Neural Computing and Applications
https://doi.org/10.1007/s00521-020-05485-3

ORIGINAL ARTICLE

The detection, tracking, and temporal action localisation of swimmers for automated analysis

Ashley Hall[1] · Brandon Victor[1] · Zhen He[1] · Matthias Langer[2] · Marc Elipot[3] · Aiden Nibali[1] · Stuart Morgan[4]

Received: 17 December 2019 / Accepted: 27 October 2020

Abstract
It is very important for swimming coaches to analyse a swimmer's performance at the end of each race, since the analysis can then be used to change strategies for the next round. Coaches rely heavily on statistics, such as stroke length and instantaneous velocity, when analysing performance. These statistics are usually derived from time-consuming manual video annotations. To automatically obtain the required statistics from swimming videos, we need to solve the following four challenging computer vision tasks: swimmer head detection; tracking; stroke detection; and camera calibration. We collectively solve these problems using a two-phased deep learning approach, we call Deep Detector for Actions and Swimmer Heads (DeepDASH). DeepDASH achieves a 20.8% higher F1 score for swimmer head detection and operates 6 times faster than the popular Faster R-CNN object detector. We also propose a hierarchical tracking algorithm based on the existing SORT algorithm which we call HISORT. HISORT produces significantly longer tracks than SORT by preserving swimmer identities for longer periods of time. Finally, DeepDASH achieves an overall F1 score of 97.5% for stroke detection across all four swimming stroke styles.

Keywords Object detection · Tracking · Temporal action recognition · Deep learning · Convolutional neural networks

1 Introduction

Swimming coaches depend on stroke count, instantaneous velocity, stroke rate and distance per stroke per sections of the race to analyse the performance of athletes. Currently, these statistics are manually annotated from video at great expense. Using computer vision algorithms to automatically annotate races will significantly scale up the annotation of swimmers. Using an algorithm to determine the real-world coordinates of swimmers can also improve consistency as manually estimating velocity from a 2D image is error-prone.

We thank the Australian Institute of Sports, Swimming Australia and Optus for providing the research innovation grant used to carry out this research.

✉ Zhen He
z.he@latrobe.edu.au

Ashley Hall
ash.hall@latrobe.edu.au

Brandon Victor
b.victor@latrobe.edu.au

Matthias Langer
matthias@kanzhun.com

Marc Elipot
marc.elipot@ausport.gov.au

Aiden Nibali
a.nibali@latrobe.edu.au

Stuart Morgan
Stuart.Morgan@ausport.gov.au

[1] Department of Computer Science, La Trobe University, Bundoora, Australia

[2] Career Science Lab (CSL), BOSS ZhiPin, Metzingen, Germany

[3] Swimming Australia, Canberra, Australia

[4] Australian Institute of Sport, Canberra, Australia

Published online: 18 November 2020

In this project we take a video of the entire pool and use computer vision algorithms to automatically track each swimmer and detect each stroke. Using this information we can derive the stroke count, instantaneous velocity in the x-axis (along the swimming lane), and distance per stroke for each swimmer. This requires solving the following four computer vision problems: object detection, tracking, temporal action localisation, and camera calibration. It is very challenging to solve all four problems in near real time for all swimmers in a race. We use a two-phased deep learning model to simultaneously detect the swimmers and their actions: creating swimmer head proposals, then refining, and detecting strokes. We then apply a tracking algorithm to associate the head locations across time to track the swimmers. Using calibrated camera parameters, we are able to project the image-space tracks into 3D pool space and extract useful analytical metrics with real-world units. We name our overall system Deep Detector for Actions and Swimmer Heads (DeepDASH).

Deep learning algorithms are central to the success of DeepDASH. Recently, deep learning algorithms have been successful in a wide range of application areas, including object detection [9, 18, 38, 40], multi-object tracking [42, 57], credit scoring [36, 37], authentication [17], activity recognition [48], etc.

Figure 1 shows the output of DeepDASH run on previously unseen video, where the white lines are the model-estimated trajectories, and the blue circles are the model-estimated swimmer head locations of when strokes occur. Although the model is not perfect, it achieves high accuracy for the vast majority of predictions. The relatively small swimmers, glare from the sun, and occlusions from audience and camera operators make the task challenging.

Similarly to the popular Faster R-CNN [40] object detection model, our deep learning model uses distinct proposal and refinement phases. We found the use of the refinement phase improves swimmer head detection accuracy dramatically, almost doubling the F1 score. Object detection in DeepDASH differs from Faster R-CNN by exploiting contextual information from multiple input frames, predicting point locations rather than bounding boxes, and using a different non-maximal suppression method. Our head detection algorithm outperforms Faster R-CNN [40] by 20.8 F1 score and is more than sixfold faster. This shows the benefits of using a custom object detector instead of an off-the-shelf alternative like Faster R-CNN.

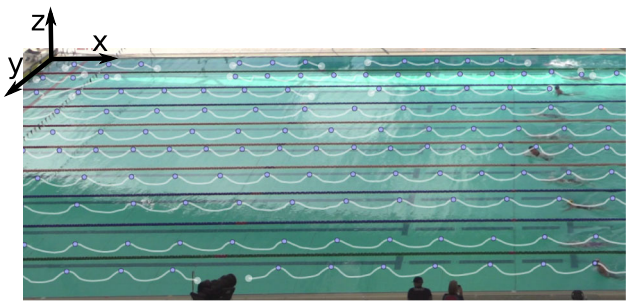


Fig. 1 Example tracking and stroke detection results on a previously unseen video. The white lines are the model-estimated trajectories, and the blue circles are the model-estimated locations when a stroke occurred (colour figure online)

Due to the need to track all swimmers in a race in near real time, we have to use a fast tracking algorithm, such as Simple Online and Realtime Tracking (SORT) [4]. However, we found that SORT produced many disjoint short tracks. We therefore developed the hierarchical SORT algorithm (HISORT) which iteratively joins shorter tracks created by SORT using increasingly more relaxed joining constraints. HISORT produces significantly longer tracks than SORT. This is evidenced by improving the IDF1 tracking metric [25] (how well identities are preserved on tracks) from 60.4 to 74.1

To count strokes and stroke length we need to know the exact moment when each stroke starts. We frame this as a temporal action localisation problem. We follow the work by Victor et al. [49] and turn the binary labels (start of stroke or not) into a smooth signal in the shape of a truncated sinusoid. In the refinement phase of DeepDASH we use multitask learning to simultaneously compute the stroke probability of the swimmer as well as the swimmer's head location. This minimises additional computational overhead for predicting stroke probabilities and achieves more than 97% F1 stroke prediction accuracy across all four stroke styles.

We make the following key contributions:

1. Most existing works solve a single computer vision problem, in contrast this paper proposes DeepDASH which collectively solves the challenging tasks of swimmer head detection, tracking, stroke detection, and camera calibration.
2. We propose a novel CNN based solution which is significantly more accurate than Faster R-CNN for swimmer head detection.
3. We propose a tracking algorithm called hierarchical SORT (HISORT), which produces significantly longer tracks than the popular SORT tracking algorithm.
4. We show that DeepDASH detects and tracks all swimmers in a race faster than real time.

The rest of the paper is organised as follows. We present the related works in Sect. 2. Section 3 describes how DeepDASH solves the four problems: swimmer head location tracking; stroke detection; swimmer tracking; and 2D image space to 3D pool space projection. Section 4

describes the experimental setup. Section 5 describes the experimental results comparing DeepDASH to alternatives. Finally, in Sect. 6 we summarise our findings and discuss areas of future work.

2 Related works

In this section we review existing work in object detection, multi-object tracking and temporal action localisation for untrimmed video. In addition we review existing work on applying computer vision techniques to swimming analysis.

2.1 Object detection

The aim of object detection is to predict a bounding box for every object and its associated class from an image. All recent high performing object detection algorithms use deep learning models and can be broadly categorised as either *two-stage detectors* or *one-stage detectors*.

Two-stage detectors first output region proposals, followed by refinement to produce bounding boxes and associated classes. R-CNN [15], Fast R-CNN [14], Faster R-CNN [40], Mask R-CNN [18], and feature pyramid networks [29] have progressively improved the accuracy and speed of two-stage object detection algorithms.

One-stage object detectors such as YOLO [38], YOLO9000 [39], SSD [31], RetinaNet [30], CornerNet [23], FCOS [45], and CenterNet [9] predict the object bounding boxes and class in a single step without explicit region proposals. Hence, they are usually faster than two-stage predictors. However, they are not as accurate as the best two-stage detectors.

In this paper we propose a two-stage object detector that is fast enough to run in real time. In contrast to most existing approaches, we take multiple input frames and output the coordinates of each swimmer's head instead of the coordinates of a bounding box. Thus, our detector uses information from adjacent frames to estimate the head location of the middle frame, and this mitigates the negative effects of occlusion caused by water splashes. Our pointwise detection approach allows us to directly position the swimmer within the pool.

Generic object detection datasets have a wide range of object sizes and shapes, and many generic object detectors [18, 39, 40] use several differently sized anchor boxes as an initial guess of an objects size and shape. In this work, all of our objects (swimmers) are similarly sized and shaped, and thus, we do not use anchor boxes.

2.2 Multi-object tracking

Most state-of-the-art multi-object tracking algorithms use the tracking-by-detection approach. In this approach, object bounding boxes are detected first, and then, the tracking algorithm associates the detections across time to create trajectories.

Most tracking-by-detection approaches score how similar two detections are between adjacent frames. Then, an assignment algorithm (such as the Hungarian algorithm [22]) is used to associate pairs of bounding boxes (one from each frame). The similarity score can be based on the location information [35, 42, 55], motion information [24, 26, 42], and appearance features [42, 50, 57]. Recent methods [42, 57] have used RNNs to determine the similarity score by using the appearance and movement history of each trajectory to associate it with a bounding box in the next frame.

A popular simple and fast tracking-by-detection multi-object tracking algorithm is the Simple Online and Realtime Tracking (SORT) [4] algorithm. SORT works by assigning a distance value between detections in neighbouring frames. It uses predicted object motion instead of appearance features to compute the distance measure. This distance value is then used by the Hungarian algorithm to associate the next detection to an existing track. Deep SORT [51] improved upon SORT by using CNNs to extract appearance features which are then used to compute the distance measure for association.

In contrast to these existing works we track the head locations of swimmers instead of bounding boxes of objects. The zoomed out view of the single camera covering the entire pool means the size of each swimmer's head is very small and similar in appearance, thus severely limiting the usefulness of appearance features for association. Hence, our tracking algorithm does not use appearance features but instead just uses the distance between head detections in adjacent frames as the similarity metric. Inspired by the simplicity and fast processing speed of the Simple Online and Realtime Tracking (SORT) [4], we extended it to create the hierarchical SORT tracking algorithm, which differs from the simple SORT [4] tracking algorithm in a number of ways as described in Sect. 3.3.1.

2.3 Temporal action localisation for untrimmed video

Temporal action localisation takes untrimmed video and predicts when actions occur and also classifies them. Existing methods can be divided into two-stage and one-stage approaches. Our problem of detecting the start of a

swimming stroke is an example of the temporal action localisation problem.

In two-stage approaches class agnostic action proposals are generated and then classified separately. Some work focuses on the proposal generation stage [6, 13, 28, 44], while others focus on the classification stage [8, 53, 56]. Shou et al. [44] proposed segment CNN (S-CNN), which make predictions at the temporal segment grain which they later extended [43] to make fine grained (per frame) predictions instead. Cao et al. [7] adapt the ideas of anchors from Faster R-CNN to solve the temporal action localisation problem.

One-stage approaches integrate proposal and classification into a single step. These models typically extract per frame features which are then fed into a 1D CNNs [27] or RNNs [5] to make localisation and classification predictions together. In contrast, the method proposed by Huang et al. [20] solves the localisation and classification tasks in two separate branches of the model.

Nibali et al. [32] performed temporal action localisation using a 3D CNN model to find the start, middle and end of dives and then used two more 3D CNNs to separately detect the location of the diver and classify each dive.

The above works either do not do any tracking or just track one person at a time [32]. In contrast, we need to track and detect actions for multiple people simultaneously. Hence, we need to associate detections across time in order to produce tracks, whereas the other works do not.

2.4 Computer vision for swimming analysis

There are a number of papers focused on estimating the pose keypoints of swimmers [10, 54] and also classification of swimming poses [12]. In contrast, we are interested in detecting the start of a stroke instead of swimmer pose.

Our earlier work [49] was focused on just detecting the beginning of swimming strokes from a zoomed in panning camera focused on a single swimmer. The problem of this paper is much more challenging for two main reasons. First, we require detection and tracking because of the zoomed out video of the entire pool. Second, to derive meaningful analytical metrics in real-world coordinates we must project the swimmers position from image space into 3D pool space.

Tsumita el al. [47] study the problem of estimating each swimmer's position using a single video of the entire pool as input. They use adaptive background modelling to help extract swimmer regions. Then, they estimate the location of a swimmer as the centre of a Gaussian distribution of the swimmer region. In contrast, we use deep neural networks to detect the location of swimmers.

Like our work, Hakozaki et al. [16] first performs swimmer detection on zoomed out footage of the entire swimming pool and also detects the beginning of strokes. They locate the swimmers using background subtraction and iteratively update the swimmer's position with a Kalman filter. Next they detect the start of strokes using an LSTM. In contrast to their work we use a multitask deep neural network to detect the swimmer's heads and the beginning of their stroke simultaneously. They do not measure tracking accuracy and use only freestyle videos. In contrast, we measure tracking accuracy and consider all four swimming styles.

3 DeepDASH

In this section we present how DeepDASH performs its four tasks: detecting swimmer head locations; detecting the start frame of swim strokes; tracking swimmers for the duration of the race; and mapping swimmer head locations from 2D images space to 3D pool space. We will begin by describing how DeepDASH uses a two-phased approach with multitask learning to simultaneously detect swimmer head locations and the start of strokes, and then proceed to describe the tracking and coordinate space mapping afterwards.

Figure 2 shows the two phase architecture used by DeepDASH for swimmer detection. DeepDASH takes multiple input frames and passes the data through two phases of processing to predict the stroke start probabilities and head coordinates for all swimmers in the middle input frame. In the first (region proposal generation) phase, a convolutional neural network (CNN) is used to output a confidence heatmap of the estimated swimmer location. A confidence threshold is applied to the heatmap in order to determine midpoint locations for cropping sub-images containing swimmer heads. In the second (refinement) phase, the crops resulting from the first phase are fed into a CNN body that is connected to two separate CNN heads. The first CNN head outputs the probability of the crop containing a swimmer executing the start of a stroke. The second CNN head outputs a refined location of the swimmer's head in each cropped image.

We run a tracking algorithm on the head locations generated from the refinement phase to produce tracks of swimmers. We then project the coordinates of the associated head positions from 2D image space into 3D pool space in order to give an estimate of the swimmer's velocity along the x-axis (how quickly the swimmer travels along the length of the pool). Swimmers executing breaststroke and butterfly have a large amount of vertical movement. Ignoring vertical motion for these styles can greatly impact the accuracy of the projected x-axis positions. We demonstrate this problem in Sect. 3.5 and specify

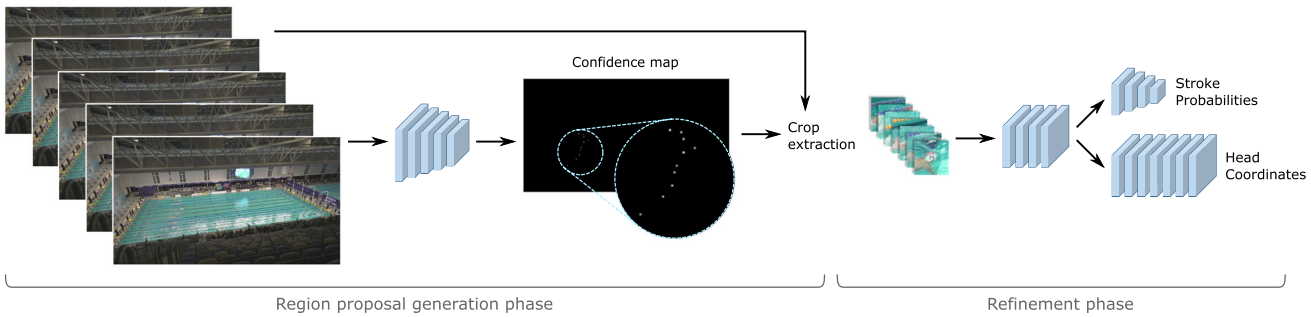


Fig. 2 Diagram showing the architecture of DeepDASH for detecting head location and detecting the start of swimming strokes. The detailed model specifications for the region proposal generation phase and refinement phase are presented in Tables 1 and 2, respectively

Table 1 Fully convolutional CNN architecture used for the region proposal generation phase

Block	Layers
1	3×3 conv, 16 output channels
	3×3 max pool, 2 stride
	4 groups of groupnorm
	Relu activation
2	3×3 conv, 16 output channels
	3×3 max pool, 2 stride
	4 groups of groupnorm
	Relu activation
3	3×3 conv, 32 output channels
	3×3 max pool, 2 stride
	8 groups of groupnorm
	Relu activation
4	3×3 conv, 32 output channels
	3×3 max pool, 2 stride
	8 groups of groupnorm
5	3×3 conv, 1 output channel
	Sigmoid activation

All convolutions had 3×3 kernels, a stride of 1 and padding of 1. All max pooling layers had a padding of 1. The input to the model is $[n * d, h = 1080, w = 1920]$, where $n = 5$ is the number of fused frames and $d = 3$ is the number of channels (R, G, B)

how we solve the problem by mapping coordinates from 2D image space to 3D pool space.

3.1 Region proposal generation phase

The region proposal generation phase of our approach is analogous to the region proposal stage commonly found in two-stage object detectors, but differs in two key ways: the input is multiple 1080p resolution video frames instead of one frame, and our system outputs pointwise coordinate predictions instead of bounding box predictions.

Multiple input frames The output of our entire system is the stroke probability and head position for each swimmer for a single frame. However, we take as input n frames centred around the frame of interest. This is because a single video frame often does not contain enough visual information to accurately predict a swimmer's location and stroke probability. This is due to environmental effects such as submersion and splashing. Furthermore, predicting stroke probability for symmetrical strokes such as breaststroke depends on detecting the peak of the swimmer's vertical elevation, which would otherwise have a high degree of ambiguity using single-frame input.

We use early fusion to combine the information from the n input frames by feeding multiple frames as separate input channels to a 2D CNN. Therefore, each input example has shape $[n * d, h, w]$, where n is the number of fused frames, $d = 3$ is the number of channels for each video frame, and h and w are the frames' height and width, respectively. In our experiments we found that setting $n = 5$ gave a good balance between computational expense and accuracy. We use every third frame at 25fps and maintain the time between frames for other frame rates.

Our previous work [49] showed that early fusion gave good performance for encoding temporal video information without the need to use more complex architectures such as 3D CNNs. By stacking video frames depth-wise, the model is able to combine visual features from a neighbourhood of frames and leverage implicit temporal video consistencies.

Pointwise coordinate predictions. To derive velocity, some specific part of the body must be consistently tracked, and thus, we ultimately predict a single $(x; y)$ coordinate pair for each swimmer's head. This contrasts with generic object detectors such as Fast/Faster R-CNN [14, 40], YOLO9000 [39] and SSD [31] which typically predict bounding boxes for objects of interest. Bounding boxes around the whole swimmer are not viable as the extension of a swimmer's arm would result in the reshaping of their bounding box, artificially changing their position and

Table 2 Fully convolutional CNN architecture used for the refinement phase

Body	Coordinates head	Stroke detection head
3×3 conv, 16 out ch	3×3 conv, 128 out ch, 16 dilation	3×3 conv, 128 out ch, 16 dilation
Groupnorm	Groupnorm	Groupnorm
Relu activation	Relu activation	Relu activation
3×3 conv, 32 out ch, 2 dilation	3×3 conv, 256 out ch	3×3 conv, 256 out ch, 2 stride
Groupnorm	Groupnorm	Groupnorm
Relu activation	Relu activation	Relu activation
3×3 conv, 64 out ch, 4 dilation	3×3 conv, 1 out ch	3×3 conv, 1 out ch, 2 stride
Groupnorm	Groupnorm	Spatial global max pool
Relu activation	Relu activation	Sigmoid
3×3 conv, 128 out ch, 8 dilation	soft-argmax	
Groupnorm		
Relu activation		

All convolutions have a stride of 1 and a padding matching their dilation rate (e.g. 4 pad for 4 dilation). All groupnorms used groups equal to the number of channels divided by 4. The smallest number of groups is lower bounded by 1. A shared body feeds into two separate heads to perform the multitask learning of stroke detection and swimmer head coordinates regression. The input to the refinement phase is $[n * d, H = 48, W = 80]$ crops of the original 1080p input image centred around the head of the swimmer, where $n = 5$ is the number of fused frames and $d = 3$ is the number of channels (R, G, B)

subsequently their velocity. However, we can use generic object detectors to derive velocity information if we place a bounding box centred on a swimmer's head and use the centre of the predicted bounding boxes for evaluation.

We took a pre-trained Faster R-CNN model and fine-tuned it on our dataset, but found that it did not perform well. Small objects are known to be difficult for generic object detectors [31], and by the standards of VOC2012 [11], all of our swimmers are very small (frequently <60 pixels high at 4k resolution). Additionally, these generic object detectors [14, 31, 40] operate only a single frame at a time. Thus, we create our architecture to be specific to our task and use early fusion to utilise the motion information as in [49]. In Sect. 5, we show that our architecture performs significantly better than Faster R-CNN on our task.

3.1.1 Heatmap prediction

Due to the translationally equivariant nature of convolutional neural networks, it is not immediately obvious how to output swimmer locations as numerical coordinates. Owing to the varying number of swimmers detected in the region proposal generation phase, we first produce a heatmap of swimmer detection likelihoods to indicate coarse regions of interest, thus allowing the use of a separate coordinate regression model to predict a single swimmer location per region.

Table 1 contains a precise description of the CNN architecture used in the region proposal generation phase. Each sample x_i—consisting of five early-fused video frames—is fed into a fully-convolutional network, producing a single-channel feature map, to which the sigmoid activation function is applied. The resulting output is a heatmap, $\hat{H}_g$, which is interpreted as a two-dimensional grid of swimmer detection likelihoods. Each block in this network consists of a 2D convolution, max-pool with a stride of 2, group normalisation, and ReLU activation function. The number of blocks is determined such that the inputs have been spatially downsampled by a factor of 16 at the end. The dimensions of the output heatmap $\hat{H}_g$ are therefore [1, h/16, w/16], where h and w are the height and width of the original image in pixels.

It was decided that our model must be usable on a single modern GPU and run in real time. Our input data are HD images (1920×1080) which are very large images for deep learning models. As the deep learning model, and especially the region proposal generation, is the most computationally expensive part of our system, we deliberately constrained our region proposal architecture to a very small network. We chose to use the minimum number of layers needed to downsample by a factor of 16. Each point in the heatmap is a potential crop position, and thus, each point in the heatmap can only ever result in a single prediction. A downsampling factor of 16 is the largest power of 2 which is still small enough that there is almost no chance of two heads appearing in the same region. Following standard practice, when we downsample, we increase the number of features of the next convolution

layer. We empirically determined that group normalisation [52] was marginally better than batch normalisation [21].

Targets We build a two-dimensional target heatmap H_g per example y_c, by rendering circular 2D-Gaussian centred on each of the ground truth coordinates $\mathcal{N}(y_{c_i}, 0.425)$ and taking the max per pixel. The use of a Gaussian-shaped soft target for representing ground truth locations is well-established in the existing keypoint detection literature [46].

It is important to note that the values in the target heatmap are heavily skewed with a high percentage of zero values (typically less than 0.5% of the output is nonzero). If we employ a standard mean- square error loss to train a model to output the heatmap, it would result in the model outputting zero values for all locations. A common way to address class imbalance is to adopt a weighted loss. Hence, we employ the following weighted mean squared error loss:

$$\mathscr{L}_1(\hat{H}_g, H_g) = \big(H_g \cdot (\gamma - 1) + 1\big)\big(H_g - \hat{H}_g\big)^2 \tag{1}$$

where γ is a scalar hyper-parameter which determines the weighting coefficient applied to positive cells relative to negative cells. This value can be increased to favour recall over precision, with too low a value resulting in the model converging to an all-zero heatmap. In our experiments we found $\gamma = 12$ gave the best trade-off between precision and recall.

3.1.2 Extracting crop coordinates

The aforementioned network produces a heatmap $\hat{H}_g$ per example (represented as a depth-1 feature map with individual elements in the range [0, 1]). To convert this map into an arbitrary length list p_i of (x, y) predictions, we list all coordinates within $\hat{H}_g$ that have values above 0.2 (empirically determined to yield best results).

For each of these predicted points, a crop is taken from the early-fused input frames which corresponds to a 3×5 region in the heatmap, centred on the predicted point. Due to the downsample factor of 16, each cropped image is of shape $[n * d, H = 48, W = 80]$.

3.1.3 Crop proposal refinement

Non-maximum suppression As a single swimmer head may appear in more than one cell, simple confidence thresholding often results in positive predictions in multiple adjacent cells for the same swimmer. To mitigate this, the heatmap has a form of non-maximum suppression applied. A window with the same spatial dimensions as the desired crop is slid over the heatmap, where the centre value is zeroed out if there exists a higher confidence cell within the window.

An example failure case for this is when a swimmer's foot is incorrectly identified as the head because it is far enough away. Other types of erroneous detections like this are typically removed during tracking (see Sect. 3.3).

Jittering During training the ground truth swimmer crop coordinates are known, resulting in the ability to create perfectly centred crops. If each crop that is fed into the refinement phase is centred perfectly on a swimmer's head, then the refinement phase will just learn to always return the centre coordinate of each crop as the head location. This would render the regression portion of the refinement phase redundant. At inference time we do not know the precise ground truth coordinates of each head, and hence, we will not be able to centre the crops perfectly on the head. Therefore to mimic the conditions during inference, we jitter the location of the crops during training by randomly translating the crop coordinates by up to 90% of the size of the window in both directions, effectively moving the swimmer's head randomly within the crop with each training example.

3.2 Refinement phase

While the region proposal generation phase was concerned with quickly determining the approximate positions within the whole image, the refinement phase is concerned with determining the precise locations and actions of the swimmers. The refinement phase takes small crops ($[n * d, H = 48, W = 80]$) of the original images and produces high-detail head location heatmaps for individual swimmers, $\hat{H}_c$. The smaller crops allow us to use more convolutional layers than the region proposal generation phase while still maintaining real-time inference speed. As in the region proposal generation phase, we use early fusion to combine crops from a window of frames, using the same crop location on each frame in that window. Unlike the region proposal model, we use dilated convolutions in the refinement phase, with an increasing dilation factor through the network (see Table 2) to allow each neuron to have a larger receptive field.

The region of interest heads in Fast/Faster R-CNN take as input image features after running a whole CNN on the input image. In contrast, our refinement phase takes as input crops of the original image. All R-CNN variants use hundreds or even thousands of region proposals per image with a lot of overlap. This overlap means that they get a large speed improvement by running the initial convolution layers on the whole image once and performing the region of interest pooling (RoI) on those features. We found that using the first phase features in addition to the input images

as input to the refinement phase produced a lower F1 score (see Sect. 5.1). And for our task, we found that we could produce a precise set of high-quality region proposals with only a few convolution layers. These regions cover a very small proportion of the image ($<1\%$) and do not overlap often, so we would not get any significant speed improvement from using the first phase features, either.

Multitask learning is used to output both the swimmer location and the action probabilities. This is implemented with a shared body of convolution operations and separate heads for each predicted property (see Table 2 for detailed model specifications). Here we only predict swimmer location and strokes, but it would be trivial to introduce more heads for predicting additional actions (e.g. breaths). By sharing several layers of computation between these tasks we increase the speed of the refinement phase. We believe that there is sufficient representational overlap between the tasks (e.g. both need to separate the swimmer from the background) that the accuracy would increase too.

3.2.1 Swimmer coordinates

We considered three ways to translate the resultant heatmaps of the swimmer head location into coordinates: a fully connected layer; argmax; and soft-argmax [33]. For DeepDASH we opted to use soft-argmax as it has several desirable properties for precise location prediction. Soft-argmax works by using a softmax to convert the heatmap into a 2D probability distribution, then finding the expected location within the domain $[-1, 1]$ for both dimensions (x and y) to produce differentiable normalised sub-pixel coordinates. Soft-argmax is translationally equivariant but a fully connected layer is not.[1] Where argmax effectively quantises predicted coordinates to whole pixel values (corresponding to several centimetres in real-world coordinates), soft-argmax allows regression to arbitrary precision. This is especially important in our case since the ground truth annotations are at a higher resolution (4K) than the input to the model (1080p). Hence, soft-argmax has the potential to output coordinates at the 4K pixel grain whereas argmax can only output at the 1080p pixel grain.

The primary coordinate loss ($\mathscr{L}_c$) is simply MSE between the predicted coordinates and the target coordinates:

$$\mathscr{L}_c\left(\hat{H}_c, y_c\right) = \mathrm{MSE}\left(\mathrm{softargmax}\left(\hat{H}_c\right), y_c\right) \tag{2}$$

where $\hat{H}_c$ are the coordinate heatmaps produced by the model and y_c are the true coordinates.

[1] The convolution layers preceding this operation are not completely translationally equivariant due to image boundary effects, but a fully connected layer only exacerbates this problem.

Training with soft-argmax and only using the primary coordinate loss is under-constrained since there are many different heatmaps that can result in the same numerical coordinate predictions. Nibali et al. [33] found that a secondary heatmap loss that compares the predicted heatmap to an idealised heatmap with a Gaussian target centred on the ground truth location produces more reliable predictions. Thus, we also use a secondary heatmap matching loss ($\mathscr{L}_h$) calculated using Jensen–Shannon divergence (JSD):

$$\mathscr{L}_h\left(\hat{H}_c, y_c\right) = \mathrm{JSD}\left(\hat{H}_c, \mathcal{N}(y_c, \sigma)\right) \tag{3}$$

The standard deviation of the target Gaussian, σ, was set to 0.85 in our experiments based on findings of a hype-parameter search (see Sect. 5.5). This value creates Gaussian blobs approximately the size of a head in the average case.

3.2.2 Stroke probabilities

For predicting swimmer stroke actions, we follow our earlier work [49]. In our earlier work we found that the stroke detection model is able to achieve much higher accuracy when the problem is changed to a regression problem where the target label is softened using a truncated sinusoid with its peak aligned with the start of a stroke, instead of just performing binary classification (where target is 1 at the start of the stroke and 0 everywhere else). The rationale for this is that the "start of stroke" frame is very similar in appearance to neighbouring frames, and therefore, it would hinder the optimisation procedure to penalise the model for producing a high stroke probability on these neighbouring frames. The softened targets, $G(n)$, are defined as follows:

$$G(n) = \begin{cases} \cos\left(\dfrac{|l_n - n|}{2z+1}\pi\right) & \text{if } |l_n - n| < z \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

where n is the frame number, l_n is the frame number of the nearest labelled event and z controls the spread of the soft targets. In our experiments we use $z = 5$, which means that the extent of nonzero labels surrounding each action event is 11 frames wide.

Let $y_s = \{G(n) | n \in F\}$ be the set of transformed target probabilities for the set of all video clip frames F used for training. The loss ($\mathscr{L}_s$) between the predicted probability $\hat{y}_s$ and the transformed target probability y_s is computed by the equation below:

$$\mathscr{L}_s(\hat{y}_s, y_s) = \mathrm{MSE}(\hat{y}_s, y_s) \tag{5}$$

In our earlier work we found training a single model to predict strokes for all swimming styles worked just as well as training a separate model for each swimming style

individually. Hence, we take that same approach in this paper.

3.2.3 Combined loss

The loss for the refinement phase is the sum of the previously mentioned losses, multiplied by a coefficient $\lambda = 0.2$ to balance it with the proposal generation phase's loss. That is,

$$\begin{aligned}\mathscr{L}_2(\hat{H}_c, \hat{y}_s, y_c, y_s) = \; & \mathscr{L}_c(\hat{H}_c, y_c) \\ & + \mathscr{L}_h(\hat{H}_c, y_c) \\ & + \mathscr{L}_s(\hat{y}_s, y_s)\end{aligned} \tag{6}$$

$$\begin{aligned}&\mathscr{L}(\hat{H}_g, \hat{H}_c, \hat{y}_s, y_c, y_s) \\ &\quad = \mathscr{L}_1(\hat{H}_g, y_c) + \lambda \mathscr{L}_2(\hat{H}_c, \hat{y}_s, y_c, y_s)\end{aligned} \tag{7}$$

where $\mathscr{L}_1$, $\mathscr{L}_c$, $\mathscr{L}_h$, $\mathscr{L}_s$ are defined as in Eqs. 1, 2, 3, and 5, respectively.

3.3 Tracking

Once per-frame swimmer detections have been obtained, a tracking algorithm is required to track individual swimmers through time. These tracks are required to associate strokes with swimmers and to derive analytical metrics (e.g. average velocity). The accuracy of tracking algorithms are greatly affected by the accuracy of the detections used, and many complex tracking algorithms are designed to cope with more noisy detections. Our detections were found to be quite accurate, and hence, a simple tracking algorithm like SORT [4] is sufficient for achieving high accuracy. However, we found that SORT has a tendency to produce short, fragmented tracks, which can be improved with an additional hierarchical joining procedure. We call this tracking approach hierarchical SORT (HISORT), which will be explained in detail in Sect. 3.3.1.

Tracking can be used to improve the detection results. Once detections have been associated with a continuous identity, it is trivial to filter out points that do not belong to any swimmer and to interpolate missed detections. Tracking improves recall using interpolation to fill in missing detections and improves precision by removing erroneous detections. However, tracking can also reduce recall by removing isolated but correct detections. We found in the experiments that overall tracking is able to improve the F1 score.

Our tracking algorithm outputs a set of tracks for a video v, which we define as a set of points with associated frame numbers,

$$T_k^v = \{(f_j, p_j) | f \in [f_{\text{start}}, f_{\text{end}}], p \in \mathbb{R}^3\} \tag{8}$$

where p is a triple of $(x, y, \text{stroke\_prob})$ and f_{start} and f_{end} are the beginning and end frames of a given video.

3.3.1 Hierarchical SORT (HISORT)

As mentioned earlier, our HISORT is based on the SORT tracking algorithm. The original implementation of SORT uses bounding boxes to represent objects, which is a common property among existing object tracking algorithms. In contrast our work considers each object (swimmer's head) to be a single point. Therefore, we used Euclidean distance instead of intersection over union (IoU) when computing the distance between two objects used for association. SORT estimates object movement using a constant velocity assumption across frames. It then computes the distance between the estimated object's bounding box with all detected objects in the next frame. This is then fed into the Hungarian algorithm to find the optimal overall association among all pairs of detections between the adjacent frames. The constant velocity assumption of SORT is not an especially good fit for swimmer motion due to the high prevalence of nonlinear movement in swimming styles such as breaststroke and butterfly. Since the movement of swimmers between frames is relatively small, we opted to make a zero velocity assumption instead and found that to work well. Following similar reasoning, using a linear Kalman filter for estimation was not found to improve tracking results (and hence we do not take this approach either).

Algorithm 1: Algorithm used to finding initial set of tracks based the SORT algorithm.

Input: $p \leftarrow$ predicted points grouped by frame,
$m_d \leftarrow$ maximum distance threshold,
$m_f \leftarrow$ maximum frames without detection
Output: set of tracks

```
O ← {} // open tracks;
C ← {} // closed tracks;
foreach p_i ∈ p do
    p_i ← predicted points on frame i;
    e ← estimated next points from O;
    find a distance matrix D which holds the
      distance between every p_{i,j} and e_k;
    associate points to tracks using the Hungarian
      algorithm on D;
    add associated points to tracks in O;
    move old tracks from O to C;
    create new tracks from points in p_i that weren't
      already accounted for and add to O;
end
```

Algorithm 1 shows the algorithm we used to find the initial set of tracks based on the SORT algorithm. The algorithm uses two hyper-parameters: the maximum threshold for joining tracks to points m_d and the maximum

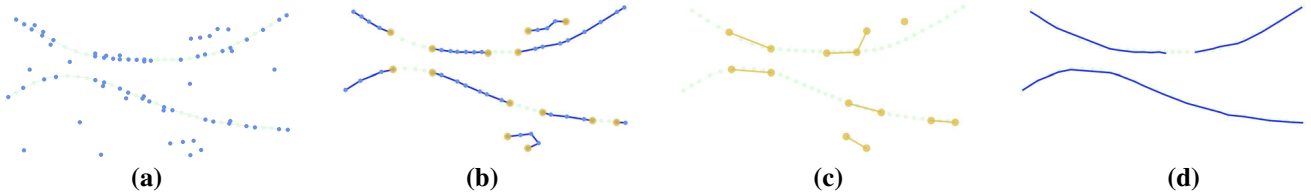


Fig. 3 Illustrative example of how HISORT works **a** The majority of our detections are quite close to the ground truth, but it is still not trivial to join them into entire tracks. The smaller blue dots are detections and the faint green lines are the ground truth. **b** We start by running Algorithm 1 to get some likely tracks (blue lines). Sometimes small tracks appear not on any swimmers. **c** We iterate by running Algorithm 1 on the beginnings and ends of likely tracks (larger yellow dots) with relaxed constraints on joining to get second-order tracks (yellow lines). **d** Once we have used the second-order tracks to join the first-order tracks from the first iteration, short tracks are removed. We can iterate multiple times to cover remaining gaps (colour figure online)

number of frames without an associated detection m_f before a track is closed. These values can be considered as measures of how much risk you are willing to take when associating a track to a point. As these values decrease, the risk of including an erroneous detection in an otherwise correct track decreases, but this also results in shorter tracks since we are taking less risk. We could not find any setting for these values which reliably created long tracks without also losing track of the swimmer. Hence, we developed the HISORT algorithm which hierarchically joins tracks to create longer tracks with each iteration.

With a single pass of SORT using conservative values for m_d and m_f, we can create many shorter, high-confidence tracks. The fastest recorded average speed for a swimmer in a race was <2.5 m/s. A swimmer may exhibit faster burst motion over short distances (between frames), so we choose the maximum distance threshold for joining points to tracks to be equivalent to 3 m/s for the first iteration.

Next we iteratively join these higher confidence tracks to create longer tracks. To do this we consider the first and last point in every track as a potential position to join another track to. Then this sub-problem is almost the same concept that we started with: we need an algorithm to associate points. Thus in the second pass, the input to SORT is a list of points on different frames and the output is a set of tracks; we call the tracks made from the original detections first-order tracks, and we select m_d and m_f such that we have high confidence in these tracks. By selecting the first and last point from each track, we can create another list of points on different frames. We then put these points into Algorithm 1 again to get another set of tracks; we call these second-order tracks. These second-order tracks are interpreted as a set of joins between our original tracks. Since SORT does not implicitly constrain second-order tracks to be valid joins, we must explicitly filter these tracks. That is, we only keep second-order tracks that are of length 2 and would join a first-order track's end to a different first-order track's beginning. We can repeat this process and for each iteration, we can relax the constraints on joining points to tracks and increase our threshold on minimum track length to get rid of erroneous tracks. Figure 3 shows an illustration of how this iteration works.

Tracking performance is quite sensitive to the choice of m_d, and thus, it is important to choose a reasonable value. We translate all detections to world space using the method described in Sect. 3.5, assuming $z = 0$. This allows us to use physical intuition to select a reasonable m_d. This also normalises distances in image space across different camera perspectives.

3.3.2 Post-processing of tracks

For practical reasons we apply the following post-processing steps after running HISORT to produce final tracks that are more useful for coaches:

1. Linearly interpolate track coordinates such that there are no missing predictions for any of the frames it spans.
2. Apply a Butterworth filter to independently smooth x and y coordinates. We used a sample frequency of 25 Hz and cut-off frequency of 4 Hz.
3. Remove tracks that do not have a large enough extent in the x-axis (2.5 m). This rule rejects swimmers from the previous race hanging on the lane ropes, and some of the people standing beside the pool during the race.

3.4 Stroke detection

We perform stroke detection after the head tracking is completed. To perform stroke detection we take the stroke probabilities in the resulting tracks and do some post-processing to identify the frame number for the start of each stroke. We take the same approach as our previous work [49], which is specified as follows:

1. Smooth the signal produced by the model by convolving a Gaussian kernel (width of 5 and σ of 1).

2. Take the middle frame of any consecutive set of frames predicted with probability over 0.5 to be a discrete prediction of a stroke.

We know that the strokes are roughly periodic for swimming, so we use some simple rule-based operations to edit (add and delete) the strokes produced using the above steps to produce more accurate predictions. The rules are as follows:

1. If the frame gap (number of frames between successive stroke detections) is less than half the median frame gap for the section, remove the later stroke.
2. If the frame gap is more than 1.55× (and less than 6×) of the median frame gap for the section, add strokes equidistant between the detected strokes.

A track may include longer periods where there are no strokes (dive in, turns, etc.), so we only apply the above rules for a section of a track with several detections not more than 3 times the median frame gap (for the track) away from each other.

3.5 Mapping from image to pool space

In order to derive useful metrics with real-world units for swimming race analysis, it is necessary to map swimmer detections from image space into pool space. Pool space is defined relative to the surface of the pool, as in Fig. 1: the x-axis runs along the length of the pool (parallel to the lane ropes), the y-axis runs along the width of the pool (perpendicular to the lane ropes), and the z-axis runs normal to the surface of the pool (perpendicular to the other axes).

In general, a homogeneous point in pool space, $\boldsymbol{x}$, is related to its image in image space, $\boldsymbol{x}'$, by the following equation:

$$\boldsymbol{x}' = KE\boldsymbol{x} \tag{9}$$

where K and E are matrices encapsulating the intrinsic and extrinsic parameters of the camera, respectively. We

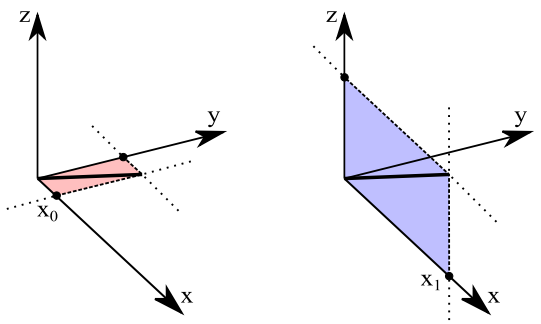


Fig. 4 Same image-space point projected onto the xy (red) and xz (blue) planes. There is a pronounced difference in x-axis displacement between the two inferred world-space coordinates x_0 and x_1 (colour figure online)

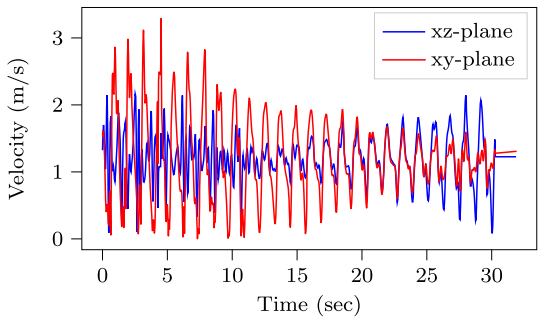


Fig. 5 Two plots of breaststroke velocity obtained using different constraints (*xy-plane*: constrained to pool surface, and *xz-plane*: constrained to lane centre). The error caused by the *xy*-plane constraint manifests as a systematic drift in velocity as the swimmer moves along the camera's field of view

calibrate the camera to obtain approximations of these matrices. In our particular implementation we assume a pinhole camera model with zero lens distortion.

Once we have computed K and E, projecting points from pool space into image space is a simple matter of applying Eq. 9. However, the inverse transformation is not so trivial since back-projecting a point from 2D space produces a ray in 3D space [58]. So in addition to back-projecting the 2D location, we will also need to find the point along the ray that corresponds to the pool-space location of the swimmer.

First, we use the pseudo-inverse of K to calculate $\hat{\boldsymbol{x}}$, the direction of the ray in camera space which passes through the camera origin and point $\boldsymbol{x}'$ on the image plane.

$$\hat{\boldsymbol{x}} = K^{+}\boldsymbol{x}' \tag{10}$$

This ray can be transformed into pool space by using the pseudo-inverse of E to transform both its origin, (0, 0, 0), and direction, $\hat{\boldsymbol{x}}$, from camera space.

$$\boldsymbol{u} = E^{+}\boldsymbol{0} \tag{11}$$

$$\boldsymbol{v} = E^{+}\hat{\boldsymbol{x}} \tag{12}$$

The location of the swimmer, $\boldsymbol{x}$, lies somewhere on this ray. That is, $\boldsymbol{x} = \boldsymbol{u} + \gamma\boldsymbol{v}$, where $\gamma \in \mathbb{R}^{+}$.

In order to solve for γ and obtain an unambiguous location in 3D space, it is necessary to impose an additional constraint. The choice of constraint is very important, as it directly influences how the swimmer is positioned along the length of the pool (Fig. 4), and hence affects key analytical metrics such as swimmer velocity.

One might initially consider constraining swimmer locations to be co-planar with the surface of the pool (effectively setting $z = 0$ for all points, which is equivalent to using a homography between the image plane and pool surface plane). While this works well for swimming styles with minimal vertical head movement, such as freestyle and backstroke, other styles like butterfly and breaststroke

Table 3 This table shows the distribution of annotated frames across the different stroke types

		Total	Train	Validation	Test
Venues		9	7	1	1
Frames	All	327,003	263,647	39,330	24,026
	Backstroke	70,355	52,534	13,616	4205
	Breaststroke	57,960	43,209	5546	9205
	Butterfly	61,595	47,518	9750	4327
	Freestyle	117,065	100,358	10,418	6289
	Medley	20,028	20,028	0	0

The train, validation and test splits each have a distinct set of venues. As we can see there is substantially more freestyle frames than other stroke types. This is because in every competition there is more freestyle events than others

suffer from inaccuracies as the swimmer's head rises and falls (see Fig. 5).

We can develop a much better constraint by observing that professional swimmers typically exhibit very little periodic side-to-side (y-axis) motion during a race. So in order to find the location of a swimmer in pool space, we select the point of intersection between the back-projected ray and the plane at $y = c$, where c is the y-position of their lane's centre. Under this assumption, the value of γ can be calculated like so:

$$c = u_y + \gamma v_y \Rightarrow \gamma = \frac{c - u_y}{v_y} \tag{13}$$

The lane-centre constraint yields accurate positions along the length of the pool for all stroke styles. Individual swimmer tracks are associated with lanes by mapping all points in the track to the pool's surface (using the $z = 0$ constraint) and associating the track with the lane in which the majority of its points are contained.

Derived Metrics To gain insight into the data obtained, we derive the swimmer velocity and stroke rate. To compute the velocity v, we perform numerical differentiation using the difference quotient formula at each time step:

$$v_i = d_{t+1} - d_t \tag{14}$$

where d_t is the swimmer's x-position at time step t. The swimmer's stroke rate is measured in "stroke cycles per minute", where each stroke cycle is completed either every two strokes for asymmetrical strokes (backstroke and freestyle), or every one stroke for symmetrical strokes (butterfly and breaststroke). This stroke rate r is computed similarly to the velocity, using the following equation:

$$r_i = s_{i+1} - s_i \tag{15}$$

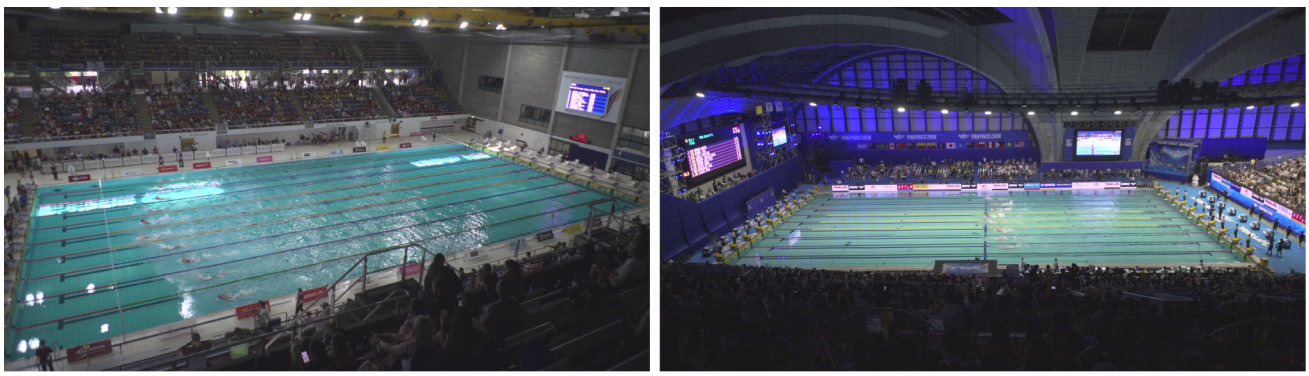
where s_i is the time at which stroke i occurred.

4 Experimental setup

We used the Adam optimiser with learning rate $\eta = 10^{-4}$ for 500k training steps with a batch size of 1 window of frames during training for all experiments. We trained models (individually; not distributed) on Nvidia Titan X and RTX 2080 ti GPUs. Depending on the hardware configuration, this took between 3 and 6 days to train each model. We used the Pytorch [34] deep learning framework to implement and train our CNN models.

4.1 Dataset

We used a dataset consisting of 249 videos from nine different venues (see Table 3), of which seven venues were selected for training. The training venues were manually selected to have best variation in environmental conditions such as time of day and indoor/outdoor pool.



(a) Example of a race with 10 swimmers. **(b)** Example of a race with 8 swimmers.

Fig. 6 Two example frames from our dataset. Notice the camera angles are quite different for the two different races in different venues. Also note the lighting variations and the variations in the glare off the water surface between the two venues

Figure 6 shows two example frames from our dataset. Notice the difference in camera angle and number of swimmers for the two races. We annotated every swimmer for every frame of the video. The videos in our dataset had a frame rate of 25 fps.

The dataset included all four swimming stroke styles (breaststroke, butterfly, freestyle, and backstroke). When stratifying by swimming style, we ensured that our train/validation/test splits were preserved. Further, we ensured that the validation and test sets each contain footage from venues not present in the training set, as we believe this to be a compelling test of generalisation to new camera angles and colour variations.

The data are 4K videos taken from a fixed camera at a relatively high position at various swimming competitions over the past few years. Due to memory constraints we first downsampled the video to 1080p versions before performing any processing.

4.2 Training

During training, examples are randomly drawn from the video recordings in our training dataset such that each annotated window of frames has an equal probability of selection.

Although our dataset contains video footage from a limited number of distinct venues, we observe that lighting conditions vary considerably across different venues. In order to help our model generalise to new venues, we perform significant uniform random colour augmentation on the original images before both phases during training. We use a contrast factor ($[-0.8, 0.8]$), global additive brightness ($\left[\frac{-40}{255}, \frac{40}{255}\right]$), additive brightness per colour channel ($\left[\frac{-20}{255}, \frac{20}{255}\right]$), random noise per pixel ($[-0.02, 0.02]$), and random horizontal flips to randomly transform each training example in an online fashion.

During training, the input is cropped using the ground truth coordinates to provide input for the refinement phase. These crops are taken such that the head location is randomly placed within the central 90% region of the crop along each axis. This process, which we refer to as "jittering", ensures that the second phase is robust to variations of head location within the crop.

The significant difference in scale of the swimmers' height in the image means that any crop size chosen will occasionally have multiple swimmers in view (for the distant lanes).

The two phases were trained simultaneously, each batch of examples representing a single training step for both phases. Training for 500K iterations takes 90 h on a GTX 1080.

4.3 Evaluation metrics

We use the $F1$ score to evaluate both head detection and stroke detection. The definitions used are as follows:

$$\text{Recall} = R = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{16}$$

$$\text{Precision} = P = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{17}$$

$$F1Score = F1 = \frac{2\text{TP}}{2\text{TP} + \text{FP} + \text{FN}} \tag{18}$$

Head detection. For head detection our target data consist of one head per lane per frame of the video. We calculate F1 on the subset of predictions which are in the same lane and same frame for each target. We consider only the distance in the x dimension since the coaches are only interested in the velocity of swimmers along the lane. The track identities are used only to ascribe each point to a lane, and a positive match is determined by any prediction being within 15cm of each ground truth. That is the TP (True Positive), FN (False Negative), and FP (False Positive) values are all calculated truth-to-prediction. We call this metric lanewise, framewise, and pointwise F1 (LFPF1).

Stroke detection. For stroke detection, we evaluate each predicted track separately, and only compare against ground truth stroke predictions that exist in the same lane and time as the predicted track. A positive match is determined by any prediction on that track being within a five-frame window of each ground truth stroke annotation. Unlike in LFPF1, the ground truth stroke annotations may account for more than one TP each, as there may be multiple predicted tracks in each lane, and we did not want to penalise the stroke detection metrics for a tracking error.

Tracking. When evaluating the tracking results we use the MOT challenge [2] MATLAB evaluation script. We report the two most prominently used metrics for tracking performance: MOTA [3] and IDF1 [41].

The MOTA metric is the most widely used tracking metric that incorporates the number of false positives, false negatives, and ID switching. It is defined as follows:

$$\text{MOTA} = 1 - \frac{\sum_t (\text{FN}_t + \text{FP}_t + \text{IDSW}_t)}{\sum_t g_t} \tag{19}$$

where IDSW measures the number of ID swaps between tracks, i.e. the number of times the identity of the closest ground truth track to a predicted point changes, summed across predicted tracks.

The IDF1 metric describes how well identities are preserved on tracks. To calculate the IDF1 metric, first each ground truth track is matched to a predicted track. Then, a positive match is a point in the ground truth track which is

within the distance threshold of the corresponding point in the matched predicted track, and the matched predicted track is the predicted track which maximises the IDF1 metric. We refer the reader to [41] for more details of this process.

5 Experimental results

We performed experiments to verify the effectiveness of DeepDASH on head/stroke detection and tracking separately. For detection, we compared our model to Faster R-CNN and performed an ablation study to investigate the relative performance impact of various aspects of our approach. For tracking, we performed a separate ablation study to investigate the impact of hierarchical operations and track post-processing. We additionally break down our head detection results for our base model stratified by stroke type and compare stroke detection results with/without extra processing rules. All results reported in this section are on the test set described in Table 3.

5.1 Head detection ablation and stratification study

In this experiment we test the accuracy of our head detection algorithm using the LFPF1 metric as defined in Sect. 4.3.

Our head detection accuracy is measured after tracking (including all extra processing rules). The tracking can remove false positive detections and also interpolate missing detections. We always know where the lane boundaries are in image space (as a result of the camera calibration), so we assign every track to the lane which the majority of its points fall into (using the 2D space transformation). Then, we only consider predicted points from tracks in the same lane as the targets. We evaluate the head detection on a frame-by-frame basis and do not consider predictions for adjacent frames.

Table 4 shows the results of our ablation study. In the ablation study, we start with the full algorithm labelled as DeepDASH. Then, for each other row of the table we remove one feature to gauge the impact of the absence of the feature on the overall performance.

The results show the model performs much worse when we remove the refinement phase, thus showing the importance of this second phase. To remove the refinement phase for our experiments, we apply soft-argmax on the window of the region proposal generation phase that would normally be used to indicate the crop position. By investigating the predictions qualitatively, we found that the model without a refinement phase exhibited a strong bias towards taking the centre point of the window as the swimmer location, leading to poor evaluation performance.

Table 4 Head detection ablation study results

	LFPF1 (%)	MOTA	IDF1
DeepDash	**93.0**	**66.9**	**74.1**
No refinement phase	51.3	− 33.7	14.6
10% Jitter	69.7	− 43.3	19.9
No NMS	89.7	57.9	59.5
First phase features	86.0	61.2	65.3

The second to fifth rows shows the results when one feature was removed from DeepDASH. The table shows both the effects of removing these features on detection metric (LFPF1) and tracking metrics (MOTA and IDF1)). "No Refinement Phase" refers to directly using the output of the region proposal phase as the head detections. "10% Jitter" refers to reducing the range of random head location jitter from 90 to 10% of the crop size when training the refinement phase. "No NMS" refers to turning off non-maximal suppression at the end of the region proposal generation phase. "1st Phase Features" refers to the refinement phase taking as input the features from the regional proposal generation phase instead of crops of the original 1080p image

Bold indicates the best result for each column

Normally, during training we deliberately choose crops such that the head appears randomly within the central 90% of the crop. Restricting that random variation to only 10% of the crop size drastically drops the model's performance. We observed that this setting causes the model to learn a strong bias to predicting in the centre of the crop at test time, regardless of where the swimmers head actually was. This validates the importance of jittering a large proportion of the crop size during training.

The results show using non-maximal suppression (NMS) at the end of the region proposal generation phase improves the results but by a smaller margin compared to the inclusion of other features. The reason that NMS is not as critical to our solution as typical object detection algorithms is that the thresholding on the heatmaps in the region proposal generation phase is quite effective at removing false detections.

Lastly, when the refinement phase uses first phase features (from the region proposal CNN) instead of cropping the original 1080p image, the results are much worse. This may be because disconnecting the two phases allows the two separate CNNs to specialise more on their individual tasks (proposal and refinement). The first phase has to distinguish a swimmer from everything that can be in the background, while the second phase only has to distinguish the swimmer from the water and the lane ropes.

Table 5 shows the results of using DeepDASH to predict swimmer head location across the different stroke styles. The results show a single model trained on all four different stroke styles works well on each separate stroke

Table 5 Results from using DeepDASH to predict swimmer head location across the different stroke styles

Stroke style	LFPF1 (%)
Backstroke	92.9
Breaststroke	89.8
Butterfly	94.0
Freestyle	96.6
Overall	93.0

A single model was trained for all four strokes styles. The results show how well that single model performs for each of the four different stroke styles separately

style. The breaststroke model exhibits the worst performance because of the large vertical movement of the swimmer's head.

5.2 Tracking algorithm results

Table 6 shows the results of the experiment that compares our hierarchical SORT (HISORT) and the existing SORT algorithm (incorporating the modifications described in Sect. 3.3.1). SORT+ and HISORT+ are the SORT and HISORT tracking algorithms with the post-processing rules from Sect. 3.3.2 applied.

The results show that adjusting the swimmer head detections using the output of tracking algorithms had a negligible impact on the head detection metric LFPF1. This implies our two-phased approach to swimmer head detections was already very accurate.

Our results show that HISORT performs similarly to SORT for the MOTA tracking metric but significantly outperforms SORT for the IDF1 metric. The reason that the MOTA results are similar is due to the accurate object detections which makes two of the three components of the MOTA formula (false positive and false negative detections) high for both tracking algorithms. IDF1, which describes how well identities are preserved on tracks, shows that HISORT is able to generate longer tracks by preserving the identities of swimmers. Furthermore, applying the extra post-processing rules described in Sect. 3.3.2 to HISORT (HISORT+) further improved the IDF1 results.

Table 6 Tracking comparison

	LFPF1 (%)	MOTA	IDF1
No tracking	92.5		
SORT	92.6	66.4	60.5
SORT+	89.6	63.2	64.3
HISORT	92.8	66.7	72.6
HISORT+	**93.0**	**66.9**	**74.1**

LFPF1 measures the accuracy of the swimmer head detection, and MOTA and IDF1 are tracking metrics used in the MOT challenge [25]. See Sect. 4.3 for more detailed description of the metrics. SORT+ and HISORT+ are the SORT and HISORT tracking algorithms with the post-processing rules from Sect. 3.3.2 applied

Bold indicates the best result for each column

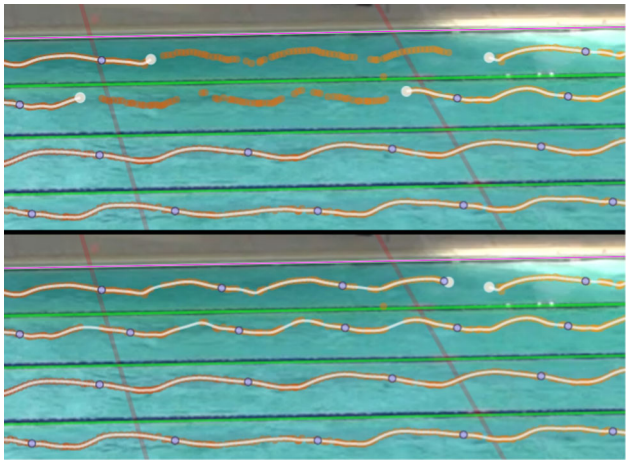


Fig. 7 This figure shows a qualitative comparison of SORT+ (above) and HISORT+ (below). HISORT+'s hierarchical tracking allows for the inclusion of short tracks, where SORT fails to do so

SORT+ had a worse LFPF1/MOTA compared to SORT. This is due to a worse head detection recall as a result of the post-processing rules removing detections.

Figure 7 shows a qualitative example of tracks produced by SORT+ versus HISORT+. As we can see from the figure, SORT+ produced shorter tracks in the two lanes where large sections of the lanes were not tracked. In contrast, HISORT+ successfully tracked almost the entire track for all lanes except for a small gap in the top lane.

5.3 Comparing our head detection algorithm to faster R-CNN

In order to compare the performance of our head detector with an off-the-shelf object detector, we used the Torchvision implementation of the popular Faster R-CNN [40] object detection model. This model has a ResNet-50 backbone [19] which was pretrained on the COCO 2017 dataset [1].

The region proposal network is trained from scratch on our swimming dataset. The rest of the network is fine-tuned on our dataset. Faster R-CNN requires bounding boxes as targets, so we translated our position targets to bounding box targets that are the same size as the crop in the second phase (48, 80) and centred on the positions. This means that all predicted bounding boxes for the Faster R-CNN are the same size. Faster R-CNN should quickly learn to only

Table 7 Comparison of head detection performance of Faster R-CNN and DeepDASH

	LFPF1 (%)	Recall (%)	Precision (%)
Faster RCNN	62.0	83.5	49.3
Faster RCNN + track	72.2	71.7	72.7
DeepDASH	92.5	**89.1**	96.2
DeepDASH + track	**93.0**	88.5	**97.9**

The results include the effect of using the HISORT+ tracking algorithm to add and delete detections

Bold indicates the best result for each column

output one bounding box size, and we can take the centre point of the predicted bounding boxes as the predicted head locations. We trained Faster R-CNN with two classes: "swimmer" and "background".

Table 7 shows the detection results of comparing DeepDASH to Faster R-CNN. The results show that DeepDASH achieves much higher LFPF1 score compared to Faster R-CNN. We believe the discrepancy in performance comes largely from Faster R-CNN being restricted to operating on a single frame at a time, whereas DeepDASH is able to incorporate motion information by operating on a window of five frames. Furthermore, the detection of small objects is a well-known weakness of all general object detection algorithms including Faster R-CNN, and therefore, the small appearance of swimmer heads (especially in far lanes) is likely a contributing factor to its poor performance. In contrast, DeepDASH with its use of the heatmap in the region proposal generation phase and the soft-argmax in the refinement phase is better able to adapt to smaller objects.

These results also show that adding and deleting the detections based on the tracking algorithm (HISORT+) significantly improved the performance of Faster R-CNN whereas it made a negligible difference to DeepDASH. This can be explained by the fact that applying tracking to Faster R-CNN predictions removed many of the false positive detections (as evidenced by the observed increase in precision). In contrast, DeepDASH already produced very high precision detections with very few false positives and consequently does not benefit much from the tracking.

Table 8 shows that our HISORT+ tracking algorithm performs significantly better when using DeepDASH's head detection algorithm rather than Faster R-CNN. This result is due to the much higher head detection accuracy of DeepDASH. (It is well known that the quality of tracking by detection is very dependent on object detection accuracy.)

Table 8 compares the inference speed of DeepDASH and Faster R-CNN. The reason that Faster R-CNN is significantly slower than DeepDASH is that Faster R-CNN

Table 8 In this table we compare the tracking performance of HISORT+ when Faster R-CNN and DeepDASH is used as the detection algorithm

	MOTA	IDF1	Frame rate
Faster R-CNN	13.8	43.1	4.8
DeepDASH	**66.9**	**74.1**	**33.1**

We also compare the inference speed of these two detection algorithms

Bold indicates the best result for each column

with a ResNet-50 backbone is significantly larger than DeepDASH, and hence requires more computational time to produce predictions. Faster R-CNN is larger because it was designed as a general-purpose object detector, and hence has a large number of weights to cater for a wide variety of object sizes, shapes, and classes.

5.4 Stroke detection

In this section we present the accuracy of DeepDASH for stroke detection. We only evaluate the accuracy of stroke detection for predicted tracks. This is because in order to detect the beginning of a stroke, we need a stroke probability signal that spans over a period of time. A stroke is considered to be correctly detected if it is predicted within a five-frame window within any ground truth label.

Table 9 shows the accuracy of DeepDASH at performing stroke detection using the F1 score as described in Sect. 4.3. The results are stratified across the different stroke styles. The results show that DeepDASH performs very accurate stroke detection for all the stroke styles. Butterfly is worst performing due to the large amount of vertical movement. The butterfly results are improved considerably when the extra rule-based operations described in Sect. 3.4 are applied (Strokes+) to remove false detections and add missing detections.

5.5 Effect of varying hyper-parameter values

In this section we show the effect on performance when varying the two important hyper-parameters σ and z. σ is used in Eq. 3 to control the size of the target 2D-Gaussian for the heatmap loss. z is the width of the truncated sine used to soften the stroke detection target (see Eq. 4).

The results for varying σ show the model is fairly robust to various sizes of the target 2D-Gaussian. Although our selected value for σ of 0.85 yields a slightly lower F1-score than for $\sigma = 0.5$, we observed that the mean and median distance between predicted and actual swimmer locations increased by a significant margin (see Table 10).

For the soft stroke target parameter z the F1 score went down dramatically with a large z value ($z = 12$), but was

Table 9 This table shows the accuracy of DeepDASH at performing stroke detection using the F1 score, where a stroke is considered correctly detected if it is detected within two frames of the ground truth (inclusive)

	Strokes (%)	Strokes+ (%)
Backstroke	97.7	97.8
Breaststroke	97.0	96.8
Butterfly	90.8	94.9
Freestyle	98.4	98.5
Overall	96.8	97.4

"Strokes+" is the result of applying the rule-based operations described in Sect. 3.4, whereas the "Strokes" results do not apply these rules

Table 10 Results of varying σ values

σ	LFPF1 (%)	Mean dist (cm)	Median dist (cm)
0.5	**93.2**	0.069	0.029
0.85	93.0	**0.036**	**0.024**
1.2	92.1	0.039	0.026

where σ controls the scale of the target 2d-Gaussian for the heatmap loss (see Eq. 3). The default σ value used in our experiments was 0.85

Bold indicates the best result for each column

Table 11 Results of varying z values

z	F1 score (%)
2	96.10
5	**97.40**
12	58.20

where z controls the spread of the soft stroke targets (see Eq. 4). The default z value used in our experiments was 5

Bold indicates the best result for each column

surprisingly robust to a narrow softening ($z = 2$) compared to our default $z = 5$. Choosing a lower z-value results in very sparse targets leading to class imbalance between positive and negative cases, whereas choosing a larger z-value results in positively labelling frames which may not have any visual features of the stroke's occurrence (see Table 11).

6 Conclusion

In this paper we presented DeepDASH, a deep learning-based automatic swimming analysis system. Tracking swimmers from a zoomed out 4K video of the whole pool was very challenging due to the small size of swimmer heads and environmental interference from splashing and submersion. We found for this situation, it was vital to design an object detector that is specifically designed to detect small objects, using multiple frames as input. Our solution can take advantage of these problem characteristics to achieve high accuracy at near real time processing speeds.

More specifically our DeepDASH approach for swimmer head detection was found to be far superior to the popular Faster R-CNN object detection algorithm in both detection accuracy and speed for this task. DeepDASH uses our HISORT tracking algorithm to output significantly longer tracks compared to the existing SORT tracking algorithm. Finally, DeepDASH is able to accurately detect strokes by achieving an overall F1 score of 97.5% for detecting strokes across all 4 stroke styles.

For future work we would like to explore using DeepDASH to detect other more challenging events such as breaths and kicks. Furthermore, adapting our solution to work for moving cameras would widen the applicability of our approach to more deployment scenarios. Another direction for future work is extending our system to other sports, which may require developing a strategy for handling multiple people in the same region proposal crop.

Compliance with ethical standards

Conflict of interest This work was funded by a competitive innovation fund from the Australian Institute of Sports. The Project is titled "A software system for automated annotation of swimming videos using deep learning". There are no other conflicts to declare.

References

1. Coco: Common obejcts in context. http://cocodataset.org/. Accessed 22 Nov 2019
2. Multiple object tracking benchmark. https://motchallenge.net/
3. Bernardin K, Stiefelhagen R (2008) Evaluating multiple object tracking performance: the CLEAR MOT metrics. EURASIP J Image Video Process 2008(1):246309. https://doi.org/10.1155/2008/246309
4. Bewley A, Ge Z, Ott L, Ramos F, Upcroft B (2016) Simple online and realtime tracking. In: 2016 IEEE international conference on image processing (ICIP), pp 3464–3468. IEEE
5. Buch S, Escorcia V, Ghanem B, Fei-Fei L, Niebles JC (2017) End-to-end, single-stream temporal action detection in untrimmed videos. In: BMVC, vol 2, p 7
6. Caba Heilbron F, Carlos Niebles J, Ghanem B (2016) Fast temporal activity proposals for efficient detection of human actions in untrimmed videos. In: Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR), pp 1914–1923
7. Chao YW, Vijayanarasimhan S, Seybold B, Ross DA, Deng J, Sukthankar R (2018) Rethinking the faster r-cnn architecture for temporal action localization. In: Proceedings of the IEEE

conference on computer vision and pattern recognition (CVPR), pp 1130–1139
8. Dai X, Singh B, Zhang G, Davis LS, Qiu Chen Y (2017) Temporal context network for activity localization in videos. In: Proceedings of the IEEE international conference on computer vision (ICCV), pp 5793–5802
9. Duan K, Bai S, Xie L, Qi H, Huang Q, Tian Q (2019) Centernet: Keypoint triplets for object detection. In: Proceedings of international conference in computer vision (ICCV)
10. Einfalt M, Zecha D, Lienhart R (2018) Activity-conditioned continuous human pose estimation for performance analysis of athletes using the example of swimming. In: 2018 IEEE winter conference on applications of computer vision (WACV), pp 446–455. IEEE
11. Everingham M, Van Gool L, Williams CKI, Winn J, Zisserman A (2012) The PASCAL visual object classes challenge (VOC2012) results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html
12. Fani H, Mirlohi A, Hosseini H, Herperst R (2018) Swim stroke analytic: front crawl pulling pose classification. In: 2018 25th IEEE international conference on image processing (ICIP), pp 4068–4072. IEEE
13. Gao J, Yang Z, Chen K, Sun C, Nevatia R (2017) Turn tap: temporal unit regression network for temporal action proposals. In: Proceedings of the IEEE international conference on computer vision (ICCV), pp 3628–3636
14. Girshick R (2015) Fast r-cnn. In: Proceedings of the IEEE international conference on computer vision (ICCV), pp 1440–1448
15. Girshick R, Donahue J, Darrell T, Malik J (2014) Rich feature hierarchies for accurate object detection and semantic segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition(CVPR), pp 580–587
16. Hakozaki K, Kato N, Tanabiki M, Furuyama J, Sato Y, Aoki Y (2018) Swimmer's stroke estimation using cnn and multilstm. J Sig Process 22(4):219–222
17. Hammad M, Pławiak P, Wang K, Acharya UR (2020) Resnet-attention model for human authentication using ECG signals. Expert Syst p e12547
18. He K, Gkioxari G, Dollár P, Girshick R (2017) Mask r-cnn. In: Proceedings of the IEEE international conference on computer vision (ICCV), pp 2961–2969
19. He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 770–778
20. Huang Y, Dai Q, Lu Y (2019) Decoupling localization and classification in single shot temporal action detection. In: IEEE international conference on multimedia and expo (ICME)
21. Ioffe S, Szegedy C (2015) Batch normalization: accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167
22. Kuhn HW (1955) The hungarian method for the assignment problem. Nav Res Log Quart 2(1–2):83–97
23. Law H, Deng J (2018) Cornernet: detecting objects as paired keypoints. In: Proceedings of the European conference on computer vision (ECCV), pp 734–750
24. Leal-Taixé L, Fenzi M, Kuznetsova A, Rosenhahn B, Savarese S (2014) Learning an image-based motion context for multiple people tracking. In: Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR), pp 3542–3549
25. Leal-Taixé L, Milan A, Reid I, Roth S, Schindler K (2015) Motchallenge 2015: Towards a benchmark for multi-target tracking. arXiv preprint arXiv:1504.01942
26. Leal-Taixé L, Pons-Moll G, Rosenhahn B (2011) Everybody needs somebody: modeling social and grouping behavior on a linear programming multiple people tracker. In: 2011 IEEE international conference on computer vision workshops (ICCV workshops), pp 120–127. IEEE
27. Lin T, Zhao X, Shou Z (2017) Single shot temporal action detection. In: Proceedings of the 25th ACM international conference on multimedia, pp 988–996. ACM
28. Lin T, Zhao X, Su H, Wang C, Yang M (2018) BSN: boundary sensitive network for temporal action proposal generation. In: Proceedings of the European conference on computer vision (ECCV), pp 3–19
29. Lin TY, Dollár P, Girshick R, He K, Hariharan B, Belongie S (2017) Feature pyramid networks for object detection. In: Proceedings of the IEEE conference on computer vision and pattern recognition(CVPR), pp 2117–2125
30. Lin TY, Goyal P, Girshick R, He K, Dollár P (2017) Focal loss for dense object detection. In: Proceedings of the IEEE international conference on computer vision (ICCV), pp 2980–2988
31. Liu W, Anguelov D, Erhan D, Szegedy C, Reed S, Fu CY, Berg AC (2016) SSD: Single shot multibox detector. In: European conference on computer vision, pp 21–37. Springer
32. Nibali A, He Z, Morgan S, Greenwood D (2017) Extraction and classification of diving clips from continuous video footage. In: Proceedings of the IEEE conference on computer vision and pattern recognition workshops, pp 38–48
33. Nibali A, He Z, Morgan S, Prendergast L (2018) Numerical coordinate regression with convolutional neural networks. arXiv preprint arXiv:1801.07372
34. Paszke A, Gross S, Chintala S, Chanan G, Yang E, DeVito Z, Lin Z, Desmaison A, Antiga L, Lerer A (2017) Automatic differentiation in pytorch
35. Pirsiavash H, Ramanan D, Fowlkes CC (2011) Globally-optimal greedy algorithms for tracking a variable number of objects. In: Proceedings of the IEEE international conference on computer vision (ICCV), pp 1201–1208. IEEE
36. Pławiak P, Abdar M, Acharya UR (2019) Application of new deep genetic cascade ensemble of svm classifiers to predict the australian credit scoring. Appl Soft Comput 84:105740
37. Pławiak P, Abdar M, Pławiak J, Makarenkov V, Acharya UR (2020) Dghnl: a new deep genetic hierarchical network of learners for prediction of credit scoring. Inf Sci 516:401–418
38. Redmon J, Divvala S, Girshick R, Farhadi A (2016) You only look once: unified, real-time object detection. In: Proceedings of the IEEE conference on computer vision and pattern recognition(CVPR), pp 779–788
39. Redmon J, Farhadi A (2017) Yolo9000: better, faster, stronger. In: Proceedings of the IEEE conference on computer vision and pattern recognition(CVPR), pp 7263–7271
40. Ren S, He K, Girshick R, Sun J (2015) Faster r-cnn: towards real-time object detection with region proposal networks. In: Advances in neural information processing systems, pp 91–99
41. Ristani E, Solera F, Zou R, Cucchiara R, Tomasi C (2016) Performance measures and a data set for multi-target, multi-camera tracking. In: Computer vision—ECCV 2016 workshops, pp 17–35. Springer International Publishing, Cham
42. Sadeghian A, Alahi A, Savarese S (2017) Tracking the untrackable: Learning to track multiple cues with long-term dependencies. In: Proceedings of the IEEE international conference on computer vision (CVPR), pp 300–311
43. Shou Z, Chan J, Zareian A, Miyazawa K, Chang SF (2017) Cdc: Convolutional-de-convolutional networks for precise temporal action localization in untrimmed videos. In: Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR), pp 5734–5743
44. Shou Z, Wang D, Chang SF (2016) Temporal action localization in untrimmed videos via multi-stage cnns. In: Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR), pp 1049–1058

45. Tian Z, Shen C, Chen H, He T (2019) FCOS: fully convolutional one-stage object detection. In: Proceedings of international conference in computer vision (ICCV)
46. Tompson JJ, Jain A, LeCun Y, Bregler C (2014) Joint training of a convolutional network and a graphical model for human pose estimation. In: Advances in neural information processing systems, pp 1799–1807
47. Tsumita T, Shishido H, Kitahara I, Kameda Y (2019) Swimmer position estimation by lane rectification. In: International workshop on advanced image technology (IWAIT) 2019, vol 11049, p 110490E. International Society for Optics and Photonics
48. Tuncer T, Ertam F, Dogan S, Aydemir E, Pławiak P (2020) Ensemble residual network-based gender and activity recognition method with signals. J Supercomput 76(3):2119–2138
49. Victor B, He Z, Morgan S, Miniutti D (2017) Continuous video to simple signals for swimming stroke detection with convolutional neural networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition workshops, pp 66–75
50. Wang M, Liu Y, Huang Z (2017) Large margin object tracking with circulant feature maps. In: Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR), pp 4021–4029
51. Wojke N, Bewley A, Paulus D (2017) Simple online and realtime tracking with a deep association metric. In: 2017 IEEE international conference on image processing (ICIP), pp 3645–3649. IEEE
52. Wu Y, He K (2018) Group normalization. In: Proceedings of the European conference on computer vision (ECCV), pp 3–19
53. Xu H, Das A, Saenko K (2017) R-c3d: region convolutional 3d network for temporal activity detection. In: Proceedings of the IEEE international conference on computer vision (ICCV), pp 5783–5792
54. Zecha D, Einfalt M, Lienhart R (2019) Refining joint locations for human pose tracking in sports videos. In: Proceedings of the IEEE conference on computer vision and pattern recognition workshops, pp 0–0
55. Zhang L, Li Y, Nevatia R (2008) Global data association for multi-object tracking using network flows. In: 2008 IEEE conference on computer vision and pattern recognition (CVPR), pp 1–8. IEEE
56. Zhao Y, Xiong Y, Wang L, Wu Z, Tang X, Lin D (2017) Temporal action detection with structured segment networks. In: Proceedings of the IEEE international conference on computer vision (ICCV), pp 2914–2923
57. Zhu J, Yang H, Liu N, Kim M, Zhang W, Yang MH (2018) Online multi-object tracking with dual matching attention networks. In: Proceedings of the European conference on computer vision (ECCV), pp 366–382
58. Zisserman RHA (2004) Multiple view geometry in computer vision. Cambridge University Press, Cambridge