Fine-Grained Visual Recognition in Mobile Augmented Reality for Technical Support

Bing Zhou and Sinem Güven, *Senior Member, IEEE*

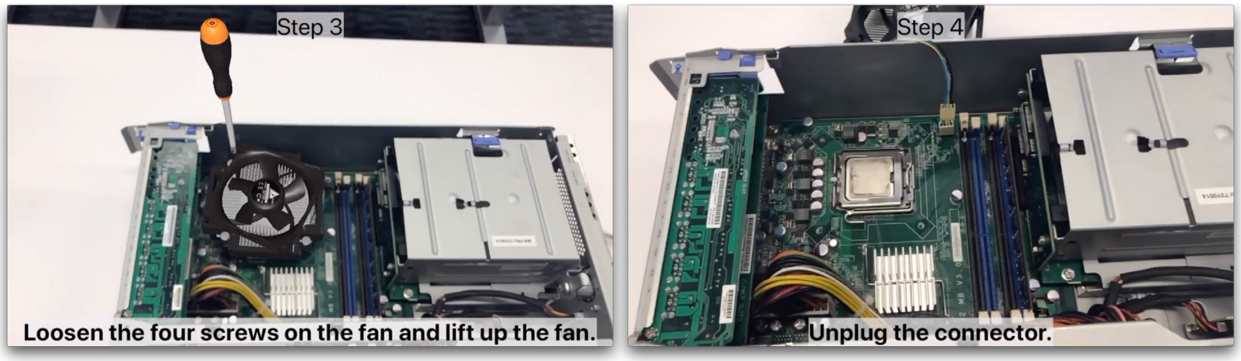

Fig. 1. Fine-grained visual recognition for augmented reality enables dynamic presentation of right set of visual instructions in the right context by analyzing the hardware state as the repair procedure evolves.

Abstract— Augmented Reality is increasingly explored as the new medium for two-way remote collaboration applications to guide the participants more effectively and efficiently via visual instructions. As users strive for more natural interaction and automation in augmented reality applications, new visual recognition techniques are needed to enhance the user experience. Although simple object recognition is often used in augmented reality towards this goal, most collaboration tasks are too complex for such recognition algorithms to suffice. In this paper, we propose a fine-grained visual recognition approach for mobile augmented reality, which leverages RGB video frames and sparse depth feature points identified in real-time, as well as camera pose data to detect various visual states of an object. We demonstrate the value of our approach through a mobile application designed for hardware support, which automatically detects the state of an object to present the right set of information in the right context.

Index Terms—Visual recognition, augmented reality, mobile

1 Introduction

Augmented Reality (AR) enhances the perception of our surroundings by overlaying media and graphics on top of what we see in the real world. Over the last decade, we have seen major progress and increased interest in AR thanks to AR SDKs, such as ARKit [5] and ARCore [9], which helped lower the barrier for entry for AR development. Recently, intelligent AR systems driven by Artificial Intelligence (AI) are beginning to emerge to enhance the AR experiences [3, 32]. Despite this progress, most AR user experiences remain primitive, and lack intelligence and automation, thereby rendering the user interaction rather unintuitive. Although AR enables tracking of virtual objects and annotations in physical spaces through computer vision techniques, it is not inherently intelligent to actually recognize semantics of what it sees. For example, in the technical support domain, traditional AR solutions can recognize a desktop motherboard in the form of a point cloud to enable tracked annotations on top of the hardware, but it does not necessarily know that it is looking at a motherboard. Nor would such system be able to understand if a desktop computer's cover was open or closed, or that the motherboard has its fan removed, or a specific connector unplugged and so on (Figure 1). The lack of semantic, fine-grained recognition requires all interaction to be driven by the user by identifying what they are looking at (e.g., through pre-labelling objects, specifying the state of an object, etc.) before they can have the relevant AR content projected to their view, and thereby significantly limits the interaction.

To address this gap and provide enriched AR user experiences, more fine-grained visual recognition, i.e., recognizing not only objects but also the visual state change of an object (or its parts), is desirable in a wide range of application scenarios, including technical support. However, there are various challenges associated with providing such fine-grained visual recognition capabilities. For example, Figure 2 shows images of a computer under repair. In this example, the technician is replacing the CPU of this computer. To render the relevant AR instructions in the right context, the visual recognition mechanism should be able to detect the state change of a part of this computer - the motherboard. Only when both the battery and the fan are removed from the motherboard, should the AR instructions for CPU removal be rendered. Compared to traditional visual recognition methods [11], implementing such state recognition requires addressing the following challenges in an AR application: *i) Camera Distance:* Depending on the target object or object parts that need to be recognized, the machine learning model requires the camera to be at varying distance to the object. For example, in Figure 2(a), in order to recognize the whole machine in the camera field of view (FoV), the camera needs to be far away from the machine, while to recognize the battery, the camera needs to be closer to the machine. This requires our solution to adjust camera distance dynamically to achieve fine-grained visual recognition. *ii) Viewing Angle:* The relevant target object parts need to be visible to be recognized. This means that the camera must capture the object at a certain angle. Figure 2(b) shows the machine being viewed from a particular angle where the battery is occluded by wires, whereas in

- *Bing Zhou is with IBM T. J. Watson Research Center, Yorktown Heights, New York, United States. E-mail: bing.zhou@ibm.com.*
- *Sinem Güven is with IBM T. J. Watson Research Center, Yorktown Heights, New York, United States. E-mail: sguven@us.ibm.com.*

Manuscript received 18 May 2020; revised 26 July 2020; accepted 17 Aug. 2020. Date of publication 17 Sept. 2020; date of current version 3 Nov. 2020. Digital Object Identifier no. 10.1109/TVCG.2020.3023635

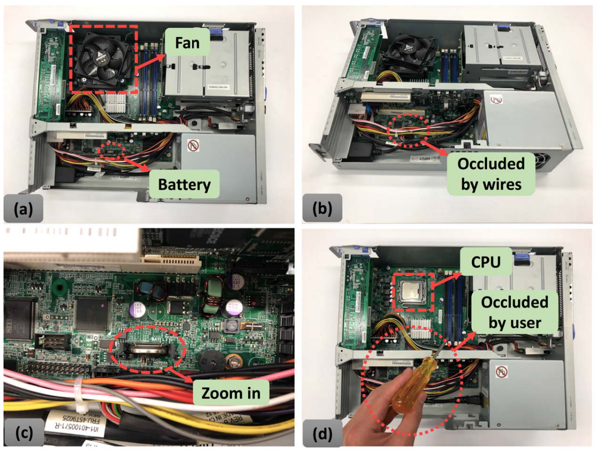

Fig. 2. Example images of a computer from different viewing angles during a repair session.

Figure 2(c), the battery is clearly visible. Unless we maintain a proper viewing angle, visual recognition naturally results in poor accuracy. *iii) Noisy Input:* As shown in Figure 2(d), a common challenge in AR-driven hardware support is that the object may be temporally occluded by the user's hand or tools. It is also common that the input images are captured by a moving camera, hence can be blurry at times. All these factors make input images noisy for the visual recognition model. So our solution needs to be robust in dealing with such noisy input, while providing reliable and stable recognition results. In addition to the challenges above, our solution must be designed to work within the resource and power constraints of mobile devices.

The ideal solution mimics the process of human perception and reasoning: to detect state changes, it enables the camera to focus on discrete local areas that change appearance in different states; prompts the user to adjust to proper viewing angles to collect images from these local areas, and makes prediction on state change only when sufficient visual data is collected. Unfortunately, most existing visual recognition solutions [11, 30] are trained and evaluated using static image data sets [8], and do not explicitly account for varying camera distance, viewing angle and specific local areas.

We propose a solution that takes advantage of AR specific data, such as real-time generated 3D feature points and camera pose, to complement the images captured by the camera for fine-grained visual recognition. We first use a set of training videos and learn Regions of Interest (RoIs), which have appearance changes that distinguish different states. We actively track the 6-DOF camera pose to ensure that the camera is kept at the right distance and viewing angle to the RoIs, minimize occlusions or other noise to the input images of the visual recognition model. To improve the robustness of recognition, we develop a discrete multi-stream Convolutional Neural Network (CNN) [19], in conjunction with bi-directional Long Short Term Memory (LSTM) [12], namely a Discrete-CNN-LSTM (DCL) model, to extract not only spatial, but also temporal data to predict state changes. In summary, our paper makes the following novel contributions:

- We study the unique problem of fine-grained visual recognition in a mobile AR setting, and propose to combine image, 3D feature point and camera pose data to actively predict object state changes.

- We generate RoI candidates from merged 3D feature points collected from AR sessions, and extract distinguishable RoIs automatically using deep CNN feature representations, which are tracked and cropped for fine-grained recognition.

- We propose Discrete-CNN-LSTM (DCL) model, which distributes RoI images on discrete multi-stream CNN branches and aggregates information with bi-directional LSTM layers. Multi-stream CNN with shared weights solves the contradictory problem between high image resolution required for fine-granularity thus larger model size, and the shrinking resolution as the number of RoIs increase. The LSTM layers aggregate the visual information in the temporal domain to further enhance the prediction stability.

- We build an iOS application using ARKit and Tensorflow [1] to demonstrate the effectiveness of our solution, and provide comprehensive evaluations using a hardware maintenance application scenario.

2 Related Work

AR for Training and Technical Support. AR is increasingly becoming a popular medium for technical support and training as it provides an intuitive way to communicate otherwise complex information. The aim is to eliminate or reduce training and ease the maintenance operation for users [23, 24, 33]. Self-Assist AR applications are of particular interest as they do not require an additional person to guide the support process. Goto *et al.* [10] proposed a task support system by displaying instructional videos on the AR workspace. Petersen *et al.* [25] developed a system for automatic creation of a step-by-step task documentation from a video demonstration for AR and leverage image distance to follow the workflow. Su *et al.* [32] presented a CNN to detect the state and pose of an object in multiple states. Although this work also detects object states, it lacks the fine-granularity required to detect very minor changes such as removed screws or connectors, which are critical in technical support scenarios. YouMove [4] leverages AR to enhance the movement training by using a large-scale augmented reality mirror. Zhu *et al.* [40] proposed a context-aware AR system to assist the operators in maintenance tasks by providing them context relevant information. Although such existing AR based solutions exhibit advantages to traditional approaches, they still lack of fine-grained visual recognition capabilities to enable fully interactive user experiences.

Fine-Grained Visual Recognition. Recognizing fine-grained categories (e.g., car models [17, 20], bird species [6, 38]) is a very challenging problem as it requires the capability of localizing and identifying marginal visual differences. Most existing work [13, 36] rely on human-annotated data sets leveraging bounding boxes to represent relevant features, but this is very difficult to scale. Weakly-supervised part models using CNNs with category labels [18] have no dependencies on bounding boxes for data labeling, hence greatly increase their applicability. Such approaches consist of two steps: part localization by training from positive/negative image patches [37] and then extraction of fine-grained features for recognition. However, these approaches are not directly applicable for our problem set because of the dynamic camera movements, which bring noise to the captured images. Similarly, Region Proposal Network (RPN) [26] does not work well in complex scenarios, such as state recognition for hardware repair, since many parts (e.g., RAM sticks, screws) may look identical, yet a combination of those can represent different state of the hardware. Such RPN-based models lack the ability to focus on specific parts of interest. All existing approaches share the same assumption that the distinguishable parts are adequately captured in input images, which may not always be the case in practice.

3D Object Recognition. Several researchers [15, 16, 31] addressed 3D object recognition by using multiple images for classification based on the 3D ShapeNets [34] dataset. RotationNet [16] estimates the object category and pose jointly using multi-views from unsupervised viewpoints. Edward *et al.* [15] proposed pairwise decomposition of image sequences for active multi-view recognition, which takes an image pair and the relative pose between them as input. MVCNN [31] takes a fixed number of 2D images taken from different angles as input to CNN, and fuses the embedding using a secondary CNN for classification. Similarly, Zhou *et al.* [39] proposed a method that requires the user to scan region of interests one by one, and recognizes the object using a similar neural network design as MVCNN. All these prior work either assume input images taken from fixed viewing angles around the object (e.g., azimuth angles with a step of 30°), or use

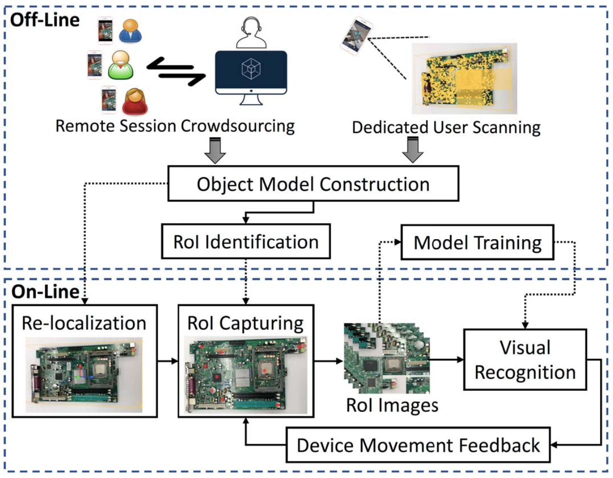

Fig. 3. The system consists of two phases: i) off-line object model construction, RoI identification and model training; ii) on-line re-localization, RoI capturing, visual recognition and device movement feedback.

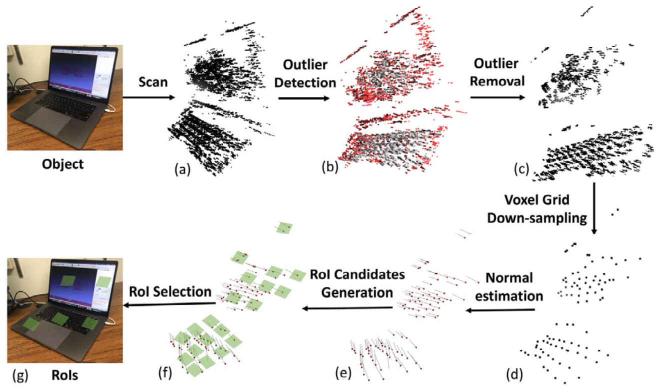

Fig. 4. Steps for object model construction and RoI identification.

random images in a group or pairwise form. Unlike the prior work on common 3D object recognition, which assumes the user knows how and where to take the input images, our system guides the user to capture images from the most informative RoIs, based on the camera pose and the 3D point cloud representing the target object.

3 Overview

We use the acronym FGVR to refer to Fine-Grained Visual Recognition. The initial step of FGVR is to identify Regions of Interest (RoIs) that represent the object state change. We then leverage camera pose tracked in AR to keep the camera focused on RoIs so that adequate video frames are captured for state recognition. Finally, we filter out noise in the frames to improve robustness of the recognition. Figure 3 illustrates the design of the proposed system, which consists of both off-line and on-line components.

In the off-line phase, we first harvest data from our AR remote collaboration sessions [14], or have a dedicated user to scan the object with a mobile device to construct the relevant object models – 3D point cloud representations of the object in different states. We also collect the camera poses, the corresponding feature points and video frames. Next, the RoI extraction module generates a set of RoIs, based on video frames collected in different states. These RoIs will determine what images should be generated from video frames to recognize object states.

In the on-line phase, FGVR first detects the object and re-localizes the mobile device with respect to the object, using the object model. The RoIs identified in the off-line phase are also mapped to the object, as described later in RoI Identification section. Next, we crop the images of these RoIs to keep only the relevant areas of the RoIs, and further process them to train the model for state recognition. During real-time recognition, the mobile app instructs the user to position the camera at the right distance and viewing angle to the object, and applies the trained visual recognition model to predict the current state. Based on the predicted state, the applicable object model is automatically selected for AR tracking, and the corresponding AR instructions are rendered accordingly.

4 Fine-Grained Active Visual Recognition

4.1 Object Model Construction

To construct an object model, we scan the physical object using ARKit's scanner functionality and extract feature points to create the 3D point cloud. The feature points extracted from video frames are accumulated during the course of scanning from different viewing angles. Note that there is a trade-off between the accuracy and computational overhead of AR tracking, determined by the density of feature points in the point cloud. In our solution, we leverage relatively sparse point clouds for AR tracking, and denser point clouds for RoI identification, as described in the next step.

We construct separate sparse and dense point clouds for physical objects in different states. This can be accomplished if the user knows *a priori* the different states of an object, and performs a separate scan in each state. Point clouds representing different states can also be obtained implicitly from our AR-driven remote collaboration sessions if the users explicitly mark the state change verbally or manually during the session. Our point cloud generation algorithms take images corresponding to each implicit state as input, and generate the relevant point clouds.

In practice, the point clouds generated may have a lot of noisy feature points due to, for example, depth estimation inaccuracy. We further filter the feature points by removing the "non-smooth" outliers from the neighborhood of each feature point. Feature points at each frame are represented as $P = \{p_0, p_1, \dots, p_N\}$, where $p_i = \{x_i, y_i, z_i\}$ is the location of a point i in 3D space, and N is the total number of points. Specifically, for each point p_i, we compute its mean distance to all its k nearest neighbor points $q_j (j = 1, 2, \dots, k)$ as $\bar{d_i} = 1/k \cdot \sum_{j=1}^{k} dist(p_i, q_j)$, and the standard deviation as $\sigma = \sqrt{\frac{1}{k-1} \sum_{j=1}^{k} \left(d_j - \bar{d_i}\right)^2}$, where $dist(p_i, q_j)$ is the euclidean distance between point p_i and q_j. Assuming the distribution of distance to neighbors is Gaussian: $\mathcal{N}\left(\bar{d_i}, \sigma^2\right)$, all the points with distances $d > \mu + \alpha \cdot \sigma$ are considered outliers, thus removed. Here α is a parameter that controls the "smoothness" of the point cloud surface. In our implementation, we set $k = 50$ and $\alpha = 1.0$. As shown in Figure 4, we first detect outliers, marked in red in (b), then derive the clean point cloud in Figure 4(c).

After this step, we obtain the point clouds that robustly represent the object in different states. For each state, a dense point cloud with more feature points is generated for RoI identification, while a down-sampled, sparse point cloud is created for efficient tracking. The point clouds generated in different states can be easily aligned in the same 3D coordinate system since they use the same sparse point cloud for re-localization. ARKit re-localization has a certain tolerance to appearance or shape changes, thus re-localization works well as long as object changes are partial or minor.

4.2 RoI Identification

RoI is a segment in the 3D space where an object's physical appearance changes due to the state change. To visually recognize the changes in an RoI, we project the RoI to a region in the 2D image taken from a certain viewing angle of the object. The simplest way to identify an RoI is to rely on human knowledge: an expert user can draw the boundaries of RoIs on given images. This is tedious in practice, and can easily lead to inconsistency in RoI or state definitions due to different human interpretations. Hence, we developed an approach to automatically identify the RoIs, given a set of images and point clouds labeled with different states, as discussed above.

4.2.1 Voxelization

We segment the 3D space into voxels of a fixed size. An RoI consists of one or multiple voxels, which may contain some (or none) feature points. To identify the RoIs, we need a consistent way to project voxels onto 2D images. For that, we define the anchor point for a voxel to be the centroid of all features points contained in this voxel. When projecting the voxel, this anchor point is first projected to the 2D image. Centered around this anchor, a square area with width w is cropped out of the image to represent the corresponding voxel. This ensures that the projected image contains sufficient visual details of the object surface. Figure 4(d) shows the anchor points for all voxels of a sample object.

To guarantee that sufficient visual details are collected for a voxel, we need to ensure that the object appears in the camera's field of view. This is enabled by estimating the normal of the anchor point, with respect to the object surface. We estimate the normal vector at the anchor point by calculating the normal of the plane tangent to the object surface at the anchor point. This can be achieved by least-square plane fitting, using libraries such as PCL [28]. Examples of the estimated normal vectors are shown in Figure 4(e).

4.2.2 RoI Image Selection

Given camera-captured images of an object, we crop out the RoI candidate images that can serve as input for object state recognition. An RoI candidate image is a square segment cropped around the voxel anchor at $\{x,y,z\}$ with width w. We project the voxel anchor and all vertices of an RoI candidate to the camera-captured image, and choose the minimum-bounding rectangle that covers the projected RoI as the cropping area. Given the camera pose (represented as a transformation matrix M_{camera}) and the coordinates of a point in 3D space P_{world}, we can project this point to be in camera coordinates with:

$$P_{camera} = P_{world} \cdot M_{camera} \tag{1}$$

where P_{camera} is the projected point in camera coordinates. We use perspective projection to project the point in camera coordinates to the image plane by a simple division of the point's x and y coordinate by the z coordinate:

$$\begin{aligned} P' \cdot x &= \frac{P_{camera} \cdot x}{-P_{camera} \cdot z} \\ P' \cdot y &= \frac{P_{camera} \cdot y}{-P_{camera} \cdot z} \end{aligned} \tag{2}$$

Then, we convert the 2D point in image space to raster space, which is represented as pixel coordinates:

$$\text{visible } = \begin{cases} \text{yes} & |P' \cdot x| \leq \frac{W}{2} \text{ or } |P' \cdot y| \leq \frac{H}{2} \\ \text{no} & \text{otherwise} \end{cases} \tag{3}$$

$$\begin{aligned} P'_{norm} \cdot x &= \tfrac{P'.x + \text{width}/2}{\text{width}} \\ P'_{norm} . y &= \tfrac{P'.y + \text{height}/2}{\text{height}} \end{aligned} \tag{4}$$

where W, H are the width and height of the canvas in raster space, and $P'_{norm} \cdot x$, $P'_{norm} \cdot y$ are normalized coordinates in raster space, which are further multiplied by the resolution of the image frame so that we can get the pixel coordinates. An RoI is within the FoV only when the four projected vertices are visible. The images for each visible RoI are cropped from the raw full resolution image frame and resized to a fixed resolution. Multiple RoIs can be cropped simultaneously, as long as they are within the FoV and visible to the camera in one single frame. In Figure 4(f), we show a few samples of the RoI candidate images after the cropping steps described above have been applied.

Next, we select from these RoI candidates the ones that can most differentiate the object states. We use Gradient-weighted Class Activation Mapping (Grad-CAM) [29] to perform RoI image selection. For each state of the object, we have collected images from different viewing angles. We crop each image to obtain the RoI candidate images. According to location of the corresponding voxels, we sort and concatenate the cropped images into single images and resize them to a fixed resolution (e.g., 224×224) as input data to Grad-CAM, as shown in Figure 5(a). For each state, a subset of these candidate images reflect the appearance change of the object.

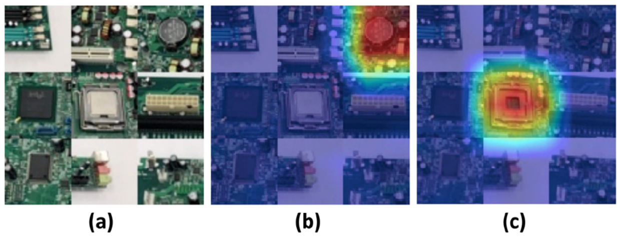
Fig. 5. Visualization of Grad-CAM map overlaid on the RoI image.

Deeper representations in a Convolutional Neural Network (CNN) capture higher-lever visual features [22]. State-specific information in the input image (i.e., from voxels with appearance change) are usually captured in the last few convolutional layers that have high-level semantic and spatial information. Grad-CAM uses the gradient information flowing into the last convolutional layer of the CNN to understand the importance of each candidate image [1]. Our goal of RoI selection is to class-discriminate localization map $G^c \in \mathbb{R}^{u \times v}$ of width u and height v for any class c. For this, we compute the gradient of a class probability y^c with respect to feature maps A^k of a convolutional layer, i.e., $\frac{\partial y^c}{\partial A^k}$. So the weights α_k^c can be global-average-pooled as:

$$\alpha_k^c = \frac{1}{Z} \sum_i \sum_j \frac{\partial y^c}{\partial A_{ij}^k} \tag{5}$$

where Z is the normalization factor and weight α_k^c captures the importance of the feature map A for a target class c. Following the same method in [29], the Grad-CAM heat-map is a weighted combination of feature maps, which is followed by a ReLU [2]:

$$G^c = \text{ReLU}\left(\sum_k \alpha_k^c A^k\right) \tag{6}$$

This results in a coarse heat-map of the same size as the convolutional feature maps (7×7 in the case of last convolutional layers of ResNet50 [11]). Then, the heat-map is normalized to 0-1 and resized to the size of input image, which is visualized by overlaying it on the input image, as shown in Figure 5.

We adopt a pre-trained ResNet50 on ImageNet [8] as the backbone model and fine-tune it to classify the concatenated images from all the states, and then generate the Grad-CAM maps. The Grad-CAM maps directly indicate the importance of each candidate image for distinguishing a specific state. As Grad-CAM is a relatively well-established algorithm, we did not directly evaluate it in terms of its heat map generation accuracy. However, its high performance is evident from the resulting state detection accuracy, as discussed later in Section 6.1. Figure 5(a) shows the concatenated image of a subset of RoI candidate images, collected for a computer motherboard. In this example, the object has three states: *default*, *battery removed*, *CPU removed*. The cropped image indicating *battery removed* is in the up right corner of the image concatenation, while that for *CPU removed* is at the center of the concatenation. Figure 5(b) shows the Grad-CAM heatmap overlaid on the image, which correctly indicates the two cropped images mentioned above are the best RoI images for differentiating these three states.

We build a super set of all the top 3 RoI candidates for each state selected from Grad-CAM and remove the duplicates. This new set of RoIs are used for final visual recognition model training and inferences during the on-line phase.

[1]We also tried other methods such as SIFT [21] and ORB [27], both of which yield significantly worse performance in identifying RoIs as such methods are searching for similarities, instead of identifying minor visual differences.

4.3 Visual Recognition Model

With the concatenated RoI images as input, we can now develop CNN models to predict the class or state of an object. For simple scenarios, where an object only has a small number of RoIs (e.g., less than 6x6 concatenated RoI images), a light-weight, shallow CNN with max pooling and batch normalization is sufficient. As the application scenario becomes more complex, more RoIs are needed to cover all states. In such cases, naively concatenating all RoI images into a single input causes problems: the resolution for each RoI reduces as the number of RoI increases as the concatenated image resolution is fixed, which leads to loss of visual details. If we retain the resolution of each RoI, the concatenated image resolution will quickly grow to be not suitable as input to the simple CNN model.

To deal with this challenge, we propose a discrete-CNN-LSTM model, which contains layers of discrete multi-branch visual feature extraction, followed by layers of spatial and temporal information aggregation for the final classification. Figure 6 shows the architecture of the DCL model. The feature extraction module consists of multiple feature extraction branches, each of which take concatenated RoI images (containing a maximum of 9 RoIs) as input. We use a pre-trained ResNet50 on ImageNet as the base model for feature extraction, but we remove the last fully connected layer of the model, thus getting a 2048-dimensional feature vector for each input image, which is then fed to a pooling layer. The ResNet50 inference only runs whenever there is an RoI image update for the concatenated input image. The image features extracted for other unchanged branches are reused. In this way, we minimize the computation overhead on feature extraction, thereby significantly reducing the computation overhead and power consumption when the model is implemented on mobile devices.

We concatenate all extracted image features after the pooling layer into a single feature vector, which presents all critical visual information for state recognition. Instead of feeding this feature vector into a fully-connected deep neural network like MVCNN [31], we use LSTM to model the temporal relationship between states: the temporal sequence of the feature vectors is fed to the LSTM layer, which predicts the object state distribution. As we shall see in later sections, leveraging the temporal relationship between states significantly improves the prediction accuracy of our model.

4.4 Active Visual Recognition

The visual recognition model described above is trained offline, using the labeled data collected a priori. When building the mobile application to detect an object's state, we not only need to implement the trained visual recognition model, but need to consider two more design aspects: *re-localization* and *RoI capturing*.

4.4.1 Re-localization

Re-localization ensures the camera pose is tracked consistently across AR sessions, with respect to fixed object coordinates. This is critical for both capturing the correct RoIs and presenting AR annotations in the right positions relative to the object. We use the sparse point cloud obtained during reference object model creation for re-localization, leveraging the object tracking capability of ARKit. In a given AR application, there are two potential state change scenarios: 1) the state change involves minor appearance change, hence the same sparse point cloud can be used for tracking; 2) the state change involves major appearance change, so that a new point cloud needs to be loaded and AR session re-localizes.

In our design, we first use a coarse-grained recognition module to detect if there is a major appearance change (e.g., cover open vs. closed) as shown in Figure 7. For this, we adapt a ResNet50 model trained on the images from data collection, which uses real time images taken from the object to detect major changes, and triggers re-localization only when it is necessary. If the appearance change is minor, neither ARKit nor the ResNet50 classifier is sufficient to detect the state of the object, hence the need for the fine-grained recognition step shown in Figure 7.

4.4.2 RoI Capturing

Previously, we described the basic idea of capturing RoIs as image segments cropped from camera view. To capture the RoI images in a robust, online manner, we need to further address the following problems: *occlusion detection*, *RoI image update*, and *device movement feedback*.

Occlusion Detection. A RoI can be only visible to the camera from certain viewing angles, or it can be occluded by other objects in the scene. Figure 8 shows such an example. The camera pose in Figure 8(a) yields valid capturing of the RoI, as indicated by the red circle, while for the camera in Figure 8(b), the RoI is occluded from the camera's view point.

A naive approach is to check if sufficient 3D feature points for a RoI are detected on each frame, so as to determine if the RoI is occluded. However, due to computation overhead, feature points are typically not updated in real time for every frame. As the mobile device moves around, the device motion is updated at a high frequency to capture the minor displacement. The feature points are then triangulated to infer their 3D positions, which are also used for calibrating the camera pose error by ARKit. So there exists a delay between the feature points detected and the current frame. As the device is moving from "visible pose" to "invisible pose" (e.g., from the pose in Figure 8(a) to (b)), the features detected in (a) also exist in (b) for a short time. To account for this latency, we examine the past consecutive 5 frames as well as the feature points detected in the corresponding time period, to determine whether the RoI is occluded.

To determine occlusion for a certain RoI, the following three conditions must be satisfied: *i) Within the camera FoV:* The projected RoI must be within the current video frame, so that it is ready to be updated. *ii) No sufficient feature points detected:* Sufficient feature points must be detected for RoI capturing. If no sufficient feature points (e.g., $<$ 10) are detected for a short period of 3 seconds, it indicates occlusion happens. We use the accumulated feature points for a short period to avoid false positives, because the feature points may be too sparse even when the RoI is not occluded. *iii) Sufficient device movement:* Because feature points rely on triangulation, which requires device movement, the feature points of a frame will remain unchanged or no feature points are generated if the mobile device remains stationary. Without sufficient feature points, it is not possible to tell whether the RoI is occluded.

RoI Image Update. When predicting object state online (i.e., while using the AR application), the visual recognition needs input images from all RoIs, except for those that have not changed visually. Meanwhile, as the camera moves around an object, not all RoIs are visible to the camera, either because some RoIs are outside the camera's FoV or because they are occluded. The occlusion has two types: *permanent occlusion*, which means an RoI is occluded by other object components permanently in an object state (e.g., a fan covers a CPU); and *transient occlusion*, which is caused by inappropriate viewing angle or temporarily occluded by other objects such as user's hand. For a specific RoI, if it is continually detected as occluded for a long time (we set an empirical 5 s in current design) when the device is actively moved by the user while the RoI is within FOV, we classify the occlusion as permanent. In such cases, the image for this RoI needs to be reset to zeros for consistency (the state of "Fan is ON" in Figure 2(a) can either happen before or after the CPU is captured, depending whether the machine is being assembled or disassembled). Similarly, when a component is removed, we also need to reset the associated RoIs. This is detected if such RoIs are within the FOV, but not detected for a period when they are not occluded.

If transient occlusion is detected or invisible RoIs are outside the FoV, we assume there is no change visually. Therefore, during model prediction, the corresponding feature extraction branches in Figure 6 will not be activated, but will reuse the previously extracted vectors, minimizing the unnecessary computations on the mobile device to prevent device heating. Usually it is natural for the user to view the object changes through camera where AR annotations are displayed, thus such changes are captured passively. In rare cases, it is possible that user has disassembled a part of the hardware which altered its

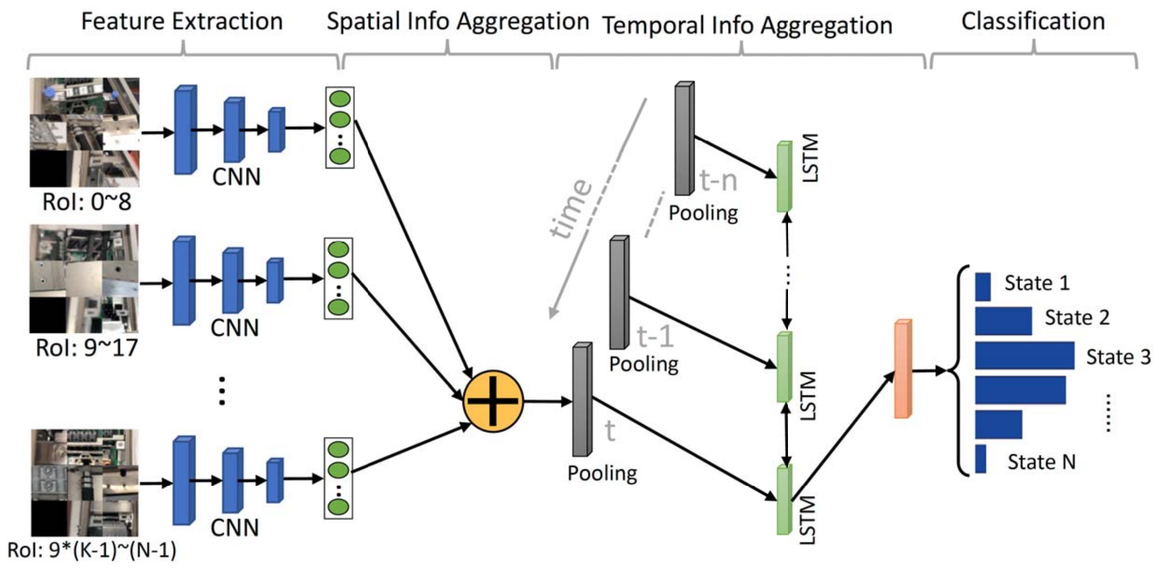

Fig. 6. The multi-branch CNN extracts spatial features from concatenated RoI images, which are aggregated by a pooling layer. An LSTM layer further aggregates the temporal information for stable predictions.

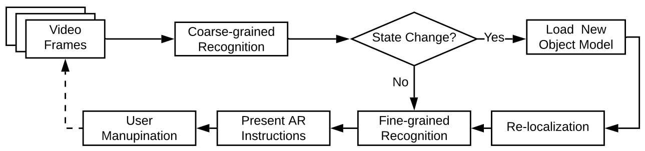

Fig. 7. The pose independent coarse-grained recognition selects the correct object model for re-localization, which enables fine-grained recognition.

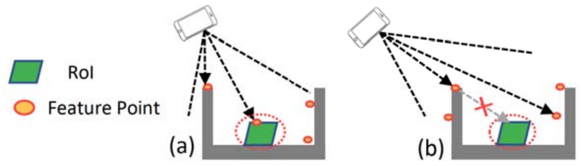

Fig. 8. RoI occlusion example.

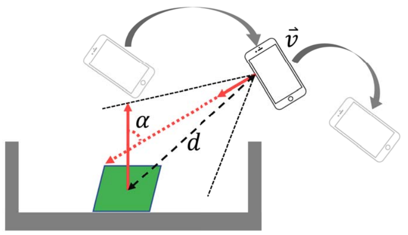

Fig. 9. RoI optimization evaluates the device's moving speed, viewing angle and distance to select the optimal captured RoI image.

visual state, but if it is occluded or outside of the FoV, this will not be considered a state change, and therefore the next set of instructions, will not be provided to the user. Thus, our app encourages the users to move the camera so as to view the altered object parts such that the RoIs can be efficiently updated, enabling responsive and accurate recognition results.

When the user holds the camera over an object, the captured video frames are likely to have varying quality (e.g., blurred images due to fast motion, unclear images due to the viewing angle, or large distance between the camera and object). To ensure robust recognition, we need to further filter out the RoI images to extract the best-quality image set as input for the visual recognition model. As shown in Figure 9, for each RoI we keep track of device movement and rotation speed $\vec{v}$, the angle between the camera normal and the RoI normal a, and the distance from camera to RoI d. Given a camera pose $\{x_p, y_p, z_p, n_{xp}, n_{yp}, n_{zp}\}$ where $\{x_p, y_p, z_p\}$ is the camera position and $\hat{n}_p = \{n_{xp}, n_{yp}, n_{zp}\}$ is the camera normal, and a RoI $\{x_r, y_r, z_r, n_{xr}, n_{yr}, n_{zr}, w_r\}$, the distance from the camera to a RoI can be calculated as $d = \sqrt{(x_p - x_r)^2 + (y_p - y_r)^2 + (z_p - z_r)^2}$ and the angle between the normals of camera and RoI is $\alpha = \frac{\hat{n}_p \cdot \hat{n}_r}{|\hat{n}_p| \cdot |\hat{n}_r|}$. The camera's moving/rotation speed is estimated with the camera pose when a frame is captured p_t and the pose at ΔT earlier $p_{t-\Delta T}$ (ΔT is set to 0.1 s in our design). The absolute combined moving/rotation speed can be calculated as $|\vec{v}| = |(p_t - p_{t-\Delta T})$. Note that we normalize the moving speed and rotation speed to balance their contribution to the combined speed before the calculation. We maintain a queue of 5 input images for every RoI, and they are updated every 300 ms if eligible RoI is captured by popping out the first RoI and pushing the new RoI. We choose 5 images because it only requires a small time window of 1.5 s and it minimizes the impact of images captured when the RoI is going out of FOV. We feed an RoI image to the queue only if $|\vec{v}| < \delta_v$, $\alpha < 60°$ and $d < 1m$, where δ_v is a threshold so that images are not blurred when device is moving within this threshold [2]. To select the optimal RoI, we choose the one with smallest α as it impacts the performance most for the eligible RoIs according to our experiments. Without such queuing and selection, the last update for an RoI usually happens when it's about to exit the FoV, which is not optimal.

Device Movement Feedback. To ensure good-quality RoI capture, the device camera needs to be properly positioned. We provide feedback through the AR app to guide the user to position the camera such that it is at the right position and that they are not moving the camera too fast or too far away from the RoIs. The feedback instructions are shown only when RoI images are not adequately captured. Specifically, if the RoIs are not updated for a period of time (e.g., 10s), which means they are not viewed from appropriate angles or distance, then appropriate instructions will be displayed to ask the user to move the mobile device. On the other hand, recognition results are ambiguous, we instruct the

[2]These are empirical numbers we get from extensive experiments, δ_v may be different under different lighting conditions.

user to move the device toward collecting more RoI images, which are associated with such ambiguous states from RoI identification, until the object state is disambiguated.

5 Implementation

We implemented FGVR in two parts: an *off-line model training* program that runs on the server, and an *on-line mobile AR app* that leverages the extracted RoIs and the trained model to perform fine-grained visual recognition.

5.1 Off-line Model Training

Training Data Generation. The RoI cropping during the training data collection produces a bunch of images for each RoI under each object state, which are concatenated as the training image for model training. To minimize the training data collection effort, we only require a few minutes of object scanning. We then further enhance the collected data in two steps:

- *Data Augmentation.* We first augment the RoI images leveraging existing approaches such as random rotation, shifts, and shear. Random rotation of the RoI images enables much robust recognition performance when the object is viewed from random angles. We also shift and shear the RoI images, because the captured RoI image may not be centered and may be off-center in a variety of different ways due to the off-set in RoI tracking caused by device pose estimation error. With such augmentation, we populate the images for each RoI under each object states.

- *Image Concatenation.* To generate the images that are consumable for model training, the individual images for each RoI need to be concatenated. Since these RoI images capture the information of the corresponding RoIs, and they may be captured at different times, and thus they are semi-independent of each other. To augment the concatenated images, we randomly draw the images from each RoI image set, and concatenate them as a new training sample. This significantly increases the training data amount, and also captures the variety of the combinations of RoI images.

We use the same techniques to generate the training data for both RoI extraction, and the final fine-grained visual recognition model training.

Model Training. To collect training data, we scan the object at each state for approximately 2 *mins*. Note that it is a one-time effort to build the model for a new object. For evaluation purposes, we run this training program off-line on a PC with GTX 1080 GPU. Keras [7] with Tensorflow backend is used for CNN construction and training. After the training, we convert the visual recognition model to CoreML [5] format to run on iOS platform.

5.2 On-line Mobile App

We developed an iOS-based mobile application for real-time testing and evaluation. ARKit is used as AR SDK for re-localization, camera pose tracking and AR experience rendering. CoreML framework is used for executing on-device machine learning inferences. We schedule a task for real-time recognition and stabilize the result via multiple trials:

- *Recognition Scheduling.* We schedule a timer to check whether the RoIs are updated every 300 *ms*, and execute the CNN feature extraction branches, whose input image is updated. We execute the spatial and temporal information aggregation and classification layers once per second if any CNN feature update is triggered. By updating each branch only when RoI changes, we reduce unnecessary computations.

- *Prediction Stabilization.* To further stabilize the state prediction result, especially during the transitions from one state to another, we leverage a rolling average prediction method. We buffer a queue of latest N prediction results, and only output a final result if one predicted state appears N-1 times out of N.

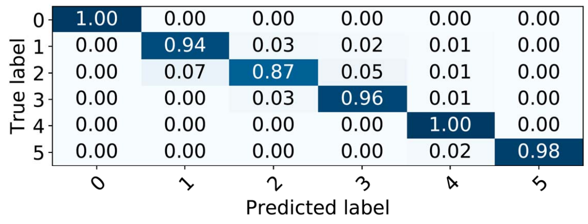

Fig. 10. Confusion matrix of state recognition results.

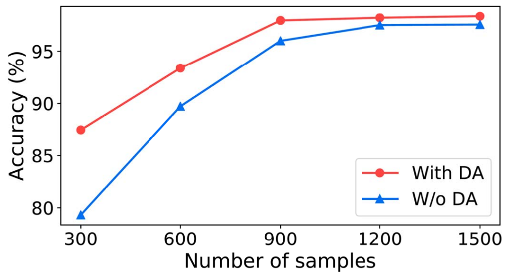

Fig. 11. Recognition accuracy comparison of data augmentation (DA) with different number of training images.

6 Evaluation

We evaluate our FGVR algorithm from multiple aspects: recognition accuracy and impact factors, comparison with existing approaches, and resource overhead.

6.1 Recognition Accuracy and Impact Factors

To validate the overall system design, we first evaluate the performance of a naive light-weight CNN model (LW model) with our RoI capturing mechanism, then followed by a comparison against DCL and VGG-based models. LW model is a two-layer CNN followed by pooling and fully connected layers for classification.

6.1.1 State Recognition Accuracy

We take the task of replacing a CPU on a desktop machine, which involves 6 steps (e.g., removing screws/fan, unplugging wires, releasing CPU latch, etc.) as our target task in this evaluation. We scan the machine under each state for ~ 2 *mins* as training data and another ~ 1 *min* for testing. Video frames are sampled every 300 *ms*, thus 1 $min's$ data collection captures ~ 200 image frames. We extract the RoIs and train the LW model for evaluation. Figure 10 shows the confusion matrix of the results. The recognition accuracy is close to or above 95% for most states, except state 3, which is mis-recognized as adjacent states. This state is easier to be confused with adjacent states, because the visual difference is very small: the fan collector hides deep in the machine, and it is only visible from a very limited viewing angles. Note that these results are based on individual recognition results. We found that error results are usually not adjacent to each other, thus we can further improve the accuracy and stability with multiple trials, which is evaluated later. Nevertheless, the overall results clearly demonstrated that FGVR is robust enough for practical use.

6.1.2 Factors Affecting Model Accuracy

Data Augmentation. We evaluate how effective data augmentation (DA) can improve the performance by populating the training images, especially when training data is limited. Figure 11 shows the recognition accuracy under different amounts of training samples from 300 to 1500. It is obvious that data augmentation improves the accuracy significantly, especially when the training samples are very limited (e.g., < 900). As the size grows, the accuracy performance becomes stable both with and without DA, although still slightly higher with DA indicating the model generalization improvement of DA.

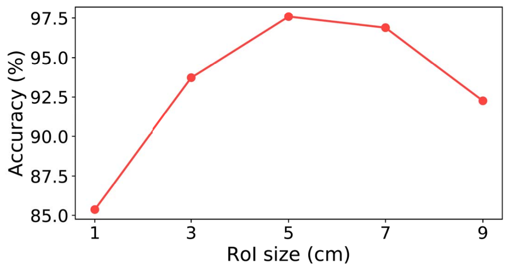

Fig. 12. Recognition accuracy with different RoI sizes.

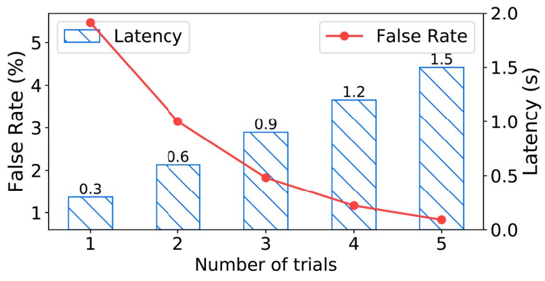

Fig. 13. False rate and latency with different number of trials.

RoI Size. The RoI size (the length/width of the square) is critical for achieving the best recognition performance. Smaller RoI size enables more focused perception of object parts, thus more fine-grained recognition, but also leads to smaller coverage area, which may miss the distinguishable parts due to offsets caused by device movements. In such cases, it produces larger errors. Larger RoI size enables larger coverage, thus less possible to miss the distinguishable parts, but at the same time, it also captures more background "noise" and shrinks the resolution of critical areas for state recognition. We evaluate the impact of different RoI sizes on the recognition accuracy. Figure 12 shows the resulting accuracy with RoI size from 1-9 *cm*. We collect data with RoI size 1,3,5,7,9 *cm*. The experiment shows 5 *cm* to be the optimal size as it achieves the highest accuracy. Depending on the actual size of the object, the optimal RoI size may differ. Dynamically choosing RoI size will be future work.

Recognition Stabilization. It is critical to have stable and accurate recognition results as any incorrect recognition would trigger irrelevant animations or instructions in the AR application. This is not only disruptive but also confuses users, leading to a poor user experience. We evaluate the false rate (i.e., inaccurate state recognition) and latency when multiple recognition trials are conducted for each cycle. Recognition trial happens every 300 *ms* thus one verdict from multiple trials is fast enough, causing no obvious delay to the user. At least $N-1$ trial must indicate the same result out of N trials in a cycle to declare state result. Figure 13 shows that more trials lowers down the false rate rapidly at the cost of larger latency. This is because more trials give the more stable results, thus reducing false results. With 5 trials, the false rate is less than 0.5%. We choose 5 trials for each recognition circle to balance the stability and latency. Note that in practical use case, the object states do not change frequently, 1-2 *s* latency is negligible.

6.2 Comparison with Existing Approaches

Accuracy Comparison. We compare the performance of LW, DCL models and a representative object recognition model VGG16 [30]. As RoI capturing is a key component for FGVR , we also compare the performance of all models with RoI capturing and raw image frames. We fine-tune the VGG16 with pre-trained weights from ImageNet [8], and freeze all the layers except the last two fully connected layers while training. Figure 14 show the results. With RoI capturing, all the three models achieved high accuracy ($> 97\%$), and DCL model has the highest accuracy of 99.87%. The VGG-based model also has an accuracy of 97.74%, which demonstrates the effectiveness of the

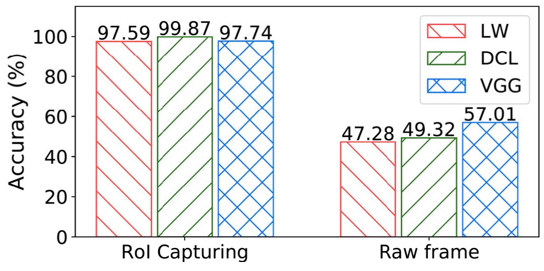

Fig. 14. Accuracy of different models with RoI capturing or using raw image frames.

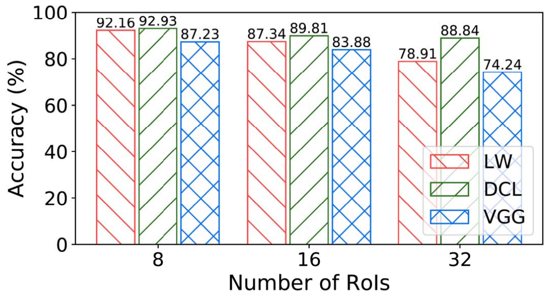

Fig. 15. Comparison of accuracy of different models, as the object becomes more complex, requiring more RoIs.

features extracted by VGG16. The results with RoI capturing are significantly higher than those using raw image frames as input, which achieve only $\sim 50\%$ accuracy. This is mainly because of the occlusions and the lack of fine-granularity. The images may look the same for some states if the changing parts are occluded, thus causing a lot of noises in the training and testing data. VGG-based model has the highest accuracy among the three using raw image frames, even better than DCL model. This is probably because DCL model takes multiple "noisy" frames as input, which is more difficult to converge.

Impact of Number of RoIs. As the number of RoI increases, if the resolution of each RoI does not change, the concatenated image resolution increases. If the concatenated image resolution is fixed, each RoI image resolution shrinks. Image with too large resolution may need to be reshaped to be fit into models such as VGG. We collect new data on a more complex object, and we manually added more RoIs. Figure 15 show the results when we have 8, 16, and 32 RoIs. As the number of RoI grows, VGG model accuracy drops to $< 75\%$ due to the resizing. LW has better performance than VGG at a cost of increased model size (increasing parameters). The DCL model has the best performance and only show marginally decrease in accuracy.

Model Size Comparison. The size of a model is critical for it to be executed efficiently, especially on a resource constrained mobile device. Figure 16 shows the number of parameters of the three models, as the number of RoI increases. The VGG-based model has a constant number of ~ 14.85 *M* as the images are resized to a fixed resolution of 224×224. The DCL model leverages VGG for feature extraction, thus has more parameters than VGG. The parameter increases by ~ 1.6 *M* every additional 9 RoIs (assuming the input image for each branch consists of 9 RoIs). The LW model has the least parameters when RoI is fewer than 25, after that it grows significantly faster then DCL model. Thus, DCL is more suitable for complex objects.

6.3 Generalization of on More Objects

Our visual recognition system is built on top of an in-production AR-based technical support system with a large volume of users. The pilot tests consist of experiments on several products, which include enterprise-level servers, ATM cash dispensers, receipt printers, laptops and PC motherboards. The tasks cover server CPU replacement, cash dispenser disassembly, refilling paper of receipt printers, laptop repair

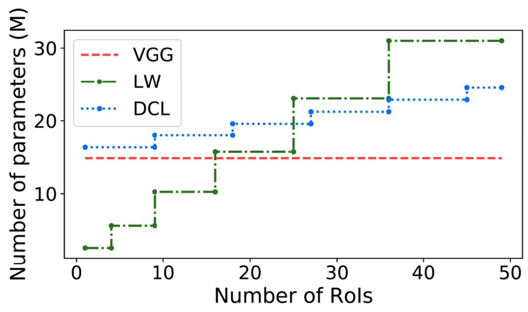

Fig. 16. The total number of parameters of different models as the number of RoI increases.

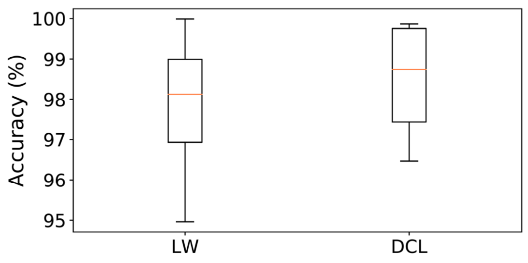

Fig. 17. The accuracy of LW and DCL models on 5 objects.

(e.g., wireless card replacement, SSD removal, battery replacement), etc. The total testing consists of 30+ procedures and there are 40+ RoIs are involved, which generates ~ 7500 recognition results (recognition result is reported every 2 seconds). Figure 17 shows the boxplot of the accuracy of both LW and DCL models, both of them are above 95%. The DCL model has a higher median accuracy over 98%.

6.4 Resource Overhead

The resource consumption is critical for good user experience and usability. We measure the FGVR resource overhead on an iPhone 8, in CPU and memory usage. We compare these metrics when the mobile app is running in different modes: AR only without visual recognition, AR with LW model inference, and AR with DCL model inference. Table 1 shows the results of typical measurements for comparison. The visual recognition only slightly raises the CPU by $\sim$ 1-1.5%, and consumes 30+ *MB* memory. With all the overheads, FGVR can easily achieve 60 fps, causing no overheating for continuous use.

Mode	CPU	Memory
AR	7.16%	134.9 *MB*
AR + LW	8.33%	168.4 *MB*
AR + DCL	8.67%	178.9 *MB*

Table 1. Resource consumption under different modes.

7 Discussion

Model Training Overhead. The model training overhead includes two aspects: data collection overhead and computation overhead. For data collection, we either have a dedicated user to scan the object under each state (each takes $\sim 1-2$ *mins*) or leverage crowdsourced data from remote assist sessions. In its current state, the crowdsourced data is noisy and requires manual pre-processing. Knowledge induction and automatic model training from such noisy data is one of our future work goals. For computation overhead, all the data collected on the mobile device is a "byproduct" of AR, thus causing minimum additional overhead. Model training is also light-weight in the backend due to the relatively small data amount for a given object.

More Sophisticated Models. The DCL model in our current design leveraged VGG16 with pre-trained weights from ImageNet as the base model, which yields reasonable results. Training a base model based on domain specific data set could further improve the accuracy and efficiency. Additionally, more sophisticated neural networks, such as spatial temporal graph convolutional networks, are promising to improve the performance [35].

Limitations. We discuss the limitations of current design and future directions.

1) Feature-less object appearances. The current design has shortcomings when dealing with objects with limited feature points, such as objects with large glossy surfaces. In such cases, AR tracking usually becomes unreliable as well, so this limitation is not specific to our work alone. For hardware repair scenarios, however, the focus areas tend to be inside the machines, which are usually very rich in features, and therefore, this limitation does not impact the utility of our system in technical support settings. Improving the robustness on feature-less objects will be our future work.

2) Minor vs. major appearance changes. In current design, we rely on the content creator to determine whether the appearance changes between steps are minor or major during data collection to prepare the data for coarse-grained state recognition model training, and create an AR reference object model after each major change. We would like to explore automating this process as future work to eliminate the need for input from content creators.

8 Conclusion

In this paper, we proposed an Active Fine-Grained Visual Recognition algorithm that combines both 2D video frames and 3D feature points in AR to recognize the changing visual state of 3D objects, even when the change is fairly subtle. We showed that it is robust to random camera movements, and provides reliable and stable visual recognition results by implementing and testing it through an iOS-based mobile application for hardware maintenance. As the next step, we plan to run user studies to test the effectiveness of state detection in terms of speeding up user tasks and reducing errors during hardware repair process.

Acknowledgments

We would like to thank Rohan R. Arora, Hui Kang, Noah Zheutlin, Gerard Vanloo, and Ever Olivas Orduna for the technical discussions and creating the 3D content for the demo video recording.

References

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pp. 265–283, 2016.

[2] A. F. Agarap. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*, 2018.

[3] H. A. Alhaija, S. K. Mustikovela, L. Mescheder, A. Geiger, and C. Rother. Augmented reality meets deep learning for car instance segmentation in urban scenes. In *British machine vision conference*, vol. 1, p. 2, 2017.

[4] F. Anderson, T. Grossman, J. Matejka, and G. Fitzmaurice. Youmove: enhancing movement training with an augmented reality mirror. In *Proceedings of the 26th annual ACM symposium on User interface software and technology*, pp. 311–320, 2013.

[5] Apple. Apple developer documentation. `https://developer.apple.com/documentation`, 2018.

[6] S. Branson, G. Van Horn, S. Belongie, and P. Perona. Bird species categorization using pose normalized deep convolutional nets. *arXiv preprint arXiv:1406.2952*, 2014.

[7] F. Chollet et al. Keras. `https://keras.io`, 2015.

[8] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255. Ieee, 2009.

[9] Google. Google arcore sdk. `https://developers.google.com/ar/`, 2019.

[10] M. Goto, Y. Uematsu, H. Saito, S. Senda, and A. Iketani. Task support system by displaying instructional video onto ar workspace. In *2010 IEEE International Symposium on Mixed and Augmented Reality*, pp. 83–90. IEEE, 2010.

[11] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[12] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[13] S. Huang, Z. Xu, D. Tao, and Y. Zhang. Part-stacked cnn for fine-grained visual categorization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1173–1182, 2016.

[14] IBM. Ibm augmented remote assist. `https://apps.apple.com/us/app/ibm-augmented-remote-assist/id1451080264`, 2020.

[15] E. Johns, S. Leutenegger, and A. J. Davison. Pairwise decomposition of image sequences for active multi-view recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3813–3822, 2016.

[16] A. Kanezaki, Y. Matsushita, and Y. Nishida. Rotationnet: Joint object categorization and pose estimation using multiviews from unsupervised viewpoints. *arXiv preprint arXiv:1603.06208*, 2016.

[17] J. Krause, H. Jin, J. Yang, and L. Fei-Fei. Fine-grained recognition without part annotations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5546–5555, 2015.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[19] Y. LeCun, Y. Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[20] T.-Y. Lin, A. RoyChowdhury, and S. Maji. Bilinear cnn models for fine-grained visual recognition. In *Proceedings of the IEEE international conference on computer vision*, pp. 1449–1457, 2015.

[21] D. G. Lowe. Method and apparatus for identifying scale invariant features in an image and use of same for locating an object in an image, Mar. 23 2004. US Patent 6,711,293.

[22] A. Mahendran and A. Vedaldi. Visualizing deep convolutional neural networks using natural pre-images. *International Journal of Computer Vision*, 120(3):233–255, 2016.

[23] C. Nakajima and N. Itho. A support system for maintenance training by augmented reality. In *12th International Conference on Image Analysis and Processing, 2003. Proceedings.*, pp. 158–163. IEEE, 2003.

[24] M. Neges, C. Koch, M. König, and M. Abramovici. Combining visual natural markers and imu for improved ar based indoor navigation. *Advanced Engineering Informatics*, 31:18–31, 2017.

[25] N. Petersen and D. Stricker. Learning task structure from video examples for workflow tracking and authoring. In *2012 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 237–246. IEEE, 2012.

[26] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pp. 91–99, 2015.

[27] E. Rublee, V. Rabaud, K. Konolige, and G. R. Bradski. Orb: An efficient alternative to sift or surf. In *ICCV*, vol. 11, p. 2. Citeseer, 2011.

[28] R. B. Rusu and S. Cousins. 3d is here: Point cloud library (pcl). In *IEEE international conference on robotics and automation*, pp. 1–4. IEEE, 2011.

[29] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 618–626, 2017.

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

[31] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *Proceedings of the IEEE international conference on computer vision*, pp. 945–953, 2015.

[32] Y. Su, J. Rambach, N. Minaskan, P. Lesur, A. Pagani, and D. Stricker. Deep multi-state object pose estimation for augmented reality assembly. In *2019 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*, pp. 222–227. IEEE, 2019.

[33] J. Wang, Y. Feng, C. Zeng, and S. Li. An augmented reality based system for remote collaborative maintenance instruction of complex products. In *2014 IEEE International Conference on Automation Science and Engineering (CASE)*, pp. 309–314. IEEE, 2014.

[34] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1912–1920, 2015.

[35] S. Yan, Y. Xiong, and D. Lin. Spatial temporal graph convolutional networks for skeleton-based action recognition. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[36] N. Zhang, J. Donahue, R. Girshick, and T. Darrell. Part-based r-cnns for fine-grained category detection. In *European conference on computer vision*, pp. 834–849. Springer, 2014.

[37] X. Zhang, H. Xiong, W. Zhou, W. Lin, and Q. Tian. Picking deep filter responses for fine-grained image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1134–1142, 2016.

[38] H. Zheng, J. Fu, T. Mei, and J. Luo. Learning multi-attention convolutional neural network for fine-grained image recognition. In *Proceedings of the IEEE international conference on computer vision*, pp. 5209–5217, 2017.

[39] B. Zhou, S. Guven, S. Tao, and F. Ye. Pose-assisted active visual recognition in mobile augmented reality. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, pp. 756–758, 2018.

[40] J. Zhu, S. Ong, and A. Nee. A context-aware augmented reality system to assist the maintenance operators. *International Journal on Interactive Design and Manufacturing (IJIDeM)*, 8(4):293–304, 2014.