Pattern recognition in Cross-Country Skiing videos by means of Artificial Intelligence algorithms

Alberto Mira Criado

***Abstract*—Within the last years, mastering the technique of Cross-Country Skiing (XCS) athletes has become crucial in order to achieve better results whilst training or even in competitions. Based on this premise, this paper aims to present a software project, which merges up as a tool for coaches to obtain the five most important XCS patterns out of both lateral and frontal videos in an automated way, substituting the traditional methods, which tend to be time-consuming. The software presents several steps, going from when a video is obtained as an input, until the generation of a dataset in order to train a Neural Network (NN). In this case, three different types of NN with different capacities have been trained, resulting in the Convolutional Neural Network (CNN) model being the one showing the best performance in both frontal and lateral videos, with an overall accuracy of 81.48% and 81.78% respectively.**
This software chain has been applied to the technique Skating 1-1, showing that the easiest pattern to classify correspond to the leg push (pattern number five), achieving a general accuracy when having account of both lateral and frontal videos of 91.76%.
Finally, despite not being tested with a large dataset, an integrated single-script software has been programmed. It uses these CNN trained models in order to classify both types of video, giving as an output a folder containing the cycles found in a video under certain conditions, as well as a .txt files, where the probabilities coming from the NN model with respect to each single pattern of each cycle are shown, so, that way, coaches can better interpret the output data.

***Index Terms*—Cross-Country Skiing (XCS), Regions of Interest (ROIs), Neural Networks (NN), YOLOv3, OpenPose**

I. Introduction

Cross-country skiing (XCS), is a high intensity physical activity which requires the activation of both the upper and the lower body [1]. There are two main styles of XCS, classical and skating, and it is known that, through the years, the speed in races has notably increased, requiring improvements in technical and tactical abilities to bolster the general athlete's performance, since, on average, even 25 switches from one sub-technique to another can be required in a single kilometre [2].

The relation between physiology and technique has been already described in literature, and, among other characteristics, fast athletes often present shorter poling and thrust phases, along with longer gliding and recovery stages [3]. In fact, changes in technique and training have led them to reduce their metabolic cost by more than 50% per metre, something important having account of the fact that uphill phases represent one third of time in XCS competitions [2], as Fig. 1 shows:

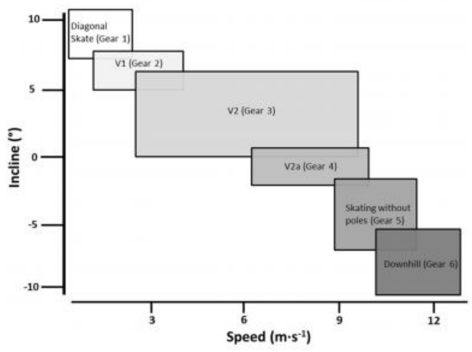


Fig. 1: Speed of different phases (known as gears) in Skating competitions. X axis represent the speed, and Y one stands for the inclination of each one of the phases [2].

Traditionally, XCS publications were mostly about physiology, but especially since the beginning of the millennium, this trend has changed, being biomechanics the most recurrent topic, something that can be due to several factors like the use of technology to obtain both kinetic and kinematics data that can be analysed afterwards, or the current interest in looking into the connection between biomechanics and physiology [4].

All in all, the aim of this work is to develop a software which is able to obtain the five most important patterns of XCS videos (Skating 1-1 concretely), since this task is normally done manually, resulting in a very time-consuming procedure. Therefore, with this new tool, this long process would be substitute for an automated method, leading to a situation where a lot of athletes could receive instant feedback about all related to their performance.

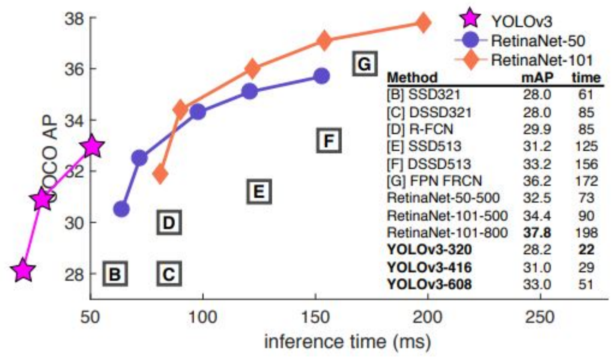
II. Methods

In this project, a whole software chain has been developed, consisting of several algorithms with a specific purpose, from the beginning, where a video is given as an input, to the final train and test of three Neural Network (NN) models, whose performance to classify Skating 1-1 data has been analysed.
The videos were provided by the Institute of Applied Training (*Institut für Trainingswissenschaft*, IAT), along with the above-mentioned five most important patterns of each video.
Finally, it is well noting that the software has been written with Python (version 3.7), and the programming environment has been PyCharm Professional 2020.2 (PyCharm Professional 2020.3 when using the MCI Super Computer), provided by

JetBrains s.r.o, with a licence *for educational use only*.

A. *Video frames, regions of interest and data augmentation*

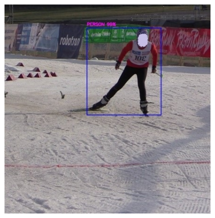
As it has been previously said, the input of this project is a video. It can be either a lateral or a frontal one, and the first step is to get its frames. This task has been performed with OpenCV [5], and along with the frame obtainment, data augmentation was coming into effect, too, by means of a function which rotated images X degrees, inspired by [6].
The exact rotation degree interval was [10,10), and rotation degrees were uniformly distributed by means of [7]. Then, for every frame obtained, two additional ones were generated, leading to a posterior increment of the dataset size.
Once the frames were obtained, it was necessary to obtain the regions of interest (ROIs) from them, meaning the place of the image were the athlete is located, task that was done with YOLOv3 [8], an object-detection software which has proven to run faster than other similar algorithms, as Fig. 2 shows:



Method	mAP	time
[B] SSD321	28.0	61
[C] DSSD321	28.0	85
[D] R-FCN	29.9	85
[E] SSD513	31.2	125
[F] DSSD513	33.2	156
[G] FPN FRCN	36.2	172
RetinaNet-50-500	32.5	73
RetinaNet-101-500	34.4	90
RetinaNet-101-800	**37.8**	198
YOLOv3-320	28.2	**22**
YOLOv3-416	31.0	29
YOLOv3-608	33.0	51

Fig. 2: Inference time comparison between YOLOv3 and other similar object-detection algorithms [8].

Along with the ROIs, YOLOv3 is able to extract four coordinates (x,y,w,h), where w stands for width, and h for height, out of the bounding boxes, and, consequently, ROIs, that it detects [8]. These four coordinates are the beginning of the feature extraction that will be necessary for the dataset creation in order to train and test the NN models.
Finally, YOLOv3 can detect up until 80 classes [8], but within the scope of this project, this has been programmed in such a way that it only detects the class *person*, so only ROIs containing the athlete can be extracted, as Fig. 3 shows:


(a)

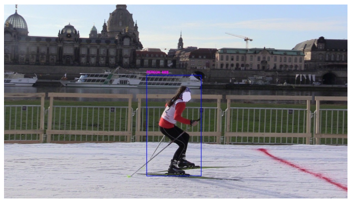

(b)

Fig. 3: Frontal a) and lateral b) detections of YOLOv3 in a frame.

B. *Data labelling*

Once the ROIs of the videos are obtained, it is time to compare them with the patterns that were provided along with the videos, so the data can be properly classified.
The comparison task has been made by using a SIFT (Scale-Invariant Feature Transform) algorithm [9]. The idea behind this SIFT algorithm is to detect the most important (x,y) keypoints, which, as described in [9], it does it by performing a Difference of Gaussians (DoG), an approximation of Laplace of Gaussians (LoG), which can be better seen in Fig. 4:

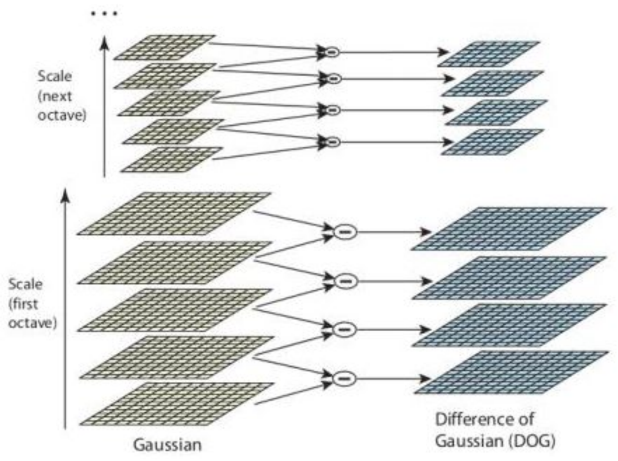


Fig. 4: Example of DoG performed by a SIFT algorithm [9].

Therefore, SIFT was applied to all the ROIs obtained from each video, as well as to the given pattern. After that, the comparison between the keypoints in both images was effected by a knnMatch Brute-Force Matcher [10], where k was set to 2. A graphical description of this comparison can be visualised in Fig. 5:

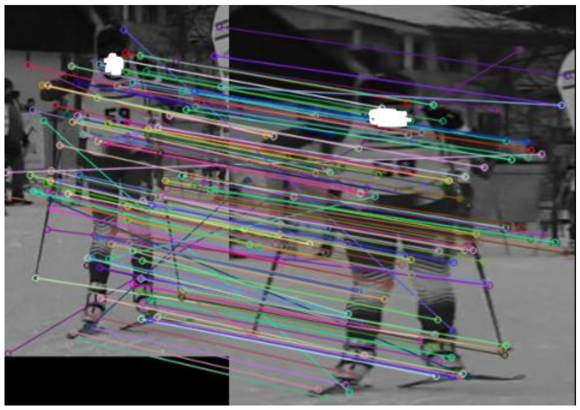


Fig. 5: Matching example between a ROI obtained from a video and a given pattern.

In the end, when all the ROIs of a video are compared with a given pattern, a matching sequence is obtained, where the ROI with the highest amount of matches would be eventually considered a pattern.

C. *Feature extraction*

As previously stated, the feature extraction process starts with YOLOv3 [8], but actually, if only those four coordinates were used to train and test a NN model, it would not likely perform well. For that reason, more features with regard to the ROIs are needed, so a more complete dataset can be offered to the NN, and, to perform this task, OpenPose [11] [12] [13] [14] was used.
OpenPose, as described in [12], is a software which is able to extract up until 135 keypoints on 2D images, with regard to the human body, feet, hands and face, although within this project, only 25 were used, which can be better visualised in Fig. 6:

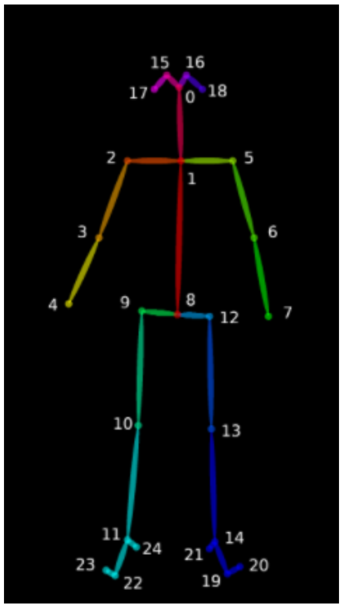


Fig. 6: Graphical description of an skeleton showing the 25 keypoints detected by OpenPose which has been used for the scope of this project [15].

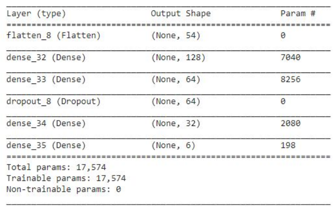
It is well noting that it is possible to use a CPU and a GPU version of OpenPose. These two were used (CPU when working locally, and the GPU version whilst using the MCI super computer), although the final dataset of the project was obtained with the GPU version. Both releases are available in [16].

D. *Dataset creation*

Now that all the information to train and test the NN models has been gathered, it is time to put it all together, and create a dataset with an uniform structure. To do that, for every ROI which was chosen as a pattern, (x,y) OpenPose coordinates of all of the 25 keypoints plus the four YOLOv3 coordinates were used. Consequently, the final dataset's size was 27x2 per pattern, which was stored in the form of NumPy arrays [17], which were eventually converted to tensors, since the library used to create the NN models was TensorFlow [18].
As a clarification, not all the three models received the same input. In fact, only two of them did, because one of the chosen models was a Convolutional Neural Network (CNN), whose input needed to be modified to (6,9,1) tensors.

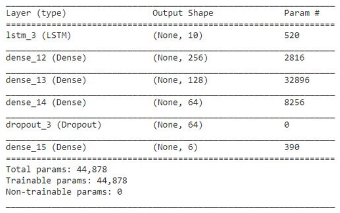
E. *Neural Network models*

In the project, as it has been mentioned, three models were created, using the library TensorFlow. Concretely, [19] was the source followed to implement them.
Since there is not an exact way to succeed when creating a NN model, the only way to go, as stated in [19] is to go on trying, generally from simpler to more complex models. This is the reason why three models with different learning capacities were tested. These three models correspond to a Simple Neural Network (sNN, the word *Simple* is just to state that it was the model with the lowest learning capacity out of the three, it does not mean that the model is inappropriate or the like, since, as it will be shown afterwards, it did not present the worst performance out of them), a LSTM (Long Short-Term Memory) Neural Network, and, finally, a Convolutional Neural Network (CNN).
The capacity of each one of the models created can be better seen in Fig.



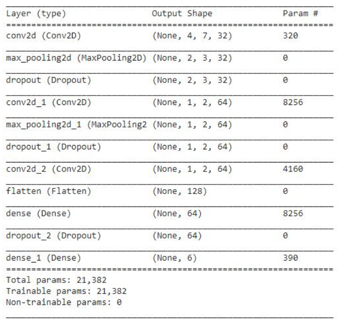
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_8 (Flatten)          (None, 54)                0
_________________________________________________________________
dense_32 (Dense)             (None, 128)               7040
_________________________________________________________________
dense_33 (Dense)             (None, 64)                8256
_________________________________________________________________
dropout_8 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_34 (Dense)             (None, 32)                2080
_________________________________________________________________
dense_35 (Dense)             (None, 6)                 198
=================================================================
Total params: 17,574
Trainable params: 17,574
Non-trainable params: 0
_________________________________________________________________
```

(a)



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_3 (LSTM)                (None, 10)                520
_________________________________________________________________
dense_12 (Dense)             (None, 256)               2816
_________________________________________________________________
dense_13 (Dense)             (None, 128)               32896
_________________________________________________________________
dense_14 (Dense)             (None, 64)                8256
_________________________________________________________________
dropout_3 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_15 (Dense)             (None, 6)                 390
=================================================================
Total params: 44,878
Trainable params: 44,878
Non-trainable params: 0
_________________________________________________________________
```

(b)



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 4, 7, 32)          320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 2, 3, 32)          0
_________________________________________________________________
dropout (Dropout)            (None, 2, 3, 32)          0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 1, 2, 64)          8256
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 1, 2, 64)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 1, 2, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 1, 2, 64)          4160
_________________________________________________________________
flatten (Flatten)            (None, 128)               0
_________________________________________________________________
dense (Dense)                (None, 64)                8256
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 390
=================================================================
Total params: 21,382
Trainable params: 21,382
Non-trainable params: 0
_________________________________________________________________
```

(c)

Fig. 7: Architecture of the models used in the project, showing their capacity: a) sNN, b) LSTM, and c) CNN.

F. *Software as a whole*

As it can be appreciated from the last steps, this project is about a compounding of several software steps to arrive

at a goal, which is the generation of a model that is able to generalise well when it is presented a video that it has not been seen beforehand.
The results of the models will be explained in detail afterwards, but, the purpose of this *Software as a whole* is for anyone who might want to use this software after seeing the output. Therefore, a single script has been made.
The outline of it is similar is just a gathering of all last steps, but adapted to this purpose. First of all, the input, as always, is a video (either lateral or frontal). After reading the video, the software gets the frames of the video by means of OpenCV [5]. Then, YOLOv3 [8] gets the ROIs of the athletes presented in those frames and extract the first four coordinates.
Once the ROIs are obtained, it directly computes those ROIs with OpenPose [11] [12] [14] [13] and obtain the 25 keypoints. It is important to state that it might happen that more than one person appears in a ROI. For that reason, more than one skeleton would be detected by OpenPose, and, consequently, the information related to those skeletons which are non-desirable for the project is also gathered. Thus, this script, also gets rid of those skeletons and keeps only the one that is related to the athlete and creates the arrays with the information coming from both OpenPose and YOLOv3.
So far, the procedure is similar to the previous steps that have been explained throughout the methods sections, but from now on, it differs a little bit more, since, right after that, a pre-trained model is opened (CNN model, which, as it will be discussed in the results section, it was the one that presented the best accuracy in both lateral and frontal videos), and the dataset which has been just obtained is classified.
The output of the model, as reflected in [19], is a vector of probabilities, corresponding to each one of the patterns of the project. Therefore, it is had, for every array (tensor once it is converted) which is directly related to a ROI, a probability to be pattern 1, another one to be pattern 2, and so on. Thus, by means of the function *find_peaks()* [20], patterns and, consequently, cycles, are found.
The starting prominence of this function is set to 0.9, so, if there is a cycle with all the patterns over that prominence, the script returns it, otherwise, the prominence starts decreasing in steps of 0.05 up until the algorithm is able to find a cycle accomplishing the prominence restriction.
In the end, the output is a folder (following [21], as always in this project), containing the athlete's cycle found under a certain prominence, as well as a .txt file where it is reflected the individual probability of each one of the patterns of the cycle found, i.e., how sure the model is about pattern 1 being really a pattern 1, pattern 2 being really a pattern 2, and so on.
Finally, this software part can be summarised like that:

- Video as an input and get its frames
- Get ROIs and their coordinates
- Get keypoints and select the desirable skeleton per ROI in case there is more than one
- Create the dataset and open the pre-trained models
- Find cycles along with their probabilites
- Folder as an output

III. Results

Before going into detail with all the results part, it is important to first state that despite there are lateral and frontal videos, the same model architecture of each one of the three model was use for both cases. For instance, as an example to clarify this, when it comes to LSTM, there are not two different LSTM model architectures, but just one, and this model is used to learn from lateral and frontal videos, so, eventually, two models with the same architecture are had.
Despite the models are the same for lateral and frontal, since the dataset is different, the weigths associated to them will be different. This means that, if the same model is trained with two different dataset, as it happened here, the result will be two models with the same architecture, but they will behave different beside a video once trained.
All the models were trained for 1000 epochs to see the point where the validation set accuracy did not improve by epoch anymore. After that, the model was then trained with up until that number of epochs. In this paper, only the CNN results were shown as figures, since this was the model that performed the best in both cases (frontal and lateral videos). Despite that, these results will be compared to the ones obtained with the other two models, so, although not all of the results will be shown graphically, they will be detailed anyway.
Finally, one of the graphical results that will be shown with regard to the CNN will be confusion matrix. It is important to state that the code to implement it was not retrieved from [19], but from [22].

A. Dataset

Since the number of lateral videos was not the same as the frontal one, the lateral dataset and the frontal one have got different size. This can be better seen in TABLE I:

Type	Number of tensors
Lateral	2562
Frontal	2521

TABLE I: Dataset of the frontal and lateral videos.

In both cases, the dataset was splitted, having got the 80% of it as a train set, and the lasting 20% as a test set. Also, out of the 80% train set, a 10% was kept as validation set.
The reason why a part of the train set has been kept as a validation set is because the accuracy of this last one will be key in terms of training the models, since, in general terms, there is a time when any model does not improve its performance any longer, even when it is being trained through more and more epochs [19]. For that reason, every model (in both cases, with the lateral and frontal dataset) will be trained, as previously said, for 1000 epochs at first, to see where that point is located. This way, they can be prevented from overfitting, a situation where the model is just mistaught, learning characteristics of the dataset that are not really generalisable [19].

B. CNN

1) Lateral videos: after training the CNN with lateral videos for 1000 epochs, the result obtained can be seen in Fig. 8:

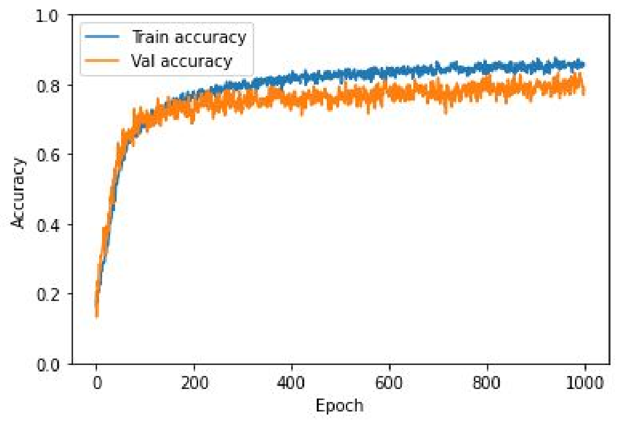


Fig. 8: Accuracy of the train and validation set after training the CNN model for 1000 epochs with the lateral dataset.

It can be visualised that around the epoch 300, the validation set starts drawing a plateau state, where the improvement is not really palpable. Besides, it is from this point that the train accuracy starts to slightly diverge from the validation one, suggesting that it is from here that the model starts to get overfitted.
This diversion is different in both sNN and LSTM models, being at the epoch 400 in the case of the first, and 170 when it comes to the second. A priori, this might sound logical because the learning capacity of the LSTM model is the highest, followed by the CNN, and, eventually, the sNN.

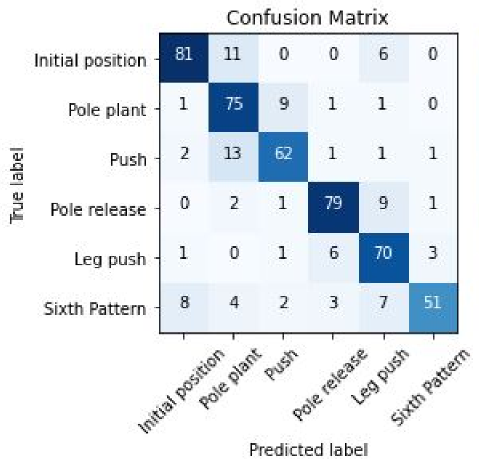
Afterwards, the CNN model was trained for 300 epochs, showing an overall accuracy of 81.48%, where each individual percentage for each pattern can be better seen in TABLE II.

TABLE II: Individual accuracy of each one of the patterns whilst classifying the lateral test set after training the CNN model.

Pattern	Accuracy
Initial position	82.65%
Pole Plant	86.21%
Push	77.5%
Pole release	85.87%
Leg push	86.42%
Pattern 6	68%

About the overall accuracy of sNN and LSTM, it was 77.00% and 51.27% respectively.

Finally, the confusion matrix for the CNN model can be seen depicted in Fig. 9:



True label \ Predicted label	Initial position	Pole plant	Push	Pole release	Leg push	Sixth Pattern
Initial position	81	11	0	0	6	0
Pole plant	1	75	9	1	1	0
Push	2	13	62	1	1	1
Pole release	0	2	1	79	9	1
Leg push	1	0	1	6	70	3
Sixth Pattern	8	4	2	3	7	51

Fig. 9: Confusion matrix of the CNN model once trained with the lateral dataset

It can be appreciated that the best performance correspond to pattern 5 (leg push), being the worst accuracy the one corresponding to pattern 6, something that will be discussed afterwards.

2) Frontal videos: when it comes to the frontal videos, the output after training the CNN model was similar to the lateral one, since the point where the validation set starts to become stable and the overfitting starts is also around the epoch number 300. This can be better seen depicted in Fig. 10:

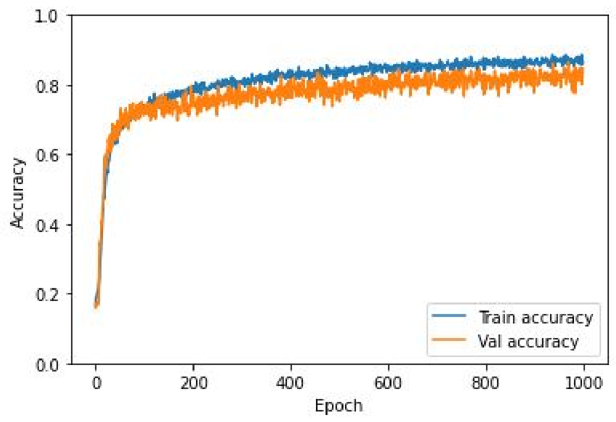


Fig. 10: Accuracy of the train and validation set after training the CNN model for 1000 epochs with the frontal dataset.

About the sNN and LSTM models, the diversion point starts at 300 and 200 respectively.
Therefore, same as before, the models were trained up until they start to overfit, and in the case of the CNN model, the overall accuracy obtained was 81.78%, which is considerably better than the sNN model, presenting a 66.34% of accuracy, and finally, the LSTM one, whose accuracy percentage falls until 63.76%.
The performance of the CNN model in each one of the patterns can be better visualised in TABLE III.

TABLE III: Individual accuracy of each one of the patterns whilst classifying the frontal test set after training the CNN model.

Pattern	Accuracy
Initial position	87.18%
Pole Plant	59.15%
Push	81.43%
Pole release	85.26%
Leg push	97.09%
Pattern 6	73.86%

Finally, the confusion matrix associated to the CNN frontal model is (see Fig 11):

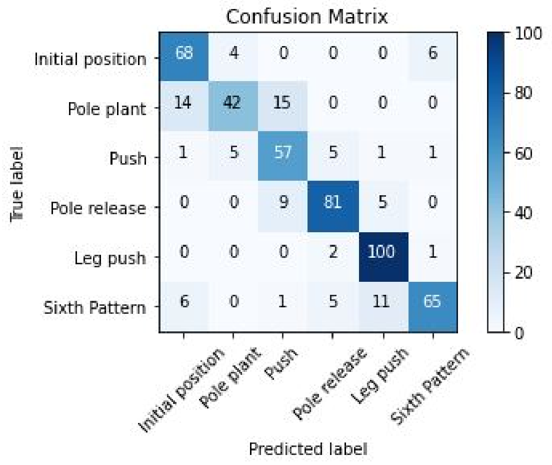


Fig. 11: Confusion matrix of the CNN model once trained with the frontal dataset

In this case, the best accuracy corresponds to the leg push once again, which it is almost 100%, by only getting confused two times with pattern 4 (pole release), and another one with the pattern six. But this time, the worst one lies in the pole plant. Despite that, the sixth pattern presents the second-worst accuracy, suggesting that it might not be a coincidence that this pattern is usually at the bottom of the best accuracies achieved by a model.
On the other hand, a deeper explanation of the findings will be given in the discussion section.

C. Software as a whole

With respect to this part, whose steps have been previously defined, it is important to say that, so far, it has not been possible to test it with large datasets, but with a few videos. This is because the videos provided by the IAT were used to train and test the NN models, since, for now, that has been the priority of the project.
The output of the software is a folder [21] with the same structure as the input folder containing both the frontal and lateral video, e.g., if the original folder has got the structure *Lateral* $\rightarrow$ *Athlete A* $\rightarrow$ *Video of athlete A*, then the layout of the output folder would be *Lateral* $\rightarrow$ *Athlete A* $\rightarrow$ *Cycles of athlete A found.*

IV. Discussion

The results of the three models that has been programmed for the scope of this project have been presented, leading to the conclusion that the model which performed the best was the CNN, which, in fact, is not either the one with the highest capacity or the lowest, but it remains in the middle.
The models were trained 1000 for epochs to better understand the point where the overfitting starts, and then, they were recompiled and trained up until that point.
In general terms, the model that needed the lowest number of epochs to "understand" the dataset was the LSTM, something that was expected, since it is the one with the highest capacity. When it comes to both the sNN and the CNN, the points were similar with the frontal dataset, and a little bit more different when it comes to the lateral one. Despite that, it can be said that their learning performance behaved similar, something that, a priori, is understandable, because despite their learning capacity is, as said, different, they are closed to each other with respect to this, than anyone of them is with LSTM.
The results obtained clarify that a highest capacity does not necessarily translate into a better performance, but it depends on the situation, and, in the end, on the dataset, since LSTM, being the one with the highest capacity, presented the worst accuracies once it was presented the test set.
On the other hand, it can be appreciated that, in general terms, the best accuracies always lied in the fifth pattern (leg push), something that was expected, since it is the most distinguishable movement in terms of the athlete's position. Conversely, the worst ones tended to pertain to the sixth pattern, which was also expected, since this pattern is more heterogeneous than the others, because, from pattern 1 to 5, the movement that the athlete performs is constant and clear, but, when it comes to the sixth pattern, the ROIs chosen were those ones which were not classified as a pattern 1,2,3,4 or 5. Besides that, the sixth pattern has got, beforehand, a disadvantage, and it is that despite being more heterogeneous, the dataset associated to it cannot be much larger than any other pattern, because, if that happens, it can happen that the model gets really biased because of the dataset, meaning that it would generalise really well in all about the sixth pattern, but then, the accuracies with respect to the other patterns would be lower, and that is not desirable in this case, because, the goal of the project, as it has been previously explained, is to obtain the five most important patterns of a XCS (Skating 1-1 in this case) video. Therefore, the utmost priority must be these first five patterns at all time, which is what has been done throughout this project.

Finally, the desirable output of the project would be a software which is able to directly read the content of an input folder, and directly gather the five most important patterns of the videos (content of the folder). This part has been also accomplished, although, as it has been described, it has not been possible to test this with large datasets, but with just some few videos, since the provided videos were used to train and test the models the way it has been explained.

V. Conclusion

This project comprises two main different parts: the first one is mostly about the reason why this work merges up, and it is the most theoretical part. In it, it has been stated that XCS is a high intensity physical activity (HIPA) that requires the activation of both the upper and the lower body [1]. From here, an outline of how the topics in literature with respect to XCS are changing, being traditionally more related to physiology, and, currently, more linked to biomechanics, being one of the main reasons the fact that, by means of technology, it is now possible to obtain kinematic and kinetic data, for a posterior analysis of them, so, in the end, a better understand of the relation between physiology and biomechanics will come up [4].
This trend is that strong, that even there are some articles in literature which introduce a term which exactly describes the scope of this project, which is *digital coaching* [23], since, in the end, the aim of this project is to create AI models that can generalise well beside unknown input data. This way, coaches could use this software to provide the athletes with instant feedback about their performance, substituting the manual traditional methods, which are really time consuming and require a lot of effort from coaches to have got to watch every single video and retrieve the five most important patterns by hand.

The second part of this project is the software one, which based on the above theory, aims to fulfil this need.
A whole software chain has been designed, accomplishing each one of the steps necessary to train and test NN models. Every part of this chain comprises different goals.
Throughout these steps, it is possible to find well-known software, like YOLOv3 [8] for object detection, or OpenPose [11] [12] [14] [13], for feature extraction, as well as also widely-used libraries like OpenCV [5] for frame obtainment, or TensorFlow [18], to build all related to the NN models.
Along the process there have been parts of code that needed to be created, such as a function for YOLOv3 in order to obtain the information that is useful for this project, or the selection of the right skeletons in case there are more than one in a ROI found, among others. In the end, this all led to the NN, which was the last step, and, for the scope of this work, three different types of NN (sNN, LSTM and CNN) with different learning capacities have been implemented, obtaining that the most ideal one for this project is the CNN, showing an overall accuracy of 81.48% for lateral videos, percentage that increases up to 81.78% for frontal ones, arriving to the conclusion that the pattern 5 (leg push) seems to be the easiest, in general terms, to classify, showing an overall accuracy of 91.76% if both lateral and frontal videos are taken into account. Conversely, the sixth pattern seemed to be the hardest one to differentiate, possibly because of the heterogeneity it presents. Therefore, this model has been chosen to be part of the final software that, despite it needs more testing with large datasets, it returns all the information related to the cycles found in a video.

Acknowledgment

First of all, I would like to thank my supervisor, Bernhard Hollaus, for introducing me to the topic and his support along these months, as well as Ronny Fudel, who, on behalf of the Institute for Applied Training Science (*Institut für Angewandte Trainingswissenschaft (IAT)* in German), provided the videos and the patterns related to the videos, so this project could be carried out.
Last but not least, I would also like to thank everyone who has supported me somehow all along this process. It would have not been the same without you.

References

[1] J. A. Laukkanen, S. K. Kunutsor, C. Ozemek, T. Mäkikallio, D.-c. Lee, U. Wisloff, and C. J. Lavie, "Cross-country skiing and running's association with cardiovascular events and all-cause mortality: A review of the evidence," *Progress in cardiovascular diseases*, vol. 62, no. 6, pp. 505–514, 2019.

[2] T. Losnegard, "Energy system contribution during competitive cross-country skiing," *European journal of applied physiology*, vol. 119, no. 8, pp. 1675–1690, 2019.

[3] C. Zoppirolli, K. Hébert-Losier, H.-C. Holmberg, and B. Pellegrini, "Biomechanical determinants of cross-country skiing performance: A systematic review," *Journal of sports sciences*, vol. 38, no. 18, pp. 2127–2148, 2020.

[4] S. Lindinger, P. Komi, O. Ohtonen, and V. Linnamo, "Developments and methodological aspects in cross-country skiing research," *Science and Noridc Skiing II–Proc 2nd Int Congr Science and Nordic Skiing (ICSNS). University of Jyväskylä, Jyväskylä*, pp. 13–22, 2013.

[5] G. Bradski, "The opencv library," *Dr Dobb's J. Software Tools*, vol. 25, pp. 120–125, 2000.

[6] Paperspace. Data augmentation for object detection. Accessed 2021-07-31. [Online]. Available: https://github.com/Paperspace/DataAugmentationForObjectDetection

[7] Numpy. numpy.random.uniform. Accessed 2021-07-31. [Online]. Available: https://numpy.org/doc/stable/reference/random/generated/numpy.random.uniform.html#numpy-random-uniform

[8] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[9] OpenCV. Introduction to SIFT (Scale-Invariant Feature Transform). Accessed 2021-07-31. [Online]. Available: https://docs.opencv.org/master/da/df5/tutorial_py_sift_intro.html

[10] ——. Feature Matching. Accessed 2021-07-31. [Online]. Available: https://docs.opencv.org/4.5.2/dc/dc3/tutorial_py_matcher.html

[11] Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh, "Realtime multi-person 2d pose estimation using part affinity fields," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7291–7299.

[12] Z. Cao, G. Hidalgo, T. Simon, S.-E. Wei, and Y. Sheikh, "Openpose: realtime multi-person 2d pose estimation using part affinity fields," *IEEE transactions on pattern analysis and machine intelligence*, vol. 43, no. 1, pp. 172–186, 2019.

[13] T. Simon, H. Joo, I. Matthews, and Y. Sheikh, "Hand keypoint detection in single images using multiview bootstrapping," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2017, pp. 1145–1153.

[14] S.-E. Wei, V. Ramakrishna, T. Kanade, and Y. Sheikh, "Convolutional pose machines," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2016, pp. 4724–4732.

[15] J. Scott, C. Funk, B. Ravichandran, J. H. Challis, R. T. Collins, and Y. Liu, "From kinematics to dynamics: Estimating center of pressure and base of support from video frames of human motion," *arXiv preprint arXiv:2001.00657*, 2020.

[16] CMU-Perceptual-Computing-Lab. openpose. Accessed 2021-07-31. [Online]. Available: https://github.com/CMU-Perceptual-Computing-Lab/openpose

[17] NumPy. numpy.array. Accessed 2021-07-31. [Online]. Available: https://numpy.org/doc/stable/reference/generated/numpy.array.html

[18] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[19] TensorFlow. Tensorflow core. tutorials. Accessed 2021-07-31. [Online]. Available: https://www.tensorflow.org/tutorials

[20] SciPy. scipy.signal.find_peaks. Accessed 2021-07-31. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.find_peaks.html

[21] T. point. Python os.mkdir() Method. Accessed 2021-07-31. [Online]. Available: https://www.tutorialspoint.com/python/os_mkdir.htm

[22] Deeplizard. Keras - Python Deep Learning Neural Network API. Accessed 2021-07-31. [Online]. Available: https://deeplizard.com/learn/video/km7pxKy4UHU

[23] J. Tjønnås, T. M. Seeberg, O. M. H. Rindal, P. Haugnes, and Ø. Sandbakk, "Assessment of basic motions and technique identification in classical cross-country skiing," *Frontiers in psychology*, vol. 10, p. 1260, 2019.