Recognizing Actions in Videos from Unseen Viewpoints

AJ Piergiovanni
Indiana University
ajpiergi@indiana.edu

Michael S. Ryoo
Stony Brook University
mryoo@cs.stonybrook.edu

Abstract

Standard methods for video recognition use large CNNs designed to capture spatio-temporal data. However, training these models requires a large amount of labeled training data, containing a wide variety of actions, scenes, settings and camera viewpoints. In this paper, we show that current convolutional neural network models are unable to recognize actions from camera viewpoints not present in their training data (i.e., unseen view action recognition). To address this, we develop approaches based on 3D representations and introduce a new geometric convolutional layer that can learn viewpoint invariant representations. Further, we introduce a new, challenging dataset for unseen view recognition and show the approaches ability to learn viewpoint invariant representations.

1. Introduction

Activity recognition with convolutional neural networks (CNNs) has been very successful [2, 41, 13] when provided sufficient diverse labeled data, like Kinetics [21]. However, one major limitation of these CNNs is that they are unable to recognize actions/data that are outside of the training data distribution. This is most notably observed for unseen classes (objects, activities, etc.) which has been heavily studied in zero-shot and few-shot learning literature. In this work, we look at a related, but different problem of *unseen viewpoint* activity recognition, where the actions are the same, but occur from different camera angles.

To motivate this problem, let us consider an example. Given a labeled dataset of a person performing actions with one camera angle, we train a CNN to recognize this action. Now, suppose we have new videos to recognize, but from a different camera view. This could be as simple as a different camera placement in the environment, or an entirely different camera and setting (e.g., Fig. 1). In this case, a trained CNN, in general, fails to recognize the action. As a simple experiment, we use the Human3.6M dataset [14], which contains videos of a person performing an action from 4 different camera angles. As shown in Table 1, when training on one view and testing on another, the model is unable to recognize the action. However, humans are able to recognize these actions regardless of viewpoint and studies have found that this is likely because humans build invariant representations of actions in their minds [39] .

Figure 1: Examples of the seen, static broadcast camera in MLB-YouTube and examples of the new, unseen viewpoints of the same actions. This dataset is quite challenging, adding new views, people, etc.

Further, this problem frequently occurs in real data (e.g., YouTube videos). Existing smaller datasets such as Toyota SmartHome [8], Charades-Ego [38], NTU [35] and others all provide videos in multiple viewpoints to study this effect. Large video datasets like Kinetics [21] naturally contain many views, however, there is no annotation of the view and each video only provides a single view. Other datasets like MLB-YouTue [31] only contains the single broadcast camera view baseball games. As collecting video data is already challenging, designing CNNs that generalize to unseen viewpoints is critical, especially for applications where diverse view data is limited or unavailable. It would be practically impossible to build datasets for many desirable settings that enumerate all possible (or sufficiently large number of) viewpoints to fully model activities.

There are many potential ways to address this problem. One hypothesis is that by training on a large-scale video datasets, such as Kinetics, the model could implicitly learn multi-view representations of actions. However, as shown in Table 1, we empirically find that while it improves per-

Method	Seen	Unseen
Random	9.1%	9.1%
2D ResNet-50	86.4%	9.1%
3D ResNet-50	100%	9.1%
3D ResNet-50 + Kinetics	100%	38.2%
Ground Truth 3D Pose	100%	100%

Table 1: Experiments on Human3.6M with unseen viewpoints. Standard CNNs are unable to recognize actions with different viewpoints, however, using global 3D pose allows the models to recognize the actions.

formance, it is still lacking. A second hypothesis is that by using 3D human pose information, we can recognize actions in a global representation space, unconstrained by camera views. A key drawback to this approach is estimating 3D pose from video itself is a challenging problem, especially when multiple people are present. It further requires estimating camera pose in order to build a 'world'/global camera invariant 3D representation. Further, it is unclear what the right representation of 3D pose is (e.g., coordinates of joints, limbs, motion difference of joints between frames, etc.).

Building on this hypothesis and observation, we present and evaluate several approaches for recognizing actions in unseen viewpoints. The basic approach relies on estimating 3D pose directly from the videos, then explores using different representations of it for recognition. Since directly estimating accurate real-world 3D pose is often difficult, we also present an approach of learning *latent* 3D representations of an action and its multi-view 2D projections. This is done by imposing the latent action representations to follow 3D geometric transformations and projections, in addition to minimizing the action classification loss. We learn such view invariant action representations *without* any 3D or view ground truth labels.

We also introduce a challenging dataset building on the MLB-YouTube dataset [31]. The MLB-YouTube dataset contains actions from a single camera and these actions are all in the same environment (e.g., a professional baseball stadium). Our extended dataset contains evaluation samples of the same actions, but from many different viewpoints and a variety of different settings: batting cages, little league (children's baseball games), high school games, etc. These use different camera (e.g., cell phones), in very different environments. The goal is to learn a representation from the single view dataset that generalizes to these challenging, unseen viewpoints. Examples are shown in Fig. 1.

To summarize, the contributions of this paper are:

- A computationally efficient, geometric-based layer and learning to learn view invariant representation.
- Thorough evaluation of multiple approaches to unseen viewpoint action recognition.
- A challenging new dataset for unseen viewpoint recognition in unseen environments.

2. Background - 3D Geometric Camera Model

First we briefly review the standard 3D geometric camera model used in computer vision, which we build on in this work. We begin with a standard pinhole camera model. Given the pixel coordinates p, the 3D world coordinates p_w are represented as

$$p = \boldsymbol{K} \left[\boldsymbol{R} \mid \boldsymbol{t} \right] \left[p_w \mid \mathbf{1} \right]^T \tag{1}$$

where $\boldsymbol{K}$ is the 3×4 camera projection matrix (intrinsic camera matrix) mapping a 3D point into a 2D camera view. $\boldsymbol{R}$ and $\boldsymbol{t}$ are the camera rotation (3×3) and translation (1×3) in the world space (extrinsic camera matrix) that transform the points between different 3D camera views. $\mid$ is the matrix concatenation operator. In many cases in computer graphics and computer vision, it is assumed that $\boldsymbol{K}$, $\boldsymbol{R}$ and $\boldsymbol{t}$ are known. These matrices can be used to compute the inverse as $[\, \boldsymbol{R}^T \mid \boldsymbol{R}^T\boldsymbol{t} \,]$ (the inverse of a rotation matrix is its transpose). Thus, given a 3D coordinates p, the view-invariant world coordinates can be computed as $p_w = p \cdot \left[\boldsymbol{R}^T \mid \boldsymbol{R}^T\boldsymbol{t} \right]$.

Importantly, points in the 3D world coordinate system are, by design, viewpoint invariant. However, in many settings, including activity recognition, the camera matrices are unknown. In some cases the intrinsic camera matrix $\boldsymbol{K}$ might be known (e.g., when the camera has been previously calibrated), but the extrinsic matrices as well as the definition of the world coordinate system are not. The core of this paper is exploring methods to learn and represent these.

3. Basics - Using 3D Pose

We first design and investigate a straight forward approach of using 3D human pose estimation and its projection for action classification (Fig. 2). Many works have explored estimating 3D human pose from videos [30, 19, 24, 25, 28], even multi-person 3D pose [27]. We begin by using PoseNet [27] to estimate 3D coordinates. PoseNet provides 3D coordinates in camera space, so directly using the coordinates will not yield viewpoint-invariant recognition. For this, the 3D coordinates need to be transformed into world-space.

However, estimating the extrinsic matrix from a single, random image is a challenging problem. We use CalibNet [16] to obtain estimations of $\boldsymbol{R}$ and $\boldsymbol{t}$. This approach is quite limited by the accuracy of CalibNet, if it provides a poor estimation, the rest of the network will fail. Since there is limited camera calibration training data, we observe than for in-the-wild videos, CalibNet often gives inaccurate results and does not generalize well.

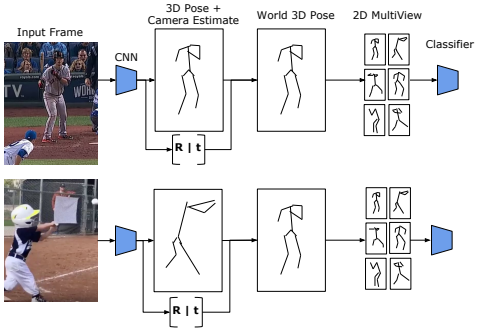

Figure 2: Overview of the process to learn global 3D pose and 2d multi-view projections of it for classifying unseen viewpoints.

3.1. Recognition with 3D representation

Given the estimated 3D pose in world space, these values can be directly used as input to the model. However, directly using 3D coordinates may not be the best feature, as scale changes (e.g., person size), speed of which the action occurs, etc. will all impact the features. Previous works have studied representations of 3D pose for skeleton action recognition [9, 46, 22, 23], such as joint angles [29] showing the difficulty of this task.

Instead of directly using the 3D pose as input to a classification model, multi-view 2D projections of it could be used. Research focusing on designing strong CNNs for understanding 2D image input has been one of the mainstream areas, and it often is more advantageous to use 2D image inputs rather than 3D. Further, by using multiple views, the model can see the input from different angles, instead of a single one. To do this, we assume we have an intrinsic camera matrix K that projects the 3D coordinates into 2D. We follow the standard pinhole camera model and learn a camera rotation and projection to generate multiple 2D views. Inspired by previous works, like Potion [6], we take the 2D projections of pose and render skeletons capturing the motion. These images are used as input to the model for activity classification. An overview of this approach is shown in Fig. 2.

4. Network for Latent 3D Representations

The previous approach is an engineered combination of existing CNNs (pose, camera estimation and action recognition) and relies on multiple components correctly functioning. If any of these networks fail or gives slightly incorrect results, the rest of the model will fail. However, we draw inspiration from the geometric based approach, and design geometric CNN layers to learn and replicate similar transformations. To do this, we begin by learning a representation that contains and uses both 3D information and the extrinsic information. We then combine this information to get a 3D world representations of the actions and provide a CNN architecture. We introduce a loss function terms to learn such 3D view-invariant representations from a dataset with (unlabeled) views.

4.1. Neural Projection Layer (NPL): Building a Latent 3D World Representation

First, we take a feature map F which is a $W \times H \times (C+3)$ tensor, where C is the channels in the feature map plus the CNN estimated 3D camera space coordinate. Formally, $F_{x',y'} = [p_{x',y'}, f_{x','y}]$, where $p_{x',y'} = [x, y, z]$ at location x', y' in the feature map and $f_{x',y'}$ is the C-dimensional feature at the location.

Next, we use a fully-connected layer to estimate $\boldsymbol{R}$ and $\boldsymbol{t}$ (rotation and translation) from each video. These are used to transform the video camera view into the world coordinates as $p^w_{x',y'} = p_{x',y'} \cdot [\, \boldsymbol{R}^T \mid \boldsymbol{R}^T \boldsymbol{t} \,]$ for each p in the feature map. This gives the 3D world coordinates, i.e., camera invariant coordinate, of each point p. Note that the 3D world coordinate system is the same for all videos, thus $\boldsymbol{R}$ is different for each video, depending on the camera viewpoint. $\boldsymbol{R}$ plays the role of aligning features (e.g., humans) in different scenes, so that the losses are minimized. This allows the model to learn to map each video into the same global coordinates.

The world 3D representation is then created as:

$$F^W_{x,y,z} = \sum_{i=0,j=0}^{W,H} \mathbb{1}(p^w_{i,j} = [x, y, z]) F_{i,j} \tag{2}$$

where $\mathbb{1}(p^w_{i,j} = [x, y, z])$ is the indicator function for when p^w matches location x, y, z.

That is, we can create a new feature map F^W which has a shape of $W \times H \times Z$ (in practice, $W = H = Z$, for example, 64). Given one of the points $p_{x',y'}$ and its associated feature vector $f_{x',y'}$ from the original feature map, we compute the 'world coordinate' of that point $x, y, z = p^w_{x',y'}$ and then set $F^W_{x,y,z} = F^W_{p^w_{x',y'}} = f_{x',y'}$. I.e., we set the values in the feature map based on their location in the world coordinate reference. This transforms the original latent 3D representation into the rotation and translation invariant 3D world representation, resulting in a representation that is viewpoint/camera invariant.

However, since x, y, z are integers and $p^w_{i,j}$ is likely not an integer and we want to implement this as a differentiable function to be learned with gradient descent, we slightly modify this to:

$$F^W_{c,x,y,z} = \sum_{i=0,j=0}^{W,H} (1 - |x - p^w_{i,j}[x]|) \\ (1 - |y - p^w_{i,j}[y]|)(1 - |z - p^w_{i,j}[z]|) F_{c,i,j} \tag{3}$$

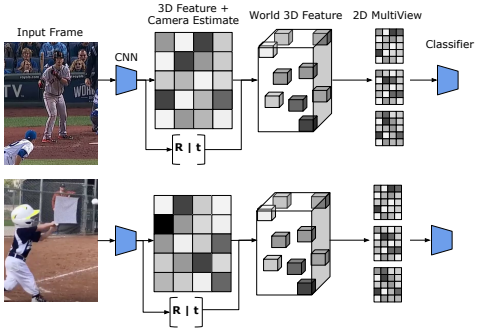

Figure 3: Learning latent 3D representation. A CNN produces a 3D feature: each location has a feature value and x, y, z location. The network also produces camera parameters, allowing the construction of a viewpoint invariant 3D feature. Then multiple cameras are learned, allowing the creation of multi-view 2D projections of the features. These are stacked on the channel axis and used for classification. The world 3D feature and 2D MultiView features are learned to be identical for the same action.

This is similar to Eq. 5 introduced in the spatial transformer network [17]. In the implementation, we first set $p_w = \tanh p^w$ to ensure its values fit in the feature map space, similar to the transformer network.

Once we obtain F_W, we use it directly as input to the remaining CNN for classification. We note that F_W is a 4-dimensional tensor (channels followed by 3D coordinates), so we use 3D convolution on top of this representation.

4.1.1 Multi-view 2D Projections

Instead of directly working with a 3D feature map, we can follow the ideas from Section 3.1 and generate multiple 2D projections of the features. This is shown in Fig. 3.

Assuming we have a camera matrix $\boldsymbol{K}$, which we represent as

$$\boldsymbol{K} = \boldsymbol{R} \begin{pmatrix} s_x & 0 & x_0 \\ 0 & s_y & y_0 \\ 0 & 0 & 1 \end{pmatrix} \tag{4}$$

where s_x, s_y are the focal lengths and x_0, y_0 are the offsets, and $\boldsymbol{R}$ is the 3×3 camera rotation matrix. Note $\boldsymbol{R}$ can be represented with 3 parameters: yaw, pitch and roll which generate the full 3×3 rotation matrix. This process uses the same components and projections as in Section 4.1, but instead of estimating $\boldsymbol{R}$ using a layer for each video, here these are learned parameters of the model. This allows the model to generate the same 2D views from the view-invariant world 3D space. We can then model a 2D projection of the 3D points as:

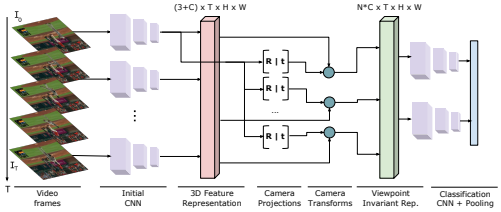

Figure 4: Illustration of the full viewpoint invariant recognition model.

$$F^W_{c,x,y} = \sum_{i=0,j=0}^{W,H} (1-|x-\boldsymbol{K}p^w_{i,j}[x]|)(1-|y-\boldsymbol{K}p^w_{i,j}[y]|)F_{c,i,j} \tag{5}$$

As these operations are differentiable, we can learn all these parameters with gradient descent. This allows the model to learn the optimal arrangement of cameras to capture views from the latent 3D representations for action recognition. Further, by increasing the number of cameras (N), we can learn multiple 2D projections of the representations, which can be stacked on the channel axis for recognition. In the next section, we describe the training loss that enables learning of the 3D representation without any 3D or camera calibration ground truth data.

We note that in this setting, some views will have occluded objects/features. An assumption of the approach is that using multiple cameras will naturally capture different viewpoints that will minimize the effects of occlusion. Another approach could be to use tomographies of each view to remove the effect of occlusions, which we leave as future work.

4.2. Recognition with 3D Representation

The full model, as illustrated in Fig. 4, begins by applying several conv. layers to a video input. At some point in the network (we experimentally evaluate where), we generate the 3D feature map and apply the geometric transformations described above. We then use either 2D or 3D conv. layers followed by a fully connected layer to classify the video clip.

5. Learning Latent 3D Representations

The key challenge with this approach is learning the camera matrices that generate view-invariant representations. We propose an approach to learn this view-invariant representation from a dataset with action videos (e.g., Kinetics [21]). Importantly, we do not assume any ground truth viewpoint or 3D data is provided, but only that the videos naturally contain multiple views. Unlike the first approach where we first learn 3D human pose estimation and extrinsic camera calibration from specific datasets, this approach only requires action-labeled data, available in many large-scale

public datasets.

Given two videos V and U of the same action, we compute their 3D representations $F_W(V)$ and $F_W(U)$ and use a loss to make their representations the same:

$$\text{3d\_loss}(V, U) = ||F_W(V) - F_W(U)||_F \tag{6}$$

Intuitively, this loss term makes it so that two videos likely with different camera views have the same 3D representation after the projections. This encourages the representations from different viewpoints of the same action result in the same 3D representation.

For multi-view 2D projection, we apply that loss on each 2D view, as well as adding a regularizing term to make each camera different:

$$cam\_reg(c_1, c_2) = \max(-||c_1 - c_2||_F, \alpha) \tag{7}$$

where c_1, c_2 are camera matrices and α is the margin, or desired max difference. We note that this constraint forces the cameras to be different, but does not ensure that they are facing the scene. However, we observed that during training, the cameras converged to views that were facing the scene, as shown in Fig. 5.

5.1. Training Loss Function

Our final loss is a combination of these terms plus a standard classification loss. Given the set of cameras in the model $\mathcal{C}$, let V be a video and l be the binary vector indicating the class label of a video. Let $M(V)$ be the application of the network to video V giving the predictions for each class (e.g., a c-dimensional vector where c is the number of classes). In particular, $M(V)$ is some video CNN that produces the classification. More details are shown in Fig. 4 and specific architecture details are in the appendix. Given the set of training videos and labels, $\mathcal{V}$, The final loss function is:

$$\begin{aligned}\mathcal{L}(\mathcal{V}) = &\sum_{(V,l)\in\mathcal{V}} \left(-\sum_{i}^{c} l_i \log M(V)_i \right) \\ &+\lambda_1 \left(\sum_{(U,k)\in\mathcal{V}} \begin{cases} \text{3d\_loss}(V, U) & l = k \\ 0 & otherwise \end{cases} \right) \\ &+ \lambda_2 \left(\sum_{c_1 \neq c_2 \in \mathcal{C}} cam\_reg(c_1, c_2) \right)\end{aligned} \tag{8}$$

The combination of the geometric structure imposed by the NPL and the components of the loss function encourages the model to learn viewpoint invariant representations. This formulation enables learning 3D representations with only activity-level labels and the geometric constraints.

6. Experiments

We conduct various experiments to understand the various components of the approach on multiple datasets. The model was implemented in PyTorch (code in appendix) and pretrained on Kinetics-400 [21]. We used Kinetics to pretrain as it is large and naturally has many viewpoints, allowing for the evaluation of this approach. We then use the network to extract features and train a small two-layer network on each specific dataset. Training was done for 25 epochs with the learning rate set to 0.01. For learning the rotation matrices, we learn 3 parameters: yaw, pitch and roll, which we convert to the rotation matrix.

Datasets: We evaluate this approach on three datasets containing multi-view data. On Human3.6M, we train the model on one camera view for one subject and test on one of the other views. The model is trained to recognize 11 different activities. We perform this experiment for 9 subjects and 2 different seen/unseen view combinations and report the average over each setting.

For the unseen MLB (baseball) videos, we train on the broadcast camera videos (original MLB-YouTube dataset [31] and test on the new, unseen views. We newly created this dataset of unseen views for testing only. It consists of 500 videos from YouTube for testing of 4 different baseball actions (swing, hit, pitch, bunt). This data is quite challenging as it has drastically different viewpoints, people, backgrounds, activity speeds, etc. Examples of these views are shown in Fig. 1.

For Toyota SmartHome (TSH), we follow the CV_1 protocol [8] where the model is trained using camera 1 and tested on camera 2. We also report results on the CV_2 protocol where it is evaluated on camera 2 but trained on multiple cameras. This dataset has 16.1k videos taken from 7 different camera viewpoints. It contains 31 classes of human daily activities in real-world environments. Similarly, we also compare on NTU-RGB-D [35] following the standard settings.

In Table 2, we compare the different proposed approaches. We find that for some datasets, like Human3.6M, using 3D pose (Section 3) is a very good feature, as this dataset contains actions focused on human motion. However, on TSH, which has many object-dependent actions, using pose degrades performance. For example, actions such as 'pick up cup' and 'pick up plate', the pose motion is nearly identical for both of those, but the object is different. Using pose gives little indication which object is being used, thus when only using pose to recognize the action, the model cannot distinguish.

On the MLB dataset, using pose harms performance on the seen viewpoint, but slightly improves performance on the unseen views. In both cases, it is only slightly better than random. This is likely due to noisy data with many people

Table 2: Comparison of the approaches. While pre-training on Kinetics improves results, the use of the geometric NPL is quite beneficial.

Method	H3.6M		MLB		TSH	
	Seen	Unseen	Seen	Unseen	Seen	Unseen
Random	9.1%	9.1%	25%	25%	5.2%	5.2%
ResNet-50	86.4%	9.1%	49.4%	27.3%	34.6%	33.7%
ResNet-50 + Kinetics	**100%**	38.2%	55.6%	30.2%	49.8%	34.2%
3D Pose Based (Section 3)						
Ground Truth 3D Pose	**100%**	**100%**	-	-	19.6%	14.5%
Estimated 3D Pose	97.8%	78.6%	36.5%	33.6%	17.9%	11.6%
MultiView 2D Pose	98.3%	81.3%	37.6%	34.6%	18.4%	12.2%
Latent 3D Representation (Section 4)						
NPL	99.3%	84.4%	52.3%	34.5%	51.2%	34.7%
NPL + Multi-view projections	99.7%	87.5%	**58.9%**	**42.7%**	**54.5%**	**39.6%**

present, and thus the 3D pose estimation is not accurate enough. When using the learned, latent 3D representation (Section 4), we find in all cases the performance on the unseen views (and often seen views) improves, showing the benefit of the proposed approach. We note that the latent 3D representation generalizes quite well to challenging data with very different views and backgrounds (e.g., MLB data) because it is trained on large-scale video data, the model is able to learn more general projections.

We also find that training from scratch (ResNet-50) gives very poor performance on the unseen views. Somewhat surprising, pre-training on Kinetics, which has many different views of people performing actions, does slightly improve performance on unseen views, but still remains low, especially in the MLB and Toyota SmartHome datasets. This suggests that standard video CNNs are not learning 3D rotation invariant representations, even when given training data from many views, further showing the benefit of learning 3D view invariant representations.

6.1. Ablation Experiments

To better understand the effect of the NPL, we conduct a set of experiments to analyze each component's impact. The results are shown in Table 3. Adding the multi-view projection without any of the loss constraints slightly reduces performance. Adding the 3D loss enforces the geometric constraints to learn the viewpoint invariant representation, without this, the model struggles to learn the representation. Further adding the the camera regularization loss improves performance. Based on this observation, we also try a baseline with representation matching (RepMatch) where we apply the 3D loss (Eq. 6) to feature maps from a ResNet-50 without using any of the geometric layers. The findings shown no real benefit of RepMatch over standard pre-training, while the proposed geometric approach shows meaningful benefit.

Table 3: Comparison of the components of the approach. All models are based on a ResNet-50 and pretrained on Kinetics-400.

Method	MLB		TSH	
	Seen	Unseen	Seen	Unseen
ResNet-50 Baseline	55.6	30.2	49.8	34.2
RepMatch	55.7	30.3	48.7	33.8
+ MultiView Proj. (MVP)	52.7	28.5	47.8	33.7
+ MVP + 3D loss (Eq. 6)	57.9	35.5	52.3	37.8
+ MVP + cam reg (Eq. 7)	54.7	30.8	50.7	34.4
+ MVP + 3D loss + cam reg	58.9	42.7	54.5	39.6

We also study the effect of the number of cameras in Fig 6, finding that using just 1 camera projection is very helpful while more than 4 no longer improves performance. This intuitively makes sense, as a single camera will already result in a viewpoint invariant representation, and the amount of new data introduced with additional cameras decreases as more are added. In Fig. 7, we compare the effect of placing the geometric layer at different locations in the network. Overall, the performance is fairly stable regardless of where it is added. In Fig. 5 we visualize the learned cameras from Sec 4.1.1. In the figure, the red rectangle represents the world 3D representations space which contains the CNN features F_W (Eq. 3). The brown, blue and green markers indicate the different learned camera matrices that capture different 2D views of the space (Eq. 5).

6.2. Comparison to other approaches

We compare the proposed approach to other approaches for unseen viewpoint activity recognition. The results are

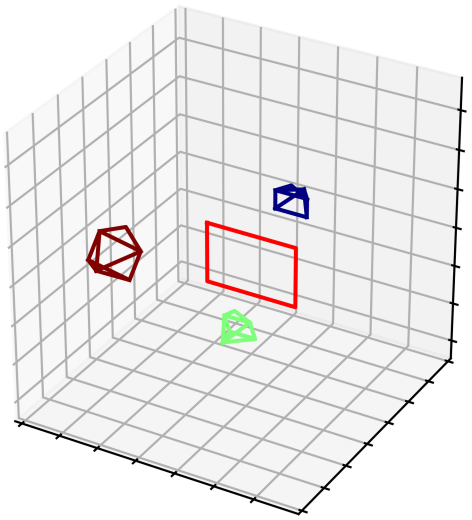

Figure 5: Visualization of the learned 2D multi-view cameras (Sec. 4.1.1). The red square represents the origin of the world coordinate system, the cameras are drawn using their intrinsic and extrinsic matrices using the matlab PlotCamera function.

Table 4: Comparison to other approaches on Toyota SmartHome (TSH) and the Unseen MLB Views and NTU-RGB+D.

Method	TSH		MLB	NTU
	CV_1	CV_2	Unseen	CV
IDT [43]	20.9	23.7	27.3	-
Pose LSTM [8]	13.4	17.2	-	-
I3D [2]	34.9	45.1	30.1	-
STA [8]	35.2	50.3	-	94.6
PEM [26]	-	-	-	**95.2**
Ours	**39.6**	**54.6**	**42.7**	93.7

shown in Table 4, showing that the proposed approach outperforms the existing ones. Importantly, the added runtime of our approach is small, processing a 3 second video clip in 120ms (ours) vs. 105ms (baseline ResNet-50), enabling practical use.

7. Related Works

Representation Invariant Networks: Many works have studied representation invariant networks. The spatial transformer network [17] and Equivarient CNNs [11] introduced an operation to make CNNs invariant to 2D translation, scale, rotation and more generic warping. Spherical CNNs [10, 7] took advantage of spherical representation that are invariant to 3D rotation transformations of objects in the camera view. Our approach shares some similar ideas and motivations to spherical CNNs, i.e., trying to learn a rotation invariant representation. But differs in the goal of learning object rotation invariance in spherical CNNs vs. world space representations in this work. Another difference is in the design of the representation: spherical CNNs rely on convolutions in the spherical harmonic domain, while the proposed approach uses traditional geometric computer vision to learn a representation. Other works like geometry-aware RNNs [5] propose the related idea of 'unprojection' for learning 3D representations by utilizing ground truth 3D data.

View Invariant Action Recognition: Many works have studied view invariance in action recognition [34, 1, 36, 12, 37, 15]. Several works have studied using multiple views during training to learn view invariant representations [45] or 'hallucinating' features (e.g., HOG) in different viewpoints to recognize actions in unseen views [4]. Several works explored using cross-view similarity to recognize actions in various viewpoints [18] or trajectory curvature [1]. Other works [33] explored using 3D Pose for recognition, but as described, pose has limitations when interacting with objects.

3D Representations: Other works have designed CNNs to specifically model 3D shapes, such as RotationNet [20] and others (e.g., ShapeNet, PointNet, etc.) [40, 3, 32]. However, these works focused on learning 3D models, rather than applications to noisy, real videos in various environments where no 3D data is directly given.

Works such as SynSin [44] propose similar ideas of using geometric projections on CNN representations, showing the promise of such ideas.

8. Conclusions

We presented a new geometric based layer to learn 3D viewpoint invariant representations within CNNs. We also introduced a new, challenging dataset to evaluate camera view invariance. We experimentally showed the benefit of the proposed layer on multiple datasets.

Acknowledgement

This work was supported in part by the National Science Foundation (IIS-2104404 and CNS-2104416).

References

[1] Faisal I Bashir, Ashfaq A Khokhar, and Dan Schonfeld. View-invariant motion trajectory-based activity classification and recognition. *Multimedia Systems*, 12(1):45–54, 2006. 7

[2] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 1, 7

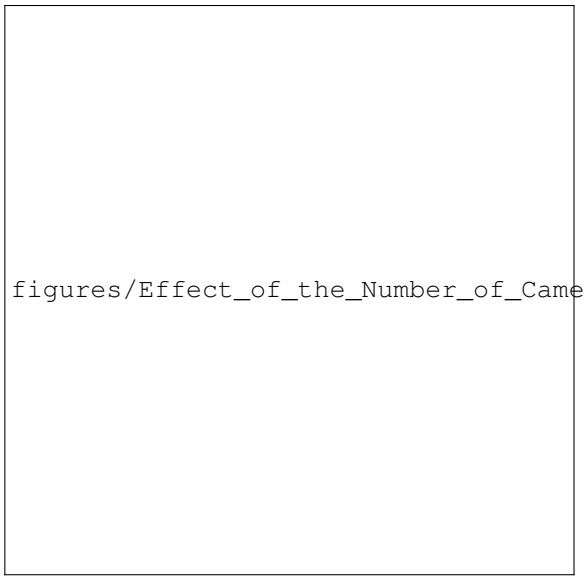

Figure 6: How many cameras to use.

Figure 7: Where in network to add layer.

[3] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015. 7

[4] Chao-Yeh Chen and Kristen Grauman. Inferring unseen views of people. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2003–2010, 2014. 7

[5] Ricson Cheng, Ziyan Wang, and Katerina Fragkiadaki. Geometry-aware recurrent neural networks for active visual recognition. In *Advances in Neural Information Processing Systems*, pages 5081–5091, 2018. 7

[6] Vasileios Choutas, Philippe Weinzaepfel, Jérôme Revaud, and Cordelia Schmid. Potion: Pose motion representation for action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7024–7033, 2018. 3

[7] Taco S Cohen, Mario Geiger, Jonas Köhler, and Max Welling. Spherical cnns. *arXiv preprint arXiv:1801.10130*, 2018. 7

[8] Srijan Das, Rui Dai, Michal Koperski, Luca Minciullo, Lorenzo Garattoni, Francois Bremond, and Gianpiero Francesca. Toyota smarthome: Real-world activities of daily living. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 833–842, 2019. 1, 5, 7

[9] Yong Du, Wei Wang, and Liang Wang. Hierarchical recurrent neural network for skeleton based action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1110–1118, 2015. 3

[10] Carlos Esteves, Christine Allen-Blanchette, Ameesh Makadia, and Kostas Daniilidis. Learning so (3) equivariant representations with spherical cnns. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 52–68, 2018. 7

[11] Carlos Esteves, Yinshuang Xu, Christine Allen-Blanchette, and Kostas Daniilidis. Equivariant multi-view networks. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 1568–1577, 2019. 7

[12] Ali Farhadi and Mostafa Kamali Tabrizi. Learning to recognize activities from the wrong view point. In *Proceedings of European Conference on Computer Vision (ECCV)*, pages 154–166. Springer, 2008. 7

[13] Christoph Feichtenhofer, Haoqi Fan, Jitendra Malik, and Kaiming He. Slowfast networks for video recognition. *arXiv preprint arXiv:1812.03982*, 2018. 1

[14] Catalin Ionescu, Dragos Papava, Vlad Olaru, and Cristian Sminchisescu. Human3.6m: Large scale datasets and predictive methods for 3d human sensing in natural environments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2014. 1

[15] Alexandros Iosifidis, Anastasios Tefas, and Ioannis Pitas. View-invariant action recognition based on artificial neural networks. *IEEE transactions on neural networks and learning systems*, 23(3):412–424, 2012. 7

[16] Ganesh Iyer, R Karnik Ram, J Krishna Murthy, and K Madhava Krishna. Calibnet: Geometrically supervised extrinsic calibration using 3d spatial transformer networks. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (‘)*, pages 1110–1117. IEEE, 2018. 2

[17] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 2017–2025, 2015. 4, 7

[18] Imran N Junejo, Emilie Dexter, Ivan Laptev, and Patrick PÚrez. Cross-view action recognition from temporal self-similarities. In *Proceedings of European Conference on Computer Vision (ECCV)*, pages 293–306. Springer, 2008. 7

[19] Angjoo Kanazawa, Jason Y Zhang, Panna Felsen, and Jitendra Malik. Learning 3d human dynamics from video. In

Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 5614–5623, 2019. 2

[20] Asako Kanezaki, Yasuyuki Matsushita, and Yoshifumi Nishida. Rotationnet: Joint object categorization and pose estimation using multiviews from unsupervised viewpoints. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5010–5019, 2018. 7

[21] Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, et al. The kinetics human action video dataset. *arXiv preprint arXiv:1705.06950*, 2017. 1, 4, 5

[22] Qiuhong Ke, Mohammed Bennamoun, Senjian An, Ferdous Sohel, and Farid Boussaid. A new representation of skeleton sequences for 3d action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3288–3297, 2017. 3

[23] Chao Li, Qiaoyong Zhong, Di Xie, and Shiliang Pu. Skeleton-based action recognition with convolutional neural networks. In *2017 IEEE International Conference on Multimedia & Expo Workshops (ICMEW)*, pages 597–600. IEEE, 2017. 3

[24] Sijin Li and Antoni B Chan. 3d human pose estimation from monocular images with deep convolutional neural network. In *Asian Conference on Computer Vision*, pages 332–347. Springer, 2014. 2

[25] Sijin Li, Weichen Zhang, and Antoni B Chan. Maximum-margin structured learning with deep networks for 3d human pose estimation. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 2848–2856, 2015. 2

[26] Mengyuan Liu and Junsong Yuan. Recognizing human actions as the evolution of pose estimation maps. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1159–1168, 2018. 7

[27] Gyeongsik Moon, Ju Yong Chang, and Kyoung Mu Lee. Camera distance-aware top-down approach for 3d multi-person pose estimation from a single rgb image. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 10133–10142, 2019. 2

[28] Francesc Moreno-Noguer. 3d human pose estimation from a single image via distance matrix regression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2823–2832, 2017. 2

[29] Eshed Ohn-Bar and Mohan Trivedi. Joint angles similarities and hog2 for action recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 465–470, 2013. 3

[30] Dario Pavllo, Christoph Feichtenhofer, David Grangier, and Michael Auli. 3d human pose estimation in video with temporal convolutions and semi-supervised training. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7753–7762, 2019. 2

[31] AJ Piergiovanni and Michael S. Ryoo. Fine-grained activity recognition in baseball videos. In *CVPR Workshop on Computer Vision in Sports*, 2018. 1, 2, 5

[32] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 5099–5108, 2017. 7

[33] Hossein Rahmani and Ajmal Mian. 3d action recognition from novel viewpoints. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1506–1515, 2016. 7

[34] Cen Rao and Mubarak Shah. View-invariance in action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages II–II. IEEE, 2001. 7

[35] Amir Shahroudy, Jun Liu, Tian-Tsong Ng, and Gang Wang. Ntu rgb+ d: A large scale dataset for 3d human activity analysis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1010–1019, 2016. 1, 5

[36] Yuping Shen and Hassan Foroosh. View-invariant action recognition using fundamental ratios. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–6. IEEE, 2008. 7

[37] Yuping Shen and Hassan Foroosh. View-invariant action recognition from point triplets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(10):1898–1905, 2009. 7

[38] Gunnar A Sigurdsson, Abhinav Gupta, Cordelia Schmid, Ali Farhadi, and Karteek Alahari. Charades-ego: A large-scale dataset of paired third and first person videos. *arXiv preprint arXiv:1804.09626*, 2018. 1

[39] Andrea Tacchetti, Leyla Isik, and Tomaso Poggio. Invariant recognition drives neural representations of action sequences. *PLoS computational biology*, 13(12):e1005859, 2017. 1

[40] Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. Multi-view 3d models from single images with a convolutional network. In *Proceedings of European Conference on Computer Vision (ECCV)*, pages 322–337. Springer, 2016. 7

[41] Du Tran, Lubomir D Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. C3d: generic features for video analysis. *CoRR, abs/1412.0767*, 2(7):8, 2014. 1

[42] Du Tran, Heng Wang, Lorenzo Torresani, Jamie Ray, Yann LeCun, and Manohar Paluri. A closer look at spatiotemporal convolutions for action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6450–6459, 2018. 11

[43] Heng Wang, Alexander Kläser, Cordelia Schmid, and Cheng-Lin Liu. Action recognition by dense trajectories. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3169–3176. IEEE, 2011. 7

[44] Olivia Wiles, Georgia Gkioxari, Richard Szeliski, and Justin Johnson. Synsin: End-to-end view synthesis from a single image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7467–7477, 2020. 7

[45] Xinxiao Wu, Han Wang, Cuiwei Liu, and Yunde Jia. Cross-view action recognition over heterogeneous feature spaces. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 609–616, 2013. 7

[46] Sijie Yan, Yuanjun Xiong, and Dahua Lin. Spatial temporal graph convolutional networks for skeleton-based action

recognition. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*, 2018. 3

A. Architecture Details

Our base model used a standard (2+1)D ResNet-50 [42]. The camera transform is inserted into the network usually after the 3rd block (in the main paper we compared all locations). Usually this network used 256 channels for the representation and we used 3 cameras (i.e., 3 different 2D projections). The total number of parameters of the 3 main models is summarized in Table 5. Our layer adds only 280k parameters (only about 1% of the parameters), but significantly improves performance on unseen views. It further has significantly better runtime performance than spherical CNNs.

Table 5: Comparison of the number of parameters in the 3 main models. Adding the geometric projection layer only adds 280k parameters, but greatly improves performance.

Model	# params	Δ
(2+1)D ResNet-50	21.3M	0
(2+1)D ResNet-50 + Ours	21.5M	280k
Spherical CNNs	21.2M	-123k

B. Full Results

The full numerical results from plots in the paper are provided here.

Table 6: How many cameras to use.

Method	MLB		TSH	
	Seen	Unseen	Seen	Unseen
Baseline	55.6	30.2	49.8	34.2
1 Cam	57.4	38.6	53.2	38.5
2 Cams	58.1	41.8	53.9	39.1
4 Cams	58.9	42.7	54.5	39.6
8 Cams	58.7	42.7	54.5	39.4

Table 7: Where in network to add layer.

Method	MLB		TSH	
	Seen	Unseen	Seen	Unseen
Block 1	57.8	42.1	54.3	39.2
Block 2	58.3	42.4	54.4	39.2
Block 3	58.9	42.7	54.5	39.6
Block 4	57.4	41.7	53.8	38.9
Block 5	57.1	40.9	53.3	37.7

C. PyTorch Implementation

We provide the code here to implement the camera projection layer.

```
import numpy as np

import torch
import torch.nn as nn
import torch.nn.functional as F

device = torch.device('cuda')

def rotation_tensor(theta, phi, psi, b=1):
  """
    Takes theta, phi, and psi and generates the
    3x3 rotation matrix. Works for batched ops
    As well, returning a Bx3x3 matrix.
  """
  one = torch.ones(b, 1, 1).to(device)
  zero = torch.zeros(b, 1, 1).to(device)
  rot_x = torch.cat((
    torch.cat((one, zero, zero), 1),
    torch.cat((zero, theta.cos(), theta.sin()), 1),
    torch.cat((zero, -theta.sin(), theta.cos()), 1),
  ), 2)
  rot_y = torch.cat((
    torch.cat((phi.cos(), zero, -phi.sin()), 1),
    torch.cat((zero, one, zero), 1),
    torch.cat((phi.sin(), zero, phi.cos()), 1),
  ), 2)
  rot_z = torch.cat((
    torch.cat((psi.cos(), -psi.sin(), zero), 1),
    torch.cat((psi.sin(), psi.cos(), zero), 1),
    torch.cat((zero, zero, one), 1)
  ), 2)
  return torch.bmm(rot_z, torch.bmm(rot_y, rot_x))


class CameraProps(nn.Module):
  """
    Generates the extrinsic rotation and translation matrix
    For the current camera. Takes some feature as input, then
    Returns the rotation matrix (3x3) and translation (3x1)
  """
  def __init__(self, channels):
    super(CameraProps, self).__init__()
    self.cam = nn.Conv2d(channels, 128, 3)
    self.cam2 = nn.Linear(128, 32)
    self.rot = nn.Linear(32, 3)
    self.trans = nn.Linear(32, 3)

  def forward(self, x):
    x = F.relu(self.cam(x))
    # averages x over space,time
    # then provides 3x3 rot and 3-dim trans
```

```
    x = torch.mean(torch.mean(x, dim=2), dim=2)
    x = F.relu(self.cam2(x))
    b = x.size(0)
    r = self.rot(x)
    return rotation_tensor(r[:,0], r[:,1], r[:,2], b), self.trans(x).view(b,3,1,1)

class CameraProjection(nn.Module):
  """
    Does the camera transforms and multi-view projection
    Described in the paper.
  """
  def __init__(self, num_cameras):
    super(CameraProjection, self).__init__()
    self.cameras = nn.ParameterList()
    self.cam_rot = nn.ParameterList()
    for c in range(num_cameras):
      self.cameras.append(nn.Parameter(torch.rand(4)*2-1))
      self.cam_rot.append(nn.Parameter(torch.rand(3)*np.pi))

  def forward(self, x, rot, trans):
    # X is a list of [F, x,y,z] feature maps
    # or X is a [C, W, H] feature map
    # rot, trans are the extensic camera parameters
    if isinstance(x, list):
      # if it is a list, process each feature map
      # resulting in a [C, W, H] as input
      output = [self.forward(f, rot, trans) for f in x]
      return torch.cat(output, dim=1) # channels is dim1
    # x is now a [F, x,y,z] input where F is the feature
    fts = x[:, :-3] # get feature value, a B x F x H x W tensor
    pt = x[:, -3:] # get 3D point locations, a B x 3 x H x W tensor

    # rot is a 3x3 matrix
    # pw is 3x3 matrix applied along dim
    pw = torch.einsum('bphw,bpq->bqhw', pt, rot)
    pw += trans # add 3D translation

    # pw is now world coordinates at each feature map location
    # we do 2d projection next
    views = []
    for r,c in zip(self.cam_rot, self.cameras):
      rot = rotation_tensor(r[0].view(1,1,1), r[1].view(1,1,1), r[2].view(1,1,1))
      cam_pt = torch.einsum('bphw,pq->bqhw', pw, rot.squeeze(0))

      proj = torch.stack([(cam_pt[:, 0]*c[0] + c[2]),
                          (cam_pt[:, 1]*c[1] + c[3])], dim=-1)
      proj = torch.tanh(proj) # apply tanh to get values in [-1,1]
      views.append(F.grid_sample(fts, proj))
    return torch.cat(views, dim=1)
```

This layer can easily be inserted anywhere into a CNN. For example, assume the following code generates a ResNet. Then the camera transform is used as:

```
class Net(nn.Module):
    def __init__(self, ...):
```

```
    self.layers = # ResNet Layers
    self.cam_props = CameraProps(channels)
    self.camera_proj = CameraProjection(num_cams)

  def forward(self, video):
    x = video
    for i,layer in enumerate(self.layers):
      x = layer(x)
      if i = apply_camera_layer_loc:
        rot, trans = self.cam_props(x)
        x = self.camera_proj(x, rot, trans)
    return x
```