Learning to detect robots from artificial images

1st Christoph Heindl
Visual Computing
PROFACTOR GmbH
4407 Steyr, Austria
christoph.heindl@profactor.at

2nd Sebastian Zambal
Machine Vision
PROFACTOR GmbH
4407 Steyr, Austria
sebastian.zambal@profactor.at

6th Josef Scharinger
Institute of Computational Perception
Johannes Kepler University
4040 Linz
josef.scharinger@jku.at

***Abstract*—Impressive results have been achieved in computer vision via machine learning methods over the last years. However, for many highly specialized industrial applications these methods cannot directly be applied due to the lack of large amounts of training data. In this paper we propose a software framework that provides artificial data. Such artificial data opens the door for machine learning even when it is hard or impossible to acquire real data. The presented system is flexible in the sense that it supports image data generation in a flexible way. We show how artificial images can directly be used in pytorch to perform deep neural network training.**

***Index Terms*—machine learning, artificial data, data augmentation**

I. Introduction and Related Work

The impressive results of supervised deep learning in the field of robotics and computer vision are to large extent due to the availability of annotated data sets. Up until recently most of the annotation was carried out by domain experts. This time consuming process significantly slows down the progress of deep learning efforts. In many tasks it is difficult, in most niche areas impossible, to obtain strong supervision because of the high costs.

Exposed to this bottleneck, new fields of research have opened up. Active learning [1]–[3] attempts to use human experts in a more targeted way by focusing on samples that seem to of highest value for learning task. Semi-supervised learning [4]–[6] combines large unlabeled data sets with smaller labeled data sets through structural assumptions, such as smoothness or low-dimensionality constraints. Transfer learning [7]–[9] exploits the fact that models trained on specific tasks can be adapted to related problems using fewer training samples. Weak supervision [10], [11] is motivated by leveraging less precise, higher level supervision that is often easier to obtain. This includes heuristics, weak or biased classifiers, unreliable non-experts.

In contrast to the methods mentioned above, synthetic data generation drops the necessity for real world annotated data in favor of artificial data sources. Synthesized data is available in virtually infinite variety and is de-facto auto-labelled. This has led to variety of successful applications in different domains: Jaderberg et al. [12] have demonstrated natural scene text recognition from synthetic data, Peng et al. [13] trained object detectors from 3D models and Johnson et al. [14] used photo-realistic computer rendering captured from a game engine to train car detectors.

A common theme to all of these synthetic approaches is the following: (a) a simulator engine generators artificial training data, (b) the generated data is stored to disk and (c) the offline data stack is used for training. The shortcomings to this procedure are early training data commitment and the lack of adaptability of data generation to training demands.

In this paper we propose method of generating artificial training data from a simulator engine that is directly looped into the training procedure, thereby avoiding early data commitment. A bi-directional communication channel enables adaptive random data generation. We demonstrate the usefulness of our approach to the detection of robot poses in color images trained from non-realistic computer generated images.

II. Method

Our approach is outlined in Figure 1. Without loss of generality we consider a supervised regression task. One or more simulator engines generate training tuples $\{(\mathbf{x}, y, \mathbf{h})\} \sim P(\mathbf{X}, Y, \mathbf{H}; \theta)$ by sampling from a probabilistic graphical model parametrized by θ. The training samples $\{(\mathbf{x}, y)\}$ are used for mini-batch updating the parameters ϕ of a neural network $P(Y|\mathbf{X}; \phi)$. Finally, the simulation parameters θ are adapted via a feedback signal emitted from control unit.

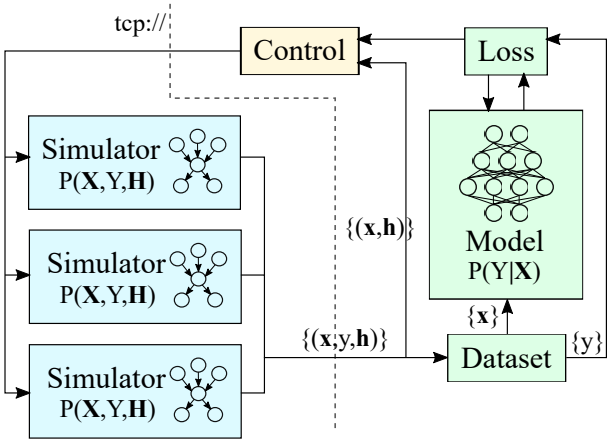


Fig. 1: Architecture of our system.

Identify applicable funding agency here. If none, delete this.

A. Artificial data and data loaders

TODO: Outline Python-support of Blender, Explain scripting in Blender, creation of meta-data (joint locations, etc.)... TODO: Explain basic concepts of PyTorch and DataLoaders, etc. TODO: Explain ZMQ pipeline, publish/subscribe

B. Localization Model and Training

Joint localization deals with the task of estimating positions of articulated body parts that constitute kinematic chains. In this work, we define joint localization to be the process of predicting 2D pixel coordinates of robotic revolute joints from color image input.

Our model, outlined in Figure 2, is based on the architecture proposed by Heindl et al. [15], which in turn is an extension to methods developed for human pose estimation [16], [17]. . Instead of predicting sparse joint pixel coordinates, we train the network to predict dense belief maps per joint $\hat{\mathbf{B}} \in \mathbb{R}^{J \times H' \times W'}$ in the following way. Color images $\mathbf{I} \in \mathbb{R}^{3 \times H \times W}$ are input to a pre-trained VGG network [18] to extract base features $\mathbf{F} \in \mathbb{R}^{C \times H' \times W'}$. A series of convolutional image-to-image networks refines belief predictions by allowing each network to work on the basis of outputs from the previous network and VGG base features.

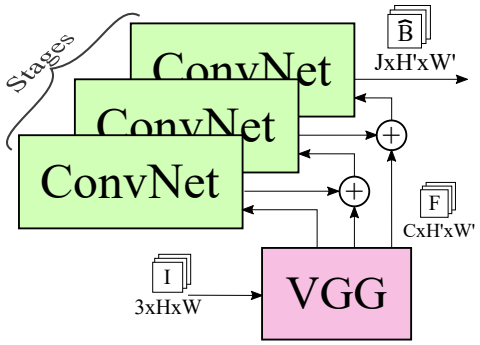


Fig. 2: Architecture of the localization module. Each stage predicts joint belief maps. Successive stages are fed a channel concatenated stack of VGG features and the current estimate.

We generate target belief maps $\mathbf{B}$ by pixel-wise maximization over the contributions from J squared exponential kernels

$$k_{\text{SE}}(\mathbf{x}; \mathbf{z}) = \sigma^2 e^{-\frac{\|\mathbf{x}-\mathbf{z}\|_2^2}{l^2}} \tag{1}$$

centered at sparse image joint locations $\{\mathbf{z}_0, \ldots, \mathbf{z}_J\}$. The output variance σ^2 determines the average distance away from its mean, the length-scale l determines the width of the kernel. During training, a objective functions after each stage penalizes pixel-wise differences between belief predictions $\hat{\mathbf{B}}$ and targets $\mathbf{B}$. Usually, L_2 or smooth L_1 loss functions are used. These intermediate losses help the network to train more effectively, as the effect of vanishing gradients in deep architectures is minimized [17], [19].

When predicting from the network only the output of the last stage is used. For partial scale invariance of image features, we additionally average the belief predictions of input images scaled to different resolutions.

C. Feedback Mechanism

A key idea to our work is a feedback mechanism that enables adaptive sampling from simulator engines. As outlined in Figure 1, a controller updates the simulation based on information of the current training mini-batch. The update can be in principle of any kind, however, in this work we focus on modifying probability distributions of the graphical model controlling the simulation process. In particular we modify distributions of hidden variables in such a way that more examples $\{(\mathbf{x}, y, \mathbf{h})\}$ of high value to the training are produced. Once a subset of important examples $\mathcal{S} = \{(\mathbf{x}, y, \mathbf{h})\}$ from the training batch has been selected, we infer $P(\mathbf{H}, Y | \mathbf{X}; \mathcal{S})$ (approximately, analytically or in a heuristic fashion) and continue sampling from the updated distribution.

We propose two ways to measure the value of specific samples for training. The first utilizes the loss function to assign more importance to examples of high error. This trivially leads to updated sampling strategies putting more weight into regions of high training error. The second method uses prediction uncertainty to select sample candidates. In contrast to Bayesian learning for neural networks [20], deterministic regression networks do not have any meaningful measure of uncertainty build in or are not well calibrated for uncertainty estimation (in the classification case) [21]. One way to overcome this limitation is perform approximate Bayesian uncertainty estimation as proposed in Gal et al. [22]. Samples selected by uncertainty are intrinsically determined by the model without consulting a loss function.

D. Implementation Notes

We use Blender [23] as a simulator engine for robotic scenes. Blender offers great scripting support, that allows us to hook into the rendering process. Our localization model is defined and trained using PyTorch [24]. As communication layer we use the distributed messaging library ZeroMQ [25]. In particular, publisher/subscriber patterns are used for broadcasting simulation results to the training loop. We limit the number of outstanding messages at the subscriber to prevent the training loop from running out of memory. The control signal is based a simple request/reply pattern. As outlined in figure 1, TCP/IP is used as transport layer.

III. Results

In the following we describe the training procedure, evaluation of our method on synthetic and real world data, and provide runtime metrics.

A. Training

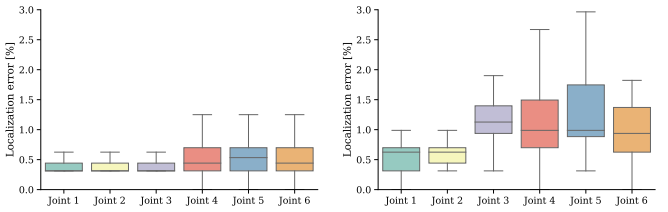
For training the localization model we sampled a set of 10.000 samples per epoch from two simulator engines. The first simulator generated uniform random poses, while the second simulator produced importance samples as described in Section II-C according to a loss metric. Two distinct collections of random background images (city and industrial theme) were used as synthetic training- and test-set backgrounds. We trained the joint localization model using six stages with input

images of size 320×240. A VGG network, pre-trained on detection, was used to generate base features. We used Adam [26], $\eta = 1 \times 10^{-3}$, optimization with mini-batch learning for a total of 30 epochs.

B. Evaluation

We challenge joint localization model trained on artificial images on synthetic and real world image. For comparison we extract joint locations from prediction belief maps $\hat{\mathbf{B}}$ by performing smoothed non-maximum suppression, followed by extracting the maximum peak per joint channel. For real-world images we asked a human export to annotate the joint locations. Synthetic images are auto-labelled by our simulation engine. We define the localization error to be the Euclidean distance between predicted and target pixel locations and report it in percent of image diagonal.

Figure 3 shows the localization error on two distinct test sets: synthetic with random backgrounds (Figure 3a) and real-world images (Figure 3b). As expected our system performs better on synthetic data than real data.



(a) Localization errors of joint prediction on a synthetic test data set with auto-labelled ground truth.

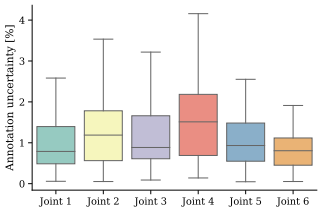
(b) Localization errors of joint prediction for a real world test set. Target locations annotated by a human domain expert.

Fig. 3: Localization errors of robot joint prediction. Results are in percent of image diagonal.

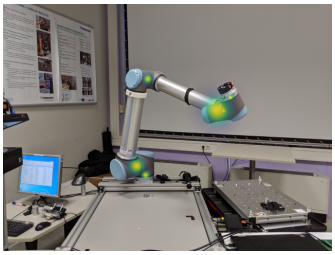
We attribute two effects for this outcome: (a) our simulation is non-realistic and (b) human annotation is error-prone. To underpin the latter issue, we have conducted a pilot study in which 10 people (all with reference to robotics) annotated 12 real world images of an UR-10 robot in different poses and from different viewpoints. We than computed the inter-rater spread as measure of confidence for human annotations in our domain. Figure 4a shows the average uncertainty per joint over all images. We find that the middle joints, often blocked by view and robot pose, are the most difficult to annotate precisely. These results indicate that our method performs better than indicated by figure 3b, because this figure does not take into account human annotation errors.

C. Effects of Feedback

We evaluate the effectiveness of our feedback mechanism to generate adapted samples in the following way: a training is run twice, once without feedback enabled and one time with adaptive feedback for 30 epochs each. Each run spawns two simulator engines. In the feedback enabled run, a high L_2 loss indicates examples that should be resampled. We



(a) Human uncertainty estimates in pose annotation. Results are in percent of image diagonal.



(b) Uncertainty estimates superimposed as Gaussian kernels on one of the images to be annotated.

Fig. 4: Uncertainty estimates from human annotation. Results are in percent of image diagonal.

then adapt the prior distributions over joint angles of the second simulator to generate more samples in close vicinity of important joint angle configurations. To avoid too heavy exploitation behavior in training, the generated samples are interleaved at the subscriber. This ensure half of the training examples come from each of the simulators. Figure 5 shows the beneficial effects of our approach. Note, we only change the distribution of joint angles and leave all distributions unchanged. As described in Section II-C, one should ideally update $P(\mathbf{H}, Y|\mathbf{X}; \mathcal{S})$ for all hidden variables.

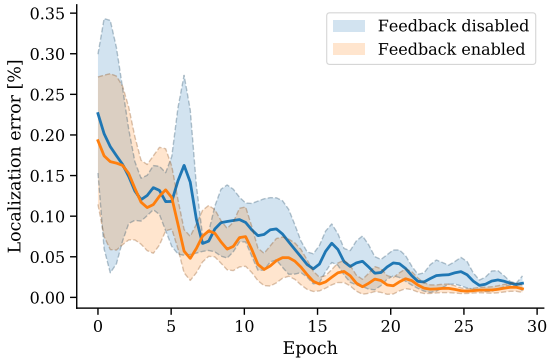


Fig. 5: Effects of applying adaptive simulator feedback comparing two trainings over 30 epochs. We see faster convergence if we let one of the simulator instances generate more examples in the neighborhood of training samples with high errors. Errors are in percent of image diagonal.

D. Runtime Analysis

All experiments are carried out on a server instance containing 2x Intel Xeon E5-2650v4 12-Core and 4 NVIDIA Tesla V100 SXM2 32 GB. Table I summarizes the individual runtimes of different parts of our system. The duration per training step is longer when using 8 simulation instances rendering a simple robot scene in 320×240. The bottleneck in training remains the forward/backward passes of the neural network.

Task	Mean [ms]	Std [ms]
Simulation step	250.32	10.21
Batch Collating (8/8)	321.23	24.76
Train step	521.23	73.76
Prediction	95.21	14.32

TABLE I: Runtimes of different parts of our pipeline for 320×240 input images. Timings in milliseconds. Batch Collating (8/8) refers to batch-size 8 with 8 simulator instances.

ACKNOWLEDGMENT

Work presented in this paper has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 721362 (project "ZAero").

REFERENCES

[1] G. Druck, B. Settles, and A. McCallum, "Active learning by labeling features," in *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1-Volume 1*, pp. 81–90, Association for Computational Linguistics, 2009.

[2] B. Settles, "Active learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 6, no. 1, pp. 1–114, 2012.

[3] M. Cakmak and A. L. Thomaz, "Designing robot learners that ask good questions," in *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction*, pp. 17–24, ACM, 2012.

[4] O. Chapelle, B. Scholkopf, and A. Zien, "Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]," *IEEE Transactions on Neural Networks*, vol. 20, no. 3, pp. 542–542, 2009.

[5] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," in *Advances in Neural Information Processing Systems*, pp. 2234–2242, 2016.

[6] X. Zhu, "Semi-supervised learning literature survey," *Computer Science, University of Wisconsin-Madison*, vol. 2, no. 3, p. 4, 2006.

[7] L. Y. Pratt, "Discriminability-based transfer between neural networks," in *Advances in neural information processing systems*, pp. 204–211, 1993.

[8] D. Ventura and S. Warnick, "A theoretical foundation for inductive transfer," *Brigham Young University, College of Physical and Mathematical Sciences*, 2007.

[9] S. J. Pan, Q. Yang, *et al.*, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.

[10] A. J. Ratner, C. M. De Sa, S. Wu, D. Selsam, and C. Ré, "Data programming: Creating large training sets, quickly," in *Advances in neural information processing systems*, pp. 3567–3575, 2016.

[11] Z.-H. Zhou, "A brief introduction to weakly supervised learning," *National Science Review*, vol. 5, no. 1, pp. 44–53, 2017.

[12] M. Jaderberg, K. Simonyan, A. Vedaldi, and A. Zisserman, "Synthetic data and artificial neural networks for natural scene text recognition," *arXiv preprint arXiv:1406.2227*, 2014.

[13] X. Peng, B. Sun, K. Ali, and K. Saenko, "Learning deep object detectors from 3d models," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1278–1286, 2015.

[14] M. Johnson-Roberson, C. Barto, R. Mehta, S. N. Sridhar, K. Rosaen, and R. Vasudevan, "Driving in the matrix: Can virtual worlds replace human-generated annotations for real world tasks?," in *2017 IEEE International Conference on Robotics and Automation, ICRA 2017, Singapore, Singapore, May 29 - June 3, 2017*, pp. 746–753, 2017.

[15] C. Heindl, S. Zambal, T. Pönitz, A. Pichler, and J. Scharinger, "3d robot pose estimation from 2d images." Submitted to Digital Image and Signal Processing 2019.

[16] Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh, "Realtime multi-person 2d pose estimation using part affinity fields," in *CVPR*, vol. 1, p. 7, 2017.

[17] A. Newell, K. Yang, and J. Deng, "Stacked hourglass networks for human pose estimation," in *European Conference on Computer Vision*, pp. 483–499, Springer, 2016.

[18] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

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

[20] R. M. Neal, *Bayesian learning for neural networks*, vol. 118. Springer Science & Business Media, 2012.

[21] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks," *arXiv preprint arXiv:1706.04599*, 2017.

[22] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation," *arXiv preprint arXiv:1506.02157*, 2015.

[23] Blender Online Community, *Blender - a 3D modelling and rendering package*. Blender Foundation, Blender Institute, Amsterdam, 2019.

[24] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[25] P. Hintjens, "ZeroMQ: The Guide," 2010.

[26] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.