Fast Convergence for Object Detection by Learning how to Combine Error Functions

Benjamin Schnieders
Department of Computer Science
University of Liverpool
L69 3BX Liverpool, United Kingdom
Email: bsc@liv.ac.uk

Karl Tuyls
Department of Computer Science
University of Liverpool
L69 3BX Liverpool, United Kingdom
Email: ktuyls@liv.ac.uk

***Abstract*— In this paper, we introduce an innovative method to improve the convergence speed and accuracy of object detection neural networks. Our approach, CONVERGE-FAST-AUXNET, is based on employing multiple, dependent loss metrics and weighting them optimally using an on-line trained auxiliary network. Experiments are performed in the well-known RoboCup@Work challenge environment. A fully convolutional segmentation network is trained on detecting objects' pickup points. We empirically obtain an approximate measure for the rate of success of a robotic pickup operation based on the accuracy of the object detection network. Our experiments show that adding an optimally weighted Euclidean distance loss to a network trained on the commonly used Intersection over Union (IoU) metric reduces the convergence time by 42.48%. The estimated pickup rate is improved by 39.90%. Compared to state-of-the-art task weighting methods, the improvement is 24.5% in convergence, and 15.8% on the estimated pickup rate.**

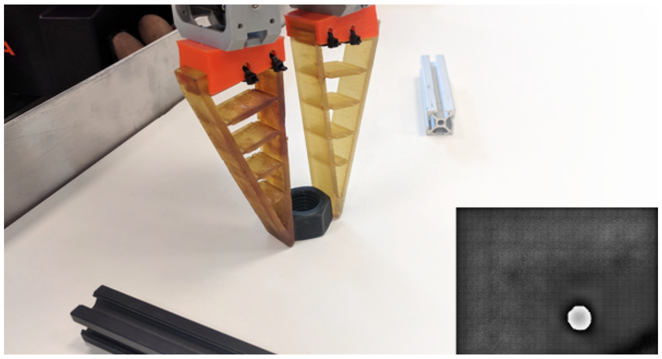


Fig. 1. Picking an object from the position indicated by the activations of the DCNN (lower right). The dominant axis of the activations is regarded as the facing of the object, and the gripper oriented perpendicular to it.

I. Introduction

Although Deep Convolutional Neural Networks (DCNNs) form the state of the art in object detection [1], they typically require a high number of training iterations to converge. In object detection and robotic pick-up frameworks like the *Factory of the Future* [2], a combination of low training times and accurate object detection is vital to ensure minimal turnaround times.

This paper introduces an innovative method for fast convergence of DCNNs performing object detection tasks, using an auxiliary network dubbed CONVERGE-FAST-AUXNET. Our method combines multiple, synergistic loss functions to improve both the accuracy and convergence speed of DCNNs. We use an image segmentation DCNN for object detection, and use the auxiliary neural network to learn optimal weights for its multiple loss functions during training. No pre-processing is required on the error metrics, and no additional hyper-parameters are introduced. The approach is tested in a RoboCup@Work [3] setup, simulating the requirements for the Factory of the Future. Figure I shows a typical pickup operation in the RoboCup@Work competition. Results illustrate that CONVERGE-FAST-AUXNET outperforms the state of the art in accuracy, convergence speed, and stability. Necessary training iterations are reduced on average by 24.5%, or roughly an hour of training on a NVIDIA Titan X. The probability of a successful picking operation is improved by 15.8% on average (9% absolute difference). The IoU measure is improved by 10.5%. Our main contributions can thus be summarized as follows:

i the introduction of the AUXNET approach, which learns optimal weights on-line in a multi-objective environment,
ii the combination of the commonly used Intersection over Union (IoU) metric and a Euclidean distance loss for faster convergence,
iii providing a labeled object classification and detection benchmark dataset, which is publicly available.[1]

The remainder of this paper is organized as follows: Section II provides background in object detection and presents related work. Section III introduces the CONVERGE-FAST-AUXNET approach and describes the introduced loss functions in detail. Section IV describes the experimental setup. Finally, Section V presents and compares the experimental results and Section VI concludes the paper.

II. Background and Related Work

Object detection is a well-researched task in the field of computer science and artificial intelligence. The purpose is to output a list of pre-trained objects that are present in a given image, along with their positions in the image [4]. Major breakthroughs using DCNNs in the last years have advanced the performance of neural network object predictors such

[1]See `https://airesearch.de/ObjectDetection@Work/`

that DCNNs are now the leading type of classifier in many image recognition [5] or detection benchmarks, most notably, the Pascal VOC Challenge [6]. State-of-the-art approaches such as Faster-RCNN [7] or YOLO9000 [8] pass an image through the network once, then propose regions for bounding boxes which are subsequently refined. Another way to detect objects, avoiding bounding box creation, is using semantic segmentation [9]. A semantic segmentation network features input and output layers of the same dimension, with each output pixel denoting which of the trained classes it belongs to. This effectively allows a different object to be predicted at every pixel. Further processing, such as clustering, is required to obtain the position of each object. The DCNN used in this research is based on the segmentation approach.

Multi-task learning can be used to improve both training time and prediction accuracy by learning a shared representation for multiple objectives simultaneously [10]. Multiple tasks can be learned by a single network by combining the respective task loss functions. Commonly, multi-task learning is employed to learn related, but independent tasks by a shared model, such as object detection and classification [11]. Combining error functions typically consists of scaling, weighting and finally summing them. These weights are conventionally modeled as hyperparameters [12]. Kendall et al. [13] present a statistically sound way to learn optimal weights, assuming that the epistemic error in the model will be eliminated with enough learning, and the dominant form or error remaining will be the homoscedastic uncertainty, which can be captured by measuring the variance of the loss over time. An error function with lower variance will then gain a higher relative weight, and vice versa. We will refer to this weighting method as *KGC-weighting*, after the initials of the authors. The combined loss $\mathcal{L}_{KGC}$ is defined in terms of the individual losses $\mathcal{L}_i$, and their respective variances σ_i^2 are defined as shown in Equation 1.

$$\mathcal{L}_{KGC} = \sum_i \frac{1}{2\sigma_i^2}\mathcal{L}_i + \log \sigma_i^2 \tag{1}$$

An immediate limitation of this approach is that functions with extremely low variances, such as an IoU with little or no overlap, may not train well as they can produce exploding gradients. Section V-A presents two approaches to address this issue: $KGC_{+\epsilon}$ and $KGC_{/_{Mean}}$-weighting.

III. Approach

In this work, we will show that combining mutually *dependent* error functions can provide a significant improvement in both convergence time and the resulting classifier accuracy. Common practice is to train multi-objective networks using mutually *independent* error functions, that is, when training one independent function, the values of the others are not affected.

While [10] hints that tasks can be too similar to gain improved performance from training a shared model, we will demonstrate that in an object detection framework, training a shared model minimizing two mutually dependent loss functions, i.e., loss functions that are dependent in such a way that training one may also reduce the other, provides a significant improvement over learning just one.

We then present Converge-Fast-Auxnet, a method that models the learning of optimal error function weights as an auxiliary task [14]. The joint error function can be described as minimizing the scaled, summed up error of all the available error functions, while optimizing their weights for fastest reduction. As an auxiliary network, we use a fully connected neural network with input features being the current value, the average, and standard deviation of every error function. The hidden layer consists of 24 ReLU neurons, which are combined into the two weights used to scale the two error functions. Figure 2 displays the auxiliary network layout. While the number of neurons in the hidden layer may be seen as an additional hyper-parameter, they should only depend on the number of error functions to be weighted, not the type or scale of the functions used.

The *total loss* of the DCNN, $\mathcal{L}_{Total}$ is derived from summing all weighted individual functions. The learned weights for every individual error function are applied to the respective loss value in a manner similar to *KGC-weighting*, i.e., dividing each loss value $\mathcal{L}_i$ by the respective weight w_i and adding the natural logarithm of the weight, as shown in Equation 2. Weighting the terms as shown introduces self-regulating properties of the weights compared to a simple multiplication with the loss. The auxiliary network is maximizing the rate of decline in *total loss*, the term to be minimized is provided in Equation 3. $\mathcal{L}_{Total}$ and $\mathcal{L}_{\text{AUXNET}}$ are optimized with two different Adam instances, both initialized with the same parameters as listed in Section II. The individual error functions to be weighted and subsequently collectively minimized are presented in Section III-B.

$$\mathcal{L}_{Total} = \sum_i \frac{1}{w_i}\mathcal{L}_i + \log w_i \tag{2}$$

$$\mathcal{L}_{\text{AUXNET}} = \frac{\mathcal{L}_{Total} - \overline{\mathcal{L}_{Total}}}{\overline{\mathcal{L}_{Total}}} \tag{3}$$

A. Network

The layout of the DCNN used was inspired by the networks described in [15], but was extended to feature the traditional hour-glass shape of semantic segmentation networks [9]. The derived network features four instead of the common three input channels, with the fourth channel encoding distance information obtained by using an Intel SR300 3D camera for construction of the dataset. The number of filters per layer was tweaked by hand, and the number of output layers reflects the number of different objects to be distinguished by the network. Before being presented to the network, all images are scaled down from the native resolution of 640×480 pixels to a more manageable size of 256×192 pixels. All activation functions are ReLUs. Regularization is introduced in the form of dropout regularization, with a 15% dropout chance in the convolutional layers 6 and 7. The innermost layer doubles as a classification vector. If a specific object is to be retrieved, only the activations on

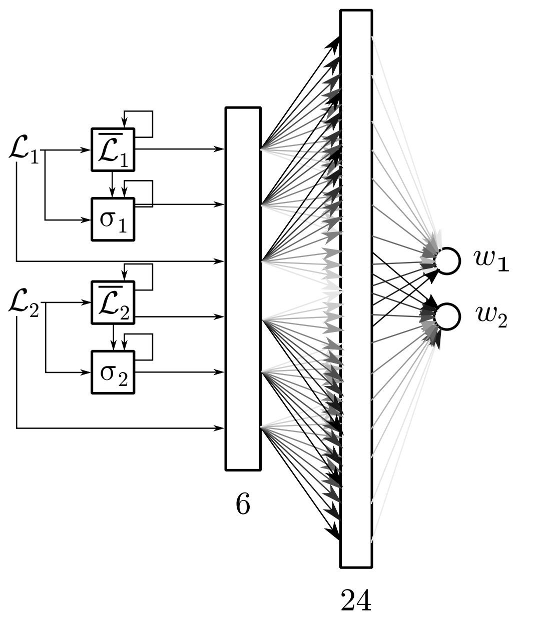


Fig. 2. The layout of AUXNET. The current losses $\mathcal{L}_i$ are averaged to $\overline{\mathcal{L}}_i$ over the course of the training session using exponential moving averages. The standard deviation σ_i is obtained likewise. The fully connected hidden layer produces output weights, which are inserted into Equation 2. The network optimized to the gradient of the DCNN error reduction, as detailed in Equation 3.

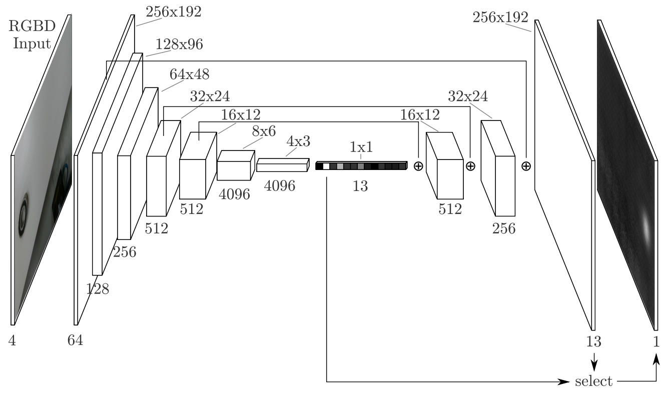


Fig. 3. The network layout used. Numbers above the layers indicate pixel dimensions, numbers below the image denote the number of channels or filters. All layers are convolutional/inverse convolutional, using ReLU activation functions. It is trained on either minimizing the IoU or distance error for a queried object on the corresponding output layer, or on a combination of both using Equation 2.

the corresponding layer are taken into account. A detailed description of the network can be found in Figure 3.

B. Error Functions

Commonly, object detection or semantic segmentation networks are trained on minimizing the cross entropy between the network output and the labeled ground truth. Training to maximize the IoU between the predicted and actual bounding boxes is not possible out of the box, as this function is not differentiable; however, one can use an approximate IoU measure instead, which [16] argue converges more quickly. Note that in this paper we are mainly minimizing

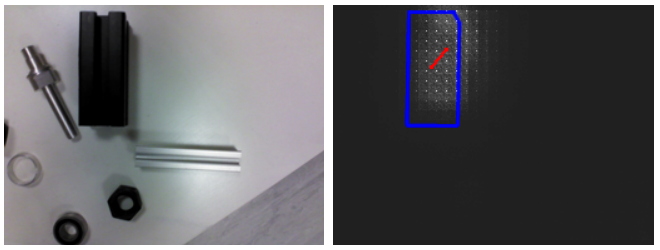


Fig. 4. Left: An example input image. Right: The output of the layer detecting 40×40 mm aluminium profiles. The ratio of activations inside the area of the object, displayed in blue, divided by the sum of all activations and the size of the blue area produces the IoU. The Euclidean distance error between weighted center of activations and ground truth annotation is displayed in red.

the *IoU Error*, defined as $1 - IoU$. Figure 4 visualizes how the IoU measure is calculated from a prediction and approximate segmentation data, obtained by a conventional object detection approach. In case the prediction and ground truth do not overlap, the IoU does not produce a usable gradient and training can get stuck. The Euclidean distance between prediction and label, however, is always well defined and produces a slope a gradient descent algorithm can follow. Because of this, and because the measure of quality for a picking operation will ultimately be a function of the distance between predicted and actual position of the object, one might train the network on this measure immediately. The object prediction is produced by obtaining the weighted average of activations on the corresponding output layer, as indicated in red in Figure 4. Minimizing this *distance error* lets the network converge faster and to a lower error, as Figure 5 demonstrates. However, low-frequency filters appear to dominate the output, in turn leading to activations being spread over the greater part of the image, as opposed to tightly localized around the desired position. Figure 6 shows this behavior. The distinction between multiple objects on the same layer using clustering is therefore impossible. As the IoU measure produces strongly localized activations, but the distance error converges much faster, we hypothesize that a combination of both can produce superior results.

An inherent weakness of the *distance error* in pixels or centimeters is its lack of expressiveness. Although it does describe the visual offset between prediction and label, it fails to capture the quality of this prediction: While halving an error from e.g. $0.8\,\mathrm{cm}$ to $0.4\,\mathrm{cm}$ will not result in any increased chance of a successful pick-up operation, halving the error from $2\,\mathrm{cm}$ to $1\,\mathrm{cm}$ will greatly increase the probability. The relation between a successful pickup and the distance error is therefore not linear. During empirical testing of the picking operation, we observed that for small objects, a distance error of $1\,\mathrm{cm}$ results in a failed grasp 50% of the time; for a distance offset of $3\,\mathrm{cm}$, the probability of failure rises to 90%. We thus model the *pickup error* by applying a negative exponent power function to the error in centimeters, coinciding in $E_{Distance}(0) = E_{Pickup}(0) = 0$, and E_{Pickup} asymptotically approaching 1 for $E_{Distance} \to \infty$, resulting in Equation 4.

$$E_{Pickup} = \frac{-1}{(E_{Distance})^2 + 1} + 1 \quad (4)$$

Figure 7 plots the distance error metric versus the pickup error in the usually occurring range. Using the pickup error metric readily scales the distance error in the same range as the IoU, between 0 and 1. Early results showed that training on pickup error directly was unsuccessful, due to the gradient of the function being too low for higher error values. However, we will use the pickup error and the *pickup rate*, defined as $1 - E_{Pickup}$, as an accurate scaling method when comparing or adding pickup and IoU errors.

IV. Experiments

To measure the convergence speed and resulting quality of employing different error functions and combinations thereof, each setup was trained for 100000 steps ($\approx$ 67 epochs) with a batch size of 8. An Adam optimizer [17] was used to minimize the loss term, with a learning rate of 1×10^{-5}, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 1 \times 10^{-8}$. Initial tests showed that all methods tested could train and converge with these values, so the hyperparameters for the neural networks to be trained were frozen at these values. In order to achieve statistical significance, 20 networks were trained for every method. A full training run with 100000 iterations takes about 10 hours to complete on an NVIDIA Titan X graphics card, regardless which of the methods we present is used. Every 500 iterations during training, the networks were tested on the entire validation set to get an accurate estimate of their performance on unknown data. This data forms the basis for all results to be presented.

A. Dataset

Employing the RoboCup@Work challenge as a Factory of the Future simulation, the up-to-date version of team *smARTLab@Work*'s previously world-cup winning [18] hard- and software was used to produce a dataset. More than 35000 RGBD 3D images were taken with an Intel SR300 3D camera, and over 14000 images were manually labeled for object detection. Alternative RGBD benchmark datasets containing RoboCup@Work objects are scarse. Distinct training and evaluation sets were recorded, mimicking the Industry 4.0 requirements such that both are comparatively small with about 12000 images for training, and 2000 images for validation. Each of the 13 RoboCup@Work object types is recorded from multiple angles, placed on platforms as defined in the official rule book[2]. Figure 8 shows an example. The training and validation sets were manually labeled with the center of gravity, or "pickup point", of each object. The dataset can be retrieved from `https://airesearch.de/ObjectDetection@Work/`.

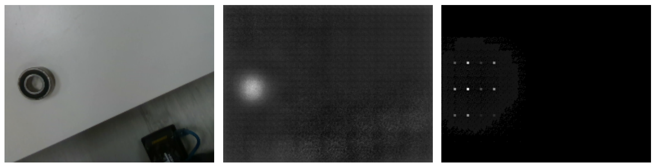


Fig. 6. Left: The input image leading to the activations shown right. Middle: Activations on the "Bearing" layer for a network trained on IoU. Right: Activations when trained on distance error alone. The object position is predicted accurately by constructing the weighted average of the activations, however, the majority of the filter weights are zero, and most of the activations are not close to or within the object. Note that the size of the activated pixels was increased for better visibility.

V. Results

The results of the experiments conducted are split into analyzing how and which weights are derived by the available methods to weight error functions, and their respective performance regarding convergence speed and error reduction.

A. Optimal Weight Learning

In order to compare *KGC-weighting* to the weighting learned by the auxiliary task network AUXNET, we are testing both methods to empirically evaluate their respectively derived combinations of the distance error and IoU error

[2]Available on `http://www.robocupatwork.org/`

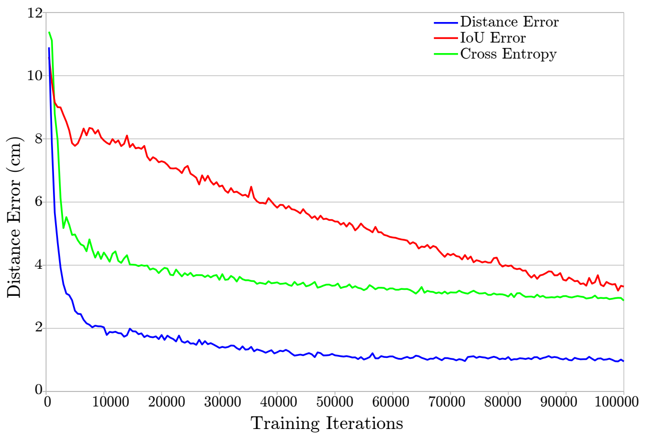


Fig. 5. Training directly on the distance error metric (blue) leads to faster convergence and lower error than training on cross entropy (green) or IoU (red). Averaged from 20 runs with equal hyperparameters.

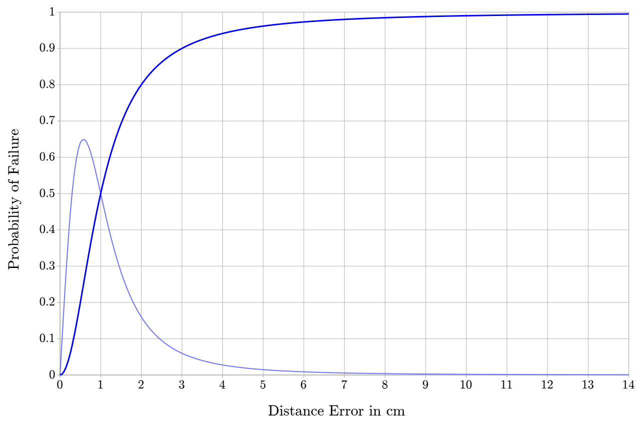


Fig. 7. The probability of a failed pickup (blue) and its derivative (light blue) plotted in a range of 0 to 14 cm of distance error.

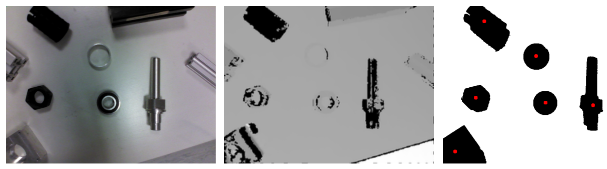


Fig. 8. Left: An example image from the dataset, displaying multiple RoboCup@Work objects. Center: The corresponding depth image. Right: Imperfect segmentation mask obtained by the current RoboCup@Work software. Manually labeled pickup positions marked in red.

measures. To produce a valid *KGC-weighting*, scaling the error functions between 0 and 1 is required. The IoU error is produced in this range by design, but the pixel distance error may range from 0 to 320 pixels, i.e. the diagonal of the image. In practice however, distances over 70 pixels do not occur. As a compromise, we divide the pixel distance by 100 in order to scale it in the required range. Tests using the pickup error as scaled distance error term during training were unsuccessful, evidently due to the lack of slope of the error function throughout most of its range.

Training the network on the *KGC-weighted* downscaled pixel distance and IoU posed another challenge, as frequently, the network would irrecoverably die out, most certainly due to the ReLU activation functions getting stuck with negative weights, a well known problem with ReLU activation function and steep gradients [19]. In order to get 20 successful training runs, 30 networks had to be trained, as one third of them did not converge. A contributing factor to the dying ReLUs appears to be the extreme weights derived from the task variance. Dividing by the very low values of initial IoU variance leads to extreme weights, which in turn can lead to a gradient explosion. Adding an epsilon of 1×10^{-3} to both variances mitigated the neuron decay, however, with a detrimental effect on the prediction performance. We call this method $KGC_{+\epsilon}$*-weighting*.

Another way to combat exploding gradients is to divide the error variance by the error mean, assuming that an error function with a higher error in general will also feature a higher absolute variance. This $KGC_{/Mean}$*-weighting* performs comparable to regular *KGC-weighting*, and does not require explicit scaling of the error functions to a 0 to 1 range. It also appears to generate more sensible weights, as the network converged in all 20 out of 20 runs. Table I shows the extreme values of the learned weights. Figure 9 presents the weight history over a full training episode. For these charts, the weights were normalized and inverted, so a larger area means a higher contribution of the weight to the loss term. As $KGC_{/Mean}$ and AUXNET are trained on the raw pixel distance rather than a scaled distance between 0 and 1, their weights are scaled accordingly by 100. Figure 10 shows the effect of the different weight learning methods on error reduction during training.

B. Convergence Speed and Accuracy

To compare the trained classifiers, convergence speed and error after convergence are measured separately, as in

Learning method	$W_{Distance}$	W_{IoU}
KGC-weighting		
min	6.14×10^{-6}	2.26×10^{-7}
max	8.57×10^{-2}	2.18×10^{-1}
$KGC_{+\epsilon}$*-weighting*		
min	2.25×10^{-1}	1.00×10^{-3}
max	8.29×10^{2}	2.19×10^{-1}
$KGC_{/Mean}$*-weighting*		
min	7.98×10^{-1}	3.98×10^{-5}
max	6.44×10^{1}	2.92×10^{-2}
AUXNET		
min	2.71×10^{-4}	4.95×10^{-3}
max	7.76×10^{-3}	5.17×10^{-2}

TABLE I

EXTREME VALUES FOR THE ERROR FUNCTION WEIGHTS LEARNED BY DIFFERENT METHODS. *KGC-weighting* CAN PRODUCE THE MOST EXTREME GRADIENTS, DIVIDING THE LOSSES BY THE SMALLEST WEIGHTS. AUXNET APPEARS TO PROVIDE THE MOST STABLE WEIGHTS.

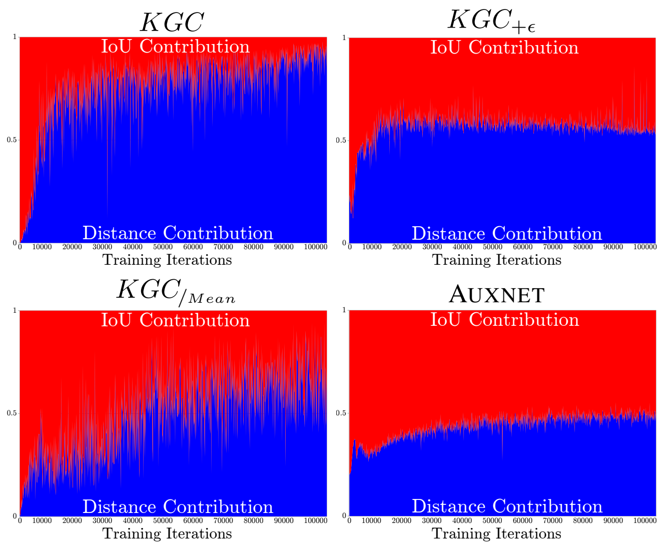
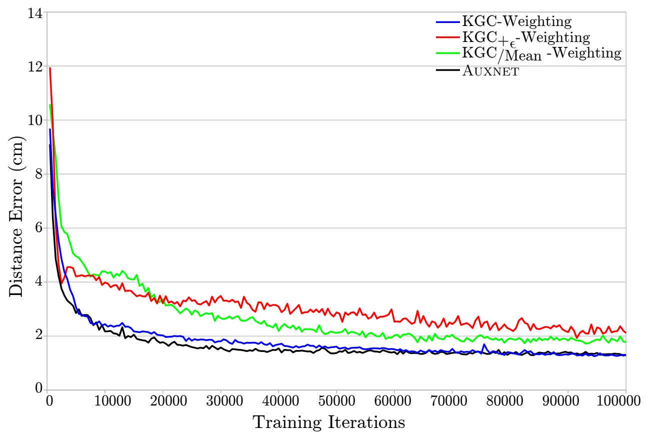


Fig. 9. Normalized and inverted weights derived from the different methods. Positive contribution to distance error is shown in blue, contribution to the IoU error is shown in red. All KGC-measures start off with training almost exclusively on IoU. Adding an epsilon balances the weights more. AUXNET features more balanced weights from the start, explaining the fast early convergence. The weights appear to be more stable than KGC's.

different scenarios, users may value time and quality constraints differently. Using dropout regularization effectively prevented our networks from overtraining; however, it also introduced fluctuations of the error value after convergence. These fluctuations prohibit the use of an epsilon threshold on the discrete derivative of the error as convergence measure, as the relation between selected epsilon and the resulting convergence point is highly nonlinear, and varies for networks trained on different error functions. For our convergence point determination method, we instead assume a normal distribution of errors around the mean error after convergence. Further, we assume that the longer we train a network, the more it converges, i.e., that it does not diverge and also that it always reaches convergence within 100000 iterations. Visual analysis of the error charts supports these assumptions, as hinted by 15 randomly selected charts all converging, shown



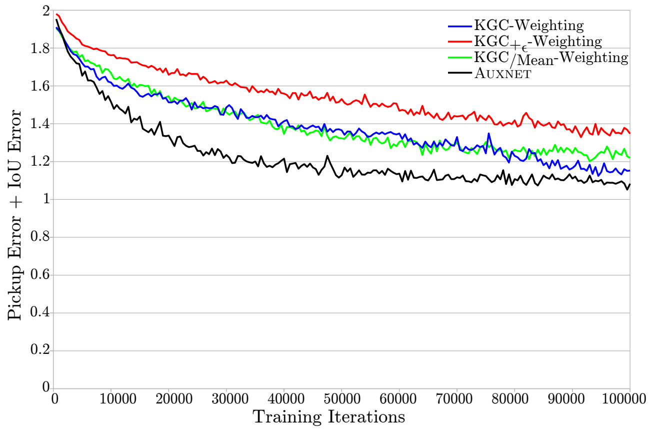


Fig. 10. Charts comparing the reduction of different error measures for (blue) *KGC-weighting*, (red) $KGC_{+\epsilon}$*-weighting*, (green) $KGC_{/Mean}$*-weighting* and (black) weighting by AUXNET. The first chart plots the distance error in cm, the second shows the reduction of an added pickup- and IoU error measure. Charts are averaged from all 20 runs and obtained on the entire evaluation set during training.

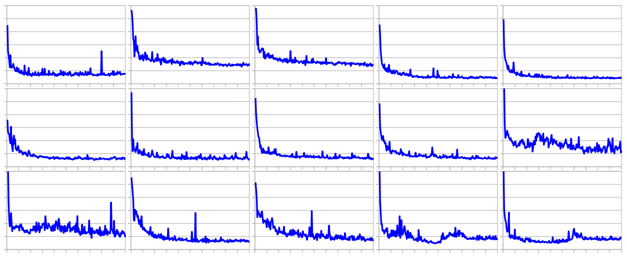


Fig. 11. All test runs were examined for proper convergence. 15 randomly selected, successfully converging charts are shown here for convenience.

in Figure 11. We then iteratively remove the earliest data points that do not fit a normal distribution around the mean error with regards to the standard deviation, and shrink the acceptance window to the newly obtained mean and standard deviation. Figure 12 illustrates this process. The first entry of the remaining data is considered to be the convergence point, and the average of the remaining data can be considered the average error after convergence.

Figure 13 displays the convergence speed and resulting error after convergence in a double box plot. This plot shows that training on the distance error alone yields the highest error, although it trains comparatively fast. Training on IoU error alone converges very slowly, but to a lower error compared to the distance error metric. This is due to the fact that training on IoU error reduces both the IoU and distance error, but training on the distance error does not lower the IoU error. Training on a simple addition of distance error and IoU results in reduced error compared to either. Both *KGC-weighting* and $KGC_{/Mean}$*-weighting* significantly reduce the combined error, but do not improve the convergence time. While AUXNET features a higher deviation in both convergence speed and quality, the majority of the runs perform significantly better than any other combination method, reducing the average summed error by 15.85% compared to *KGC-weighting*; it also proved to be more stable, as all 20 test runs converged. AUXNET manages to weight the error terms such that the network can converge on average in 26800 iterations, much faster than *KGC-weighting* with an average of 34850 iterations – reducing it by nearly 25%, or saving about one hour of training on an Nvidia Titan X. Table II shows the relative improvement of AUXNET compared to just training on the distance error, IoU error, on a simple addition of IoU and Distance Error, and *KGC-weighting*.

AUXNET vs.	Convergence Time	Pickup Rate	IoU
Distance Error	+0.48%	+11.83%	+35.31%
IoU Error	−42.48%	+39.90%	+3.31%
Distance+IoU	−16.90%	+21.53%	+24.41%
KGC-weighting	−24.53%	+15.85%	+10.48%

TABLE II
RELATIVE IMPROVEMENTS OF AUXNET OVER SELECTED OTHER METHODS. DATA IS AVERAGED FROM 20 RUNS. AUXNET CONVERGES FASTER AND PRODUCES A HIGHER PICKUP RATE AND IOU THAN THE OTHER TESTED METHODS, AN EXCEPTION BEING THE DISTANCE ERROR CONVERGING MARGINALLY FASTER.

The high variation of quality and convergence can be explained by analyzing the charts of the more slowly converging outliers. Figure 14 shows that these network experienced near-death scenarios, in which part of the network died out, from which they never fully recovered. It shall be noted that optimal weights for error functions will automatically also pose a higher threat of neuron death for the network, as optimal weights will produce steeper gradients, which in combination with Adam's learned momenta may lead to ReLU neurons getting stuck with negative activations and dying. An obvious solution may be converting the DCNN to leaky ReLUs.

C. Significance

The distributions of error values and convergence points were tested against each other with a two-sample Kolmogorov-Smirnov (KS) test, using $\alpha = 0.05$, the null hypothesis being that the different sources can not be distinguished. All error distributions were statistically significantly different. Table III shows the results of the convergence tests. With distributions too similar or variations too high in either

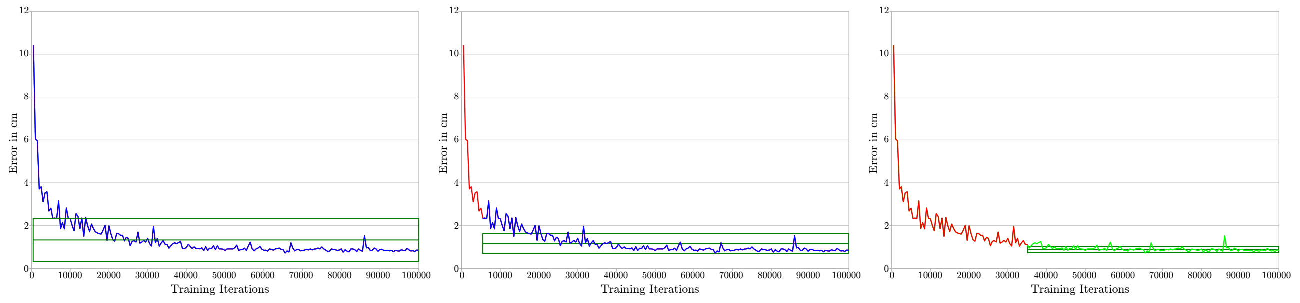


Fig. 12. The convergence point determining process in detail. Left: An acceptance window (green) is created from the average error and the standard deviation. Middle: While rejecting points (red), the acceptance window shrinks. Right: The leftmost point within the acceptance window becomes the convergence point. For the amount of noise present in our data, a window of $\pm\sigma/2$ produced best results.

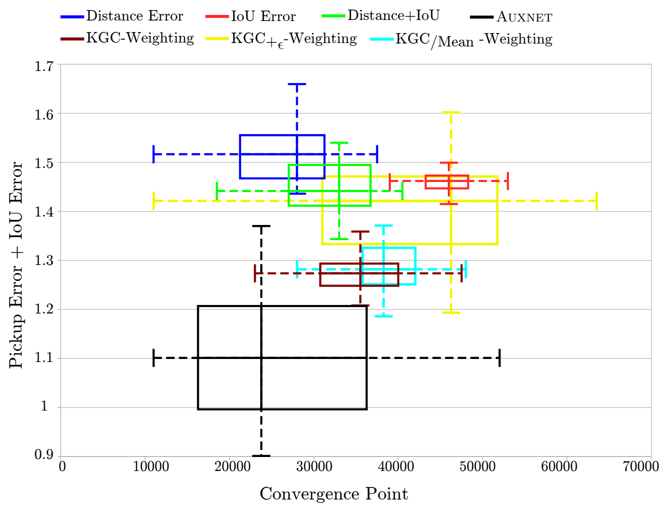


Fig. 13. A double box plot showing the median and quartiles from twenty networks trained on each error function or combination thereof. Extents along the X-Axis describe variation in convergence speed, while the Y-Axis shows the summed *pickup*- and IoU error. The median convergence and error measures for AUXNET are significantly better than for any compared method.

sample, some of the null hypotheses could not be rejected for the number of test runs.

VI. Conclusion

We have shown that error functions do not have to be independent in order to speed up convergence and produce lower error. We have introduced the CONVERGE-FAST-AUXNET approach, a method to learn a weighting between these error functions in a manner that improves convergence and the reduction of error even more. Repeated test runs have shown that improvements on state-of-the-art methods to derive optimal weightings are significant. In conclusion, object detection tasks can be improved in accuracy and training times by further constraining the network using a combination of available error metrics, and using approaches such as AUXNET to learn optimal weights between them. The *Factory of the Future* benefits especially from the faster convergence, enabling it to train object detectors for each new task in near real time.

Future work includes extending CONVERGE-FAST-AUXNET to a recurrent network that can learn the decrease along the gradients of the network, and modify the output weights in accordance. We also plan to test our approach on common datasets such as the Pascal VOC Challenge or the Cityscapes Dataset [20]. As the use of ReLU neurons appeared to pose problems to the combination of weight learning (as in *KGC-weighting* or AUXNET) and the Adam optimizer, a next logical step would be to test other activation functions for the DCNN, such as leaky ReLUs. Lastly, we are planning to investigate further why training on the pickup error was unsuccessful, and aim to modify the function such that it still reflects pickup probabilities, but can also be used as a better scaled version of the distance error for KGC-weighting.

References

[1] J. Schmidhuber, "Deep Learning in Neural Networks: An Overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.

[2] H. Lasi, P. Fettke, H.-G. Kemper, T. Feld, and M. Hoffmann, "Industry 4.0," *Business & Information Systems Engineering*, vol. 6, no. 4, p. 239, 2014.

[3] G. K. Kraetzschmar, N. Hochgeschwender, W. Nowak, F. Hegger, S. Schneider, R. Dwiputra, J. Berghofer, and R. Bischoff, "RoboCup@Work: Competing for the Factory of the Future," in *RoboCup 2014: Robot World Cup XVIII*, ser. Lecture Notes in Computer Science, R. A. C. Bianchi, H. L. Akin, S. Ramamoorthy, and K. Sugiura, Eds. Springer International Publishing, 2015, vol. 8992, pp. 171–182.

[4] Y. Guo, Y. Liu, A. Oerlemans, S. Lao, S. Wu, and M. S. Lew, "Deep Learning for Visual Understanding: A Review," *Neurocomputing*, vol. 187, pp. 27–48, 2016.

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *IEEE Conference on Computer Vision and Pattern Recognition, 2009.* IEEE, 2009, pp. 248–255.

[6] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The Pascal Visual Object Classes (VOC) Challenge," *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, 2010.

[7] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, 2017.

[8] J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* IEEE, 2017, pp. 6517–6525.

[9] J. Long, E. Shelhamer, and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3431–3440.

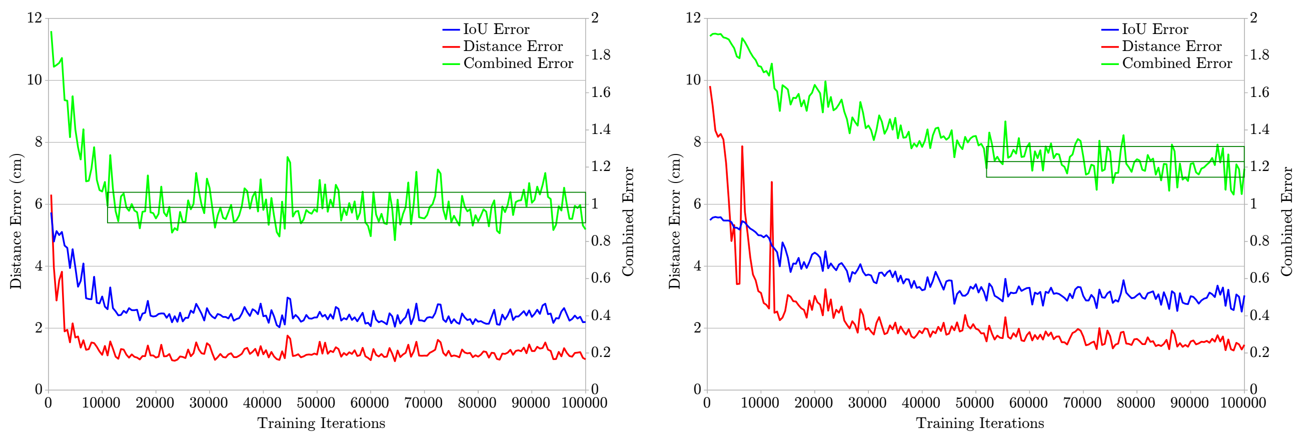


Fig. 14. Convergence plots of the fastest (left) and slowest (right) converging networks trained using AUXNET. Distance error (red) is converted to a pickup error and added to the IoU error (blue) to form the combined error (green). The dark green box indicates the converged region, its average error, and standard deviation. The distance error plot on the right side shows that the network suffered traumatic experiences during early training, from which it could never fully recover, resulting in overall higher error and slower convergence.

Learning method	Distance Error	IoU Error	Distance+IoU	KGC	$KGC_{+\epsilon}$	$KGC_{/Mean}$	AUXNET
Distance Error		✓	✓	—	✓	—	—
IoU Error	✓		✓	✓	✓	✓	✓
Distance+IoU	✓	✓		✓	✓	✓	✓
KGC	—	✓	✓		✓	—	✓
$KGC_{+\epsilon}$	✓	✓	✓	✓		✓	—
$KGC_{/Mean}$	—	✓	✓	—	✓		✓
AUXNET	—	✓	✓	✓	—	✓	

TABLE III

CHECKMARKS INDICATING WHETHER H_0 COULD BE REJECTED BY A TWO-SAMPLE KS-TEST, $\alpha = 0.05$. ALL NETWORKS WERE COMPARED WITH RESPECT TO THEIR CONVERGENCE. WITH FEW EXCEPTIONS, MOST DIFFERENCES ARE SIGNIFICANT, MOST IMPORTANTLY, BETWEEN KGC AND AUXNET.

[10] R. Caruana, "Multitask Learning," in *Learning to Learn*. Springer, 1998, pp. 95–133.

[11] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. Lecun, "Overfeat: Integrated Recognition, Localization and Detection using Convolutional Networks," in *International Conference on Learning Representations (ICLR2014), CBLS, April 2014*, 2014.

[12] R. Girshick, "Fast R-CNN," in *2015 IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2015, pp. 1440–1448.

[13] A. Kendall, Y. Gal, and R. Cipolla, "Multi-Task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[14] J. Zhang, G. Tian, Y. Mu, and W. Fan, "Supervised Deep Learning with Auxiliary Networks," in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2014, pp. 353–361.

[15] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[16] A. Rahman and Y. Wang, "Optimizing Intersection-Over-Union in Deep Neural Networks for Image Segmentation," in *International Symposium on Visual Computing*. Springer, 2016, pp. 234–244.

[17] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *Proceedings of the International Conference on Learning Representations (ICLR), San Diego, 2015*, 2015.

[18] B. Broecker, D. Claes, J. Fossel, and K. Tuyls, "Winning the RoboCup@Work 2014 Competition: The smARTLab Approach," in *RoboCup 2014: Robot World Cup XVIII*. Springer, 2014, pp. 142–154.

[19] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier Nonlinearities Improve Neural Network Acoustic Models," in *ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, vol. 30, no. 1, 2013, pp. 3–8.

[20] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The Cityscapes Dataset for Semantic Urban Scene Understanding," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 3213–3223.