

# **tinyHulk: Lightweight Annotation for Wearable Cognitive Assistance**

**Chanh Nguyen      Roger Iyengar      Qifei Dong**  
**Jim Blakley      Mahadev Satyanarayanan**

February 2023  
CMU-CS-23-314

School of Computer Science  
Carnegie Mellon University  
Pittsburgh, PA 15213

## **Abstract**

Wearable Cognitive Assistance (WCA) employs DNNs models to provide real time step-by-step guidance to end-users for specific tasks. Collecting and annotating training data for such DNNs is painful and takes massive human effort and time. In this work, we propose tinyHulk, an automatic annotation tool that reduces human time and effort for these labor-intensive annotation tasks. Further, the tool is equipped with the *background replacement*, an *object-aware* data augmentation mechanism to increase the size and diversity of the training set. Our experimental results show that using tinyHulk can save human time up to fourfold compared to using manual methods for labeling tasks. The computer vision models trained with data annotated and generated by tinyHulk achieve high accuracy, outperforming the models trained with real or synthetic data using conventional methods for annotation.

**Keywords:** wearable cognitive assistance, auto labeling, annotation tool, DNNs, computer vision, chroma keying, data augmentation

# 1 Introduction

Wearable Cognitive Assistance (WCA) [20] is a new emerging category of cyber-human applications that integrate wearable devices with computer vision and edge computing to augment human cognition. They provide real time step-by-step guidance and verify each single step has been performed correctly. Although the concept is nearly two decades old [44, 45], such applications have been attainable only recent years with the convergence of the three technology pillars [20]: (a) the accuracy of deep neural networks (DNNs) for computer vision;(b) the availability of reliable computing infrastructure for offloading compute-intensive operations from mobile devices (i.e., Edge Computing); and (c) the availability of wearable hardware (e.g., Smart Glasses).

The CMU Living Edge Lab has designed and built an open-source software platform, the Gabriel platform<sup>1</sup>, to simplify the development of WCA applications [20]. Gabriel factors out complex system-level functionality that is common across many applications, including network communication and data pre-processing [8]. The workflow of a WCA application built with Gabriel platform is shown in Figure 1. Sensor data (i.e., video frames) captured by an end-user’s wearable device is first encoded and compressed before transmission to a cloudlet – a server within a close network proximity – over a wireless connection. The sensor streams are further processed on the cloudlet by a collection of cognitive modules that employ compute-intensive computer vision models such as an object detector and a classifier. Cognitive module outputs are integrated by a task-specific user guidance module that performs higher-level cognitive processing. The application-specific code in the user guidance module typically consists of a task state extractor and a guidance generator. Based on the mapping between the user’s current progress and the output of the cognitive modules, the guidance generator triggers the task-appropriate visual, verbal, or tactile guidance which is then transmitted back to the end-user’s wearable device. Since the Gabriel back-end embodies all task-specific components, applications can be easily ported to different wearable devices such as Google Glass, Microsoft Hololens, etc. Thanks to the Gabriel platform, we have been exploring wearable cognitive assistance and built close to 20 applications of this genre, from a simple LEGO assembly, to a complex IKEA kit assembly<sup>2</sup>.

As discussed earlier, the core of a WCA application is the deep neural network (DNN) that detects objects captured in video frames and classifies a task state. Therefore, the accuracy of these computer vision models plays a crucial role in WCA’s success. Each specific WCA application employs a distinct DNN. For example, the DNN of the WCA assembly application for a toy car detects and classifies different car parts (wheel, car base, etc.), while the DNN of the WCA assembly application for a lego set detects and classifies various lego components. Training these DNNs requires a large number of ground-truth images, i.e., each image in the dataset is carefully annotated with a bounding box or a polygon and labeled to identify the object of interest. The current method of annotating such datasets is primarily manual. For example, the computer vision annotation tool (CVAT) [11] can be used to annotate objects of interest in each frame of an input video, which requires significant human time and effort. In one case, using CVAT to annotate an image dataset for a WCA application that could recognize 17 different states in the assembly of an IKEA

---

<sup>1</sup><https://github.com/cmusatyalab/gabriel>

<sup>2</sup><https://www.cmu.edu/scs/edgecomputing/resources/videos.html>

cart took over 50 person hours [26]. This extremely time-consuming and labor-intensive task is undoubtedly a bottleneck in the WCA development process. High-quality annotated datasets are generated with significant effort. They require proper tools to conduct a precise data annotation to ensure the highest quality at an affordable price. In most WCA assembly systems, the objectives are specific, i.e., focusing on a single object of interest and a single state of interest captured in each frame fed by the user’s device.

In this work we propose *tinyHulk*, an automatic image annotation tool to annotate image datasets for training the DNNs of WCA assembly systems. To create training sets, the user first records a video of the object of interest on a green<sup>3</sup> background surface. With this video input, the tool automatically draws a 2D bounding-box to precisely cover the object of interest in each video frame. For supervision purposes, the tool provides an interactive window, and functionality to let the user inspect and modify the annotation.

Furthermore, we develop an *image augmentation* method inspired by the *chroma keying* technology [3] in the film industry to replace the green background in each frame with new backgrounds. With this object-aware augmentation, we can increase the image training set’s size and diversity as many times as desired in just a few seconds.

We use the data annotated and generated by *tinyHulk* to train the DNNs computer vision models for two WCA assembly applications, respectively: assembling a Meccano bike, and assembling a toy plane. Our experimental results show that **annotating the image dataset using *tinyHulk* can be up to four times as fast as using CVAT**. Further, the computer vision models trained with the data annotated and generated by our tool **outperform the corresponding models trained with manually annotated data, and is competitive with models trained with a massive amount of synthetic data**.

The **key contributions** of this work are summarized as follow:

- We proposed *tinyHulk*, an automatic image annotation tool to annotate image datasets for the purpose of training the DNNs of WCA assembly systems (Section 3.1, 3.2, 3.3).
- We develop an object-aware image augmentation method to replace the green background of images with different backgrounds to increase the amount and diversity of the training set (see Section 3.4).
- We evaluate and validate the efficiency and efficacy of *tinyHulk* with respect to human time savings and the performance of the DNNs models trained (see Section 4 and 5).

## 2 Background and Related Work

In this section, we first present some ground truth (image) data generation approaches that can be applied to generate a training set for computer vision models in a WCA application. After that, we

---

<sup>3</sup>It is worth noting that our tool can work with backgrounds of any uniform and distinct color, as long as no part of the photographed objects may duplicate the color used as the background. For the sake of presentation, throughout this paper we use “*green background*” as a use case.

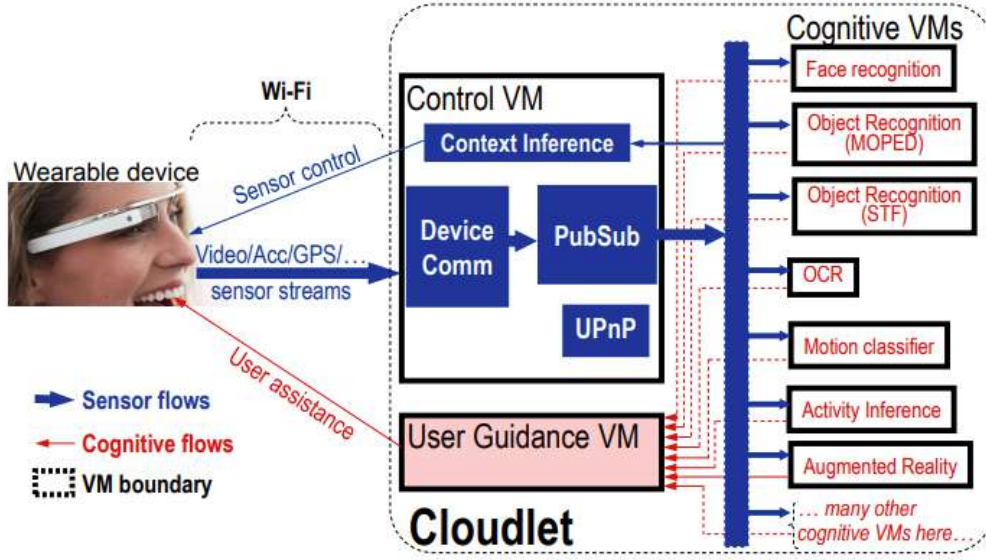


Figure 1: The WCA application architecture developed with the Gabriel platform. Reference source [20].

present some data augmentation approaches that can help increase the original training set in both size and diversity.

## 2.1 Labeled Training Data Generation

Training data generation is the workhorse behind the development cycle of machine learning algorithms and directly impacts algorithmic performance. The higher the data quality, the more accurate the trained machine learning models will be. We classify these ground truth data generation approaches into three different categories, i.e., manual, automatic, and synthetic.

**Manual Data Annotation.** The most popular image annotation tool for computer vision models is Computer Vision Annotation Tool (CVAT) [11], a free open-source annotation tool. It provides a web-based UI to let users label image and video data for many computer vision tasks such as object detection and image segmentation. With many useful features (e.g., interpolation, multi-annotation shapes, etc.), CVAT has been a common tool for manual labeling tasks. The alternates include the VGG Image Annotation Tool (VIA) [14], Label Studio [28], and COCO Annotator [6]. Although these tools provide for convenient annotation, users have to interact with the tool and manually annotate the image data, which takes a massive amount of human effort and time. Further, human variability, for example, non-uniformity in hand-drawn bounding boxes may confuse training.

**Automatic Data Annotation.** Many advanced computer vision techniques have been explored

to reduce the annotation effort. Hakan et al. [4] proposed a Weakly Supervised Deep Detection Networks (WSDDN) to perform simultaneously region selection and classification. First, a pre-trained CNN is trained on the ImageNet data [42]. After that, the last pooling layer in the last convolutional block of the pre-trained network is modified to transform it into a WSDDN to reason explicitly and efficiently about image regions. The experimental results show that the resulting object detectors are weak compared to ones trained on human annotated bounding boxes [29]. The work from Dong et al. [13] introduce Multi-modal Self-Paced Learning for Detection (MSPLD) framework that focuses on getting the most out of as little training data as possible. First, they use a few annotated bounding boxes per class (1% of the images in the entire dataset to be annotated) to train an object detector. Then, they embed multiple detection models in a unified learning scheme to improve model performance. Although the framework outperforms the state-of-the-art weak supervised object detection approaches, it cannot reach the performance of the detectors trained with human labels. Further, such learning based approaches that require annotated data beforehand, do not guarantee to scale to different data domains.

**Synthetic Data Generation.** The idea of using synthetic data to train computer vision models has been long investigated. Hinterstoisser et al. [23, 24] trained an object detector on synthetic data that outperformed an object detector trained on real data. First, they yielded background images with realistic shapes and texture, and then they rendered the objects of interest on top of the new background images. These images looked 3D, but they were not photo-realistic. Dwibedi et al. [16] avoided rendering 3D graphics altogether by cropping objects from photographs and pasting these crops into new background images. They trained models using a mix of real and synthetic images, which resulted in a better-performed model than the models trained with real or synthetic data alone. For the purpose of training computer vision models for WCA applications, Iyengar et al. [26] generated a set of synthetic (pre-labeled) images using the Unity Perception package [5]. After that, they use the synthetic images to train computer vision models for a WCA assembly application. The performance of these models was comparable to that of models trained on real images. Although this method saves human time for the labeling task, it requires as a prerequisite a 3D model of the object and good skill with rendering software.

## 2.2 Image augmentation

Wang et al. [38] states that:

*“...It is common knowledge that the more data an ML algorithm has access to, the more effective it can be. Even when the data is of lower quality, algorithms can actually perform better, as long as useful data can be extracted by the model from the original data set...”*

In fact, the statement above has proven to be true [34, 10, 56]. As a result, data augmentation techniques have been investigated and become an effective technique to increase both the amount and diversity of data by randomly *augmenting* it [10], which significantly improve the performance of the trained ML models [54, 19]. A few standard data augmentation methods are distortions, scale,

translation, and rotation [43, 46, 52]. These methods are supported by many machine learning frameworks<sup>4</sup>.

An *object-aware* data augmentation method has been investigated recently, namely the *Copy-Paste augmentation* [16, 19] – randomly pasting objects to new background images. This method has the potential to create novel training datasets for free. In essence, the step by step copy-paste augmentation, as described in [16], includes: 1) Collect object instance images which cover diverse viewpoints and have a modest background; 2) Collect scene images to be the new backgrounds; 3) Predict foreground mask for the object; and 4) Paste object instances on a randomly chosen background image. As such, to segregate each image pixel into foreground/background, this augmentation method requires a foreground/background segmentation model, for example, a fully convolutional network (FCN) based model [32]. The output of the segmentation model is then processed through an edge-aware smoothing step using the *fast bilateral solver* [1] to obtain the final object masks. The experimental result shows that, the models trained with the new data generated by the copy-paste augmentation approach achieve comparable performance with the strong baselines [15, 16].

In the following section, we will present our proposed annotation tool, namely tinyHulk. The tool automatically annotates image data, resulting in annotation results that are uniformly neat and precisely cover the object of interest without taking much human time and effort. We also extend the tool with a simple yet efficient object-aware image augmentation to help increase the training set in both volume and diversity.

### 3 tinyHulk - An Automatic Image Annotation Tool

The workflow of tinyHulk is presented in Figure 2. It is developed using the Open Computer Vision libraries (OpenCV) [37] for computer vision algorithms and the wxPython libraries for the graphical user interface (GUI)<sup>5</sup>. The tool is available online and can be found at the project page<sup>6</sup>. In essence, the main components of the proposed annotation tool include: 1) video parsing and duplicate removal; 2) bounding-box drawing; 3) background replacement; 4) annotation result inspection and modification window; and 5) labeled data generation. In the following sections, we will present technical details of each component. To create training set for a WCA, first we record a two-minute average length video that captures the final result of the step on a green background surface table. After that, each video is input to tinyHulk to generate the annotated image training sets

#### 3.1 Video Parsing and Duplicate Removal

As its name suggests, this component parses input videos to frames, which then become the input for the following annotation process. Input video must be recorded where the object of interest is solely placed on a uniform single-colored background, for example a green backdrop. Extracted

---

<sup>4</sup>[https://www.tensorflow.org/tutorials/images/data\\_augmentation](https://www.tensorflow.org/tutorials/images/data_augmentation)

<sup>5</sup><https://www.wxpython.org/>

<sup>6</sup><https://github.com/cmusatyalab>

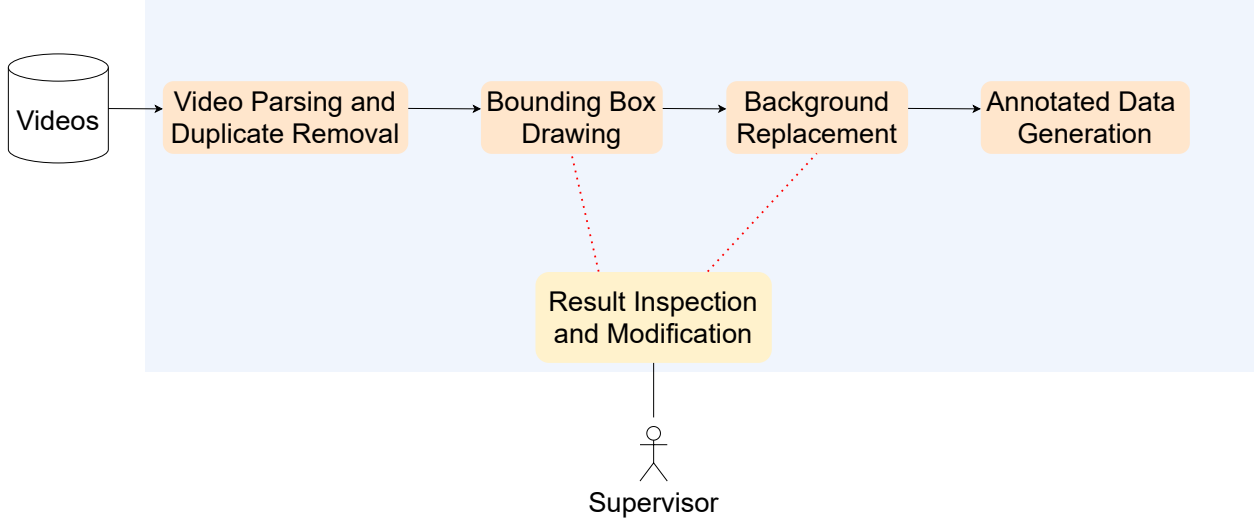


Figure 2: The workflow of tinyHulk.

frame sets from a video may involve duplication for many reasons. For example, the camera may have been accidentally moving slowly, which resulted in capturing multiple frames from the same viewpoint. Studies from the data cleaning research domain show that the existence of duplicates hurts the learning result [2, 55, 12]. Duplicates in the training set cause the trained model to learn *biases* towards the patterns specific to duplicated images (i.e., the duplicated images receives disproportionate weight during training), resulting in a model that *lacks the ability to generalize to unseen images*. Furthermore, the duplicates also add unnecessary costs (e.g., computation, storage) for data processing and increase the time and cost of training the DNNs models. Hence, it is important to eliminate such duplicates in the dataset. Many proposed approaches for removing the duplicates, for example, leverage deep learning [25, 33], or use feature matching techniques [7, 41]. While these approaches are proven to be robust, it requires significant compute resources and time for training the DNNs, which is inappropriate for a lightweight tool.

There is a fast and robust algorithm that is widely used by many industry practitioners (e.g., Google, Yandex, TinEye) [21] to detect image similarities for digital forensics, namely *perceptual hashing* [27, 35]. In essence, the perceptual hash algorithm is driven by the fingerprinting technique, which converts an image to a fingerprinting character string (e.g., a 64-bit integer hash value) and uses the *Hamming distance* algorithm [36] to calculate the similarity between the two images. To this end, we implement the perceptual hash algorithm for detecting duplicates existing in the extracted frame sets. We set a similarity threshold  $\delta$  as a cutoff for determining whether two frames are duplicated. Figure 3 presents an example where the first column presents the frames of interest, and the second column shows their duplicates found and eliminated by the tool (with  $\delta = 1$ ).



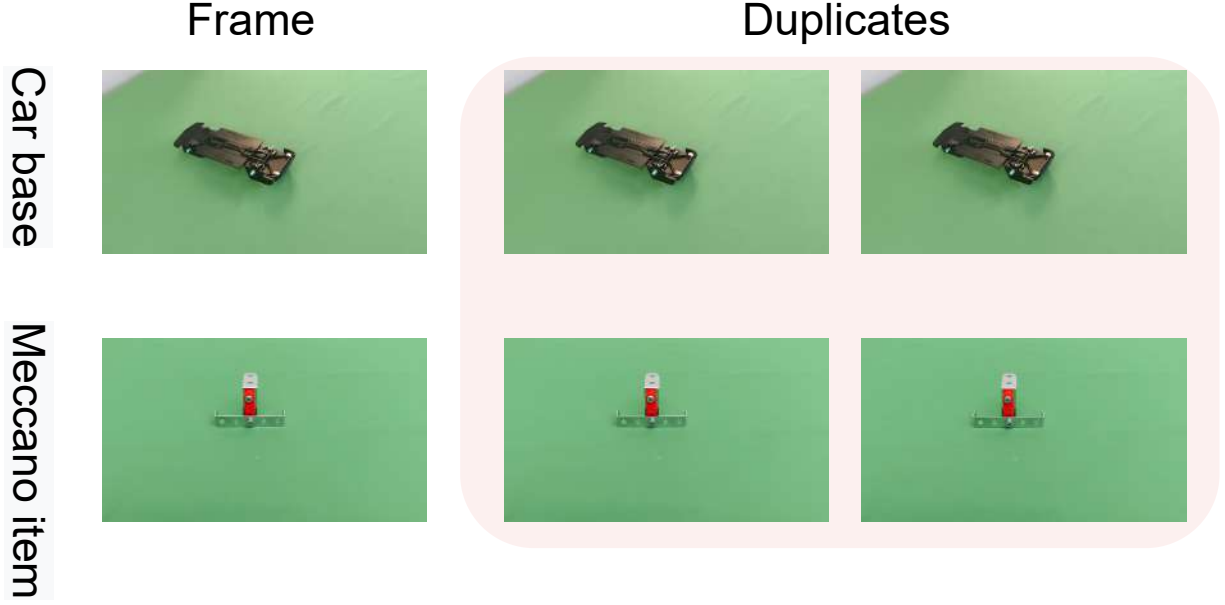


Figure 3: Example of duplicated frames found by the tool. The first row presents frames capturing the car base object; the second row shows frames capturing the Meccano item. The similarity threshold value is equal to 1.

### 3.2 Bounding-Box Drawing

After extracting and cleaning the frame sets from input videos, tinyHulk automatically annotates each frame in the dataset. The purpose of image annotation is to locate the object of interest in each image, which helps the machine learning algorithms recognize an annotated area as a distinct object or class in a given image. The most common types of image annotations are 2D Bounding Boxes, Polygonal Segmentation, Lines and Splines, Semantic Segmentation, and 3D Cuboids [49]. In the case of WCA assembly system, the annotation result does not need to be an exact outline of the object of interest in each frame (such as using precise complicated polygons to draw polygonal boundaries around the object), however it is highly required to be precise and neatly covering the object of interest. Therefore, we develop the tool to annotate frames with a 2D bounding-box. To draw a bounding-box covering the object of interest in each frame, our idea is as follow:

1. First, we employ the *adaptive threshold* technique [22, 53] to **find contours** that cover the object of interest in the input frame. Differing from the global threshold technique that defines only one global threshold for the whole image, the adaptive threshold technique determines a threshold for image's pixel based on a small region and hence different thresholds for different areas in the image. This advantage gives better results for images with varying illumination.
2. After finding all possible contours in the frame, we **remove the noise contours** (i.e, the ones are too small or in the edge of the frame).



Figure 4: Example of bounding box drawn by the tool.

3. Finally, we **extract and return the coordinates** ( $x_{min}, y_{min}, x_{max}, y_{max}$ ) **of the bounding rectangle of the final contour** (i.e., the one that covers all the valid contours).

In the end, all information of each frame (filename, height, width, class, and bounding-box coordinates) in the input video is saved to a .csv-format file. We implement the whole concept using the OpenCV library, as mentioned previously. Figure 4 shows examples of the bounding boxes drawn by the tool with inputs are frames of various objects.

### 3.3 Result Inspection and Modification

tinyHulk provides an interactive window to let the user inspect the annotation result returned. From the window, the user first selects a .csv file capturing all annotation results returned by the tool. From that, the user can browse frames and their corresponding bounding-box with a preferred browsing speed (i.e., frames per second), and modify a bounding-box (or delete the current displayed frame) if needed. To avoid missing correcting incorrect bounding boxes when browsing at a high speed, the browsing progress automatically stops at the frame suspected to be a wrong bounding box until the user confirms or corrects it. Technically, we implement this anomaly bounding-box detection mechanism using the global and local outlier detection that employs the *interquartile range* technique [51] on the bounding-box’s area and the bounding-box’s width and height. If the following frame is similar to the previous modified frame, and has an incorrect bounding-box, then its bounding-box is also automatically updated with the bounding-box information of that previous modified frame. This automatic correction saves the user time in modifying consecutive similar frames. Finally, the modified data is updated to the final annotation result (i.e., updated to the selected .csv file).

Figure 5 show the interface and different functionalities of the interactive window that the user can interact with.

### 3.4 Background Replacement

To increase the size and diversity of the original training set, we aim to develop and equip the tool with image augmentation that can change the original frame’s monotonous background to



a) Correct bounding-box

b) Suspected incorrect bounding-box

c) Suspected incorrect bounding-box

Figure 5: The inspection window. The user browses the bounding-box results with a preferred frame rate. a) shows a browsing in progress; b) and c) show browsing is in pause because the tool suspects the bounding-box is incorrect. The user must either confirm that the bounding-box is correct (click on the button Slide Show or Next) or rectify it otherwise.

different backgrounds. The copy-paste augmentation procedure [16, 19] – randomly pasting objects to new background images – has been proven to be efficient and can provide solid gains on top of strong baselines [15, 16]. However, the high computational requirement (to train a foreground/background segmentation model) prevents the copy-paste augmentation method from being used for the implementation of the background replacement component of a lightweight annotation tool.

On the contrary, **Chroma keying** [3] – a well-known background replacement technique used popularly in the film industry – does not require much computation. Therefore, to implement the background replacement component in our annotation tool, we apply the conventional chroma keying method with thresholding from human visual perception in HSV (Hue, Saturation, Value) color space. We use HSV instead of the RGB (Red, Green, Blue) color model because HSV is closer to how a human perceives color [47, 48]. The HSV space stores color information in a cylindrical representation of RGB color points, in which the Hue value varies from 0-179, the Saturation value ranges from 0-255, and the Value value ranges from 0-255. In essence, the background replacement process is as below:

- 1). Collect new background images (e.g., a wooden background, a working place background, etc.).
- 2). From an interactive window, the user chooses a combination of HSV threshold values (low and high values of Hue, Saturation, and Value, respectively) for a sample frame so that the object of interest is blended on the new background image, see Figure 6. The threshold should be chosen precisely so that the object of interest is visible on the new background and the noise is minimized.
- 3). All frames in the frameset have their green background replaced automatically by the new background with the selected threshold. All new frames are saved to a separated folder under the .jpg format. The new generated frame keeps the same bounding-box coordinate values as the cor-

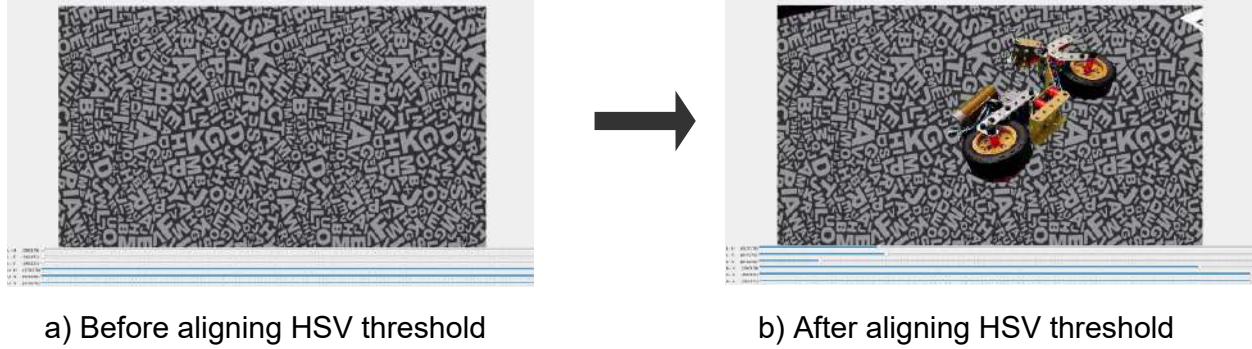


Figure 6: Example of background replacement by the tool. a) presents the window with the new background before tuning HSV threshold; b) presents the window with the object of interest (the meccano bike) being visible on the new background after tuning the HSV threshold (L-H = 47, L-S = 78, L-V = 21, U-H = 169, U-S = 255, U-V = 255).

responding original frame. In the end, all information for the newly generated frame (filename, height, width, class, and bounding-box coordinates) is saved to a *.csv*-format file.

With this mechanism, the original dataset size can be increased by as many folds as desired just in a few seconds (see Section 5 for the execution time of the background replacement component).

### 3.5 Annotated Data Generation

The final step is generating the training data under a specific format for training DNNs models. After human inspection and correction of the annotation results, the tool creates the training dataset under different data formats respecting the input requirement of various models, for example Pascal VOC [17], COCO format [9](*.json* file), or Tensorflow *tfrecord* [50].

In Section 4, we present the experiments that use the *tfrecord* output by tinyHulk as the input to the Tensorflow Framework to train different computer vision models for a WCA application.

## 4 Experiment

This section describes the experimental setting in which we use tinyHulk to annotate training datasets for the computer vision models of two WCA applications. Also, we present two other ways to collect and generate training datasets, i.e., synthetic data generation and real data, for the two models that serve as the baseline models for performance evaluation.

### 4.1 Experimental setting

We create two WCA assembly applications: (a) assembling a Meccano bike, and (b) assembling a toy plane. The step-by-step assembly of each application is shown in Figure 7.

The application must determine the current step of the assembly task shown in a camera feed and guide the end-user stepwise to completion of the assembly task. We use a two-stage vision process inspired by Gebru et al.[18]. For each image frame fed from the end-user’s camera, the first step is finding the region of the image that contains the assembly that the end-user is working on. To do so, we built an object detector for each WCA using Faster R-CNN (Region-Based Convolutional Neural Network) model [39]. Then the image is cropped around this region. Finally, the cropped image is classified using the Fast MPN-COV (Matrix Power Normalized Covariance pooling) ConvNet [30]. The Fast MPN-COV model has one output label for each step of the assembly task (i.e., 5 possible outputs for the Meccano bike (Figure 7(a)) and 10 possible outputs for the toy plane (Figure 7(b)) in our experiments). The classification result therefore indicates the task step that is shown in an image.

We use tinyHulk to generate training datasets for these models as follows. First, we record a two-minute average length video that captures the final result of the step on a green background surface table (48"  $\times$  30" rectangle table) as shown in Figure 8. After that, each video is input to tinyHulk to generate the annotated image training sets. The annotation tool runs on a desktop machine with CPU Intel Xeon(R) W-2195 CPU 2.30GHz, and 62GB RAM using a single thread. Further, to generate new-background datasets from the original green-background frame set, we first collect new scene images, such as a wooden background, a tile background, a workplace, etc. After that, we run the background replacement component over the original frame set with the scene images chosen randomly to double the size of the original training set with the new generated data. Figure 9 shows some examples of background replacement from the original training sets of the two applications (i.e., toy plane in Figure 9 (a) and Meccano bike in Figure 9 (b)).

## 4.2 Models trained with real data and synthetic data

In order to have a baseline to evaluate and compare the performance of the trained models, we also build the corresponding models trained with synthetic data and real data. The data generation and annotation for these two benchmark models are described as below:

- **Synthetic data generation:** Following the same approach as described in [26], we generated synthetic image datasets for the two applications respectively. We took the CAD models for the Meccano bike and the toy plane on GrabCAD<sup>7</sup> and Cults3D<sup>8</sup>. By rearranging the components of the CAD models using Autodesk Fusion 360, we generated one 3D model file in Filmbox (.fbx) format for each step in the assembly task. Using the Unity Perception Package [5], we placed the 3D model files into a virtual scene one at a time to create synthetic images. We applied lighting, background texture, camera position, and object orientation randomization to introduce more variations into the output data. Finally, the generated images and the metadata files containing the labels and bounding box information are combined into tfrecord files for model training. Figure 10 shows some examples in the synthetic training sets of the two WCA applications.

---

<sup>7</sup><https://grabcad.com/library/meccano-9550-002-1>

<sup>8</sup><https://cults3d.com/en/3d-model/game/toy-plane-assembled-by-bolts-and-nuts>

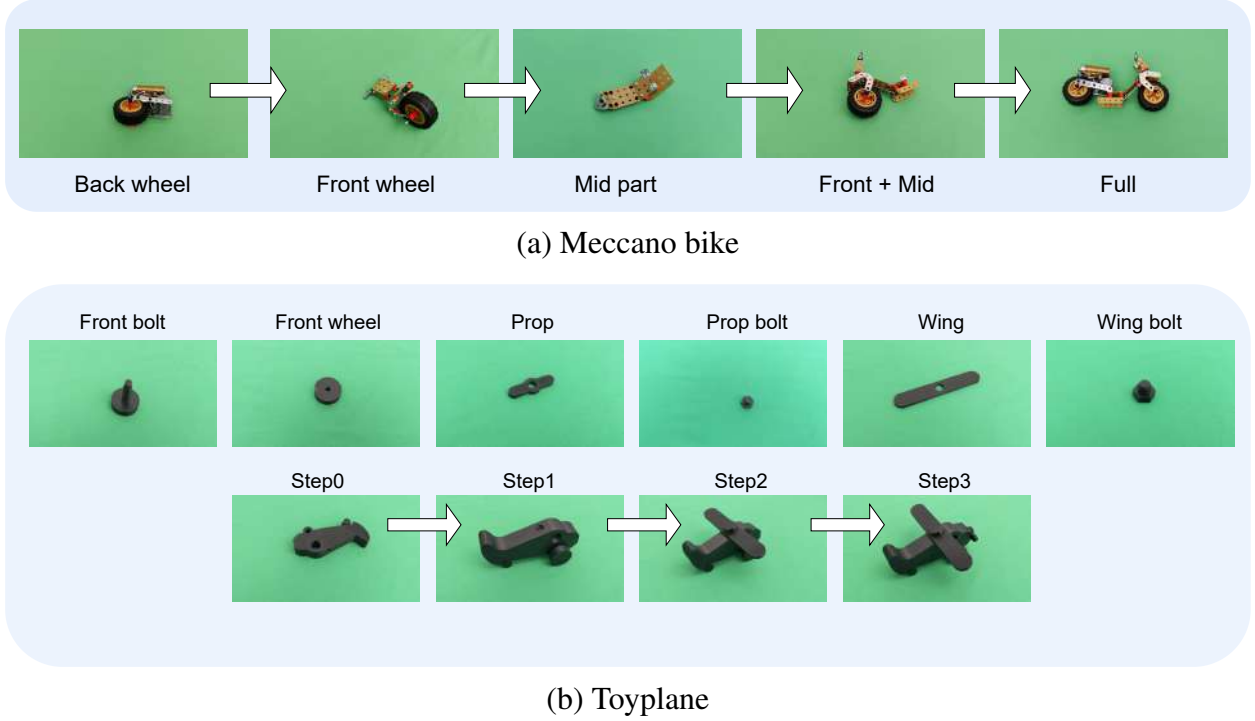


Figure 7: Step by step assembly for: (a) the Meccano bike; and (b) the toy plane.

- **Real data generation:** First, we recorded a video (with a two-minute average length) for each step in the assembly task, where the object is placed in a casual, common background, e.g., on a desk or wooden floor. After that, we used CVAT with the setting of one keyframe out of every 10 frames in the video to draw a 2D bounding box around the object of interest on each keyframe, and then had all remaining video frames annotated automatically by using interpolation of bounding boxes between keyframes. Finally, we exported the annotated video as a tfrecord dataset. Figure 11 shows some examples in the real training sets of the two WCA applications.

After collecting data, we trained the two sets of computer vision models (i.e., object detector + classifier) using the real data and synthetic data, respectively:

- R-M: The object detector and the classifier are trained with the real data labeled using CVAT tool.
- S-M: The object detector and the classifier are trained with the synthetic data.

Subsequently, we trained two sets of computer vision models (i.e., object detector + classifier) using data annotated and generated by our tool, respectively:

- G-M: The object detector and the classifier are trained with the green background data only.
- GD-M: The object detector and the classifier are trained with the green background data and the new replaced background data generated by tinyHulk.





Figure 8: Green background table where we record input videos.

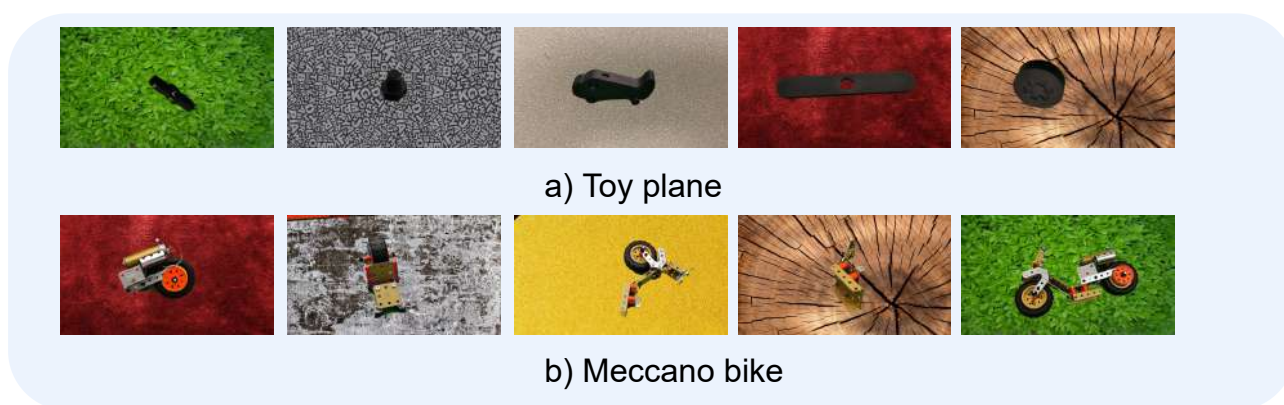


Figure 9: New frames created by tinyHulk's background replacement from the original training sets of the two applications.

Table 1: Abbreviations and descriptions.

Abbreviation	Description
<b>Model</b>	
R-M	model trained with real data, manually labeled using CVAT
S-M	model trained with synthetic data
G-M	model trained with green background data, automatically labeled by tinyHulk
GD-M	model trained with green background data and new background replaced data generated and automatically labeled by tinyHulk
<b>Test set</b>	
UG	uncluttered green test set: the object of interest is placed on a pure green background, without any other objects surrounding
CG	cluttered green test set: the object of interest is placed on a pure green background, with some other objects surrounding
UT	uncluttered table test set: the object of interest is placed on a table background (for example wooden, tile, etc.), without any other objects surrounding
CT	cluttered table test set: the object of interest is placed on a table background (for example wooden, tile, etc.), with some other objects surrounding

Table 2: Training set size of each model in the individual WCA application.

Model	Training set size	
	Meccano bike	Toy plane
R-M	15,477	39,643
S-M	75,000	50,000
G-M	21,399	13,402
GD-M	42,798	26,804

Table 1 summarizes these abbreviations and their descriptions.

Table 2 summarizes the training set sizes.

## 5 Discussion

In this section, we discuss the advantages and disadvantages of using tinyHulk to annotate the image dataset for the DNNs training. Our discussion includes annotation accuracy, human effort, and performance of the DNN model trained.

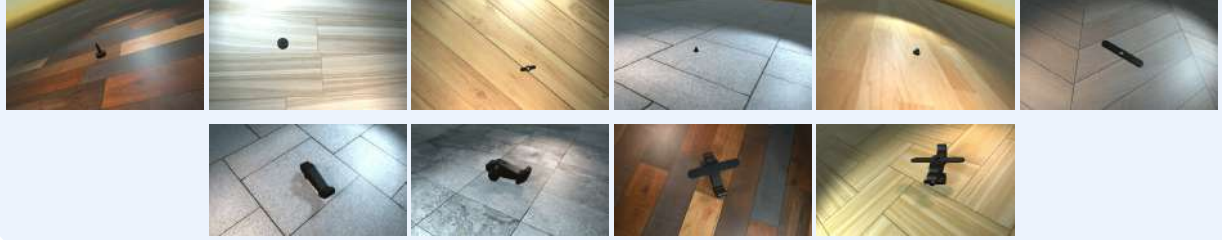
### 5.1 What annotation accuracy does tinyHulk achieve?

We first run tinyHulk to parse the input video to frames and remove near-duplicates (with the similarity threshold  $\delta = 1$ ). The fourth column of table 3 presents total number of frames remaining in each training set of the two applications after duplicate removal. After that, the tool draws





a) Meccano bike



b) Toy plane

Figure 10: Some examples in the synthetic training set of (a) the Meccano bike; and (b) the toy plane.

bounding-boxes for the cleaned datasets. We use the inspection window of the tool to observe and count inaccurate bounding-boxes from the annotation results. Technically, a bounding box is inaccurate if it is so small that it cannot completely cover the object of interest or so large that it covers unrelated information. Figure 12 shows examples of inaccurate bounding-boxes drawn by the tool.

Let  $A$  be the total number of inaccurate bounding boxes,  $B$  be the total number of frames in the dataset after duplicate removal. We calculate the annotation accuracy as  $\frac{B-A}{B} \times 100$ . Table 3 presents the accuracy result achieved with the tool. We can observe that **the tool achieves an average annotation accuracy up to 99.5% on all datasets of the two applications**. In essence, most of the inaccurate bounding boxes are due to noise in the frames, e.g., many wrinkles; too much detail of the cloth stripes in the background; or the object of interest is out of the frame, as shown in Figure 12. Inaccurate annotation can be minimized if the input video is recorded without such noise. For example, the tool achieves 100% annotation accuracy on the input videos of *Front\_bolt*, *Prop*, *Wing*, *Step0*, and *Step2* of the toy plane because these input videos avoided such noise. The tool only achieves 96.7% annotation accuracy on the input video of *Prop\_bolt* of Toyplane because the input video captured too much detail of the background’s cloth.

## 5.2 How much human time is saved with the automatic annotation tool?

To gain insight into the productivity of tinyHulk, we compare the human time spent using our tool and the traditional CVAT tool. For the case of synthetic data generation method, human effort includes many aspects prior to creation of a training set. Examples include creating 3D models of the object, tuning the 3D object dimension, etc. Hence, we do not compare the time aspect of the syn-

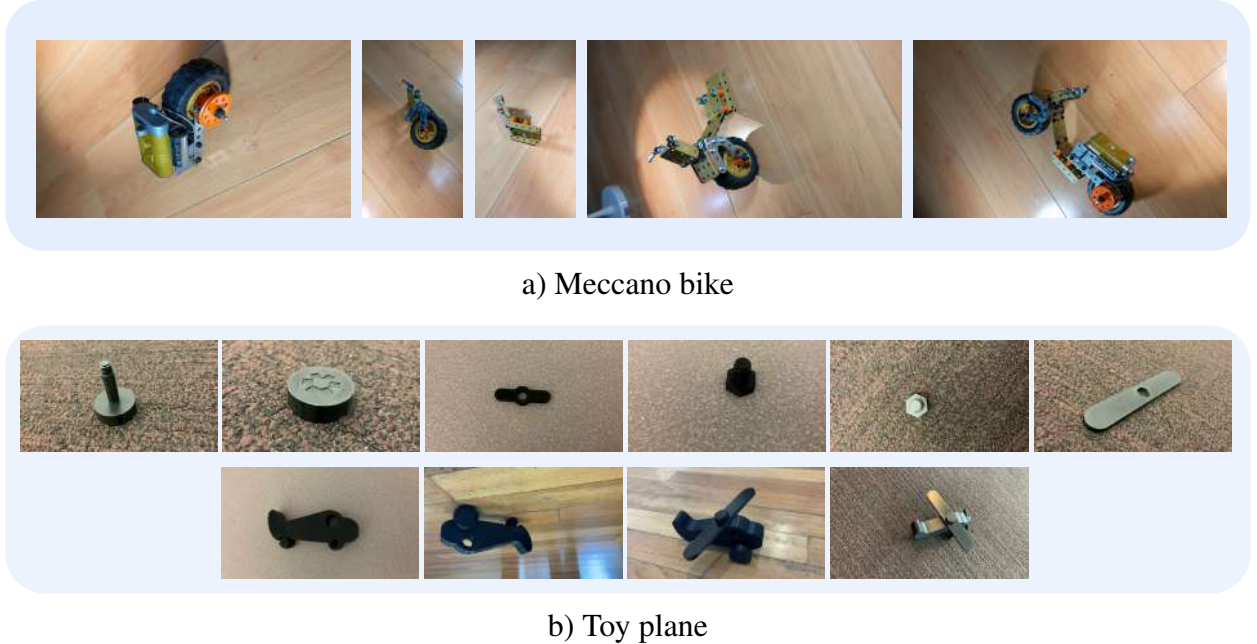


Figure 11: Some examples in the real training set of (a) Meccano bike; and (b) toy plane.

thetic data generation with the other two. Four experienced persons (who have worked intensively in related domains with high experience in using CVAT tool to annotate datasets) worked on the annotation task for each dataset of the two WCA applications. We tracked the total time  $T$  each person spent annotating the dataset (with total frames  $F$ ) using CVAT tool, and using tinyHulk, respectively. After that, we calculate the *time taken per frame*  $= \frac{T}{F}$  which is the average time a person spent to annotate a frame. Note that we count the time that a person directly manipulates the tool, i.e.:

- For the manual labeling using CVAT:  $T$  is counted from when a person annotates the first frame to the last frame of an input video.
- For the automatic labeling using tinyHulk:  $T$  is counted from when a person complete browsing (and corrects the wrong annotations if any) the first frame to the last frame from the annotation results for an input video returned by the tool.

The number of frames in each dataset of the two applications is presented in Table 3. For a fair comparison, we input the same videos recorded with a green background scene to the two annotation tools. Hence, 47,530 frames of the Meccano bike and 27,625 frames of the toy plane are annotated by the two annotation tools separately<sup>9</sup>. Table 4 presents the statistical detail of human time spent with the two annotation tools with respecting the definition above. As observed, labeling image datasets using **tinyHulk can save human time up to fourfold compared to manual labeling using CVAT**.

<sup>9</sup>CVAT is also used to annotate test sets where videos are recorded with different background scenes. The time per frame tracked from this annotation task is statistically analog to the time per frame tracked with the experiment above.

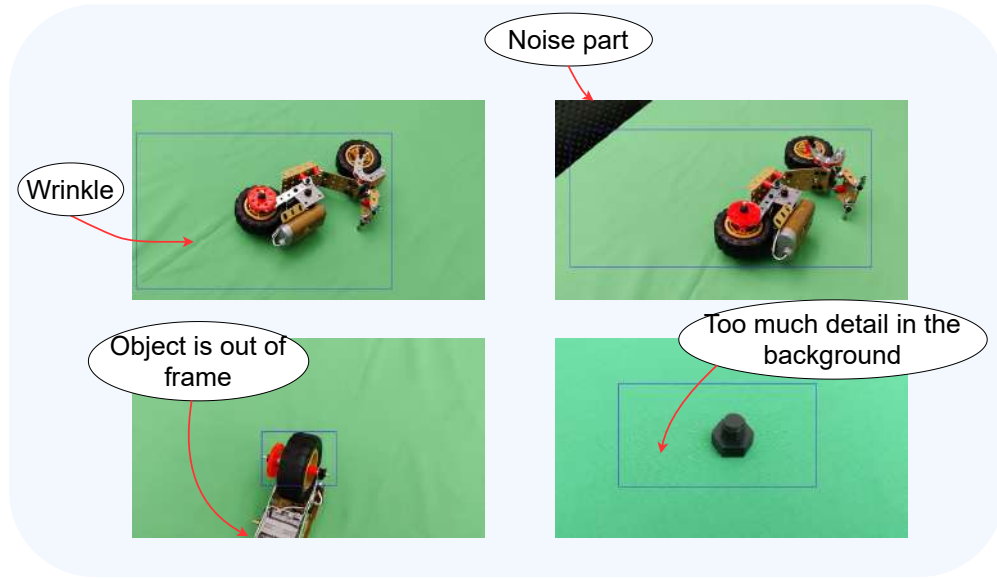


Figure 12: Inaccurate bounding-boxes and reasons.

Table 3: Annotation accuracy achieved by the tool.

Application	Dataset	#Original frames	#Frames remaining after duplicate removal	#Inaccurate bounding box	Accuracy (in percent)
<b>Meccano bike</b>	Front	8589	4014	19	99.6%
	Back	11049	4883	36	99.3%
	Mid	10471	5130	32	99.4%
	Fronttwo	8083	3545	30	99.2%
	Full	9338	3827	29	99.3%
<b>Toyplane</b>	Front_bolt	2295	1245	0	100%
	Front_wheel	2714	1297	4	99.7%
	Prop	1940	1112	0	100%
	Prop_bolt	2234	1233	41	96.7%
	Wing	2312	1181	0	100%
	Wing_bolt	2730	1269	11	99.1%
	Step0	4229	1913	0	100%
	Step1	3390	1551	1	99.9%
	Step2	2679	1193	0	100%
	Step3	3102	1408	3	99.8%

Table 4: **Human time** spent on the annotation task with each tool.

Annotation method	Time per frame (Mean(SD))(in seconds)	#Frames annotated
Manual (using CVAT)	0.4 (0.14)	- Meccano bike: 47,530
Automatic (using tinyHulk)	<b>0.1 (0.02)</b>	- Toy plane: 27,625

Table 5: tinyHulk’s **execution time**.

Individual task	Time per frame (Mean (SD))(seconds)
Video parsing and duplicate removal	0.03 (0.001)
Bounding-box drawing	0.03 (0.001)
Background replacement	0.18 (0.02)

To increase the size of the original annotated dataset, one can use the background replacement component of tinyHulk. As we mention in Section 3, a person directly manipulates the component to select HSV threshold values once for a whole dataset. The average human time spent choosing an HSV threshold for each dataset is approximately 20 seconds, regardless of the dataset’s size. This amount of time is much smaller than that used to annotate a new dataset.

To gain insight into tinyHulk’s *execution time*, we present in Table 5 the average time per frame that the tool takes for each task, i.e., video parsing and duplicate removal, drawing bounding-box, and background replacement.

### 5.3 Is duplicate removal really necessary?

To verify whether duplicate removal is important, we first run the tool to remove duplicates with different similarity threshold settings. Table 6 presents the Meccano bike dataset size in different similarity thresholds. As expected, the total number of remaining frames decreases significantly when the similarity threshold  $\delta$  is raised. We use the remaining data in each similarity threshold setting to train the object detectors. There are four object detectors with the four training sets created above:

- no-DR: the object detector is trained with the original data without duplicate removal.

Table 6: Total frames in each training set of the Meccano bike assembly application with various similarity thresholds  $\delta$  for duplicate removal.

Application	Dataset	Total frames remaining			
		$\delta = 0$	$\delta=1,2$	$\delta=3$	$\delta=4$
<b>Meccano bike</b>	Front	8,589	4,014	1,710	961
	Mid	10,471	5,130	2,209	1,216
	Back	11,049	4,883	2,021	1,160
	Fronttwo	8,083	3,545	1,485	835
	Full	9,338	3,827	1,571	920

Table 7: Average Precision (AP) and Average Recall (AR) @IoU=0.50:0.95 of the object detectors of Meccano bike assembly application on different test sets.

Application	Model	Cluttered table		Uncluttered table		Cluttered green		Uncluttered green	
		AP	AR	AP	AR	AP	AR	AP	AR
<b>Meccano bike</b>	no-DR	0.5	0.567	0.566	0.691	0.549	0.682	0.963	0.978
	DR-1	<b>0.575</b>	<b>0.652</b>	<b>0.651</b>	<b>0.687</b>	<b>0.570</b>	<b>0.695</b>	<b>0.973</b>	<b>0.981</b>
	DR-3	0.495	0.591	0.414	0.541	0.429	0.687	0.97	0.981
	DR-4	0.447	0.529	0.28	0.459	0.497	0.692	0.968	0.98

- DR-1: the object detector is trained with data after duplicate removal with the similarity threshold  $\delta = 1$  (or 2).
- DR-3: the object detector is trained with data after duplicate removal with the similarity threshold  $\delta = 3$ .
- DR-4: the object detector is trained with data after duplicate removal with the similarity threshold  $\delta = 4$ .

We collect different test sets where the object of interest is placed on different backgrounds:

- Cluttered table: The object of interest is placed on a table, some other objects surround it.
- Uncluttered table: The object of interest is solely placed on a table, with nothing surrounding it.
- Cluttered green: The object of interest is placed on a green background surface, some other objects surround it.
- Uncluttered green: The object of interest is placed on a green background surface, with nothing surrounding it.

The test sets: cluttered table, uncluttered table, and cluttered green are annotated manually using CVAT, while the uncluttered green test set is automatically annotated using tinyHulk. After that, we evaluate the four trained object detectors performance on these test sets respecting the average precision and average recall at the Intersection over Union 0.50:0.95 (*AP and AR @ IoU= 0.50:0.95*). IoU is the most popular evaluation metric used in object detection benchmarks [40, 17]. Detections were assigned to ground truth objects and judged to be true/false positives by measuring the overlap between the predicted bounding box and the ground truth bounding box. To be considered a correct detection, IoU must exceed 50% [17]. Based on the IoU metric, the average precision (AP) and the average recall (AR) are calculated accordingly. Table 7 presents results of the four object detectors on different test sets.

The results show that DR-1 achieves the best performance among the four models respecting the evaluation metrics. Also, we observe that if the similarity threshold  $\delta$  is set at a high value

(i.e.,  $\delta > 2$  in this experiment), resulting in too many images being removed from the training sets, the trained models’ performance drops (e.g., performance drops in DR-3 and DR-4). This result demonstrates that **the duplicate removal helps improve the performance of the DNNs models. However, one should be careful in choosing the similarity threshold  $\delta$  to avoid missing the critical information from the original dataset.**

To create the training sets for the following experiments, we set the similarity threshold for the duplicate removal as  $\delta = 1$ .

## 5.4 How does the performance of the models trained with data labeled and generated by tinyHulk compare with alternatives?

We compare the performance of the models trained with data labeled by tinyHulk (i.e., G-M and GD-M) with the models trained with data manually labeled (i.e., R-M), and synthetic data (i.e., S-M), separately. Let  $\mathcal{CL}$  be the total number of correct labels returned by the model on a specific test set, and  $\mathcal{GT}$  be the total number of ground-truth labels in that test set. The model’s accuracy is calculated as  $\frac{\mathcal{CL}}{\mathcal{GT}} \times 100$ .

**Meccano bike application.** We collect different test sets where the object of interest is placed on different background surfaces and in different light conditions (i.e., five categories: T1, T2, T3, T4, and T5). T5 is special – we recorded a uncluttered table test set in the same condition (background, light condition) of the training set of R-M. Table 8 summarizes the test sets and the number of ground truths in each test set. We extensively run the four models on these test sets and collect the results returned by each model. Table 9 presents the mean and standard deviation of the accuracy of the four models over the test sets of four categories T1, T2, T3, T4, and T5. Because T5 only has a single uncluttered test set, the results of the four models on T5 is presented separately in Table 10.

From the result, we observe that both G-M and GD-M outperform R-M and S-M in all cluttered and uncluttered green test sets. The performance preeminence of G-M and GD-M in these cases is explainable because they are more familiar with these test sets (i.e., the training sets of G-M and GD-M have images of objects on the green background) than are R-M and S-M. Further, GD-M outperforms R-M and is very competitive with S-M in most test cases. Recall that GD-M’s training sets consist of images of the object on the green background and the augmented data (new background scenes replace the green background) generated and automatically annotated by the tool; R-M’s training sets consist of the real and manual labeling data; while S-M’s training sets consist of a large amount of synthetic data (the training set size of S-M is 1.5 times larger than the training set size of GD-M). Especially, GD-M even outperforms R-M on the uncluttered test set of T5 – R-M’s ”home field” (i.e., this test set was recorded in the same background scene as R-M’s training set). The accuracy of GD-M and R-M on this test set of T5 is 84.3% and 83.5%, respectively.

**Toy plane application.** Similarly, we collect test sets for the toy plane where the object is placed on different background surfaces and light conditions (T1, T2, T3). Table 11 summarizes the test sets and number of ground truth in each test set. Uncluttered table test sets of T2, T3 are recorded in the same condition with the training set of R-M. Table 12 presents the mean and

Table 8: Test set for Meccano bike model.

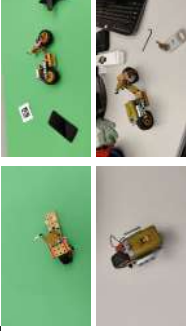

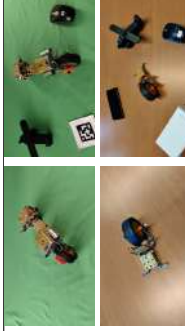


Test set	Test size				Sample
	Cluttered table	Cluttered green	Uncluttered table	Uncluttered green	
T1: white surface background for cluttered table and uncluttered table test set, normal light	4535	4914	4503	6914	
T2: white surface background for cluttered table and uncluttered table test set, dim light	4613	4736	4687	4667	
T3: Wooden surface background for cluttered table and uncluttered table test set, dark light	6462	5656	5909	5274	
T4: Black surface background	4779	None	4741	None	
T5: Same background and light condition as of training data of R-M	None	None	4490	None	

Table 9: Accuracy of the 4 models of the Meccano bike over various test sets.

Application	Model	Accuracy (Mean(SD))			
		CT	UT	CG	UG
<b>Meccano bike</b>	R-M	66.1(11.6)	84.3(3)	53.3(5.1)	24.3(1.3)
	S-M	76.1(18.8)	<b>92.8(10.5)</b>	26(3.1)	9(1.6)
	G-M	81.9(14.5)	77.1(11.7)	87.2(4.2)	<b>98.2(2.1)</b>
	GD-M	<b>84.6(10.8)</b>	89.8(5.2)	<b>87.6(2.3)</b>	<b>98.2(2.4)</b>

Table 10: Accuracy (in percentage) of the four models on T5 (only has a uncluttered table test set which is recorded in the same environment with the training set of R-M) of the Meccano bike.

Model	Accuracy (in percentage)
R-M	83.5
S-M	<b>96</b>
G-M	49.6
GD-M	84.3

standard deviation of the accuracy of the four models over the test sets of three categories T1, T2, and T3.

Similar to the result observed in the Meccano bike’s models, G-M and GD-M outperform R-M in most cluttered green and uncluttered green test sets for the toy plane application. In other test sets, G-M performs poorly compared to other models. One reason is that the training set’s size of G-M is the smallest (18,979 records), approximately half of the training set’s size of R-M (39,643 records), and one-third of the training set size of S-M (50,000 records). However, when we double the training set’s size of G-M by the augmented data (i.e., background replacement) generated by tinyHulk and use it as the training set of GD-M, the GD-M’s performance significantly improves. As observed, GD-M outperforms R-M and S-M in most test sets, even those test sets recorded in the same conditions as the training set of R-M (the uncluttered table test sets of T2, T3).

The attractive performance of GD-M is thanks to 1) the training data is labeled systematically and precisely by tinyHulk; 2) the advantage of the duplicate removal as discussed in Section 5.3; and 3) the new background data generated by the tool added to the training set that improves the model’s generalizability. At the other extreme, human mistakes (i.e., drawing incorrect bounding boxes) due to fatigue or distraction when annotating data with manual tools, and duplicates in the final training data of R-M cause its performance to be not robust in many test sets. Figure 13 presents some inaccurate manual annotations found in the training set of the two applications.

To summarize, the high performance in terms of accuracy achieved by GD-M shows that **tinyHulk helps to create labeled image training sets for computer vision models that result in a competitive model to the ones trained with real data and synthetic data at a fraction of the annotation effort.**



Table 11: Test set for the toy plane model.




Test set	Test size				Sample
	Cluttered table	Cluttered green	Uncluttered table	Uncluttered green	
T1: wooden surface background for cluttered table and uncluttered table test set, dark light for green background	8424	8460	8634	8290	
T2: white surface background for cluttered table and uncluttered table test set, dim light for green background	8551	8332	8431	7999	
T3: white surface background for cluttered table and uncluttered table test set, lighter light for green background	8467	7994	8449	8085	

Table 12: Accuracy of the 4 models of the toy plane application over various test sets in categories T1, T2, and T3.

Application	Model	Accuracy (Mean(SD))			
		CT	UT	CG	UG
<b>Toy plane</b>	R-M	46.1(7.1)	75.4(3.2)	17.9(2.2)	36(2.6)
	S-M	79.1(8.2)	81.9(9.1)	<b>68.8(11.3)</b>	81.7(9.6)
	G-M	21.8(16.9)	23.6(15.8)	39.8(11.7)	83.5(15.2)
	GD-M	<b>82.2(10.5)</b>	<b>86.1(12.6)</b>	62.4(1.5)	<b>92.7(9.1)</b>



a) Meccano bike



b) Toyplane

Figure 13: Examples of inaccurate manual annotation in the training sets of a) Meccano bike; and b) Toy plane.

## 5.5 Limitations?

tinyHulk is not free from limitations. First, most of tinHulk’s annotation errors arise from the quality of the input videos. **The tool is sensitive to noise in the background.** As we mentioned in Section 5.1, extraneous parts, background cloth wrinkles, and excessive background cloth detail are the main reasons for most of the incorrect annotations we observed. These challenges can be overcome by: 1) using a better background material, such as a light absorbent material backdrop and 2) using a large area enough surface to place the object of interest so that the recorded frame does not capture anything outside of the surface’s area. In our experiment, those small objects (e.g., front\_bolt, prop, wing of the toy plane) perfectly fit within the table size. Hence all frames from the recorded video of these objects are captured only the area within the green background surface, with no other background at the edge of the frame. Therefore, tinyHulk obtained 100% annotation accuracy on these input videos (see table 3).

Second, as we depend on the HSV threshold values for the background replacement, keeping the original background color unchanged in the whole input video is vital. However, some digital cameras automatically adjust brightness and exposure while recording videos (i.e., the automatic white balance feature [31]), resulting in frames with different background colors in a video (for example, the green color turns to the blue color). This difference causes the HSV threshold values selected for one sample frame to be incorrect for other frames, causing bad background replacement results. Figure 14 presents an example, the user selected the HSV threshold values for the ”green” color of the background of the frame at the first column, which is wrong to apply to the frame in the second column. We leave this limitation to solve in future work.

## 6 Conclusion

Data annotation is the foundation of every AI and machine learning model, requiring massive human effort and time. This labor-intensive task is a bottleneck for developing and adopting WCA applications that depend on such DNNs models. In this work, we proposed tinyHulk, an automatic annotation tool to help users significantly reduce human effort and time for these labeling tasks. The tool is equipped with background replacement, an object-aware image augmentation mechanism to increase the size and diversity of the training set. The experimental results show that using tinyHulk can save human time up to fourfold for a labeling task. The computer vision models trained with the data annotated and generated by tinyHulk outperform the corresponding models trained with the real data labeled by the manual labeling tool. They are also very competitive with the models trained with a large amount of synthetic data. Therefore, tinyHulk can replace the manual labeling tools for such labor-intensive labeling tasks, as well as substitute for synthetic data generation where a 3D-model of the object is not available. Future work involves building a web-based service for tinyHulk with a simple interactive interface. We also plan to investigate further image augmentation to create an even more diverse training set from the original dataset.

Rapid advances in edge computing and machine learning research areas open an unprecedented opportunity for WCA applications. tinyHulk can contribute to accelerating the realizing these great opportunities for WCA development and adoption.



a). Original frames in an input video



b). Background replacement with a chosen HSV threshold

Figure 14: Auto brightness and exposure adjustment in some smart camera causes different background in a recorded video (Figure a), resulting in a bad background replacement (Figure b).

## References

- [1] BARRON, J. T., AND POOLE, B. The fast bilateral solver. In *European conference on computer vision* (2016), Springer, pp. 617–632.
- [2] BARZ, B., AND DENZLER, J. Do we train on test data? Purging CIFAR of near-duplicates. *Journal of Imaging* 6, 6 (2020), 41.
- [3] BEN-EZRA, M. Segmentation with invisible keying signal. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2000 (Cat. No. PR00662)* (2000), vol. 1, IEEE, pp. 32–37.
- [4] BILEN, H., AND VEDALDI, A. Weakly supervised deep detection networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), pp. 2846–2854.
- [5] BORKMAN, S., CRESPI, A., DHAKAD, S., GANGULY, S., HOGINS, J., JHANG, Y.-C., KAMALZADEH, M., LI, B., LEAL, S., PARISI, P., ET AL. Unity perception: Generate synthetic data for computer vision. *arXiv preprint arXiv:2107.04259* (2021).
- [6] BROOKS, J. COCO Annotator. <https://github.com/jsbroks/coco-annotator/>, 2019.
- [7] BROWN, M., SZELISKI, R., AND WINDER, S. Multi-image matching using multi-scale oriented patches. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)* (2005), vol. 1, IEEE, pp. 510–517.
- [8] CHEN, Z. *An application platform for wearable cognitive assistance*. PhD thesis, Ph.D. Dissertation. Carnegie Mellon University, 2018.
- [9] COCODATASET.ORG. Common Object in COntext. <https://cocodataset.org/#download>, Accessed: 2022-09-30.
- [10] CUBUK, E. D., ZOPH, B., MANE, D., VASUDEVAN, V., AND LE, Q. V. Autoaugment: Learning augmentation strategies from data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2019), pp. 113–123.
- [11] CVAT.AI CORPORATION. Computer Vision Annotation Tool (CVAT), 9 2022.
- [12] DENG, J., DONG, W., SOCHER, R., LI, L.-J., LI, K., AND FEI-FEI, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition* (2009), IEEE, pp. 248–255.
- [13] DONG, X., ZHENG, L., MA, F., YANG, Y., AND MENG, D. Few-example object detection with model communication. *IEEE transactions on pattern analysis and machine intelligence* 41, 7 (2018), 1641–1654.

- [14] DUTTA, A., AND ZISSERMAN, A. The VIA annotation software for images, audio and video. In *Proceedings of the 27th ACM International Conference on Multimedia* (New York, NY, USA, 2019), MM '19, ACM.
- [15] DVORNIK, N., MAIRAL, J., AND SCHMID, C. Modeling visual context is key to augmenting object detection datasets. In *Proceedings of the European Conference on Computer Vision (ECCV)* (2018), pp. 364–380.
- [16] DWIBEDI, D., MISRA, I., AND HEBERT, M. Cut, paste and learn: Surprisingly easy synthesis for instance detection. In *Proceedings of the IEEE international conference on computer vision* (2017), pp. 1301–1310.
- [17] EVERINGHAM, M., VAN GOOL, L., WILLIAMS, C. K., WINN, J., AND ZISSERMAN, A. The pascal visual object classes (voc) challenge. *International journal of computer vision* 88, 2 (2010), 303–338.
- [18] GEBRU, T., KRAUSE, J., WANG, Y., CHEN, D., DENG, J., AND FEI-FEI, L. Fine-grained car detection for visual census estimation. *Proceedings of the AAAI Conference on Artificial Intelligence* (Feb. 2017).
- [19] GHIASI, G., CUI, Y., SRINIVAS, A., QIAN, R., LIN, T.-Y., CUBUK, E. D., LE, Q. V., AND ZOPH, B. Simple copy-paste is a strong data augmentation method for instance segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2021), pp. 2918–2928.
- [20] HA, K., CHEN, Z., HU, W., RICHTER, W., PILLAI, P., AND SATYANARAYANAN, M. Towards wearable cognitive assistance. In *Proceedings of the 12th annual international conference on Mobile systems, applications, and services* (2014), pp. 68–81.
- [21] HAO, Q., LUO, L., JAN, S. T., AND WANG, G. It’s not what it looks like: Manipulating perceptual hashing based applications. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security* (2021), pp. 69–85.
- [22] HE, X.-C., AND YUNG, N. H. Curvature scale space corner detector with adaptive threshold and dynamic region of support. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.* (2004), vol. 2, IEEE, pp. 791–794.
- [23] HINTERSTOISSER, S., LEPETIT, V., WOHLHART, P., AND KONOLIGE, K. On pretrained image features and synthetic images for deep learning. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops* (2018), pp. 0–0.
- [24] HINTERSTOISSER, S., PAULY, O., HEIBEL, H., MARTINA, M., AND BOKELOH, M. 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 workshops* (2019), pp. 0–0.

- [25] HOFFER, E., AND AILON, N. Deep metric learning using triplet network. In *International workshop on similarity-based pattern recognition* (2015), Springer, pp. 84–92.
- [26] IYENGAR, R., ZHANG, E., AND SATYANARAYANAN, M. Experience with using synthetic training images for wearable cognitive assistance. *Association for the Advancement of Artificial Intelligence* ([www.aaai.org](http://www.aaai.org)) (2022).
- [27] KOZAT, S. S., VENKATESAN, R., AND MIHÇAK, M. K. Robust perceptual image hashing via matrix invariants. In *2004 International Conference on Image Processing, 2004. ICIP'04.* (2004), vol. 5, IEEE, pp. 3443–3446.
- [28] LABEL STUDIO. <https://labelstud.io/>, Accessed: 2022-09-30.
- [29] LEE, J., WALSH, S., HARAKEH, A., AND WASLANDER, S. L. Leveraging pre-trained 3d object detection models for fast ground truth generation. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)* (2018), IEEE, pp. 2504–2510.
- [30] LI, P., XIE, J., WANG, Q., AND GAO, Z. Towards faster training of global covariance pooling networks by iterative matrix square root normalization. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2018), pp. 947–955.
- [31] LIU, Y.-C., CHAN, W.-H., AND CHEN, Y.-Q. Automatic white balance for digital still camera. *IEEE Transactions on Consumer Electronics* 41, 3 (1995), 460–466.
- [32] LONG, J., SHELHAMER, E., AND DARRELL, T. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2015), pp. 3431–3440.
- [33] MELEKHOV, I., KANNALA, J., AND RAHTU, E. Siamese network features for image matching. In *2016 23rd international conference on pattern recognition (ICPR)* (2016), IEEE, pp. 378–383.
- [34] MIKOŁAJCZYK, A., AND GROCHOWSKI, M. Data augmentation for improving deep learning in image classification problem. In *2018 international interdisciplinary PhD workshop (IIPhDW)* (2018), IEEE, pp. 117–122.
- [35] MONGA, V., AND EVANS, B. L. Perceptual image hashing via feature points: performance evaluation and tradeoffs. *IEEE transactions on Image Processing* 15, 11 (2006), 3452–3465.
- [36] NOROUZI, M., FLEET, D. J., AND SALAKHUTDINOV, R. R. Hamming distance metric learning. *Advances in neural information processing systems* 25 (2012).
- [37] OPENCV. <https://opencv.org/>, Accessed: 2022-09-30.
- [38] PEREZ, L., AND WANG, J. The effectiveness of data augmentation in image classification using deep learning. *arXiv preprint arXiv:1712.04621* (2017).

- [39] REN, S., HE, K., GIRSHICK, R., AND SUN, J. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems* 28 (2015).
- [40] REZATOFIGHI, H., TSOI, N., GWAK, J., SADEGHIAN, A., REID, I., AND SAVARESE, S. Generalized intersection over union: A metric and a loss for bounding box regression. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (2019), pp. 658–666.
- [41] ROSTEN, E., PORTER, R., AND DRUMMOND, T. Faster and better: A machine learning approach to corner detection. *IEEE transactions on pattern analysis and machine intelligence* 32, 1 (2008), 105–119.
- [42] RUSSAKOVSKY, O., DENG, J., SU, H., KRAUSE, J., SATHEESH, S., MA, S., HUANG, Z., KARPATY, A., KHOSLA, A., BERNSTEIN, M., ET AL. Imagenet large scale visual recognition challenge. *International journal of computer vision* 115, 3 (2015), 211–252.
- [43] SATO, I., NISHIMURA, H., AND YOKOI, K. Apac: Augmented pattern classification with neural networks. *arXiv preprint arXiv:1505.03229* (2015).
- [44] SATYANARAYANAN, M. From the editor in chief: Augmenting cognition. *IEEE Pervasive Computing* 3, 2 (2004), 4–5.
- [45] SATYANARAYANAN, M., BAHL, P., CACERES, R., AND DAVIES, N. The case for vm-based cloudlets in mobile computing. *IEEE pervasive Computing* 8, 4 (2009), 14–23.
- [46] SIMARD, P. Y., STEINKRAUS, D., PLATT, J. C., ET AL. Best practices for convolutional neural networks applied to visual document analysis. In *Icdar* (2003), vol. 3, Edinburgh.
- [47] SU, C.-H., CHIU, H.-S., AND HSIEH, T.-M. An efficient image retrieval based on hsv color space. In *2011 International Conference on Electrical and Control Engineering* (2011), IEEE, pp. 5746–5749.
- [48] SU, C. H., CHIU, H. S., HUNG, J. H., AND HSIEH, T. M. Color space comparison between rgb and hsv based images retrieval. In *Advanced Materials Research* (2014), vol. 989, Trans Tech Publ, pp. 4123–4126.
- [49] TELUS INTERNATIONAL. Five types of image annotation and their use cases. <https://telusinternational.com/insights/ai-data/article/an-introduction-to-5-types-of-image-annotation>, Accessed: 2022-09-30.
- [50] TENSORFLOW.ORG. The TFRecord format. [https://www.tensorflow.org/tutorials/load\\_data/tfrecord](https://www.tensorflow.org/tutorials/load_data/tfrecord), Accessed: 2022-09-30.



- [51] VINUTHA, H., POORNIMA, B., AND SAGAR, B. Detection of outliers using interquartile range technique from intrusion dataset. In *Information and decision sciences*. Springer, 2018, pp. 511–518.
- [52] WAN, L., ZEILER, M., ZHANG, S., LE CUN, Y., AND FERGUS, R. Regularization of neural networks using dropconnect. In *International conference on machine learning* (2013), PMLR, pp. 1058–1066.
- [53] WANG, J., ZHAO, S., LIU, Z., TIAN, Y., DUAN, F., AND PAN, Y. An active contour model based on adaptive threshold for extraction of cerebral vascular structures. *Computational and mathematical methods in medicine 2016* (2016).
- [54] WONG, S. C., GATT, A., STAMATESCU, V., AND MCDONNELL, M. D. Understanding data augmentation for classification: when to warp? In *2016 international conference on digital image computing: techniques and applications (DICTA)* (2016), IEEE, pp. 1–6.
- [55] ZHENG, S., SONG, Y., LEUNG, T., AND GOODFELLOW, I. Improving the robustness of deep neural networks via stability training. In *Proceedings of the ieee conference on computer vision and pattern recognition* (2016), IEEE, pp. 4480–4488.
- [56] ZOPH, B., CUBUK, E. D., GHIASI, G., LIN, T.-Y., SHLENS, J., AND LE, Q. V. Learning data augmentation strategies for object detection. In *European conference on computer vision* (2020), Springer, pp. 566–583.