CoKe: Contrastive Learning for Robust Keypoint Detection

Yutong Bai, Angtian Wang, Adam Kortylewski, Alan Yuille
1Johns Hopkins University  2University of Freiburg  3Max-Planck-Institute for Informatics

Abstract

In this paper, we introduce a contrastive learning framework for keypoint detection (CoKe). Keypoint detection differs from other visual tasks where contrastive learning has been applied because the input is a set of images in which multiple keypoints are annotated. This requires the contrastive learning to be extended such that the keypoints are represented and detected independently, which enables the contrastive loss to make the keypoint features different from each other and from the background. Our approach has two benefits: It enables us to exploit contrastive learning for keypoint detection, and by detecting each keypoint independently the detection becomes more robust to occlusion compared to holistic methods, such as stacked hourglass networks, which attempt to detect all keypoints jointly. Our CoKe framework introduces several technical innovations. In particular, we introduce: (i) A clutter bank to represent non-keypoint features; (ii) a keypoint bank that stores prototypical representations of keypoints to approximate the contrastive loss between keypoints; and (iii) a cumulative moving average update to learn the keypoint prototypes while training the feature extractor. Our experiments on a range of diverse datasets (PASCAL3D+, MPII, ObjectNet3D) show that our approach works as well, or better than, alternative methods for keypoint detection, even for human keypoints, for which the literature is vast. Moreover, we observe that CoKe is exceptionally robust to partial occlusion and previously unseen object poses.

1. Introduction

Semantic keypoints, such as the joints of a human body, provide concise abstractions of visual objects in terms of their shape and pose. Accurate keypoint detections are of central importance for many visual understanding tasks, including viewpoint estimation [30], human pose estimation [5], action recognition [26], feature matching [24], image classification [46], and 3D reconstruction [20]. There are many diverse approaches for keypoint detection. Common approaches include the application of a regression loss [28, 21], a classification loss [14], or combinations of either with a 3D geometric model of the object [48]. In recent years, work in contrastive learning has led to major advances in representation learning [7, 13, 27] demonstrating benefits over classical losses, such as cross-entropy, e.g. in terms of robustness and data efficiency [22]. However, most works on contrastive learning in computer vision focus on the task of image classification, and it remains unclear how contrastive learning can be applied for keypoint detection.

In this paper, we introduce a contrastive learning framework for keypoint detection (CoKe). Keypoint detection differs from other visual tasks where contrastive learning has been applied, such as face recognition [33] or unsupervised learning [13], because the input is a set of images.
in which multiple keypoints are annotated. To enable the contrastive learning of keypoint detectors, we need to represented and detected keypoints independently, such that a contrastive loss can make the keypoint features different from each other and from the background. This is very different from current popular approaches to keypoint detection, such as stacked hourglass networks [2], which attempt to detect all keypoints jointly. Our approach has two benefits: It enables us to exploit contrastive learning for keypoint detection, and by detecting each keypoint independently the detection can become more robust to occlusion compared to holistic methods (as shown in our experiments).

Our CoKe framework introduces several technical innovations. Specifically, we found that the contrastive learning of keypoint representations requires the optimization of three types of distances in the feature space (Figure 1):

1. The distance between features of the same keypoint should be small. But the computational cost to calculate the distance between all features of the same keypoint is quadratic in their number. We instead reduce the computational cost to be linear by introducing an average prototypical representation for each keypoint (stars in Figure 1).

2. The distance between features of different keypoints should be large. The number of distance comparisons between features of different keypoints is combinatorial in the number of keypoints and training images. To manage this computational burden, we introduce a keypoint bank which stores the prototypical representations of all keypoints and allows for an efficient computation of the distance between the prototypes of different keypoints (Figure 1 bold orange arrow).

3. The distance between keypoint features and features from the background clutter (Figure 1 grey squares) should be large to reduce false-positive detections. However, most of the features in an image are clutter features and it is not feasible to compute the distance to all of them. Therefore, we introduce a clutter bank that keeps track of clutter features that are spatially close to keypoint features and hence are most difficult to be distinguished from.

The proposed approximations enable an efficient contrastive learning of keypoint detectors. We evaluate CoKe on several datasets including PASCAL 3D+, MPII, and ObjectNet3D. We observe that CoKe performs on par, and often even better, compared to SOTA related work (Stacked Hourglass Networks, MSS-Net [21]) and to approaches that use additional supervision in terms of detailed 3D object geometries (StarMap [48]). These results are remarkable as CoKe works well on all these datasets while, for example, the best results on MPII are often achieved by architectures that are specialized for human keypoint detection. We also observe that when compared to related work, CoKe is exceptionally robust to partial occlusion and unseen object poses. Our main contributions are:

1. We introduce a contrastive learning framework for keypoint detection.
2. CoKe performs very well on various keypoint detection datasets for rigid and articulated objects.
3. CoKe achieves exceptional robustness to partial occlusion and previously unseen object poses.

2. Related Work

Keypoint Detection. Keypoint detection, is a widely studied problem in computer vision. Popular applications are, e.g., the detection of human joints [5, 28, 36, 37] or distinct locations on rigid objects [42, 38, 30, 48]. Early approaches relied on local descriptors [11, 32, 45, 8, 10] that are distinctive and invariant [25]. While approaches using local descriptors have proven to be robust to occlusion and background clutter, they were outperformed by deep learning approaches that were trained end-to-end [28]. Toshev et al. [37] first trained a deep neural network for 2D human pose regression and Li et al. [23] extended this approach to 3D. Starting from the work of Tompson et al. [36], regression-based approaches to keypoint detection became very popular. They perform keypoint detection by regressing a heatmap representation. These approaches achieve a particularly good performance at detecting the joints of both articulated and rigid objects [28], because they can implicitly leverage the structural information between keypoint to resolve locally ambiguous keypoint detections. Tulsiani et al. [38], proposed to integrate the structural information between keypoints explicitly by integrating 2D and 3D models, which inspired a number of follow-up works, in particular for rigid objects [48, 38, 30].

Supervised Contrastive Learning. Contrastive learning, originates from Metric Learning [6, 41, 31] and involves the learning of a representation space by optimizing the similarities of sample pairs in this space. Intuitively, supervised contrastive learning aims to reduce the distance of feature representations of the same class, while increasing the distance between samples from different classes. Popular examples use pairs of samples for loss computation [12], triplets [33], or N-Pair tuples [34].

Recently, contrastive learning has attracted attention from the research community in self-supervised learning [7, 13, 43, 27, 16]. The main difference in the self-supervised learning setting is that positive examples are usually generated using data augmentations [9] or co-occurrence [18, 35] of a query sample, whereas negative examples are chosen as other images in the same mini-batch.

While most of the supervised contrastive learning [22] focuses on learning a holistic representation of the complete image, in this paper, we target a more fine-grained task - keypoint detection. Keypoints are localized image patterns
and therefore require the learning of local feature embeddings. The main challenge is that local image patterns can be highly ambiguous (e.g. the front and back tire of a car) and therefore require a contrastive learning framework that can learn to disambiguate local representations, while at the same time being able to learn a distinct representation that can be localized accurately.

3. CoKe: Contrastive Keypoint Learning

In this section, we present our framework for contrastive keypoint learning, then discuss the intuition underlying our approach and the training pipeline and how to perform keypoint detection during inference.

3.1. Training CoKe

We use $\Phi$ to denote the feature extractor. Given an input image $\mathbf{I}$, it computes the feature map $\Phi(\mathbf{I}) = \mathbf{F} \in \mathbb{R}^{H \times W \times D}$, where $i$ is the index in the training data $\{\mathbf{I}|i \in \{1, \ldots, N\}\}$. Using the keypoint annotation we retrieve the corresponding keypoint features $\mathbf{f}_k \in \mathbb{R}^D$ for the keypoints $k \in K$ from the feature map $\mathbf{F}$. Similarly, we can (randomly) select a non-keypoint location as clutter point and retrieve a set of clutter features $\{\mathbf{f}_c \in \mathbb{R}^D|c \in \{1, \ldots, C\}\}$. We define the distance between two features as $d(\cdot, \cdot)$. During training we learn a feature extractor that optimizes the following distances in the feature space:

1. One objective for the feature extractor during training is to minimize the distance between features of the same keypoint (i.e. the intra-keypoint distance) across all training images, by minimizing the objective:

$$D_{\text{intra}}(\mathbf{f}_k) = \sum_{j=1}^{N} d(\mathbf{f}_k, \mathbf{f}_j) \approx d(\mathbf{f}_k, \mu_k).$$

(1)

However, as we described in the introduction it is unpractical to compute the distance of a keypoint’s feature vector from one image $\mathbf{f}_k$ to the corresponding vectors in all other training images $j \in \{1, \ldots, N\}$. To resolve this computational problem, we define a prototypical keypoint feature $\mu_k$, that represents the average feature of keypoint $k$. Instead of computing the full objective, we approximate it as simply the distance to the corresponding average representation $d(\mathbf{f}_k, \mu_k)$. We store the prototypical features of all keypoints $\{\mu_k\}$ in a keypoint bank and update them during training.

2. The second objective for the feature extractor is to maximize the distance between features of different keypoints (i.e. the inter-keypoint distance). This requires computing the distance between the feature representations of one particular keypoint $k$ and all other keypoints $k'$ over all training images:

$$D_{\text{inter}}(\mathbf{f}_k) = \sum_{k' \in K \setminus \{k\}} \sum_{j=1}^{N} d(\mathbf{f}_k, \mathbf{f}_j) \approx \sum_{k' \in K \setminus \{k\}} d(\mathbf{f}_k, \mu_{k'}).$$

(2)

We approximate this objective by instead computing the distance to the prototypes of the respective keypoint features from the keypoint bank.

3. The third objective for the feature extractor is to maximize the distance between the keypoint features and all clutter features. In the best case, this involves computing the distance between a keypoint feature $\mathbf{f}_k$ to every clutter feature in all training images. To avoid the computation of this large amount of distances, we instead approximate this objective by storing a subset of all clutter features $\{\theta_c, c \in C\}$ in a clutter bank. Which allows us to approximate the full objective with

$$D_{\text{clutter}}(\mathbf{f}_k) \approx \sum_{c \in C} d(\mathbf{f}_k, \theta_c).$$

(3)

These three approximations make it feasible to optimize the overall objective with a feasible computational load. As the parameters of the feature extractor will change during learning, the clutter features $\theta$ in the clutter bank as well as the prototypes $\mu_k$ need to be updated. To achieve this, we follow an EM-type optimization process. First, we initialize the clutter bank by randomly sampling clutter features from the training data, and initialize the prototypes by computing the average feature for every keypoint across the training data $\mu_k = \frac{1}{N} \sum_{i=1}^{N} \mathbf{f}_k$. Using these initial estimates, we can compute the overall objective and train the feature extractor. While training the feature extractor, we update the clutter and keypoint bank. We perform those updates in an alternating manner.

3.1.1 Keypoint and Clutter Bank Update

Figure 2, illustrates the process of updating the keypoint prototypes and the clutter bank during training.

**Keypoint Bank Update.** Computing the prototypical keypoint features $\mu_k$ while learning the feature extractor is challenging, because we want to avoid to re-compute the prototypes over all training images $\mu_k = \frac{1}{N} \sum_{i=1}^{N} \mathbf{f}_k$ after every gradient step. Instead, we approximate the sample mean via a cumulative moving average. Specifically, we update $\mu_k$ with a training batch of size $m$ using:

$$\mu_k \leftarrow \mu_k + \sum_{i=0}^{m} \frac{\mathbf{f}_k}{m} \ast (1 - \alpha).$$

(4)

**Clutter Bank Update.** The clutter bank contains a limited number $C$ of clutter features $\{\theta_c, c \in C\}$. In practice the size of the clutter bank depends on the availability of
Clutter
Bank

Bank

respectively. To optimize those terms simultaneously, we {Similarly, we can maximize that we can minimize in our model are L2 normalized. From Eq. 5 observe the last step in above equation leverages that all features of the feature extractor. To compute the distance between two feature vectors, we use the L2 distance:

\[ d(f_a, f_b) = (f_a - f_b)^2 = 2 \times (1 - f_a \cdot f_b). \] (5)

The last step in above equation leverages that all features in our model are L2 normalized. From Eq. 5 observe that we can minimize \( D_{\text{intra}}(f_k) \) by maximizing \( f_k \cdot \mu_k \). Similarly, we can maximize \( D_{\text{inter}}(f_k) \) and \( D_{\text{clutter}}(f_k) \) by minimizing \( \{f_k \cdot \mu_k \} \forall k' \in K \setminus \{k\} \) and \( \{f_k \cdot \theta_c \} \forall c \in C \) respectively. To optimize those terms simultaneously, we use a non-parametric softmax as our loss function. Thus, the loss for each keypoint feature is calculated as:

\[ \mathcal{L}(f^i_k, \{\mu_k\}, \{\theta_c\}) = \frac{e^{f^i_k \cdot \mu_k}}{\sum_{k' \in K} e^{f^i_k \cdot \mu_{k'}} + \sum_{c' \in C} e^{f^i_k \cdot \theta_{c'}}}, \] (6)

where \( \{\mu_k\} \) is the keypoint bank and \( \{\theta_c\} \) is the clutter bank.

**Clutter Sampling Loss.** A technical problem is that the features in the clutter bank are copied from a large number of training images, and therefore it is not practical to calculate gradient directly w.r.t. \( \{\theta_c\} \), and therefore the feature extractor is not optimized w.r.t. the clutter features. This technical limitation makes the optimization w.r.t. the clutter distance \( D_{\text{clutter}}(f_k) \) converge slowly, especial when the clutter bank is large. To make the training more efficient, we propose a clutter sampling loss, which uses the sampled clutter feature \( f^i_k \) from the current training batch and directly maximizes the clutter to keypoint distance:

\[ \mathcal{L}(f^i_c, \{\mu_k\}) = \sum_{k' \in K} f^i_k \cdot \mu_{k'}. \] (7)

GPU memory and we observe that the larger the bank, the better the training performance (see experiments section). We update the clutter bank during training by replacing the oldest clutter features with newly extracted features from the current training batch based on a time tag that indicates how long features were stored in the bank.

**3.1.2 Feature Extractor Training**

When training the feature extractor with SGD, we freeze the keypoint bank and clutter bank in every gradient step and use them to compute the gradient update on the weights of the feature extractor. To compute the distance between two feature vectors, we use the L2 distance:

\[ d(f_a, f_b) = (f_a - f_b)^2 = 2 \times (1 - f_a \cdot f_b). \] (5)

The last step in above equation leverages that all features in our model are L2 normalized. From Eq. 5 observe that we can minimize \( D_{\text{intra}}(f_k) \) by maximizing \( f_k \cdot \mu_k \). Similarly, we can maximize \( D_{\text{inter}}(f_k) \) and \( D_{\text{clutter}}(f_k) \) by minimizing \( \{f_k \cdot \mu_k \} \forall k' \in K \setminus \{k\} \) and \( \{f_k \cdot \theta_c \} \forall c \in C \) respectively. To optimize those terms simultaneously, we use a non-parametric softmax as our loss function. Thus, the loss for each keypoint feature is calculated as:

\[ \mathcal{L}(f^i_k, \{\mu_k\}, \{\theta_c\}) = \frac{e^{f^i_k \cdot \mu_k}}{\sum_{k' \in K} e^{f^i_k \cdot \mu_{k'}} + \sum_{c' \in C} e^{f^i_k \cdot \theta_{c'}}}, \] (6)

where \( \{\mu_k\} \) is the keypoint bank and \( \{\theta_c\} \) is the clutter bank.

**Clutter Sampling Loss.** A technical problem is that the features in the clutter bank are copied from a large number of training images, and therefore it is not practical to calculate gradient directly w.r.t. \( \{\theta_c\} \), and therefore the feature extractor is not optimized w.r.t. the clutter features. This technical limitation makes the optimization w.r.t. the clutter distance \( D_{\text{clutter}}(f_k) \) converge slowly, especial when the clutter bank is large. To make the training more efficient, we propose a clutter sampling loss, which uses the sampled clutter feature \( f^i_k \) from the current training batch and directly maximizes the clutter to keypoint distance:

\[ \mathcal{L}(f^i_c, \{\mu_k\}) = \sum_{k' \in K} f^i_k \cdot \mu_{k'}. \] (7)
Thus, the final loss for training the feature extractor becomes:

\[ L(F^i; \{ \mu_k \}, \{ \theta_c \}) = \sum_{k \in \mathbb{K}} \mathcal{L}(f^i_k; \{ \mu_k \}, \{ \theta_c \}) + \sum_{c \in \mathbb{C}} \mathcal{L}(f^i_c; \{ \mu_k \}). \]  

(8)

### 3.2. Inference with the CoKe Model

In the following, we describe the detection process for keypoints of a single category, but note that this can be trivially extended to multiple categories. Figure 3 illustrates the inference process with the CoKe. To locate the predicted keypoint position \( p_k \) for keypoint \( k \) on the feature map of a test image \( \mathbf{I} \), we apply the following steps:

1. Extract a feature map \( \Phi(\mathbf{I}) = \mathbf{F} \) using the trained feature extractor \( \Phi \). Use the prototypes \( \mu_k, k \in \mathbb{K} \) to compute the per pixel feature distance \( d(f, \mu_k) \) for all feature vectors \( f \in \mathbf{F} \) and store the detection scores in the output \( \mathbf{S} \in \mathbb{R}^{H \times W \times K} \). For each keypoint, select the position \( p \) with the highest detection score in \( \mathbf{S} \). Project \( p \) back to the original image coordinates.

### 4. Experiments

In this section, we experimentally evaluate CoKe and compare it to related works. We first describe the experimental setup, compare CoKe to related work on several diverse datasets, and study their robustness to partial occlusion. Then discuss qualitative results with ablation studies.

#### 4.1. Experimental Setup

**Evaluation Protocol.** Evaluation is done using the standard Percentage of Correct Keypoints (PCK) metric which reports the percentage of detections that fall within a normalized distance of the ground truth. We use PCK=0.1 for PASCAL3D+, ObjectNet3D and OccludedPASCAL3D+ datasets following the standard experimental protocol. For MPII, we use PCKh=0.5 as the evaluation metric. Distance is normalized by a fraction of the head size (PCKh). We follow the common protocol of evaluating each object category by computing the average accuracy on the visible keypoints over all the test images.

**Training Setup.** We use the standard train-val-test split for all the datasets. We use a batch size of 64 for training. For each image, we randomly select a group of 20 clutter points. The full clutter bank consists of 1024 such groups. We choose the clutter features to be within two pixels distance to the keypoint annotation in the feature map. We use the non-parametric softmax [43] to calculate the similarity between features and banks. The temperature parameter [17] that controls the concentration level of the distribution is set to \( \tau = 0.7 \).

**PASCAL3D+ Dataset.** The dataset contains 12 man-made object categories with totally training 11045 images and 10812 evaluation images. Different from previous works [48, 38], we use all images for evaluation, including occluded and truncated ones.

**MPII Dataset.** MPII Human Pose [1] consists of images taken from a wide range of human activities with a challenging array of articulated poses. The keypoint visibility is annotated, enabling us to report numbers for the full dataset as well as partially occluded humans.

**ObjectNet3D Dataset.** ObjectNet3D consists of common daily life objects and is notably more difficult compared to PASCAL3D+ as it contains more rare viewpoints, shapes and truncated objects with occlusion. We test on nine categories, chosen based on their high annotation accuracy, as some categories in ObjectNet3D have low quality annotations due to the complexity of this dataset.

**OccludedPASCAL3D+ Dataset.** While it is important to evaluate algorithms on real images of partially occluded objects, simulating occlusion enables us to quantify the effects of partial occlusion more accurately. We use an analogous dataset with artificial occlusion for keypoint detection proposed in [39] for object detection. It contains all 12 classes of the PASCAL3D+ dataset at various levels of occlusion. The dataset has a total of 3 occlusion levels, with Lv.1: 20-40%, Lv.2: 40-60% and Lv.3: 60-80% of the object area being occluded.

### 4.2. Performance on Various Datasets

**PASCAL3D+ and OccludedPASCAL3D+.** Table 1 shows the keypoint detection results on the PASCAL3D+ dataset for CoKe models learned from three different backbones: ResNet-50 [15], Stacked-Hourglass-Network and Res-Unet [47]. We also show the performance of StarMap [48] as reported in the original paper. Note that StarMap uses 3D models as additional supervision to jointly reason about the relative position of the keypoints. The performance of CoKe with all backbones is constantly high. The highest performance is achieved with the most recently developed architecture Res-Unet. When compared to the original Stacked-Hourglass-Network trained with a regression loss (SHG) we can clearly observe a large gain in performance. Most notably, the performance difference is very prominent for strong occlusion. We believe CoKe is more robust to occlusion, because of two reasons: 1) It is actively optimized to discriminate between keypoint features and clutter features based on their local representations only, which might help to reduce the effective receptive field size and hence reduces the negative effects of occlusions. 2) CoKe stores the clutter representation from a large number of images, and hence can better learn to distinguish keypoints from clutter. Overall, our results clearly highlight that CoKe is very competitive with related work, while being highly robust to partial occlusion.

In Table 2, we demonstrate an additional robustness to
unseen object poses. We divide the azimuth pose in the car category of Pascal3D+ into 4 bins, front, back, left and right. We train CoKe on the front and back subsets, and test on seen (7305 images) and unseen (3507 images) poses separately. Table 2 shows that CoKe is much more robust to unseen poses compared to SHG.

<table>
<thead>
<tr>
<th>Occlusion Level</th>
<th>L.v.0</th>
<th>L.v.1</th>
<th>L.v.2</th>
<th>L.v.3</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHG</td>
<td>68.0</td>
<td>46.5</td>
<td>43.2</td>
<td>39.9</td>
<td>49.4</td>
</tr>
<tr>
<td>MSS-Net</td>
<td>68.9</td>
<td>46.6</td>
<td>42.9</td>
<td>39.6</td>
<td>49.5</td>
</tr>
<tr>
<td>StarMap</td>
<td>78.6</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>CoKe-Res50</td>
<td>77.0</td>
<td>67.6</td>
<td><strong>59.9</strong></td>
<td>53.4</td>
<td>64.4</td>
</tr>
<tr>
<td>CoKe-SHG</td>
<td>78.3</td>
<td>66.3</td>
<td>58.4</td>
<td>52.3</td>
<td>63.8</td>
</tr>
<tr>
<td>CoKe-ResUnet</td>
<td><strong>80.3</strong></td>
<td><strong>68.5</strong></td>
<td>59.1</td>
<td><strong>54.0</strong></td>
<td><strong>65.5</strong></td>
</tr>
</tbody>
</table>

Table 1. Keypoint detection results on Pascal3D+ under different levels of partial occlusion (L.v.0:0%, L.v.1:20-40%, L.v.2:40-60%, L.v.3:60-80% of objects are occluded, L.0 is the original dataset). CoKe outperforms baseline models and is highly robust to partial occlusion.

MPII. We compare CoKe learned from the SHG backbone to a number of related works on MPII in Table 3. CoKe-SHG is again highly competitive and outperforms related work by a small but significant margin. Table 4 compares CoKe-SHG and SHG for both occluded and non-occluded keypoint detection on the MPII dataset. We can observe that CoKe achieves significantly higher results on occlusion scenarios compared with SHG. When compared to a recent and highly competitive baseline such as RSNs [4], we can observe that RSN achieves a higher performance on the full dataset, due to its advanced architecture that
has more than four times more parameters compared to our model. However, we observe that the model suffers from a relatively higher performance decrease when the humans are occluded, hence it is not as robust as our model that uses a standard Stacked-Hourglass backbone. We also see the potential for the performance of our model to further increase with more advanced backbones.

<table>
<thead>
<tr>
<th>MPII</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>RecurrentPose [2]</td>
<td>88.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PoseMachines [40]</td>
<td>88.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DeeperCut [19]</td>
<td>88.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PartHeatmap [3]</td>
<td>89.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHG [28]</td>
<td>90.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DualPathNetworks [29]</td>
<td>91.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PoseRegression [3]</td>
<td>91.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>CoKe-SHG</strong></td>
<td><strong>91.4</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3. Keypoint detection results on MPII compared with regression based algorithms.

<table>
<thead>
<tr>
<th>MPII</th>
<th>Full</th>
<th>Ocluded</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHGs</td>
<td>90.1</td>
<td>84.3</td>
</tr>
<tr>
<td>RSN [4]</td>
<td>92.7</td>
<td>86.9</td>
</tr>
<tr>
<td><strong>CoKe-SHG</strong></td>
<td><strong>91.4</strong></td>
<td><strong>86.7</strong></td>
</tr>
</tbody>
</table>

Table 4. Keypoint detection results on MPII of CoKe-SHG and SHG on both original and challenging scenarios.

ObjectNet3D. We compare SHG and a CoKe-SHG on ObjectNet3D in Table 5. CoKe-SHG outperforms SHG by a large margin on every category. Since ObjectNet3D dataset contains much more challenging scenarios, we claim that CoKe is more robust.

<table>
<thead>
<tr>
<th>ObjectNet3D</th>
<th>coffee</th>
<th>dryer</th>
<th>kettle</th>
<th>jar</th>
<th>wash</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHG</td>
<td>31.0</td>
<td>35.1</td>
<td>32.2</td>
<td>41.9</td>
<td>33.9</td>
</tr>
<tr>
<td><strong>CoKe-SHG</strong></td>
<td><strong>36.4</strong></td>
<td><strong>37.6</strong></td>
<td><strong>37.8</strong></td>
<td><strong>45.2</strong></td>
<td><strong>36.1</strong></td>
</tr>
<tr>
<td><strong>SHG</strong></td>
<td>66.8</td>
<td>52.3</td>
<td>44.6</td>
<td>45.8</td>
<td>42.62</td>
</tr>
<tr>
<td><strong>CoKe-SHG</strong></td>
<td><strong>70.2</strong></td>
<td><strong>57.7</strong></td>
<td><strong>51.3</strong></td>
<td><strong>49.6</strong></td>
<td><strong>46.88</strong></td>
</tr>
</tbody>
</table>

Table 5. Keypoint Detection results on ObjectNet3D+ [44].

In summary, we observe that CoKe is a general purpose framework that constantly achieves a very high performance across a wide range of backbone architectures and for a range of datasets with very different characteristics, while also being highly robust to occlusion.

4.3. Qualitative Results

Visualization of Keypoint Detection Maps. We visualize the part detection maps from CoKe-Res50 in Figure 4. Images are selected from the car category in PASCAL3D+. which has complex changes in pose, illumination and object structure. Note how all keypoints are detected accurately despite the difficulty from false positives, occlusions, rare viewpoints, different domain, irregular appearance. We can observe that CoKe is robust in these challenging scenarios.

Visualization of Detection Results under Occlusion. We visualize qualitative results in Figure 5. Overall, the illustrations demonstrate the robustness of CoKe to partial occlusions. Any keypoints that are not in the vicinity of occluders are correctly detected and not affected by the occlusion. Furthermore, keypoints that are partially occluded (e.g. the wheel) can still be located robustly, although the detections tend to move away from the occluder. Importantly, we do not observe false positive detections at locally ambiguous keypoints. This demonstrates that CoKe leverages the receptive field to disambiguate keypoints, while still being able to localize individual keypoints accurately.

Inference Time and Memory Consumption. During inference, CoKe-Res50 (params: 23M, acc: 77%) takes 0.01s per image, while SHGs (params: 25M, acc: 68%) needs 0.06s. For the memory consumption, CoKe-Res50 needs 715MB, SHGs 786MB when batch size equals to 1. CoKe has an advantage of the inference time while maintaining the competence of the memory consumption.

4.4. Ablation Study

Comparison with Other Losses. In Table 6, we ablate the effect of the contrastive learning compared to other common losses on different backbone architectures. Specifically, we compare to a regression loss as used in SHGs and a supervised classification loss. The classification baseline enables us to compare the effect of the contrastive learning with a cross-entropy loss. We implement keypoint detection as a classification problem by using the features at the ground truth keypoint locations in the training data as instances of different classes to train the backbone with the cross-entropy objective. During testing, we use the average representations from the training set as part detector, similar as in the CoKe pipeline. The main difference to the CoKe training is hence that the contrastive loss and the clutter bank are not used. From the results, we observe that our representation learning formulation of keypoint detection constantly outperforms other losses by a large margin.

Clutter Bank Mechanism In Table 7, we study the influence of the clutter features and the clutter sampling
5. Conclusion

In this paper, we study keypoint detection from the perspective of contrastive learning. Our contrastive keypoint learning framework (CoKe) makes several efficient approximations to enable the contrastive representation learning for keypoint detection: (i) A clutter bank to approximate non-keypoint features; (ii) a keypoint bank that stores prototypical representations of keypoints, to approximate the within-keypoint distance; and (iii) a cumulative moving average update to learn the keypoint prototypes while training the feature extractor. Our experiments on several diverse datasets (PASCAL3D+, MPII, ObjectNet3D) show that CoKe is very general and performs well for rigid as well as articulated objects. Compared to related work, CoKe achieves higher performance, while also being much more robust to partial occlusion and unseen object poses.

Acknowledgements. AK acknowledges support via his Emmy Noether Research Group funded by the German Science Foundation (DFG) under Grant No. 468670075.
References


