UNIVERSITY OF OULU
FACULTY OF SCIENCE

Pirkko Mustamo

Object detection in sports: TensorFlow Object Detection API case study

Bachelor's Thesis
Degree Programme in Mathematical Sciences
January 2018

Mustamo P. (2018) Object detection in sports: TensorFlow Object Detection API case study. University of Oulu, Degree Programme in Mathematical Sciences. Bachelor's Thesis, 43 p.

ABSTRACT

Object detection is widely used in the world of sports, its users including training staff, broadcasters and sports fans. Neural network based classifiers are used together with other object detection techniques. The aim of this study was to explore the modern open source based solutions for object detection in sports, in this case for detecting football players. TensorFlow Object Detection API, an open source framework for object detection related tasks, was used for training and testing an SSD (Single-Shot Multibox Detector) with Mobilenet- model. The model was tested as a) pre-trained and b) with fine-tuning with a dataset consisting of images extracted from video footage of two football matches. Following hypotheses were examined:

1) Pre-trained model will not work on the data without fine-tuning.
2) Fine-tuned model will work reasonably well on the given data.
3) Fine-tuned model will have problems with occlusion and players pictured against the rear wall.
4) Using more variable training data will improve results on new images.

The results of this study indicate that:

1) The pre-trained model was useless for detecting players in the test images.
2) A fine-tuned model worked reasonably well.
3) Problem areas were players in clusters and/or pictured against the rear wall.
4) A model trained with data from one game was able to detect players in footage from another game. The overall model performance did not much improve by training the model with data from two games.

Other model types (such as Faster R-CNN model) should be tested on the data.

Keywords: machine learning, object detection, sports, TensorFlow Object Detection API, SSD model

Mustamo P. (2018) Kohteentunnistus urheilussa TensorFlow Object Detection API:n avulla. Oulun yliopisto, matemaattisten tieteiden tutkinto-ohjelma. Kandidaatintyö, 43 s.

TIIVISTELMÄ

Kohteentunnistusta käytetään yleisesti urheilumaailmassa, mm. valmennuksessa, televisiolähetyksissä sekä fanikäytössä. Neuroverkkoihin perustuvia menetelmiä käytetään yhdessä muiden tekniikoiden kanssa. Tämän tutkimuksen päämäärä oli tarkastella moderneja avoimen lähdekoodin ratkaisuja kohteentunnistukseen urheilussa, tässä tapauksessa jalkapalloilijoiden tunnistuksessa. TensorFlow Object Detection API perustuu avoimeen lähdekoodiin ja tarjoaa työkaluja kohteentunnistukseen. Sen avulla opetettiin ja testattiin SSD (Single-Shot Multibox Detector) with Mobilenet- mallia sekä a) valmiiksi treenattuna että b) hienosäädettynä aineistolla, joka koostui kahdesta jalkapallo-otteluvideosta poimituista kuvista.
Työssä tarkasteltiin seuraavia hypoteeseja:

1) Valmiiksi opettettu malli ei toimi ilman hienosäätöä omalle aineistolle.
2) Hienosäädetty malli toimii kohtuullisen hyvin omalle aineistolle.
3) Hienosäädetyllä mallilla on ongelmia toisensa peittävien tai takaseinää vasten kuvattujen pelaajien tunnistamisessa.
4) Mallin opettaminen vaihtelevammalla aineistolla parantaa tuloksia uudenlaisia esineitä tunnistetaesa.

Tutkimuksen tulosten perusteella:

1) Valmiiksi opetettu malli oli hyödytön tämän datan käsittelyssä.
2) Hienosäädetty malli toimi kohtalaisen hyvin.
3) Hienosäädetyllä mallilla oli ongelmia toisensa peittävien tai takaseinää vasten kuvattujen pelaajien tunnistamisessa.
4) Yhdestä pelistä saadulla aineistolla opetettu malli tunnisti pelaajat toisesta pelistä kohtalaisen hyvin. Mallin toiminta ei juurikaan parantunut kun se opetettiin molemmista peleistä koostetulla aineistolla.

Muita mallityyppejä (kuten Faster R-CNN model) pitäisi testata tällä datalla.

Avainsanat: koneoppiminen, kohteentunnistus, urheilu, TensorFlow Object Detection API, SSD malli

TABLE OF CONTENTS

FOREWORD

Foreword

I would like to thank Anri Kivimäki, Sami Huttunen and Jari Päkkilä for supervising this thesis. Furthermore, I would like to thank Anri Kivimäki and Sami Huttunen for giving me the opportunity to work on this interesting topic and for introducing me to the world of object detection in sports. The great tutorial by Dat Tran [1] was used as a guide for fine-tuning the model for the football game data – I am grateful for its existence. My family has been most understanding and supportive during the last three years of studying for this BSc degree – many thanks to all of you!

I would like to dedicate this work to Sanna, with love.

Oulu, 24.01.2018

Pirkko Mustamo

ABBREVIATIONS

mAP	mean average precision
SSD	Single Shot Multibox Detector

1. INTRODUCTION

Object detection is a general term for computer vision techniques for locating and labeling objects. Object detection techniques can be applied both to static images or moving images. Computer vision techniques are already in wide use in sports today, as they can offer valuable insight about movement of individual athletes or whole teams to training staff, help in tracking ball location and help visualizing game progression during broadcasts [1].

Open source libraries such as OpenCV's DNN library and TensorFlow Object Detection API offer easy-to-use, open source frameworks where pre-trained models for object detection (such as ones downloadable from the TensorFlow model zoo) reach high accuracy in detecting various object from humans to tv monitors [2]. However, the video footage used in this study was shot with cameras located under the roof of the stadium so the view was from a high angle, the players in the images often very small or partially occluded by other players. This made it important to train the models to work with the specific data. For this study, TensorFlow Object Detection API was chosen due to its popularity in object detection and for its easy-to-approach nature.

In this study, TensorFlow Object Detection API was tested for detection of football players. The model used within the API was a SSD model with Mobilenet. The pre-trained model was trained and tested with our own data which consisted of images extracted from video footage of two football matches. Following hypotheses were examined:

1) Pre-trained model will not work on the data without fine-tuning.
2) Fine-tuned model will work reasonably well on the given data.
3) Fine-tuned model will have problems with occlusion and players pictured against the rear wall.
4) Using more variable training data will improve results on new images.

2. LITERATURE REVIEW

2.1. Modern methods in object detection

Object detection is a common term for computer vision techniques classifying and locating objects in an image. Modern object detection is largely based on use of convolutional neural networks [2]. Some of the most relevant system types today are Faster R-CNN, R-FCN, Multibox Single Shot Detector (SSD) and YOLO (You Only Look Once) [2].

Original **R-CNN** method worked by running a neural net classifier on samples cropped from images using externally computed box proposals (=samples cropped with externally computed box proposals; feature extraction done on all the cropped samples). This approach was computationally expensive due to many crops; **Fast R-CNN** reduced the computation by doing the feature extraction only once to the whole image and using cropping on the lower layer (=feature extraction only once on the whole image; samples cropped with externally computed box proposals). **Faster R-CNN** goes a step further and used the extracted features to create class-agnostic box proposals (=feature extraction only once on the whole image; no externally computed box proposals). **R-FCN** is like Faster R-CNN but the feature cropping is done in a different layer for increased efficiency [2].

YOLO (You Only Look Once) [3] works on different principle than the aforementioned models: it runs a single convolutional network on the whole input image (once) to predict bounding boxes with confidence scores for each class simultaneously. The advantage of the simplicity of the approach is that the YOLO model is fast (compared to Faster R-CNN and SSD) and it learns a general representation of the objects. This increases localization error rate (also, YOLO does poorly with images with new aspect ratios or small object flocked together) but reduces false positive rate. [3]

Single Shot Multibox Detector (SSD) [4] differs from the R-CNN based approaches by not requiring a second stage per-proposal classification operation [2]. This makes it fast enough for real-time detection applications. However, this comes with a price of reduced precision [2]. Note that "**SSD with MobileNet**" refers to a model where model meta architecture is SSD and the feature extractor type is MobileNet.

Speed-accuracy tradeoff

Many modern object detection applications require real-time speed. Methods such as YOLO or SSD work that fast, but this tends to come with a decrease in accuracy of predictions, whereas models such as Faster R-CNN achieve high accuracy but are more expensive to run. This is illustrated in Fig. 1. The cost in model speed depends on the application: With larger images (e.g. 600x600) SSD works comparable to more computationally expensive models such Faster R-CNN, even as on smaller images its performance is considerably lower [2].

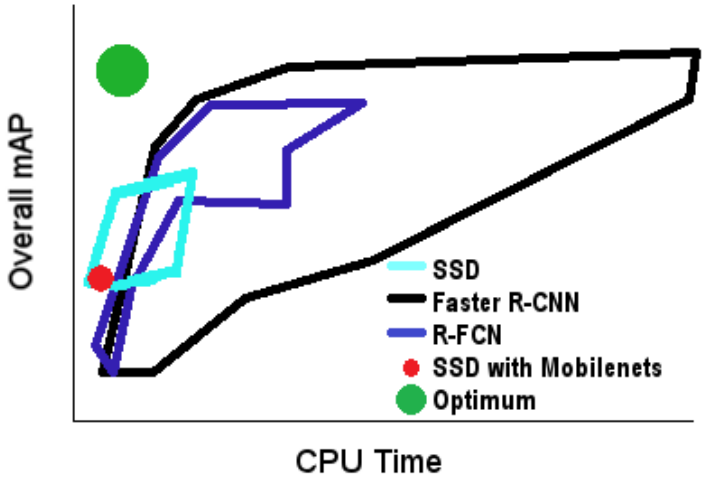


Figure 1. Overall mean average precision (mAP) vs CPU Time for some commonly used model types (based on [2], Figure 2), including the model type used in this study (SSD with Mobilenets) tested on low resolution images (red marker). Note that mAP metric is discussed in chapter 3.2.4.

2.2. Object detection in sports

Computer vision techniques are widely used in sports today. Applications include use during training (trainers and coaches can e.g. analyze movement of individual players), during matches (e.g. helping referees in tracking players and balls, visualization in broadcasting) and for audiences [5]. Commercial player detection systems can be completely manual (a person is marking the locations of e.g. players in images) or automated [5]. However, there are no commercial, fully-automated systems capable of reliably finding and labeling players who often obstruct view to each other or look similar [5]. Ball tracking systems are used e.g. tennis, baseball and football with good accuracy [5].

Use of trained classifiers has been widely used in pedestrian detection, with good results. In sports this technique has been suffering from problems such as high false positive rates (due to factors such as highly varying poses of the athletes) and high computational intensity [5, 6], and it has been shown that the classifier should be taught using scene specific data [5]. Detection of the ball has been considered difficult as the ball is small and tends to move fast [5].

In sports such as football where there is a relatively uniformly colored field and possibility for static camera usage, image differencing and background subtraction (comparing image to pre-build background model of the location) or a color-based elimination of the ground can be used [5]. However, these methods are prone both to false positives and missed detections, especially when only one camera viewpoint is available [5, 6].

The object detection systems applied to moving image are often combinations of techniques. For instance, ball detection can combine finding possible candidates for a ball and knowledge of physics of ball movement to predict the most likely trajectory of the ball [5]. One suggested method to overcome some of the aforementioned

problems of using background-subtracted foreground masks or trained classifiers is combining the two approaches [6]. Foreground masks with high detection rates could be used to produce probable candidates for objects, which could be then further confirmed or rejected using appearance-base classification. The appearance-based classifiers could be adapted to the specific scene by continuously training them with the most likely positive and negative examples provided by the foreground mask phase.

3. METHODS

3.1. Data and preprocessing

Raw data consisted of frames extracted from video footage of football games [7]. The view was from the side and middle of the field (camera view 1). To explore the effect of game specific factors such as team shirt colors, footage from two separate games (Tromsø IL – Anzhi: referred here as game 1; Tromsø IL - Strømsgodset: referred here as game 2) was included. The training data sets consisted of frames from the first half of the games, and the evaluation and test sets frames from the second half to ensure the independence of the test data from the training data. To increase variability in player positions and poses, one frame per 9 sec of video was extracted with ffmpeg as .jpg (constant rate factor = 2, final size of the images 1280x960 pixels). The final data consisted of 200 training data images, 50 evaluation data images and 50 test data images from the game 1 and 100 training data images, 50 evaluation data images and 50 test data images from the game 2 (total 500 frames). Each training image had varying number of tagged objects (players), as some of the 22 players in the field moved in and out of the camera view during the games.

Table 1 Frames extracted from the football videos for training, evaluating (eval.) and testing the player detection models.

Game	1 (Tromsø IL – Anzhi)	2 (Tromsø IL - Strømsgodset)
Training	**200**	**100**
Evaluation	**50**	**50**
Test	**50**	**50**
Testing pre-trained model	50 (unlabeled)	50 (unlabeled)
Fine-tuning the pre-trained model Model A	200 (labeled)	-
Fine-tuning the pre-trained model Model B	100* (labeled)	100 (labeled)
Evaluation during the fine-tuning process Model A, Model B	50 (labeled)	50 (labeled)
Testing with evaluation protocol Model A, Model B	25** (labeled)	25** (labeled)
Manual review of the fine-tuned model Model A, Model B	50 (unlabeled)	50 (unlabeled)

* first 100 of 200 labeled training images ** last 25 of the test data were manually labeled

The training data base was generated by manually tagging the objects (players) in the images using LabelImg [8] (Fig. 1). Referees within the line markings were tagged as players for simplicity but any non-player persons standing outside the line markings were not tacked. Players occluding each other, being shown against the rear wall or being partially out of the camera view were tagged as “difficult”. LabelImg saves the annonations as xml-files in PASCAL VOC format, so a ready-made script [9] was available for creating TFRecords (Tensor Flow record format). The 1280x960 sized images (and the corresponding annotation files) were later resized to 640x480 (25%) to increase model training efficiency.

Figure 2. Typical training data image being labeled using labelImg.

3.2. Object detection

3.2.1. TensorFlow Object Detection API with SSD model with Mobilenet

TensorFlow Object Detection API is “an open source framework built on top of TensorFlow” that aims to make it easy to “construct, train and deploy object detection models” [10]. To achieve this, TensorFlow Object Detection API provides the user with multiple pre-trained object detection models [2] with instructions and example codes for fine-tuning and using the models for object detection tasks.

TensorFlow Object Detection API can be used with different pre-trained models. In this work, a SSD model with Mobilenet (ssd_mobilenet_v1_coco) was chosen. The model had been trained using MSCOCO Dataset which consists of 2.5 million labeled instances in 328 000 images, containing 91 object types such as “person”, “cat” or “dog”[11] (Fig. 3). The ssd_mobilenet_v1_coco-model is reported to have mean Average Precision (mAP) of 21 on COCO dataset[12].

In this work, the SSD model was tested both as pre-trained without fine tuning and as fine-tuned for our own dataset.[1]

Figure 3. The pre-trained SSD model was able to recognize the cat (98%) and the potted plant (57%) but was less sure about the identity of the dog (86% dog, 75% cat). photo: S.Kallioinen

3.2.2. Object detection with pre-trained model

The pre-trained SSD model (ssd_mobilenet_v1_coco) was tested with 50 frames extracted from the game 1 footage. The resulting tacked images were visually checked.

[1] Object detection was done using a code modified from a code by [16], which in turn was modified from [17].

3.2.3. Object detection with customized model

The pre-trained SSD model (ssd_mobilenet_v1_coco) was fine-tuned for our dataset using manually labeled images a) from game 1 (model A) and b) from game 1 and game 2 (model B). A provided configuration file (ssd_mobilenet_v1_coco.config) was used as a basis for the model configuration (after testing different configuration settings, the default values for configuration parameters were used). The provided checkpoint file for ssd_mobilenet_v1_coco was used as a starting point for the fine-tuning process. The training was stopped after 17400 timesteps when the mean average precision (mAP) somewhat leveled out (Fig. 4, Fig. 5). The mAP value increased up to 8000 timesteps. However, the mAP values kept fluctuating even after that, which raised suspicion that even longer training might improve the detection results. Training the model in cPouta server took over 20 hours with CPU or, alternatively, about 2 hours with GPU for each model. The total loss value was reduced rapidly for both models A and B due to starting from the pre-trained checkpoint file (Fig. 6).

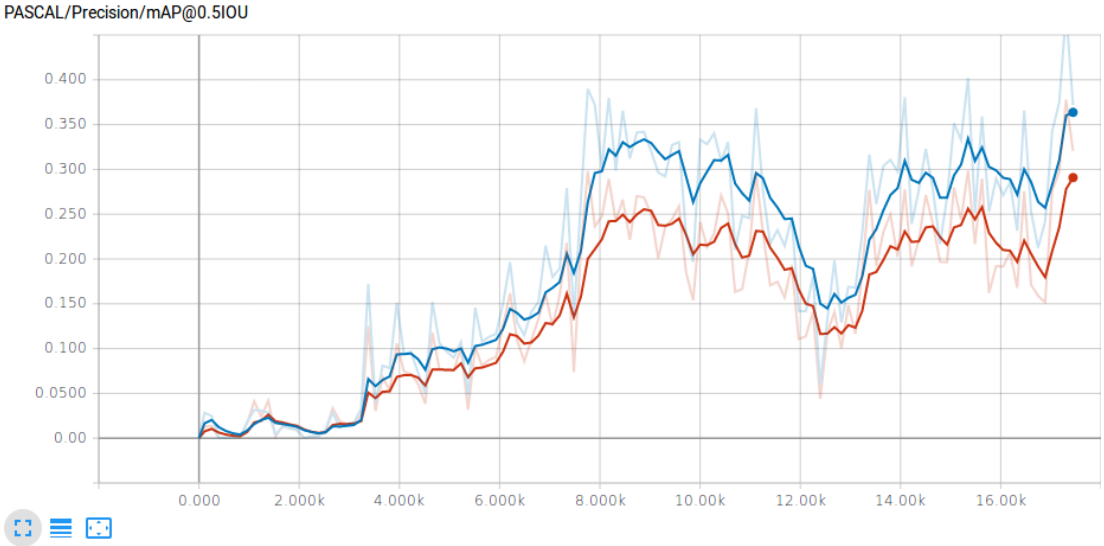


Figure 4. Development of mAP when training model A (blue = with evaluation data from game 1; red = with evaluation data from game 2, smoothing = 0.7).

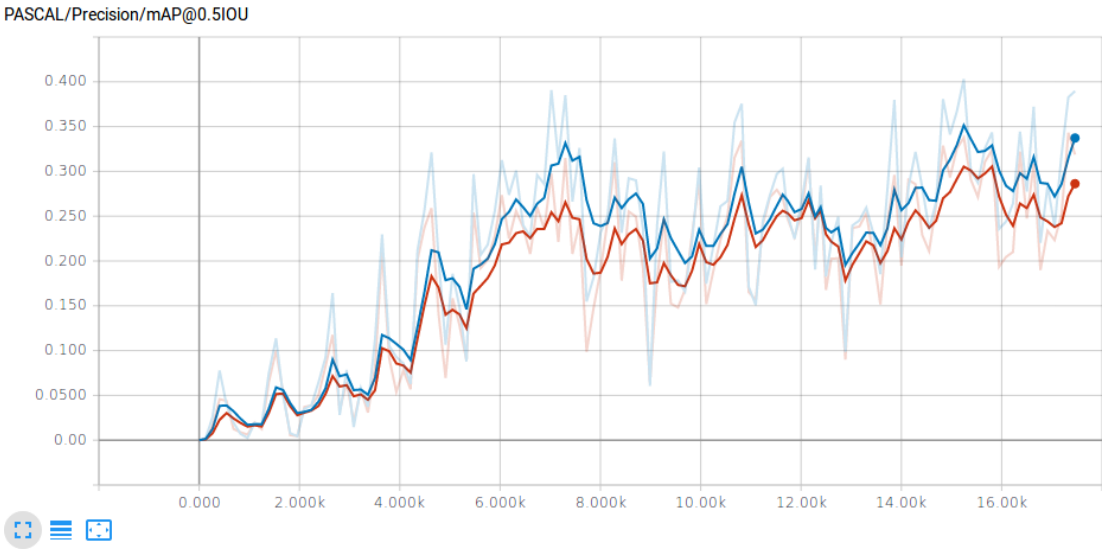


Figure 5. Development of mAP when training model B (blue = with evaluation data from game 1; red = with evaluation data from game 2, smoothing = 0.7).

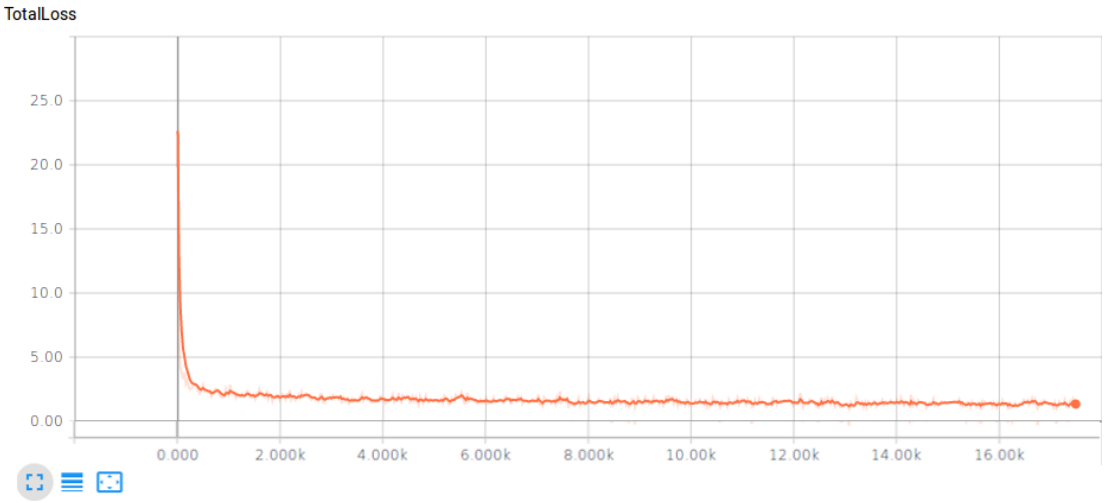


Figure 6. Development of loss when training model A.

The fine-tuned model A was fed to TensorFlow Object Detection API and tested on test data from game 1, and in addition on test data from game 2 to see how game specific factors affect the result. The model B was then tested on data from game 1 and 2 to see if the game specific fine-tuning improved the detection results.

A concern rose from the analysis of the results that the fine-tuned models had not reached the optimal number of training steps at 17400 timesteps (for instance, the fluctuation in mAP level between timeteps (Fig. 4 and 5) could indicate this). Therefore, the models were further trained up to 200 000 timesteps. The mAP values for both models showed an increasing trend until about 100 000 timesteps and leveled out after that (Fig.7 and 8). Note that the fluctuation in these images is less pronounced because of the scale of the x-axis and because the mAP was evaluated approximately every 1000 timesteps as opposed to every 100 timesteps as in Fig. 4 and 5.

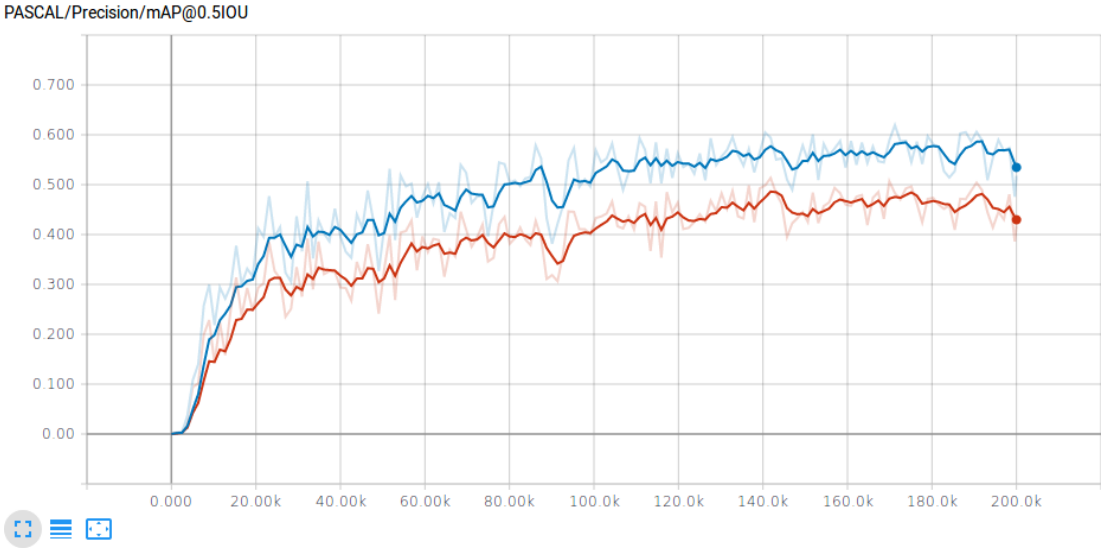


Figure 7. Development of mAP when training model A (blue = with evaluation data from game 1; red = with evaluation data from game 2, smoothing = 0.7).

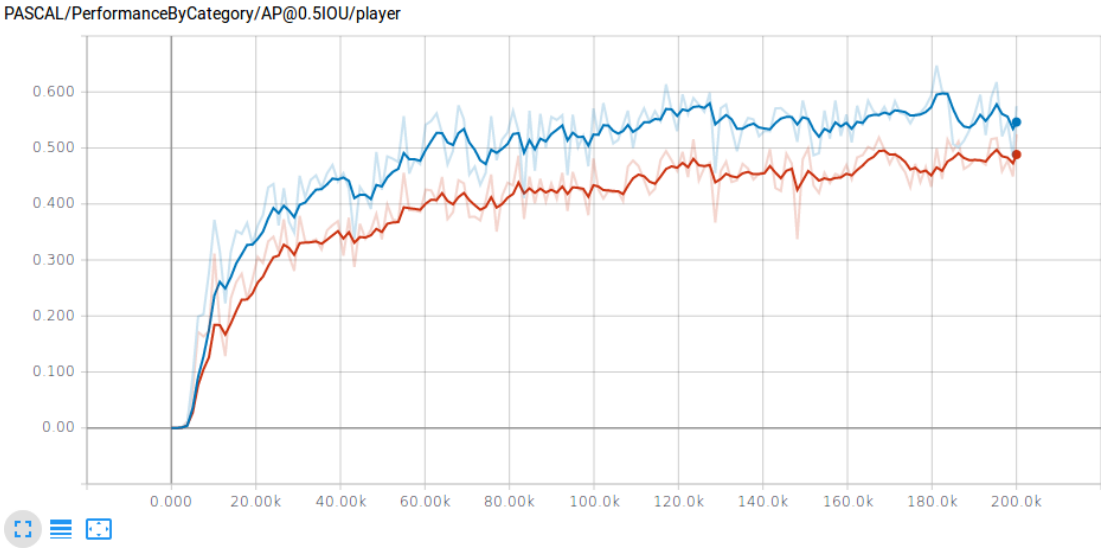


Figure 8. Development of mAP when training model B (blue = with evaluation data from game 1; red = with evaluation data from game 2, smoothing = 0.7).

3.2.4. Evaluation metrics

The most relevant evaluation metrics for this application are precision, recall, F1-score and mAP.

Precision describes how relevant the detection results are (Eq. 1):

$$precision = \frac{TP}{TP+FP} \quad (1)$$

where TP = true positive, FP = false positive

Recall describes the percentage of relevant objects that are detected with the detector (Eq. 2):

$$recall = \frac{TP}{TP+FN} \qquad (2)$$

where FN = false negative.

Generally, when precision increases, recall decreases and vice versa: a very sensitive model is able to catch large percentage of objects in an image, but it also generates high number of false positives, where as a model with high threshold for detection only produces few false positives but it also leaves a higher percentage of objects undetected. The right balance between these two depends on the application. **F1-score** is used as a single metric for these two viewpoints (Eq. 3).

$$F1 - score = 2 * \frac{precision*recall}{precision+recall} \quad (3)$$

Definition of **true positive**, **false positive** and **false negative** can change from one object detection application to other. The TensorFlow Object Detection API uses "PASCAL VOC 2007 metrics" [13] where a predicted instance is defined as a TP when Intersection over Union (IoU) is over 50% [14] (Eq.4), i.e.

$$IoU = \frac{area(true\ bounding\ box \cap predicted\ bounding\ box)}{area(true\ bounding\ box \cup predicted\ bounding\ box)} > 0.5 \quad (4)$$

One object can have only one bounding box associated with it, so if several bounding boxes are predicted for an object, only one is considered TP and the others FP. An object without any predicted bounding boxes associated with it is considered a FN. Evaluation protocol ignores objects tagged as "difficult" (here players occluding each other, being shown against the rear wall or being partially out of the camera view). [14]

Mean Average Precision (mAP) is a commonly used metrics for comparing model performance in object detection. MAP summarizes the information in "precision-recall" curve to one number. Precision-recall curve is formed by sorting all the predicted bounding boxes (over all images in the evaluation set) assigned to a object class by their confidence rating, and then for each prediction, a recall value and a precision value are calculated. Recall is defined as proportion of TPs above the given rank out of all user tagged objects. Precision is defined as proportion of TPs in predictions above the given rank. From these precision-recall-pairs a precision-recall-curve is formed. Average Precision (AP) is calculated as the "area under the precision-recall-curve", i.e. it approximates precision averaged across all values of recall between 0 and 1 (see [15] for a good explanation of this) by summing precision values multiplied by the corresponding change in recall from one point in curve to the next. However, what VOC metrics call "AP" is actually *interpolated* average precision, which is defined "as the mean precision at a set of eleven equally spaced recall levels",

the precision value corresponding to each recall interval being the maximum precision value observed in the curve within the interval [14]. MAP of a model is then the mean of AP values of all object classes. The interpolated AP is, of course, higher than the basic AP; this must be taken into consideration when comparing mAP values.

Manual review of the detection results

Detection results were reviewed manually by going through each image and counting the true positives, false negatives and false positives, and counting also the true positives and false negatives by uniform color. For model A, the true positives, false negatives and false positives were counted for players in clusters (defined as some part of their bodies touching) and pictured against the rear wall (at least their head). As the aim of this review was to roughly estimate if a player was detected at all, a very liberal definition of TP was used: detection was classified as TP if at least 50% of the area of the player was estimated to be within the predicted bounding box. Only one box could be associated with one player and vice versa: if there were more than one player within a box, the box went to the one with highest proportion of his body within the box. If the proportions of players within the box were equal, the box went to the front-most player. If there were more than one box associated with a player, one of them was counted as TP and others as FP.

4. RESULTS AND DISCUSSION

4.1. Pre-trained model

Pre-trained model worked poorly, recognizing only few players as "persons" and additionally, reporting some players with wrong labels such as "birds" (Fig. 9). It was obvious without further analysis that the pre-trained model was useless for any application using this type of images. The pre-trained model might have been more successful with larger, better-quality images.

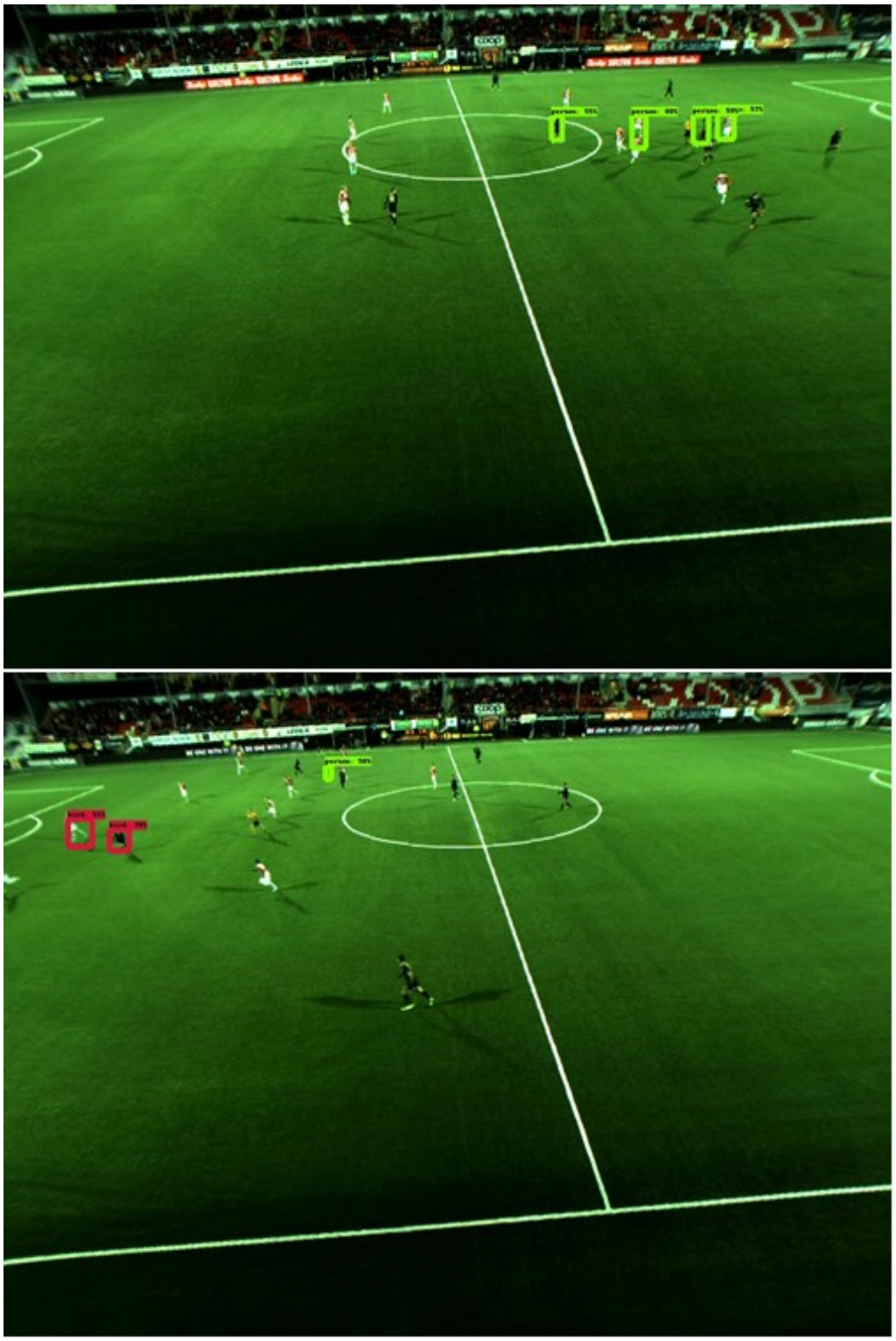


Figure 9. Only a few of the players were detected with the pre-trained model. Also, a few birds were falsely detected.

4.2. Fine-tuned models

After 17400 timesteps, the estimation protocol offered by the TensorFlow Object Detection API estimated the mAPs for model A as approximately 0.3 and 0.45 for the test sets from game 1 and game 2, respectively (however the mAP value was fluctuating between timesteps so the exact value was hard to determine (Fig. 4 and 5)). For model B, the respective mAP values were approximately 0.3 and 0.4. The results from the manual review of the detection performance of the models after 17400 timesteps are presented in Tables 2-4 and Fig. 10. After 200 000 timesteps, the estimation protocol offered by the TensorFlow Object Detection API estimated the mAPs for both model A and model B as approximately 0.45 and 0.6 for the test sets from game 1 and game 2, respectively, which was a considerable improvement from the values at 17400 timesteps. Based on the mAP graphs (Fig. 7 and 8), it appears that the optimal would be at approximately 100 000 timesteps – the mAP leveled out after this. When the progress in bounding box locations over training time was checked with a few of the test set images (Appendix 1), in some cases the models also seemed to produce more false positives after this point, which might indicate some degree of overtraining. However, this hypothesis was not supported by the mAP graphs or tested with any actual analysis; this behavior may be just normal fluctuation in model performance between timesteps during training.

Table 2. Overall precision and recall from manual review of the detection results.

Model	Model A		Model B	
Testset	Game 1	Game 2	Game 1	Game 2
Players total	911	769	911	775
Precision	0.858	0.831	0.806	0.855
Recall	0.638	0.650	0.592	0.646
F1-score	0.732	0.729	0.682	0.736

Table 3. Recall values by the uniform color from manual review of the detection results.

Model Testset	Model A Game 1	Game 2	Model B Game 1	Game 2
Players in white-red uniforms	417	326	419	332
Recall for players in white-red uniforms	0.614	0.660	0.568	0.602
Players in black uniforms	432	-	430	-
Recall for players in black uniforms (count of players)	0.644	-	0.600	-
Players in blue uniforms	-	371	-	371
Recall for players in blue uniforms (count of players)	-	0.639	-	0.687

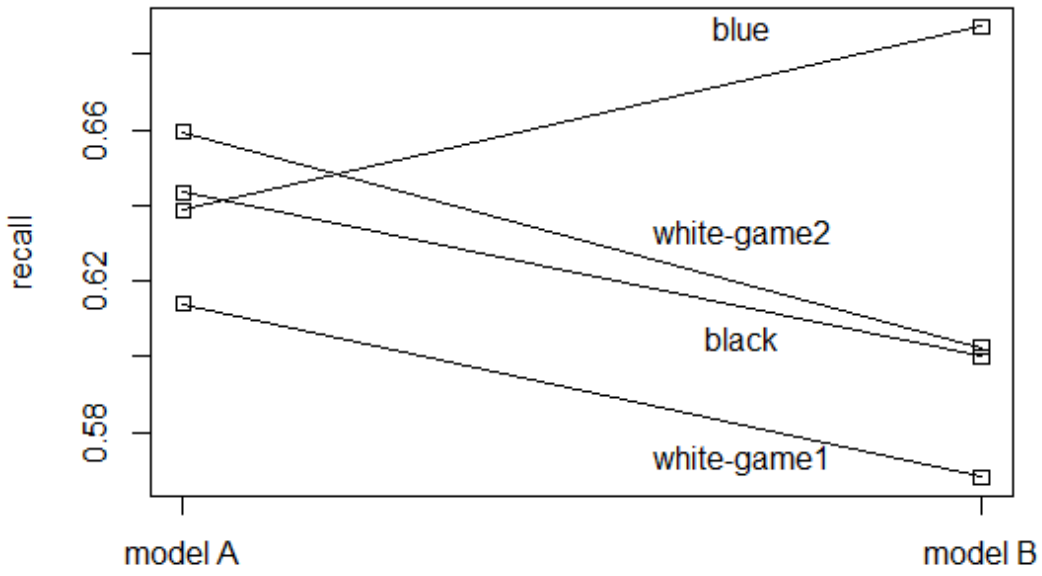


Figure 10. Change in recall from model A to B with different color of uniform.

Table 4. Effect of occlusion or being pictured against the rear wall on detection.

Model Testset	Model A Game 1	Game 2
Recall for all players	0.638	0.650
Precision for all players	0.858	0.831
Players in clusters	263	208
Recall for players in clusters	0.498	0.490
Players against the rear wall	110	57
Precision for players against the rear wall	0.725	0.525
Recall for players against the rear wall	0.336	0.368

Both model A and B (17400 timesteps) worked relatively well with the test set from the game 1, finding most of the players within the line markings (Fig. 11) and not detecting the non-player persons outside the line markings (Fig. 12). However, both models had problems with players occluding each other or standing against the rear wall (Table 4, Fig. 13, Fig. 14). These are well-documented problem areas for object detection in sports [5]. Furthermore, it seems that in some cases where players were standing close together, the predicted bounding boxes were shifted towards the other player (Fig. 15), however the cause of this tendency could not be verified by the present study.

The automatically estimated mAP values were almost the same for the models A and B (trained up to 17400 timesteps), the mAP values with game 1 data being somewhat higher than with game 2 data in the evaluation phase (Fig. 4 and 5) but the other way around for test data. This seem to indicate that the random differences between testing datasets were more influential than the differences between models A and B. The manually estimated recall value for players in blue uniform (not present in model A training data) was higher with model B compared to model A. However, the uniform color did not seem to affect the recall values for the model A (Fig. 10), and the recall values for the players with other uniform colors than blue were lower with model B than with model A. While it appears that the model B, trained with more varying images, did slightly better with the blue uniformed players than model A, the differences in mAP values and the manually estimated recall values were so small (and the related uncertainty so significant) that they are likely caused by random factors such as subjectivity in manual classification or model B training/testing data having different proportion of the "difficult" objects than the training/testing data of model A. To get more accurate idea of the differences between detection results by model A and model B, a more detailed evaluation protocol (that takes into account the factors here evaluated manually) should be developed and the model performance tested by a larger dataset than 50 images. Also, test data from more games would bring important insight (if the lighting conditions were more different, how would that affect the generalizability of the models? what if the players wore neon pink uniforms?).

SSD models are known to be less accurate with small objects [2]. For an application where model speed is not the most important consideration, a model such as Faster R-CNN model could provide better accuracy [2]. Some pre-processing (e.g. increasing contrast, filtering to sharpen the images, transformation to gray scale) of the images could also improve the results. Model B could benefit from a larger training data set, as the training data has more variability than the training set for model A.

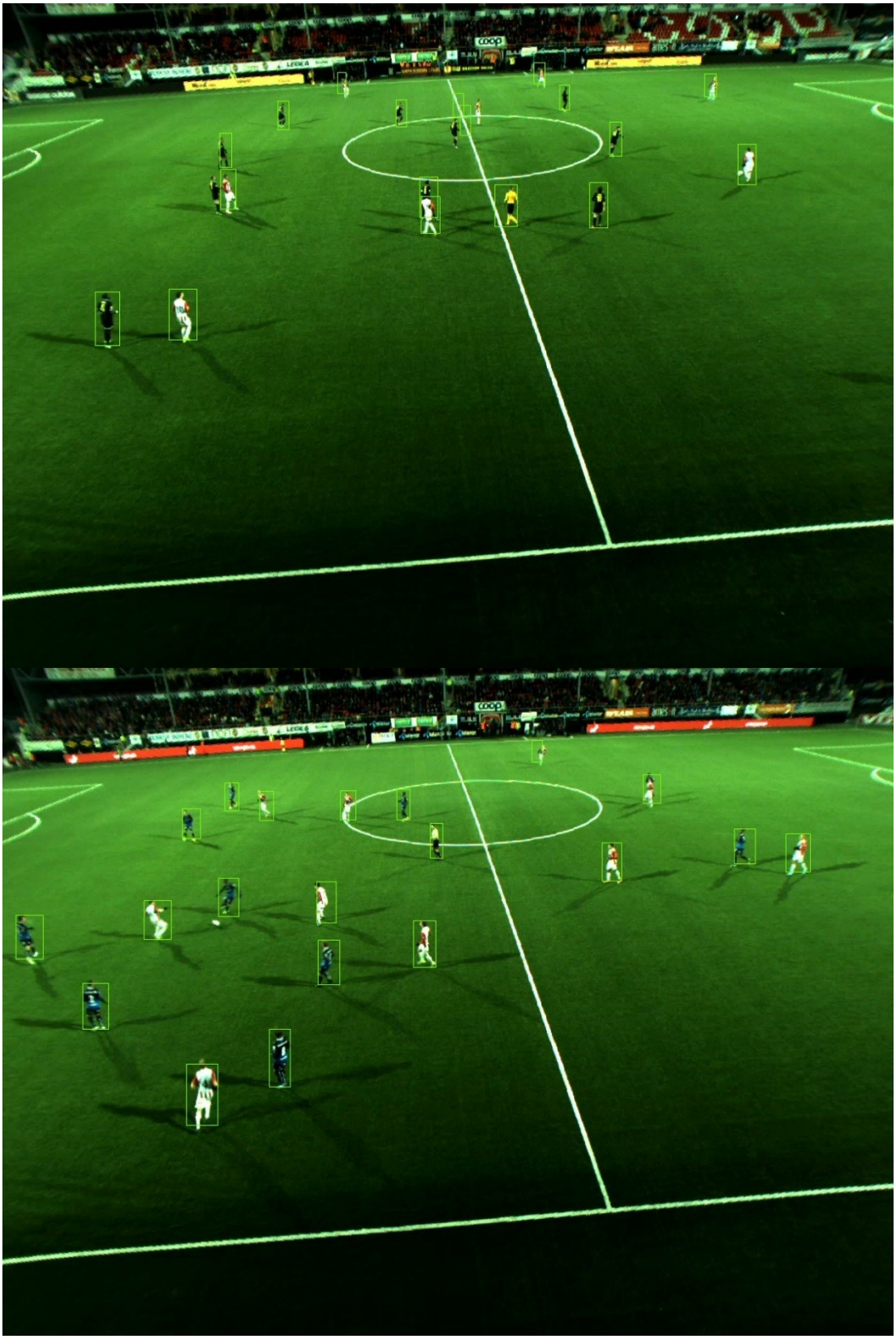


Figure 11. Player detection with fine-tuned model A (upper) and model B (lower).

Figure 12. The referee is not falsely detected as player.

Figure 13. Players close to the rear wall are poorly detected.

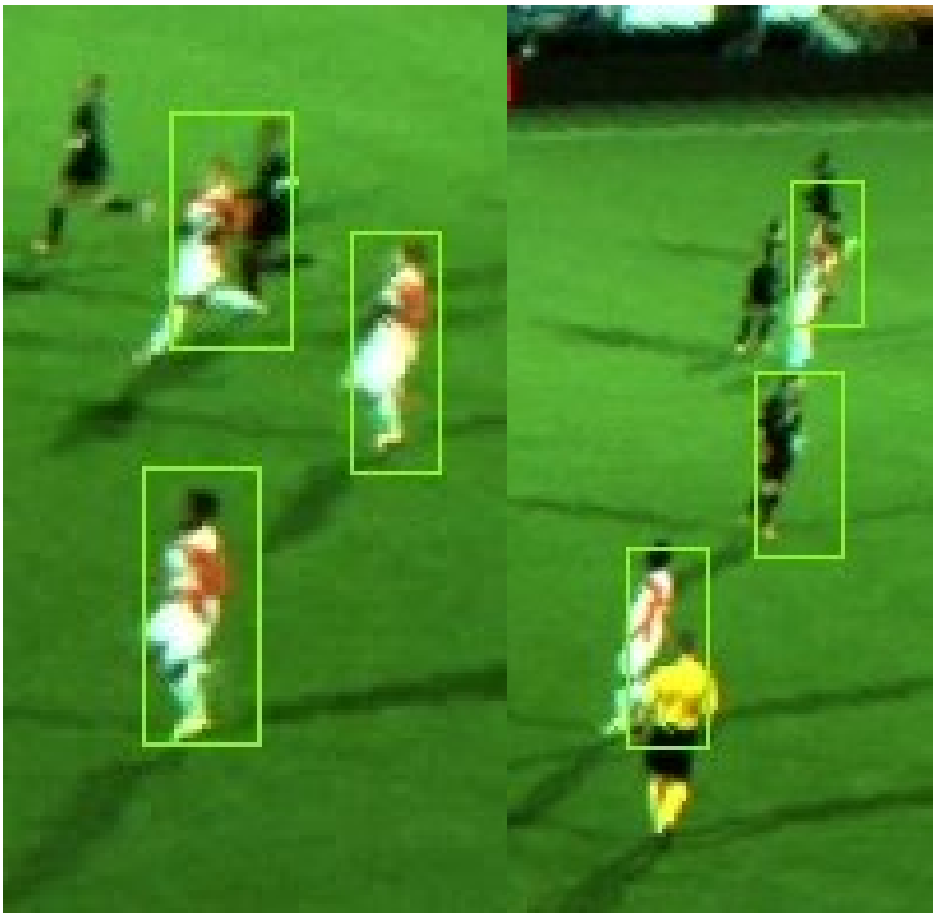


Figure 14. Occlusion causes problems for detection.

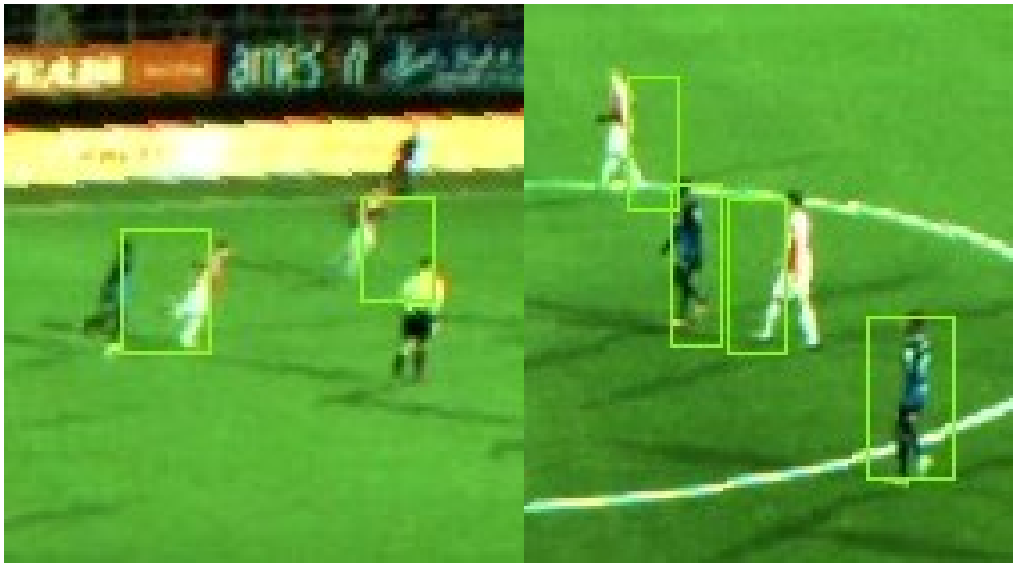


Figure 15. Some predicted bounding box location seem to be affected by nearby players.

5. CONCLUSION

A pre-trained SSD object detection model was tested for detecting football players, both as-is and as fine-tuned with a manually tagged data set. Model testing and training was done using features provided by TensorFlow Object Detection API, a freely available framework for object detection. Following conclusions were drawn based on the results:

1) The pre-trained model was useless for detecting players in the test images.
2) A fine-tuned model worked reasonably well.
3) Problem areas were players in clusters and/or pictured against the rear wall.
4) A model trained with data from one game was able to detect players in footage from another game. The overall model performance did not much improve by training the model with data from two games.
5) The model performance evaluated as mAP improved when the model was trained from 17 400 to 200 000 timesteps. Optimal number of timesteps is likely to be around 100 000 timesteps.

Topics for further study:

1) A model with another type of meta architecture such as Faster R-CNN could be better suited for detecting small objects.
2) Image pre-processing could improve detection results.
3) A more comprehensive testing protocol should be developed, and larger and more variable datasets used for training and testing the models.
4) The models should be tested on better-quality videos.

6. REFERENCES

[1] Tran D (2017) How to train your own Object Detector with TensorFlow's Object Detector API. URI: https://medium.com/towards-data-science/how-to-train-your-own-object-detector-with-TensorFlows-object-detector-api-bec72ecfe1d9. Cited November 5, 2017.

[2] Huang J, Rathod V, Sun C, Zhu M, Korattikara A, Fathi A, Fischer I, Wojna Z, Song Y & Guadarrama S (2016) Speed/accuracy trade-offs for modern convolutional object detectors. arXiv preprint arXiv:1611.10012 .

[3] Redmon J, Divvala S, Girshick R & Farhadi A (2016) You only look once: Unified, real-time object detection. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. : 779-788.

[4] Liu W, Anguelov D, Erhan D, Szegedy C, Reed S, Fu C & Berg AC (2016) Ssd: Single shot multibox detector. European conference on computer vision. , Springer: 21-37.

[5] Thomas G, Gade R, Moeslund TB, Carr P & Hilton A (2017) Computer vision for sports: Current applications and research topics. Comput Vision Image Understanding .

[6] Parisot P & De Vleeschouwer C (2017) Scene-specific classifier for effective and efficient team sport players detection from a single calibrated camera. Comput Vision Image Understanding .

[7] Pettersen SA, Johansen D, Johansen H, Berg-Johansen V, Gaddam VR, Mortensen A, Langseth R, Griwodz C, Stensland HK & Halvorsen P (2014) Soccer video and player position dataset. Proceedings of the 5th ACM Multimedia Systems Conference. , ACM: 18-23.

[8] Tzutalin (2017) LabelImg. Git code. GitHub repository .

[9] The TensorFlow Authors (2017) create_pascal_tf_record.py. URI: https://github.com/TensorFlow/models/blob/master/research/object_detection/dataset_tools/create_pascal_tf_record.py. Cited October 28, 2017.

[10] The TensorFlow Authors (2017) TensorFlow Object Detection API (readme). URI: https://github.com/TensorFlow/models/tree/master/research/object_detection. Cited November 23, 2017.

[11] Lin T, Maire M, Belongie S, Hays J, Perona P, Ramanan D, Dollár P & Zitnick CL (2014) Microsoft coco: Common objects in context. European conference on computer vision. , Springer: 740-755.

[12] Rathod V & Wu N (2017) TensorFlow detection model zoo (documentation). URI: https://github.com/TensorFlow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md. Cited November 23, 2017.

[13] The TensorFlow Authors (2017) Supported object detection evaluation protocols. URI:

https://github.com/TensorFlow/models/blob/master/research/object_detection/g3doc/evaluation_protocols.md. Cited December 12, 2017.

[14] Everingham M, Van Gool L, Williams CK, Winn J & Zisserman A (2010) The pascal visual object classes (voc) challenge. International journal of computer vision 88(2): 303-338.

[15] McCann S (2017) It's a bird… it's a plane… it… depends on your classifier's threshold. URI: https://sanchom.wordpress.com/tag/average-precision/. Cited December 10, 2017.

[16] Tran D (2017) Building a Real-Time Object Recognition App with TensorFlow and OpenCV. URI: https://towardsdatascience.com/building-a-real-time-object-recognition-app-with-TensorFlow-and-opencv-b7a2b4ebdc32. Cited November 5, 2017.

[17] Rathod V & Wu N (2017) Object Detection Demo. URI: https://github.com/TensorFlow/models/blob/master/research/object_detection/object_detection_tutorial.ipynb. Cited November 23, 2017.

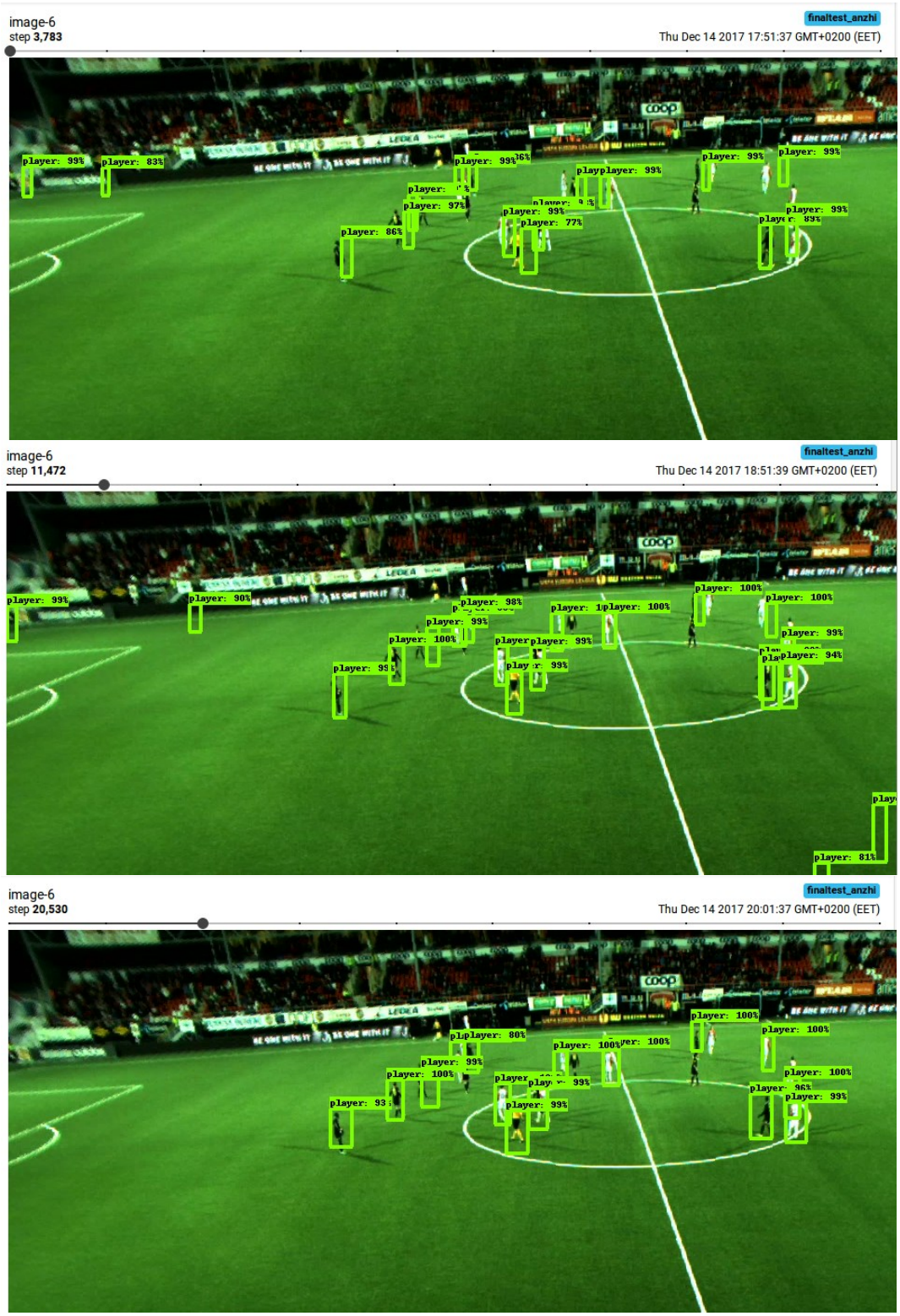
7. APPENDICES

Appendix 1. Evolution of bounding box locations over time-steps.

Appendix 2. Evolution of mAP values over time-steps.

Appendix 1. Evolution of bounding box locations over time-steps.

Model A – test data from game 1



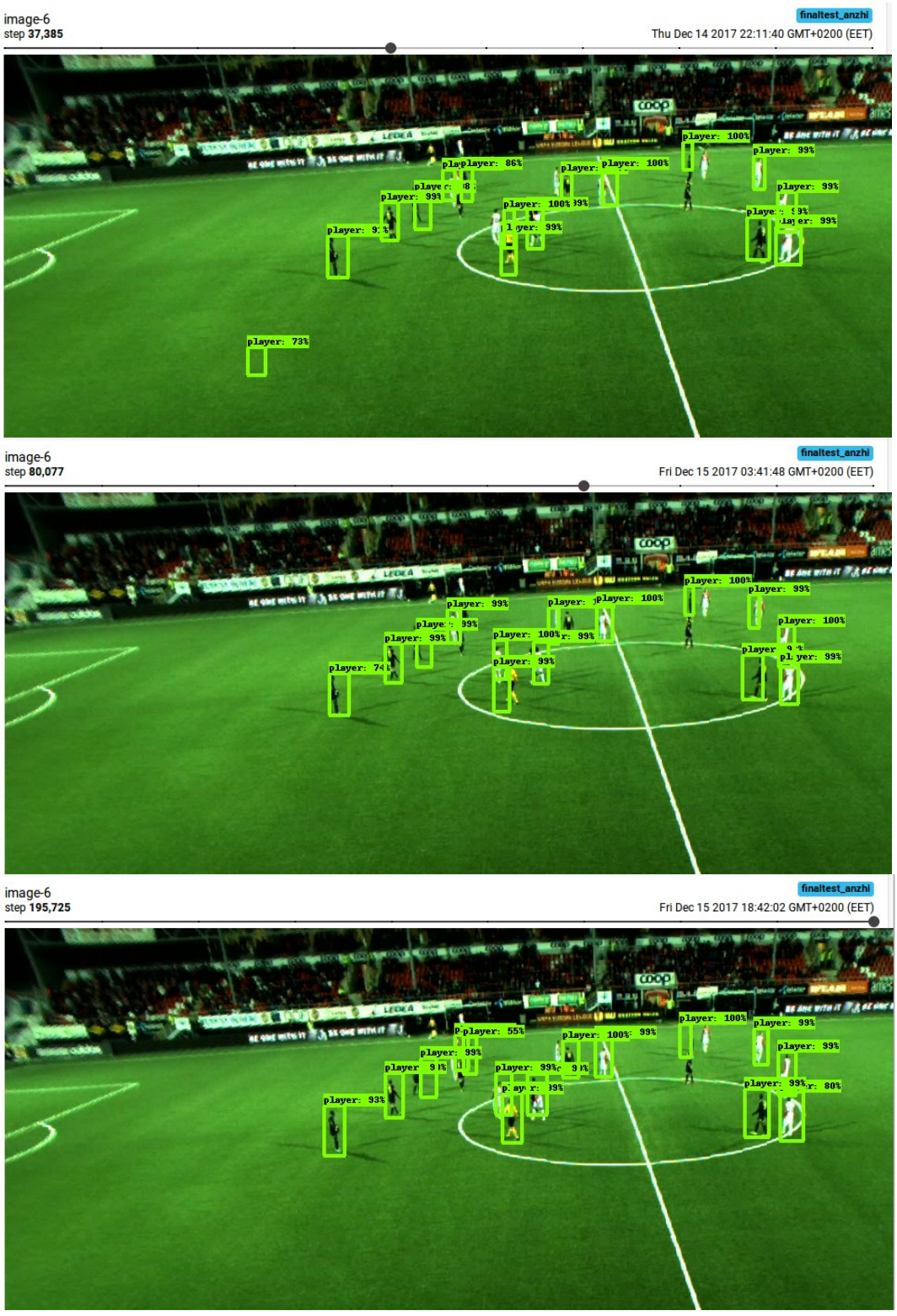


image-6
step 37,385
finaltest_anzhi
Thu Dec 14 2017 22:11:40 GMT+0200 (EET)
image-6
step 80,077
finaltest_anzhi
Fri Dec 15 2017 03:41:48 GMT+0200 (EET)
image-6
step 195,725
finaltest_anzhi
Fri Dec 15 2017 18:42:02 GMT+0200 (EET)

Model A – test data from game 2

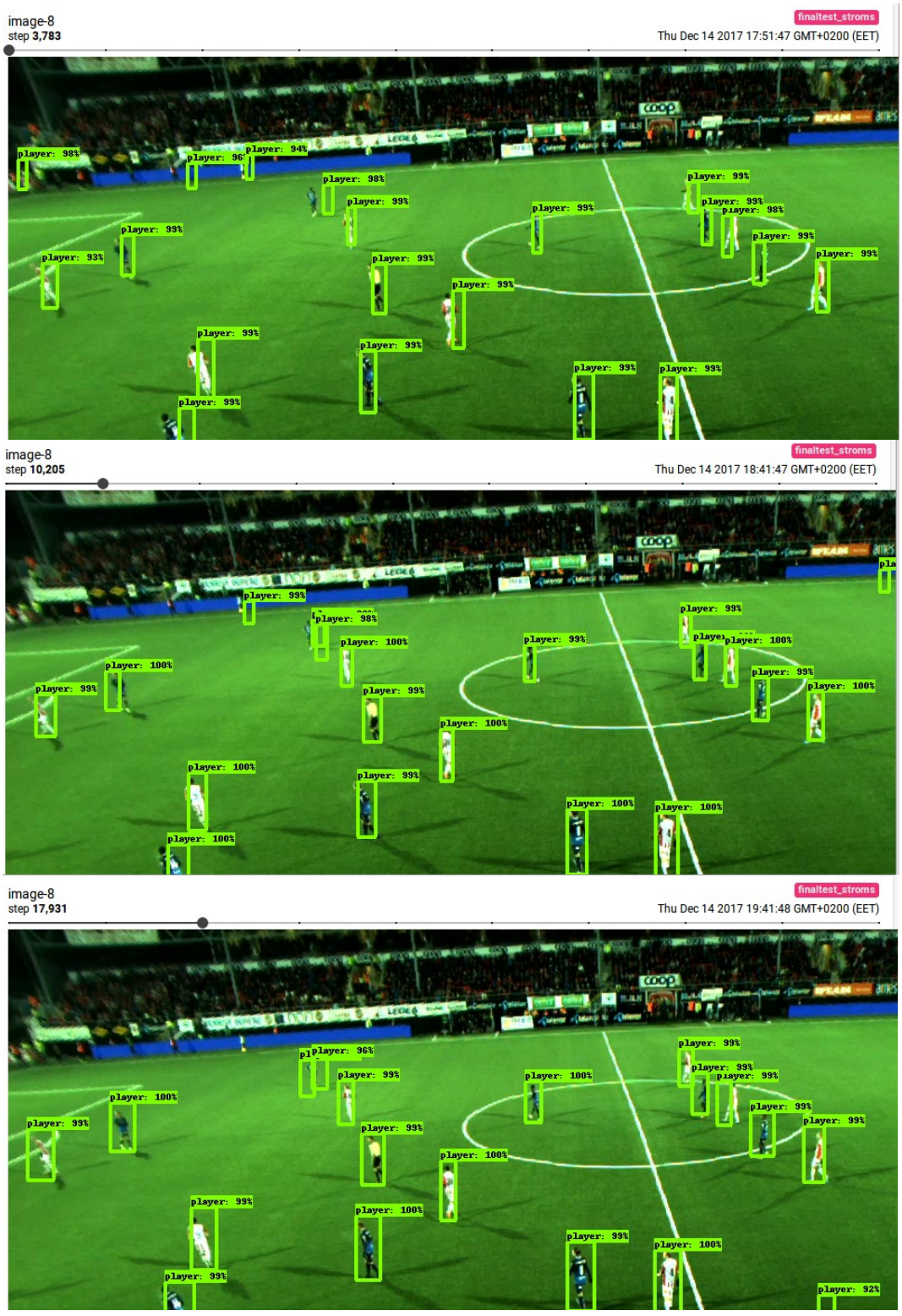



image-8
step 80,077
finaltest_stroms
Fri Dec 15 2017 03:41:58 GMT+0200 (EET)
image-8
step 200,000
finaltest_stroms
Fri Dec 15 2017 19:17:11 GMT+0200 (EET)

Model B – test data from game 1

image-1
step **25,373**

finaltest_anzhi

Sat Dec 16 2017 00:06:01 GMT+0200 (EET)

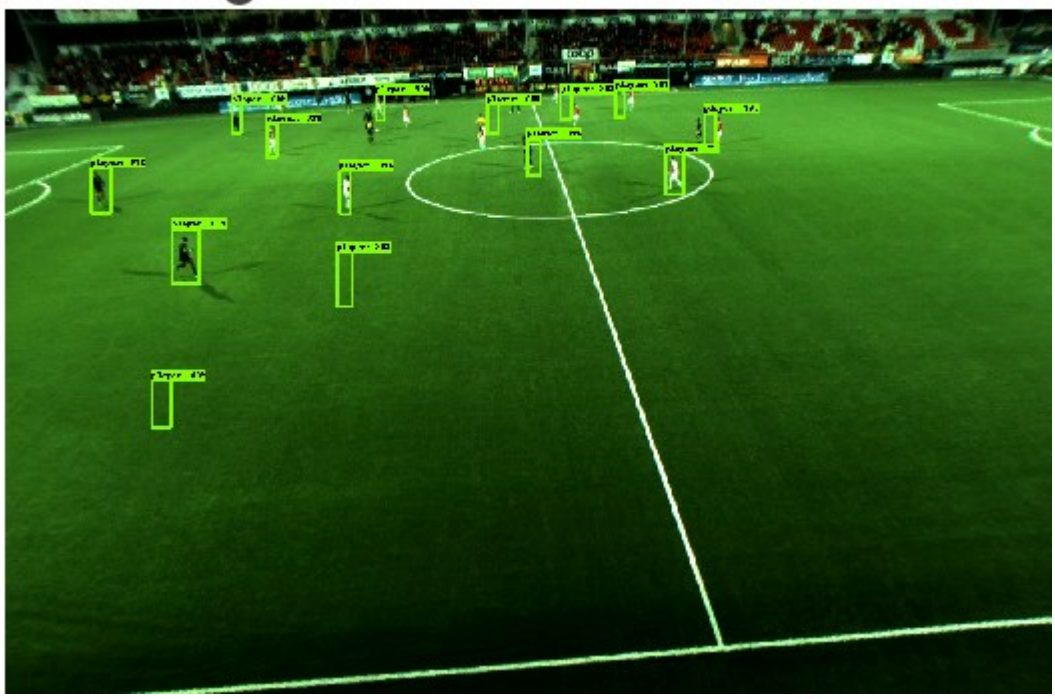


image-1
step **39,409**

finaltest_anzhi

Sat Dec 16 2017 01:56:02 GMT+0200 (EET)

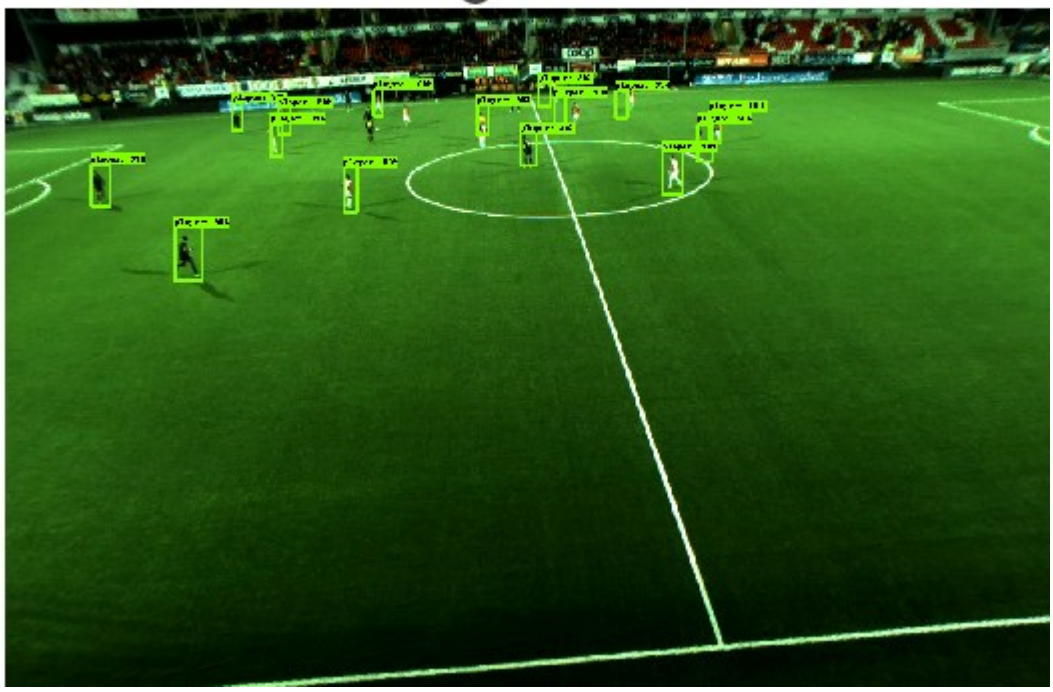


image-1
step **110,548**

finaltest_anzhi

Sat Dec 16 2017 11:16:12 GMT+0200 (EET

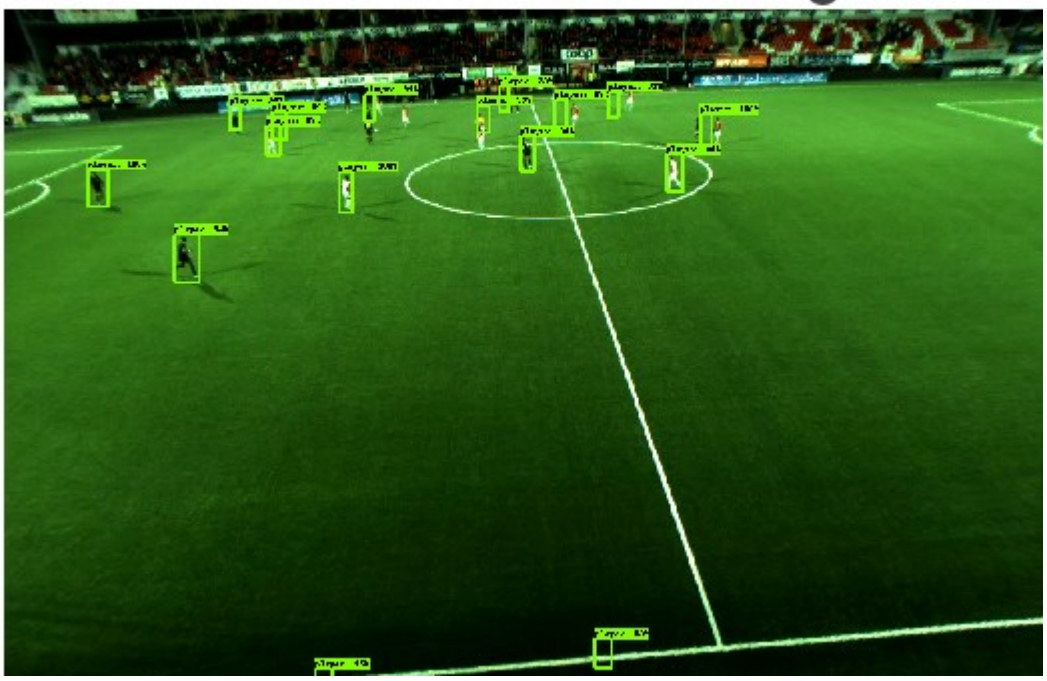
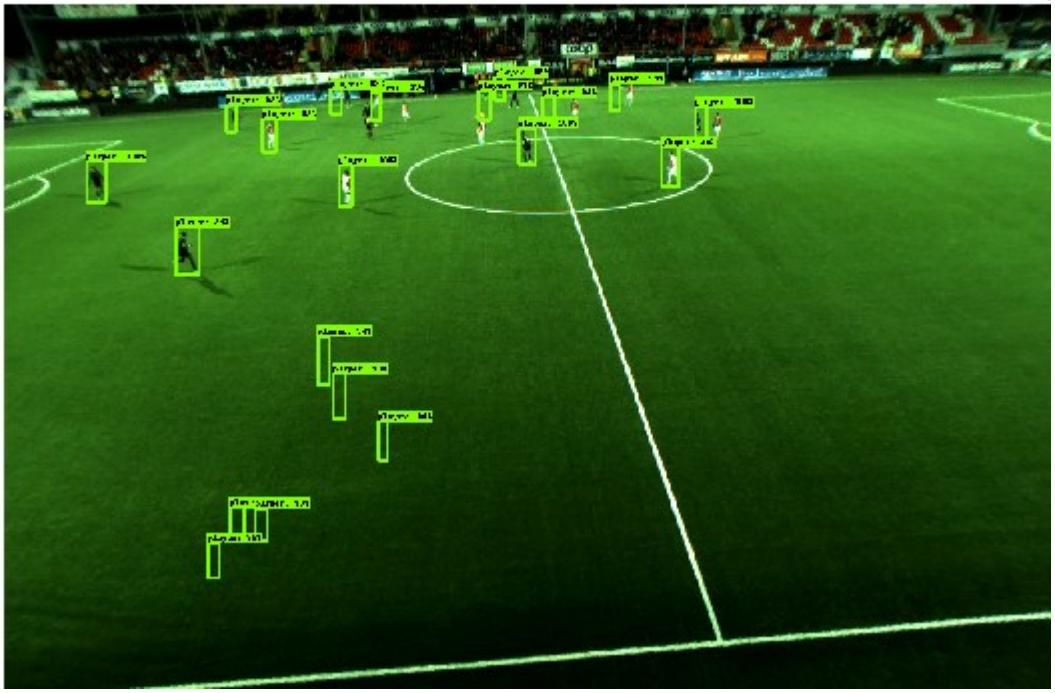


image-1
step **199,170**

finaltest_anzhi

Sat Dec 16 2017 22:46:22 GMT+0200 (EET)



Model B – test data from game 2

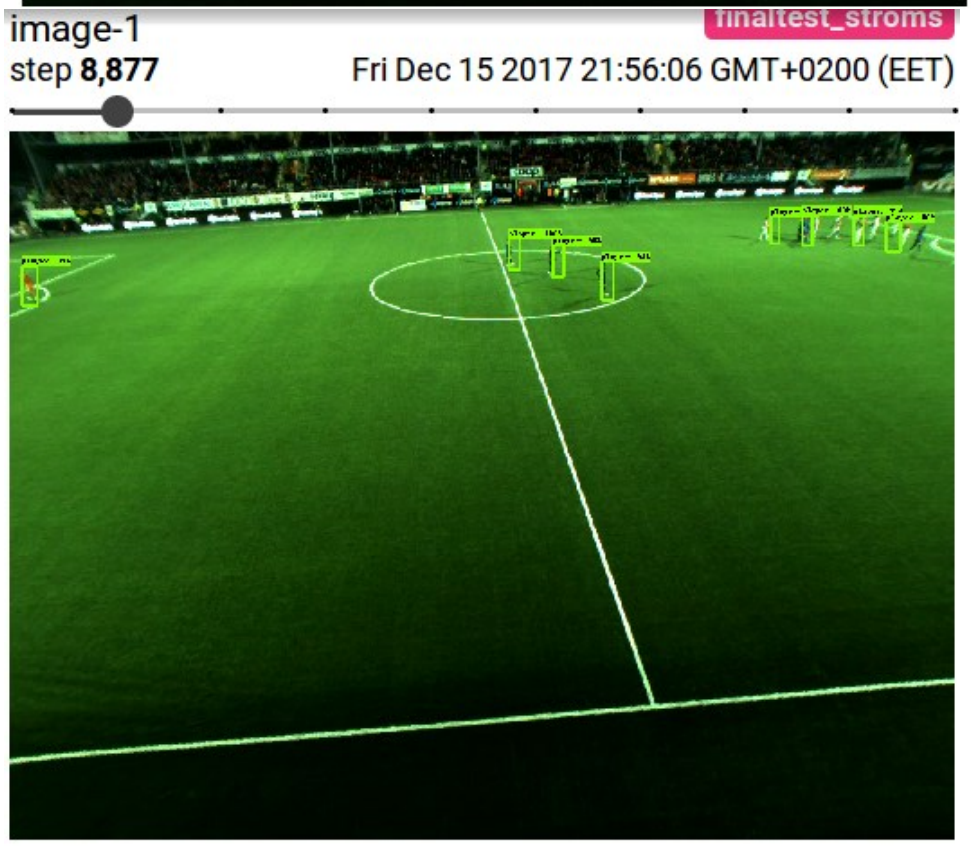


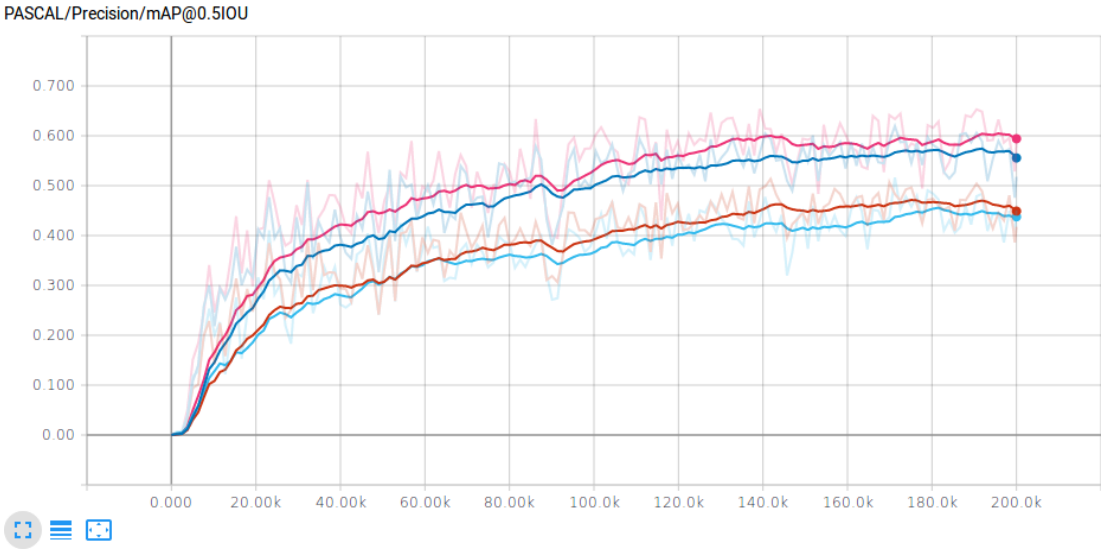

image-1
finaltest_stroms
step 19,173
Fri Dec 15 2017 23:16:08 GMT+0200 (EET)


image-1
finaltest_stroms
step 38,126
Sat Dec 16 2017 01:46:10 GMT+0200 (EET)

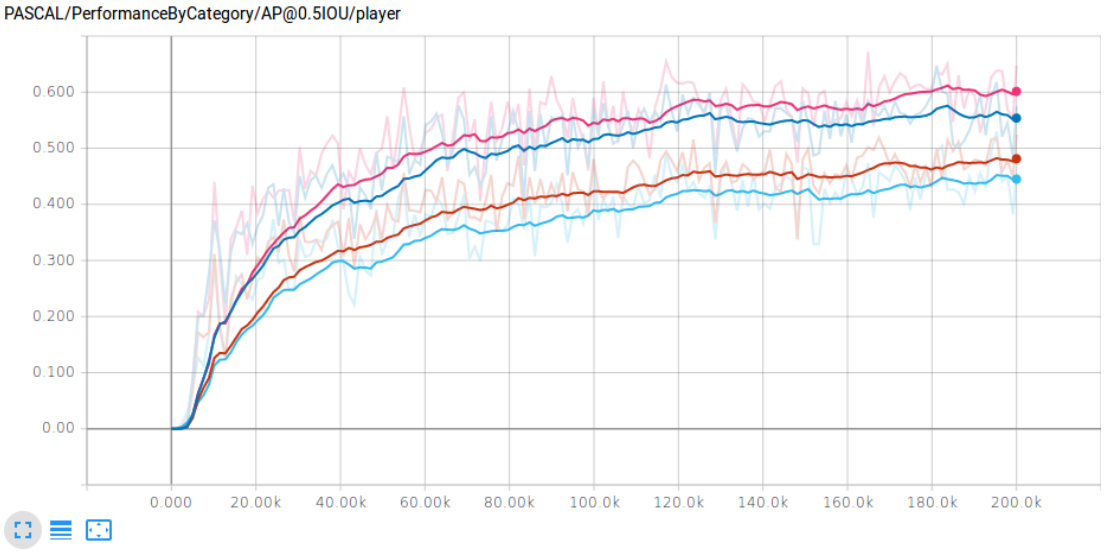

image-1
finaltest_stroms
step 110,548
Sat Dec 16 2017 11:16:19 GMT+0200 (EET)


image-1
finaltest_stroms
step 199,170
Sat Dec 16 2017 22:46:30 GMT+0200 (EET)

Appendix 2. Evolution of mAP values over time-steps.



Model A (pink = test data from game 2, blue = evaluation data from game 1, red = evaluation data from game 2, light blue = test data from game 1; smoothing = 0.9). Note that while the curves corresponding to test data from game 1 and evaluation data from game 2 (and test data from game 2 and evaluation data from game 1) seem suspiciously similar, the identity of each data set was double-checked: it seems that this is just a coincidence.



Model B (pink = test data from game 2, blue = evaluation data from game 1, red = evaluation data from game 2, light blue = test data from game 1; smoothing = 0.9). Note that while the curves corresponding to test data from game 1 and evaluation data from game 2 (and test data from game 2 and evaluation data from game 1) seem suspiciously similar, the identity of each data set was double-checked: it seems that this is just a coincidence.