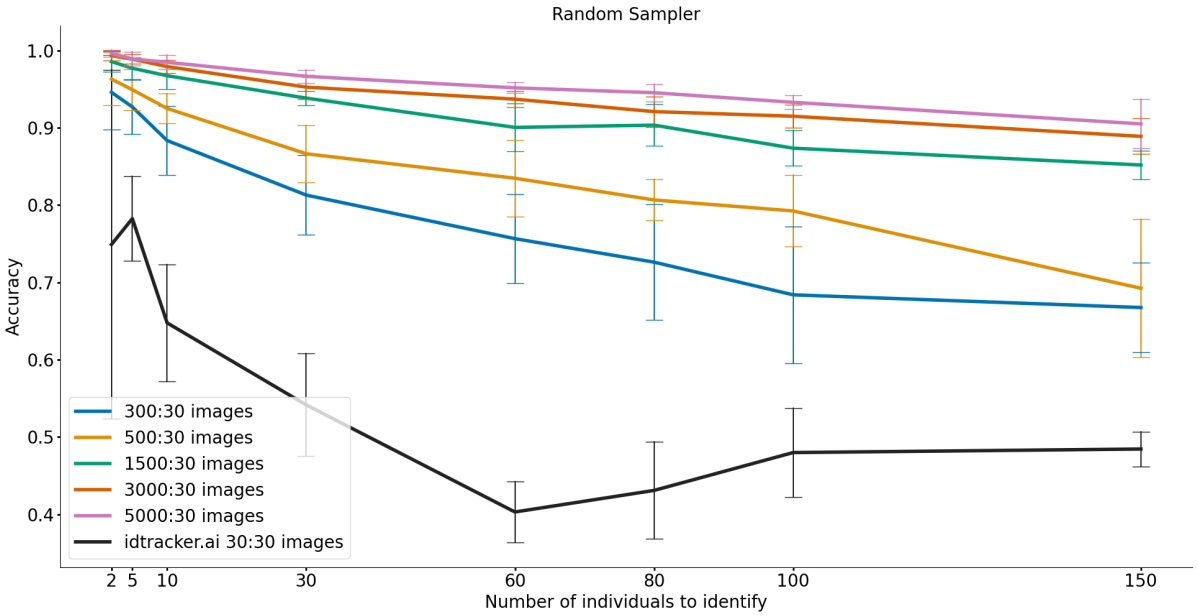


Figure C.3: Test accuracy when using the pairwise method and the Random Sampler described in Section 4.4.3, for 300, 500, 1500, 3000 and 5000 images per class for training, of which 30 images (per class) are multi-labeled. These results are compared to the idtracker.ai test accuracy for 30 images per class for training. The number of images referenced in the legend correspond to *number of images per class: number of multi-class labeled images per class*. The results here represented correspond to the mean and standard deviation of 5 repetitions for every test condition. The error bars are calculated as $mean \pm standard\ deviation$. In some test conditions one of the repetitions was not used here, as the value of accuracy was close to 0%, because the model was not able to learn the image features.

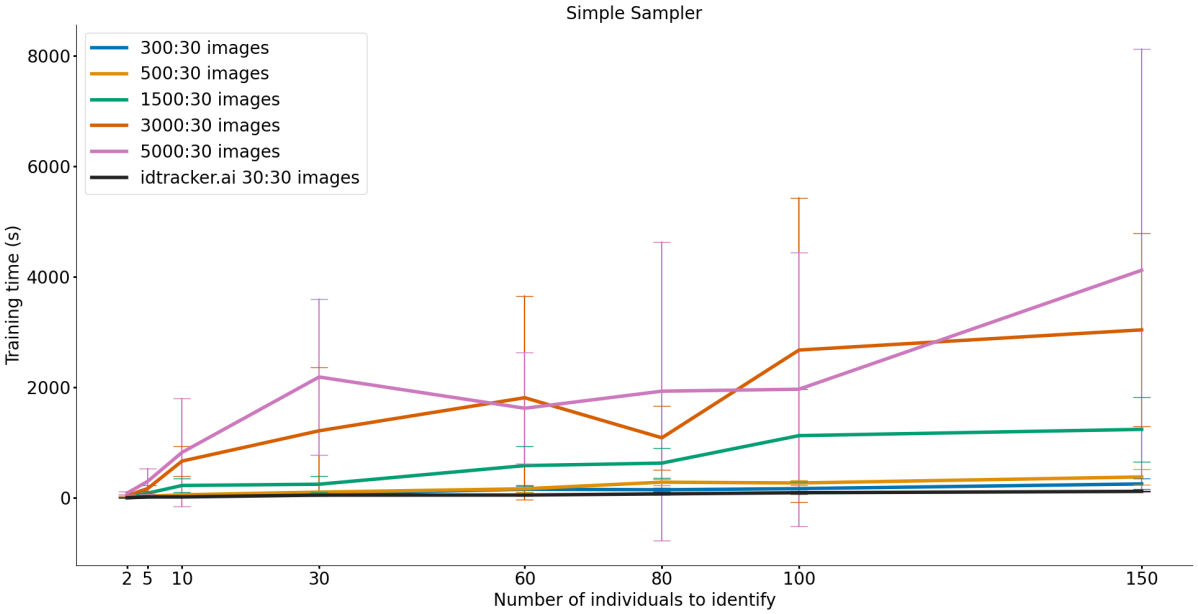


Figure C.4: Training time when using the pairwise method and the Simple Sampler described in Section 4.4.2, for 300, 500, 1500, 3000 and 5000 images per class for training, of which 30 images (per class) are multi-labeled. These results are compared to the idtracker.ai training time for 30 images per class for training. The results here represented correspond to the mean and standard deviation of 5 repetitions for every test condition. The error bars are calculated as $mean \pm standard\ deviation$.[2]

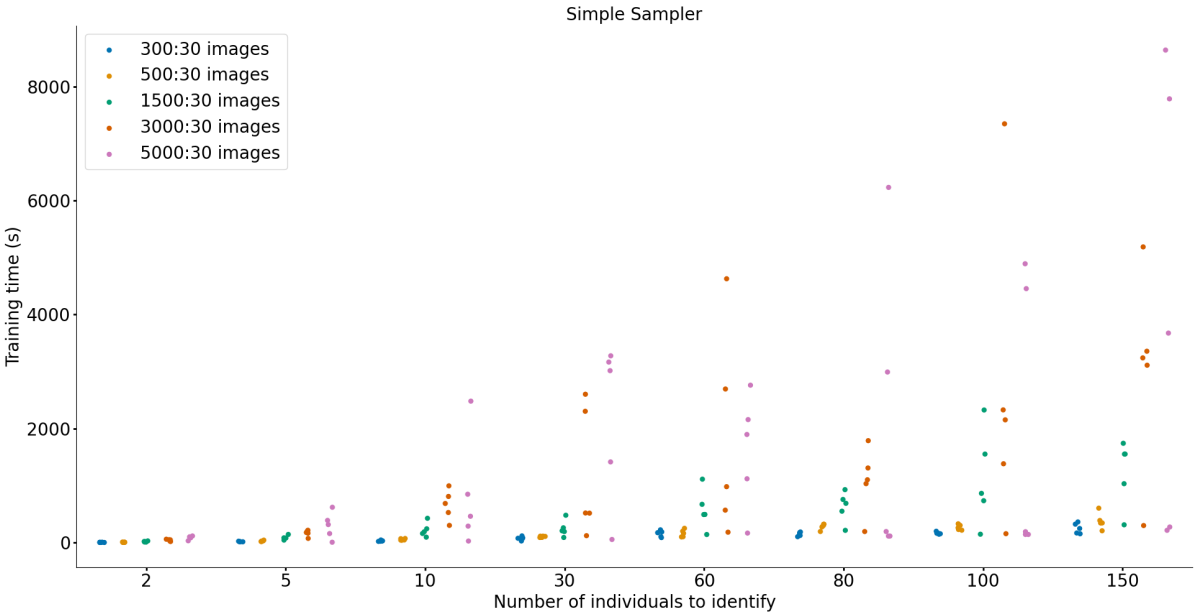


Figure C.5: Training time when using the pairwise method and the Simple Sampler described in Section 4.4.2, for 300, 500, 1500, 3000 and 5000 images per class for training, of which 30 images (per class) are multi-labeled. The number of images referenced in the legend correspond to *number of images per class: number of multi-class labeled images per class*. In this graph, we show each of the five repetitions for each test condition (the dots). Here we see the different time between repetitions of the same test condition (comparison between dots of the same color for the same number of individuals to identify). The fact that for some repetitions (of the same test condition) the time is faster than most, shows that the training stopped early on, which corresponds to the situations where the network did not converge.

[2]Similarly to what happens with the accuracy in Figure 5.6, the time of training cannot be below the 0 mark, this is just the representation of the standard deviation values that appears like this because of the odd values (faster training "runs"), corresponding to the tests where the model did not train properly.

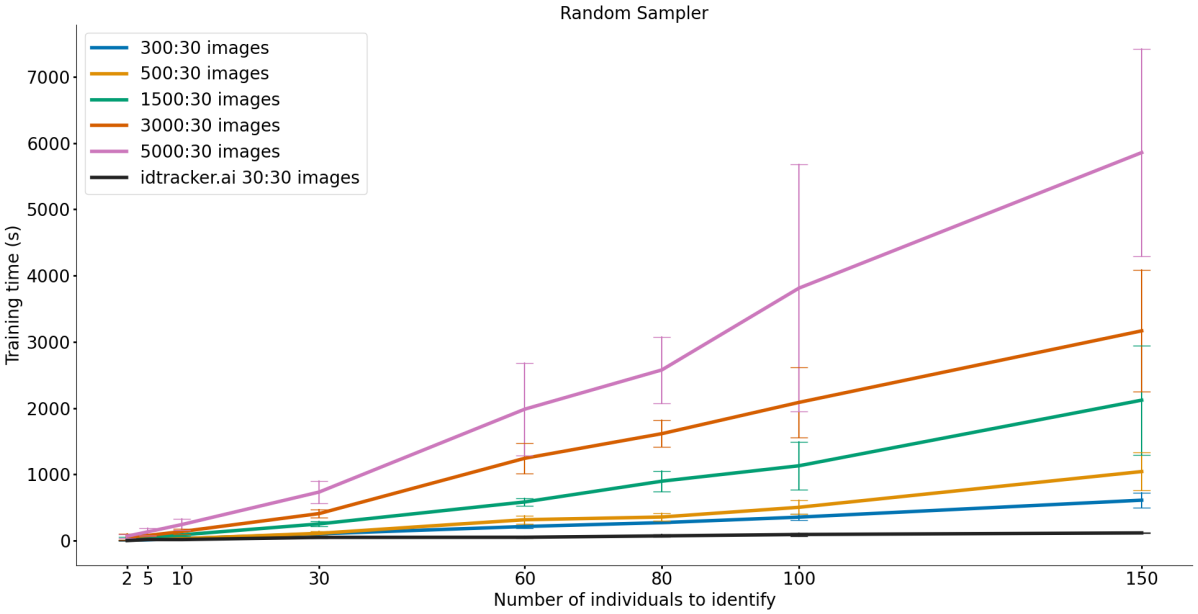


Figure C.6: Training time when using the pairwise method and the Random Sampler described in Section 4.4.3, for 300, 500, 1500, 3000 and 5000 images per class for training, of which 30 images (per class) are multi-labeled. These results are compared to the idtracker.ai training time for 30 images per class for training. The number of images referenced in the legend correspond to *number of images per class: number of multi-class labeled images per class*. The results here represented correspond to the mean and standard deviation of 5 repetitions for every test condition. The error bars are calculated as $mean \pm standard\ deviation$. In some test conditions one of the repetitions was not used, as the value of accuracy was close to 0%, because the model was not able to learn the image features.

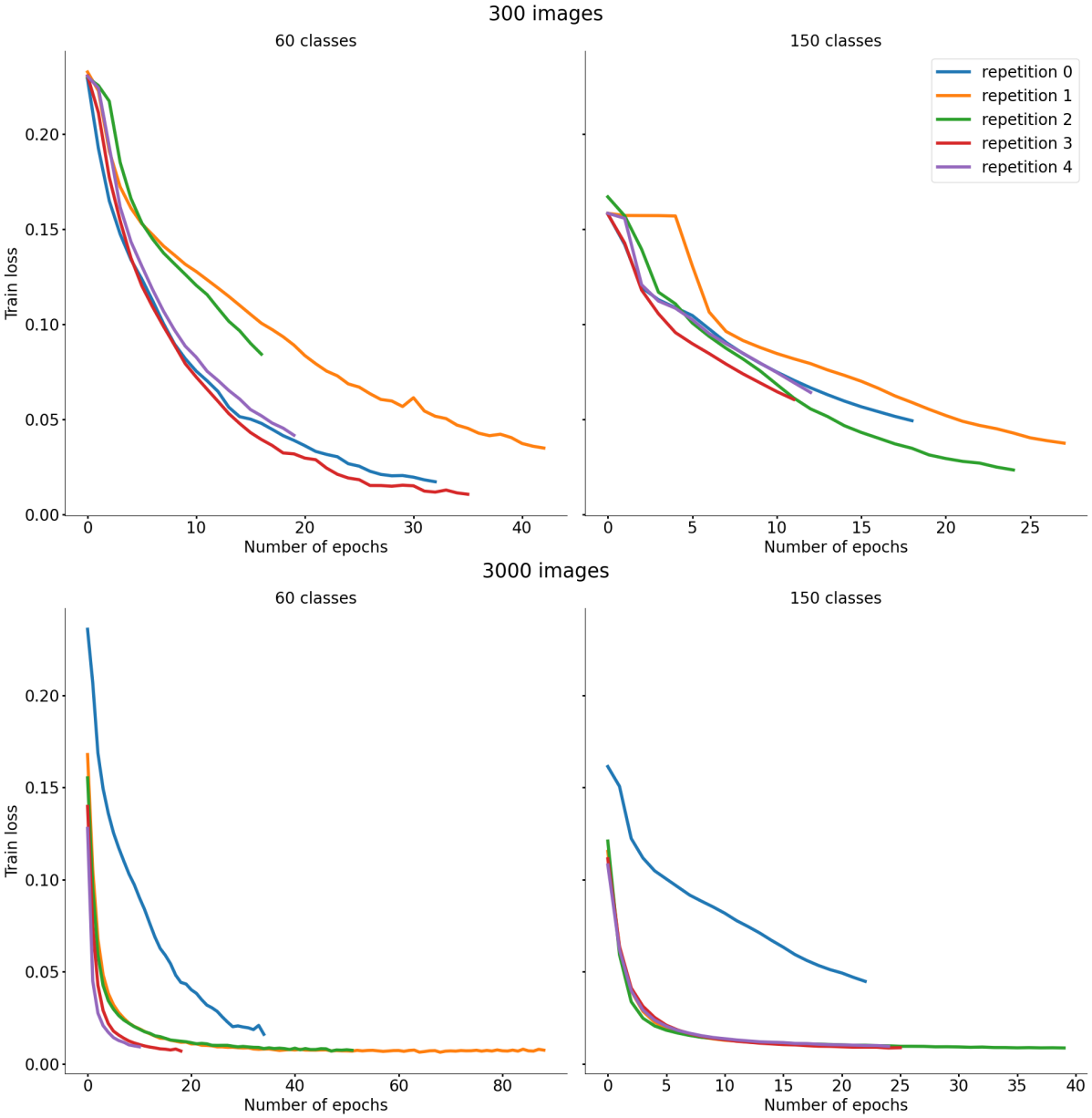


Figure C.7: Training loss when using the pairwise method and the Simple Sampler described in Section 4.4.2, for 300 and 3000 images per class and 60 and 150 classes. This figure illustrates the learning curves of the model for these specific conditions. We can observe that for some repetitions of the same test conditions, the initial loss is higher than for the others, which can result in a repetition that run for more epochs, or a repetition that did not converge. The fact that some repetitions run for more epochs, even after converging, might be related to the stopping criteria used.

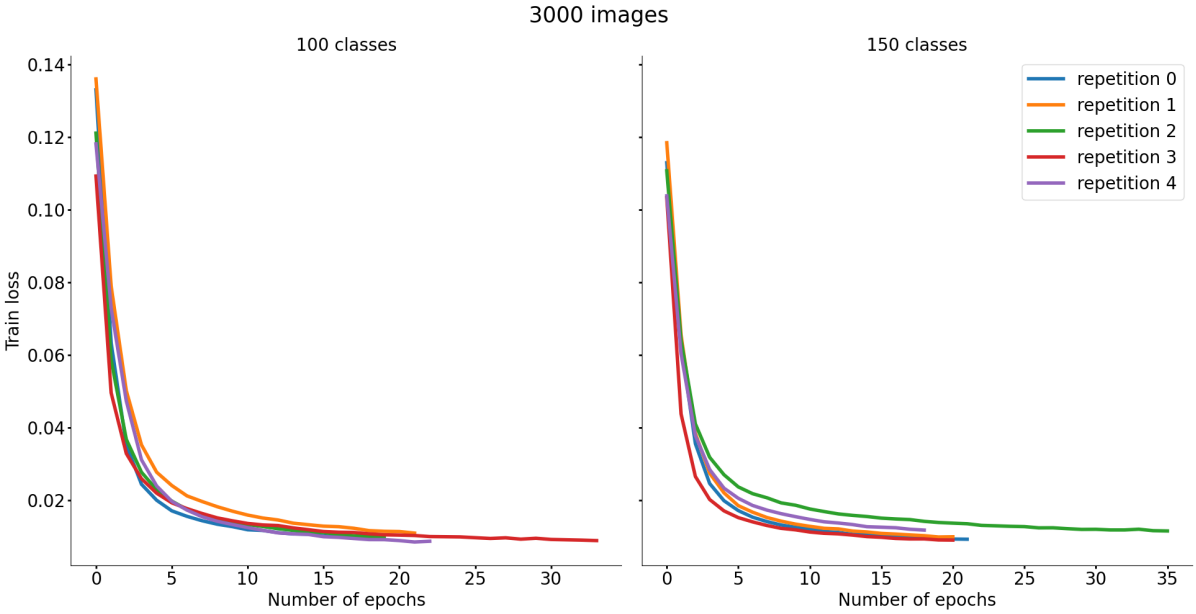


Figure C.8: Training loss when using the pairwise method and the Random Sampler described in Section 4.4.2, for 3000 images per class and 100 and 150 classes. This figure illustrates the learning curves of the model for these specific conditions. We can observe that some repetitions trained for more epochs than others, which influenced mean training time for this experiments, as we see in Figure 5.9.

Appendix D

Extra Results

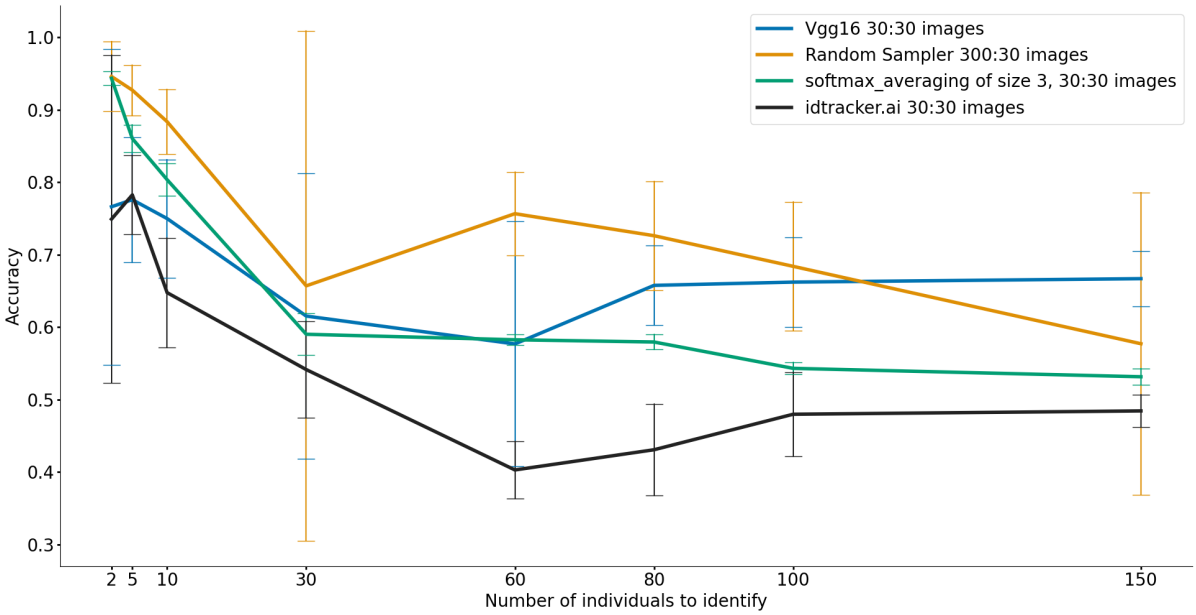


Figure D.1: Comparison of test accuracy of the best results from all previous experiments. Here, we compare the result of Vgg16 for 30 images per class; of the ensemble of size 3 for 30 images per class, more specifically the softmax averaging results; of the Random Sampler experiment for 300 images per class (of which 30 images are multi-labeled); and of idtracker.ai for 30 images per class. The error bars are calculated as $mean \pm standard\ deviation$. The number of images referenced in the legend correspond to *number of images per class: number of multi-class labeled images per class*.

D. EXTRA RESULTS

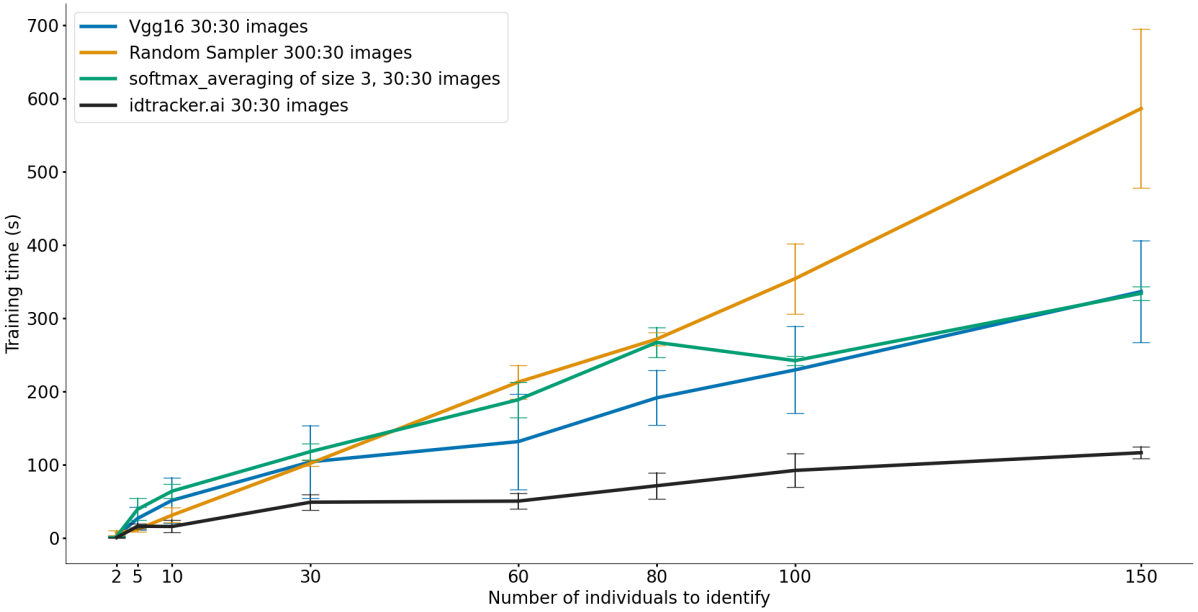


Figure D.2: Comparison of training time of the best results from all previous experiments. Here, we compare the result of Vgg16 for 30 images per class; of the ensemble of size 3 for 30 images per class, more specifically the softmax averaging results; of the Random Sampler experiment for 300 images per class (of which 30 images are multi-labeled); and of idtracker.ai for 30 images per class. The error bars are calculated as $mean \pm standard\ deviation$. The number of images referenced in the legend correspond to *number of images per class: number of multi-class labeled images per class*.