Experience with Using Synthetic Training Images for Wearable Cognitive Assistance

Roger Iyengar, Emily Zhang, Mahadev Satyanarayanan
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213
{raiyenga, satya}@cs.cmu.edu

Abstract

Wearable Cognitive Assistance (WCA) applications use computer vision models that require thousands of labeled training images. Capturing and labeling these images requires a substantial amount of work. By using synthetically generated images for training, we avoided this labor-intensive step. The performance of these models was comparable to that of models that were trained on real images.

Introduction

Wearable Cognitive Assistance (WCA) applications are a new class of cyber-human systems that leverage machine learning models, mobile devices, and edge computing (Ha et al. 2014; Chen et al. 2017; Wang et al. 2019; Pham et al. 2021). These applications guide users step-by-step through complex tasks with visual and audio guidance, just as turn-by-turn GPS navigation systems guide drivers. Using computer vision, the application determines when a step has been completed correctly, and then gives the user the next instruction. Images are captured using the camera on a head-mounted wearable device or a smartphone. These images are transmitted over a wireless network to a cloudlet (i.e., a server with close network proximity), where they are analyzed by computer vision models to determine task progress.

Most of the models used by WCA applications are deep neural networks (DNNs). Training these DNNs requires thousands of labeled images. Capturing and labeling these images requires substantial effort. Bounding boxes must be drawn around the region of each image that contains the object being assembled. The bounding box must then be labeled with the step of the assembly process that is depicted in the image. Collecting and labeling a training set of images is a major barrier to entry for anyone who wants to develop a WCA application for a new task. For example, this effort took over 50 person hours for a WCA application that could recognize 17 different states in the assembly of an IKEA cart. This time-consuming and labor-intensive aspect of WCA is the biggest bottleneck to its widespread adoption.

Previous papers have proposed the use of synthetically generated images for training sets. In this approach, pre-labeling is done by construction (Hinterstoisser et al. 2019b; Tremblay et al. 2018b; Gupta, Vedaldi, and Zisserman 2016; Hinterstoisser et al. 2019a; Rajpura, Hegde, and Bojinov 2017; Tremblay et al. 2018a; Dwibedi, Misra, and Hebert 2017). Since programs that generate synthetic images have information about the objects that are visible and their locations, there is no need for manual input of this information. In addition, synthetic images of objects can easily be rendered in a wide variety of different lighting conditions and environments. In contrast, capturing real images of objects in a variety of conditions requires the images of the object to be captured in every such environment. Overall, the use of synthetic data may save considerable manual effort.

In this paper, we examine whether synthetically generated images are an adequate substitute for real images in training sets. More specifically, we ask how well synthetic images work for creating a WCA application for guiding the assembly of a particular Meccano Erector kit. We use the following procedure to answer this question. We first generate a set of synthetic (pre-labeled) images using the Unity Perception package (Borkman et al. 2021), and then train computer vision models on this data. Next, we collect and manually label a set of real images for the same task, and then train computer vision models on this data. Finally, we compare the accuracy of these two families of models on a held-out test set of real images. Our results show that models created with a training set size of 75,000 synthetic images perform slightly better than models created with roughly 15,000 real images. However, this ordering is reversed when fewer synthetic images are used for training.

Caution is advisable in generalizing our results. They pertain to one data point: a specific example (Meccano Erector kit) of a specific class of AI tasks (WCA). Yet, it is also true that WCA is representative of an emerging class of “AI in the wild” applications. Our positive experience with synthetic training data suggests that more extensive efforts to investigate their use in real-world AI is warranted.

Background & Related Work

Wearable Cognitive Assistance

A WCA application provides just-in-time guidance and error detection for a user who is performing an unfamiliar task. Prompt error detection is also valuable for a user who is performing familiar tasks, since human errors cannot be com-

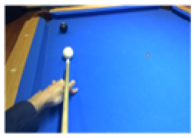
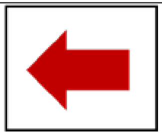
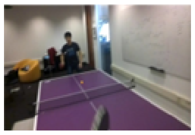
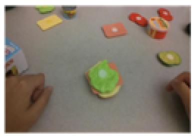
App Name	Example Input Video Frame	Description	Symbolic Representation	Example Guidance
Pool		Helps a novice pool player aim correctly. Gives continuous visual feedback (left arrow, right arrow, or thumbs up) as the user turns his cue stick. The symbolic representation describes the positions of the balls, target pocket, and the top and bottom of cue stick.	<Pocket, object ball, cue ball, cue top, cue bottom>	
Ping-pong		Tells novice to hit ball to the left or right, depending on which is more likely to beat opponent. Uses color, line and optical-flow based motion detection to detect ball, table, and opponent. Video URL: *https://youtu.be/_lp32sowyUA*	<InRally, ball position, opponent position>	"Left!"
Work-out		Counts out repetitions in physical exercises. Classification is done using Volumetric Template Matching on a 10-15 frame video segment. A poorly-performed repetition is classified as a distinct type of exercise (e.g. "good pushup" versus "bad pushup").	<Action, count>	"8"
Face		Jogs your memory on a familiar face whose name you cannot recall. Detects and extracts a tightly-cropped image of each face, and then applies a state-of-art face recognizer. Whispers the name of the person recognized.	ASCII text of name	"Barack Obama"
Lego		Guides a user in assembling 2D Lego models. The symbolic representation is a matrix representing color for each brick. Video URL: *https://youtu.be/7L9U-n29abg*	[[0, 2, 1, 1], [0, 2, 1, 6], [2, 2, 2, 2]]	"Put a 1x3 green piece on top"
Draw		Helps a user to sketch better. Builds on third-party app for desktops. Our implementation preserves the back-end logic. A Glass-based front-end allows a user to use any drawing surface and instrument. Displays the error alignment in sketch on Glass. Video URL: *https://youtu.be/nuQpPtVJC6o*		
Sand-wich		Helps a cooking novice prepare sandwiches according to a recipe. Since real food is perishable, we use a food toy with plastic ingredients. Object detection uses Faster-RCNN deep neural net approach. (Ren et al. 2015) Video URL: *https://youtu.be/USakPP45WvM*	Object: "E.g. Lettuce on top of ham and bread"	"Put a piece of bread on the lettuce"

Table 1: Example Wearable Cognitive Assistance Applications. The input frame is sent to a cloudlet, which converts it to the symbolic representation. The symbolic representation is then used to give guidance. (Source: Adapted from Satyanarayanan (Satyanarayanan 2017))

pletely avoided, especially when the user is tired or stressed. Informally, WCA is like having "an angel on your shoulder." (Ha et al. 2014) It broadens, the metaphor of GPS navigation tools that provide real-time step-by-step guidance, with prompt error detection and correction.

Table 1 lists some examples of WCA applications that we have developed. In total, we have developed over 15 WCA applications. This paper focuses on applications that that help users assemble physical objects. We have developed such applications for an IKEA lamp, a Lego kit, a toy sandwich, an IKEA cart, and a Stirling engine.

These applications are a compelling use case for edge computing. They first capture images using the camera on a mobile device, such as a smartphone or head-mounted wearable device. The images are then sent to a cloudlet for processing, using the Gabriel platform (Ha et al. 2014). The computational limitations of lightweight mobile devices that have acceptable battery life prevent us from being able to process these images using the devices' own hardware (Satyanarayanan et al. 2009). Table 2 lists the resource consumption and end-to-end latency bounds of five offloading-based WCA applications. It shows that WCA applications are simultaneously compute-intensive, bandwidth-hungry, and latency-sensitive.

	Pool	Ping-pong	Face	Lego	Sandwich
Cloudlet CPU load(%)	72.10	45.40	75.60	52.20	85.10
End-to-end latency bounds (tight–loose, ms)*	95–105	150–230	370–1000	600–2700	
Video streaming bandwidth requirement (Average / Peak, Mbps)	480p: 3.6 / 7.0 720p: 6.8 / 9.9 1080p: 8.1 / 12.7				

Table 2: CPU load, latency bounds, and the required bandwidth of example WCA applications. The implementations of the application servers (Chen et al. 2017) were tested on a laptop (with an Intel® Core™ i7-8500Y processor and 8GB RAM), running the frontend on an Android phone, using 480p, 720p, and 1080p video resolutions. (*) End-to-end latency includes both the round trip time (RTT) from the Android phone to the cloudlet and compute time in the cloudlet. Tight and loose bounds are adopted from Chen, et al, (Chen et al. 2017) where they are defined: "The tight bound represents an ideal target, below which the user is insensitive to improvements. Above the loose bound, the user becomes aware of slowness, and user experience and performance is significantly impacted."

WCA Tools

Ha et al. (2014) introduced the first version of a programming framework called Gabriel. This framework includes networking and runtime components for WCA applications. Chen et al. (2017) developed an initial set of WCA applications, determined how much latency was acceptable for these applications, and examined how changes to the network, hardware, and algorithms used can affect end to end latency. Wang et al. (2019) examined how to reduce the load imposed on a cloudlet by a single WCA user, thereby allowing many more users to share single cloudlet. Pham et al. (2021) developed a toolchain that allows people to develop WCA applications without writing any code.

Synthetic Training Data

The idea of avoiding manual labeling has a long and rich history. Hinterstoisser et al. (2019b) trained an object detector on synthetic data that outperformed an object detector trained on real data. They generated backgrounds cluttered with distractor objects. In addition, they added some distractor objects to the foreground and varied the lighting conditions that were used to render each of the images. These images looked 3D, but they were not photo-realistic. Other works have generated photo-realistic images to use as training data (Tremblay et al. 2018b; Gupta, Vedaldi, and Zisserman 2016), or used real background images (Hinterstoisser et al. 2019a; Rajpura, Hegde, and Bojinov 2017; Tremblay et al. 2018a). Dwibedi, Misra, and Hebert (2017) avoided rendering 3D graphics altogether by cropping objects from photographs, and pasting these crops into other photographs. Their models trained on synthetic images performed worse than their models trained on real images. However, they also trained models using a mix of real and synthetic images, and these performed better than models trained on real or synthetic data alone.

Application

We developed a WCA application that helps users assemble a model bike, using parts from a Meccano Erector kit. The fully assembled model is depicted in Figure 1. It is made from over 50 parts. We trained models to recognize five steps of the task for this work.

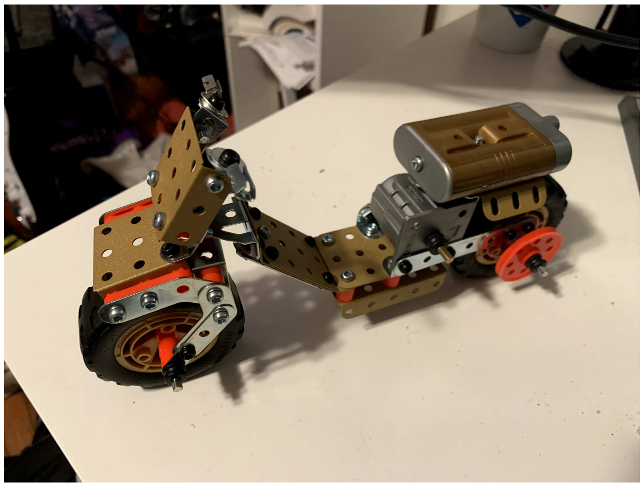


Figure 1: The fully assembled model bike. Our application guides users through the steps required to assemble this.

Our application determines which step of the assembly task is shown in the camera feed. This is accomplished using a two stage process inspired by Gebru et al. (2017). The first stage involves finding the region of the image that contains the assembly that a user is working on, using Faster R-CNN (Ren et al. 2015). Next, the image is cropped around this region, and the cropped region is classified using Fast MPN-COV (Li et al. 2018). The Fast MPN-COV model has five possible outputs, one for each step of the task that our application recognizes. The classification result therefore indicates the step of the task that is shown in an image. The architecture for this application is shown in Figure 2.

Generating Data

We found a CAD model for the Meccano kit on the community website GrabCAD[1]. This CAD model appears to have been created to replicate the physical Meccano pieces, rather than being the same model that was used to manufacture the pieces. In particular, we noticed a number of differences between the CAD model and the actual Meccano parts. We

[1]https://grabcad.com/library/meccano-9550-002-1

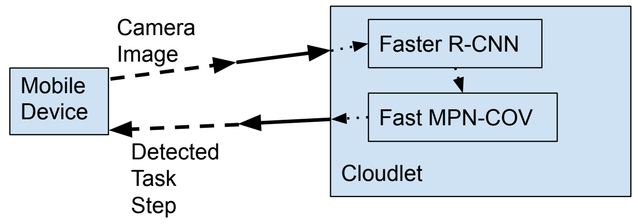


Figure 2: The architecture of our WCA application for the Meccano erector kit. The dashed lines represent a Wi-Fi connection. The solid lines represent a connection over Gigabit Lan. The dotted lines represent data transmission between components on a single cloudlet.

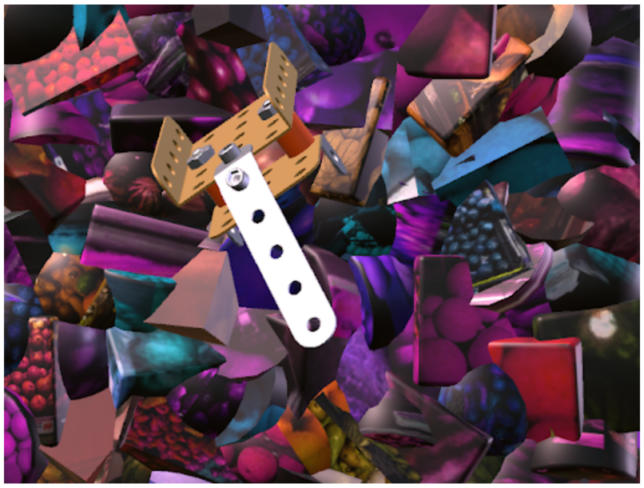


Figure 3: A synthetic image showing part of the bike model. The background is filled with distractor objects that the network should learn not to identify.

selected textures for each part of the model, trying to match the appearance of the physical object as closely as possible.

We generated synthetic images using the Unity Perception Package (Borkman et al. 2021). The default setup for this package fills the background of the images that are generated with objects that the network should learn to ignore. Figure 3 shows an image generated using this default setup.

We trained a Faster R-CNN object detector using this data. Training was done using the TensorFlow object detection API. The initial weights came from the TensorFlow model zoo, for a model pre-trained on the Microsoft COCO dataset (Lin et al. 2014). We ran transfer learning using the default hyperparameters from the TensorFlow model zoo. The Unity package creates a file with bounding box and label information, and we converted this to the format used by the TensorFlow API. The perception package drew bounding boxes tightly around the objects. We added padding to these bounding boxes, to make them more like our hand-drawn labels (which also had some padding).

Unfortunately the training process for this model did not converge. We attempted to fix this issue by removing the background objects from the image, and then we tried to

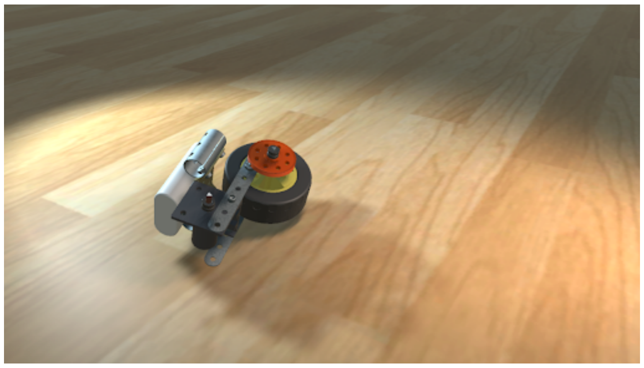


Figure 4: Our first attempt at making our synthetic images look more realistic. This image is meant to look like an object sitting on a wood floor.

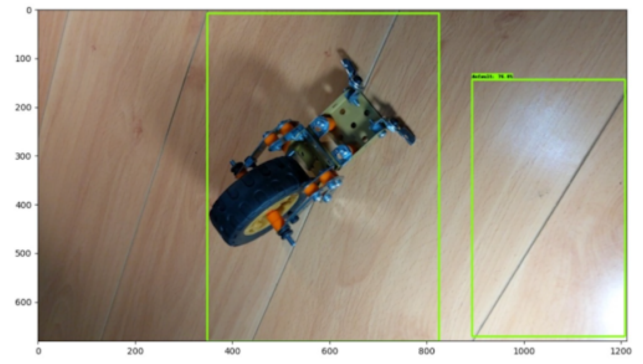


Figure 5: A case where our model incorrectly detected a line in the floor as an object of interest. The green bounding boxes are regions of the image that the model detected an object in.

make the objects look like they were sitting on a wooden floor. We accomplished this by placing the object at the bottom of the 3D scene in Unity and texturing the floor of the scene with an image of wood from Adobe's collection of stock images. Figure 4 shows one of these images.

The Faster R-CNN model trained on this data converged; however, it performed poorly. One issue that we noticed was the object detector mistakenly detected lines in the wood floor as being a model bike assembly. Figure 5 shows an example of such an erroneous detection.

We were able to correct this issue by using additional background textures and randomizing the lighting in the scene and the position of the camera. We have posted our code[2]. Figure 6 shows some examples of this data.

Results

We trained computer vision model pairs to be used in the pipeline described in the Application section. A model pair consists of a Faster R-CNN object detector and a Fast MPN-COV classifier created using the same training set. All of

[2]https://github.com/exiaohuaz/data-gen

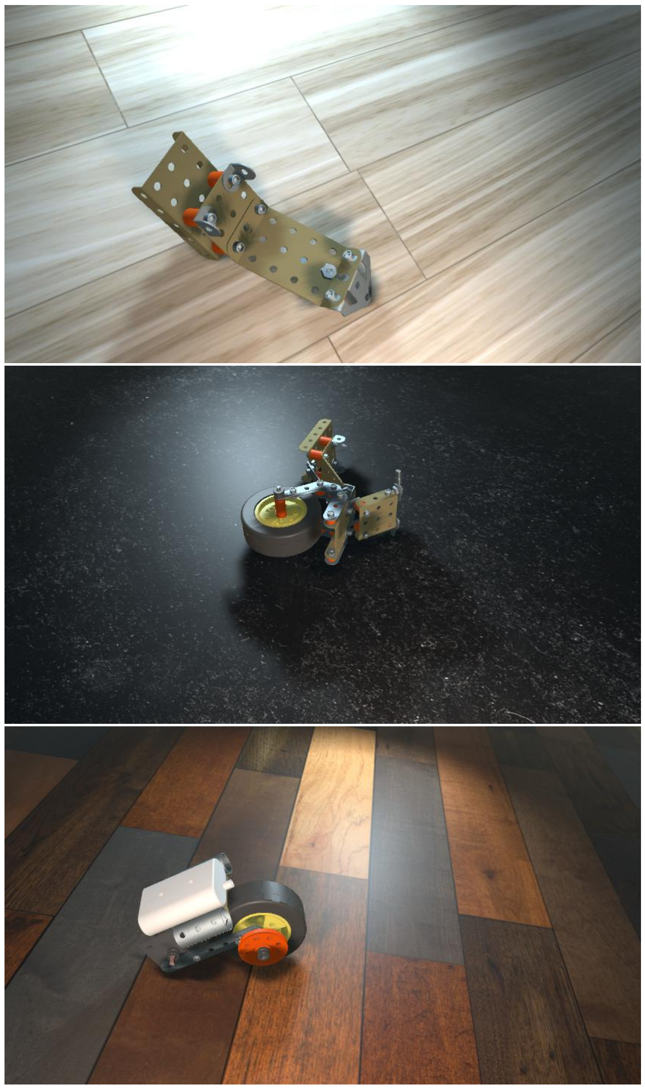


Figure 6: Synthetically generated images from our final set. The models trained on this data performed well.

our training and testing data relates to uncluttered environments with good lighting. We assume that a human using a WCA application can correct environmental issues to reduce classification complexity. For example, the user can increase the amount of light shining on an assembly, or remove clutter from the background. Assuming optimal environmental conditions for a WCA assembly task is thus reasonable.

We trained one model pair on real data that was manually labeled with bounding boxes and class labels (15,477 images). The remaining model pairs were trained on synthetic data sets of varying size (12,000, 25,000, 50,000 and 75,000 images). The labels for these images were generated by Unity. We compared the accuracy of these model pairs.

Our test set consists of 4490 real images that are not included in any training set. Table 3 presents our results. We observe that the model trained on real data performs better than the models trained on synthetic datasets with

Dataset Type	Training Set Size	Accuracy
Synthetic	12,000	69.6%
Synthetic	25,000	79%
Synthetic	50,000	84.1%
Synthetic	75,000	89%
Real	15,477	84.5%

Table 3: Classification results for our model pairs. Accuracy is the percentage of our 4490 test images that the pipeline of models classified correctly.

12,000, 25,000, and 50,000 images. However, this relationship is reversed for a model trained on 75,000 synthetic images. Somewhere between 50,000 and 75,000 images lies the cross-over point at which the increased number of synthetic images more than compensates for their lower realism.

Conclusion

Capturing and labeling training images for WCA applications is a time consuming process. Using synthetic images is less labor-intensive, and would simplify development of WCA applications. We have achieved promising results with this approach for one WCA application. Broader validation would expand the approach to other WCA assembly tasks, and include a wide range of different lighting conditions. Such broader validation would help us to better understand the robustness of this approach.

Better CAD models would also help. Our CAD model had some differences from the physical parts that we were trying to detect and classify. Using the actual CAD models used to manufacture the parts would minimize these differences.

One limitation of this work is that we are just considering models for WCA applications for assembly tasks. All of our images had good lighting, and they only contained the object that was being assembled. WCA applications for other purposes, such as repairing equipment, have to work in conditions with bad lighting and cluttered backgrounds. Training models that perform well in these conditions would likely require a more complicated process for generating synthetic images. We also did not try modifying the training configurations for any of our models, or comparing any alternative neural architectures.

Lastly, a CAD model can be used to manufacture objects out of different materials. Someone generating synthetic images from a CAD model will likely have to specify the materials that a certain object was manufactured from, and they might have to modify textures to match surfaces that result from manufacturing processes. For example, a certain type of machine might use a blade that creates a bumpy surface on a steel part. Using a 3D scanner might be a way to eliminate the need to manually specify the textures for parts. Unfortunately, 3D scanners are not common, so many people generating synthetic data will likely have to manually specify the textures of parts.

Our results for the one task we looked at offer the tantalizing promise that using synthetic data might eliminate the need to manually capture and label images in order to develop WCA applications for assembly tasks. The application described in this work represents an emerging class of applications that leverage edge computing. We see a lot of value in reducing the difficulty of developing such applications.

Acknowledgments

This research was supported by the National Science Foundation (NSF) under grant number CNS-2106862. Roger Iyengar was supported by an NSF Graduate Research Fellowship under Grants DGE1252522 and DGE1745016. Additional support was provided by AutoDesk, CableLabs, Crown Castle, Deutsche Telekom, Intel, InterDigital, Microsoft, Seagate, VMware, Vodafone, and the Conklin Kistler family fund. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the view(s) of their employers or the above funding sources.

References

Borkman, S.; Crespi, A.; Dhakad, S.; Ganguly, S.; Hogins, J.; Jhang, Y.; Kamalzadeh, M.; Li, B.; Leal, S.; Parisi, P.; Romero, C.; Smith, W.; Thaman, A.; Warren, S.; and Yadav, N. 2021. Unity Perception: Generate Synthetic Data for Computer Vision. *CoRR*, abs/2107.04259.

Chen, Z.; Hu, W.; Wang, J.; Zhao, S.; Amos, B.; Wu, G.; Ha, K.; Elgazzar, K.; Pillai, P.; Klatzky, R.; Siewiorek, D.; and Satyanarayanan, M. 2017. An Empirical Study of Latency in an Emerging Class of Edge Computing Applications for Wearable Cognitive Assistance. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, SEC '17. New York, NY, USA: Association for Computing Machinery. ISBN 9781450350877.

Dwibedi, D.; Misra, I.; and Hebert, M. 2017. Cut, Paste and Learn: Surprisingly Easy Synthesis for Instance Detection. In *2017 IEEE International Conference on Computer Vision (ICCV)*, 1310–1319.

Gebru, T.; Krause, J.; Wang, Y.; Chen, D.; Deng, J.; and Fei-Fei, L. 2017. Fine-Grained Car Detection for Visual Census Estimation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 31(1).

Gupta, A.; Vedaldi, A.; and Zisserman, A. 2016. Synthetic Data for Text Localisation in Natural Images. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2315–2324.

Ha, K.; Chen, Z.; Hu, W.; Richter, W.; Pillai, P.; and Satyanarayanan, M. 2014. Towards Wearable Cognitive Assistance. In *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '14, 68–81. New York, NY, USA: Association for Computing Machinery. ISBN 9781450327930.

Hinterstoisser, S.; Lepetit, V.; Wohlhart, P.; and Konolige, K. 2019a. On Pre-trained Image Features and Synthetic Images for Deep Learning. In Leal-Taixé, L.; and Roth, S., eds., *Computer Vision – ECCV 2018 Workshops*, 682–697. Cham: Springer International Publishing. ISBN 978-3-030-11009-3.

Hinterstoisser, S.; Pauly, O.; Heibel, H.; Martina, M.; and Bokeloh, M. 2019b. An Annotation Saved is an Annotation Earned: Using Fully Synthetic Training for Object Detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*.

Li, P.; Xie, J.; Wang, Q.; and Gao, Z. 2018. Towards Faster Training of Global Covariance Pooling Networks by Iterative Matrix Square Root Normalization. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*.

Lin, T.-Y.; Maire, M.; Belongie, S.; Hays, J.; Perona, P.; Ramanan, D.; Dollar, P.; and Zitnick, L. 2014. Microsoft COCO: Common Objects in Context. In *ECCV*. European Conference on Computer Vision.

Pham, T. A.; Wang, J.; Iyengar, R.; Xiao, Y.; Pillai, P.; Klatzky, R.; and Satyanarayanan, M. 2021. Ajalon: Simplifying the authoring of wearable cognitive assistants. *Software: Practice and Experience*, 51(8): 1773–1797.

Rajpura, P. S.; Hegde, R. S.; and Bojinov, H. 2017. Object Detection Using Deep CNNs Trained on Synthetic Images. *CoRR*, abs/1706.06782.

Ren, S.; He, K.; Girshick, R.; and Sun, J. 2015. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In Cortes, C.; Lawrence, N.; Lee, D.; Sugiyama, M.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc.

Satyanarayanan, M. 2017. The Emergence of Edge Computing. *IEEE Computer*, 50(1).

Satyanarayanan, M.; Bahl, P.; Caceres, R.; and Davies, N. 2009. The Case for VM-Based Cloudlets in Mobile Computing. *IEEE Pervasive Computing*, 8(4).

Tremblay, J.; Prakash, A.; Acuna, D.; Brophy, M.; Jampani, V.; Anil, C.; To, T.; Cameracci, E.; Boochoon, S.; and Birchfield, S. 2018a. Training Deep Networks with Synthetic Data: Bridging the Reality Gap by Domain Randomization. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 1082–10828.

Tremblay, J.; To, T.; Sundaralingam, B.; Xiang, Y.; Fox, D.; and Birchfield, S. 2018b. Deep Object Pose Estimation for Semantic Robotic Grasping of Household Objects. *CoRR*, abs/1809.10790.

Wang, J.; Feng, Z.; George, S.; Iyengar, R.; Pillai, P.; and Satyanarayanan, M. 2019. Towards Scalable Edge-Native Applications. In *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, SEC '19, 152–165. New York, NY, USA: Association for Computing Machinery. ISBN 9781450367332.