Kernel Transformer Networks for Compact Spherical Convolution

Yu-Chuan Su
The University of Texas at Austin

Kristen Grauman
Facebook AI Research
The University of Texas at Austin

Abstract

Ideally, 360° imagery could inherit the deep convolutional neural networks (CNNs) already trained with great success on perspective projection images. However, existing methods to transfer CNNs from perspective to spherical images introduce significant computational costs and/or degradations in accuracy. We present the Kernel Transformer Network (KTN) to efficiently transfer convolution kernels from perspective images to the equirectangular projection of 360° images. Given a source CNN for perspective images as input, the KTN produces a function parameterized by a polar angle and kernel as output. Given a novel 360° image, that function in turn can compute convolutions for arbitrary layers and kernels as would the source CNN on the corresponding tangent plane projections. Distinct from all existing methods, KTNs allow model transfer: the same model can be applied to different source CNNs with the same base architecture. This enables application to multiple recognition tasks without re-training the KTN. Validating our approach with multiple source CNNs and datasets, we show that KTNs improve the state of the art for spherical convolution. KTNs successfully preserve the source CNN's accuracy, while offering transferability, scalability to typical image resolutions, and, in many cases, a substantially lower memory footprint[1].

1. Introduction

The 360° camera is an increasingly popular technology gadget, with sales expected to grow by 1500% before 2022 [41]. As a result, the amount of 360° data is increasing rapidly. For example, users uploaded more than a million 360° videos to Facebook in less than 3 years [2]. Besides videography, 360° cameras are also gaining attention for self-driving cars, automated drones, and VR/AR. Because almost any application depends on semantic visual features, this rising trend prompts an unprecedented need for visual recognition algorithms on 360° images.

Today's wildly successful recognition CNNs are the result of tremendous data curation and annotation effort [6, 14, 16, 30, 35, 40], but they all assume perspective projection imagery. How can they be repurposed for 360° data? Existing methods often take an off-the-shelf model trained on perspective images and either 1) apply it repeatedly to multiple perspective projections of the 360° image [10, 37, 39, 42] or 2) apply it once to a single equirectangular projection [19, 29]. See Fig. 1(A,B). These two strategies, however, have severe limitations. The first is expensive because it has to project the image and apply the recognition model repeatedly. The second is inaccurate because the visual content is distorted in equirectangular projection.

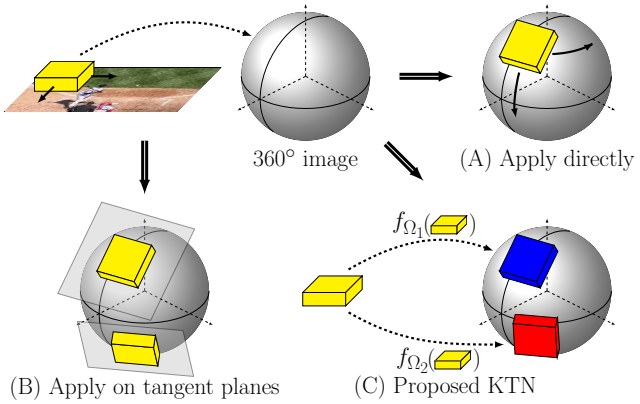

Figure 1: Our goal is to transfer CNNs trained on planar images to 360° images. Common approaches either (A) apply CNNs directly on the equirectangular projection of a 360° image or (B) project the content to tangent planes and apply the models on the tangent planes. In contrast, Kernel Transformer Network (KTN) adapts the kernels in CNNs to account for the distortion in 360° images.

To overcome these challenges, recent work designs CNN models specifically for spherical data [11, 12, 15, 36, 45]. Broadly speaking, they pursue one of three approaches. The first adapts the network architecture for equirectangular projection and trains kernels of variable size to account for its distortions [36]. While accurate, this approach suffers from significant model bloat. The second approach instead adapts the kernels on the sphere, resampling the kernels or projecting their tangent plane features [12, 45]. While allowing kernel sharing and hence smaller mod-

[1]Code and data available at http://vision.cs.utexas.edu/projects/ktn/

els, this approach degrades accuracy—especially for deeper networks—due to an implicit interpolation assumption, as we will explain below. The third approach defines convolution in the spectral domain [11, 15], which has significant memory overhead and thus far limited applicability to real-world data. All of the above require retraining to handle a new recognition task.

In light of these shortcomings, we propose the Kernel Transformer Network (KTN). The KTN adapts source CNNs trained on perspective images to 360° images. Instead of learning a new CNN on 360° images for a specific task, KTN learns a *function* that takes a kernel in the source CNN as input and transforms it to be applicable to a 360° image in its equirectangular projection. See Fig. 1 (C). The function accounts for the distortion in 360° images, returning different transformations depending on both the polar angle θ and the source kernel. The model is trained to reproduce the outputs of the source CNN on the perspective projection for each tangent plane on an arbitrary 360° image. Hence, KTN learns to behave similarly to the source CNN while avoiding repeated projection of the image.

Key highlights of the proposed KTN are its *transferability* and *compactness*—both of which owe to our function-based design. Once trained for a base architecture, the same KTN can transfer multiple source CNNs to 360° images. For example, having trained a KTN for VGG [35] on ImageNet classification, we can transfer the same KTN to run a VGG-based Pascal object detector on 360° panoramas. This is possible because the KTN takes the source CNN as input rather than embed the CNN kernels into its own parameters (unlike [11, 12, 15, 36, 45]). Furthermore, since the KTN factorizes source kernels from transformations, it is implementable with a lightweight network (e.g., increasing the footprint of a VGG network by only 25%).

Results show KTN models are orders of magnitude smaller than the most accurate competitor, SphConv [36]. Compared with Spherical U-Net [45] and SphereNet [12], KTN is much more data efficient because it does not require any annotated 360° images for training, and it is more accurate because it avoids their feature interpolation assumption.

2. Related Work

360° vision Ongoing work explores new projection models optimized for image display [5, 25, 43] or video storage [1, 4, 27, 28, 38]. We adopt the most common equirectangular projection so our algorithm can be readily applied to existing data. Other work explores how to improve the display of 360° video via video stabilization [22, 23, 26], new display interfaces [31–33], and automatic view selection [7, 10, 19, 29, 37, 39, 42]. The latter all rely on applying CNNs to 360° data, and could benefit from our method.

CNNs on spherical data As discussed above, early methods take either the expensive but accurate reprojection approach [37, 39, 42, 44], or the inaccurate but fast direct equirectangular approach [19, 29]. Recent work improves accuracy by training vanilla CNNs on the cubemap projection, which introduces less distortion [3, 7], but the model still suffers from cubemap distortion and discontinuities and has sub-optimal accuracy for tasks such as object detection.

In the last year, several methods develop new spherical CNN models. Some design CNN architectures that account for the distortion in 360° images [12, 36, 45]. SphConv [36] learns separate kernels for each row of the equirectangular projection, training to reproduce the behavior of an off-the-shelf CNN and adjusting the kernel shape based on its location on the sphere. While more accurate than a vanilla CNN, SphConv increases the model size significantly because it unties kernel weights along the rows. In contrast, SphereNet [12] defines the kernels on the tangent plane and projects features to the tangent planes before applying the kernels. Similarly, Spherical U-Net [45] defines the kernels on the sphere and resamples the kernels on the grid points for every location in the equirectangular projection. Both allow weight sharing, but they implicitly assume that features defined on the sphere can be interpolated in the 2D plane defined by equirectangular projection, which we show is problematic. Instead of learning independent kernels or using a fixed 2D transformation, our KTN learns a transformation that considers both spatial and cross-channel correlation. Our model is more compact than SphConv by sharing the kernels, and it is more accurate than SphereNet and Spherical U-Net by learning a more generic transformation.

Another strategy is to define convolution in the spectral domain in order to learn rotation invariant CNNs. One approach is to apply graph convolution and design the graph structure [24] such that the outputs are rotation invariant. Another approach transforms both the feature maps and kernels into the spectral domain and applies convolution there [11, 15]. However, orientation is often semantically significant in real data (e.g., cars are rarely upside down) and so removing orientation can unnecessarily restrict discrimination. In addition, these approaches require caching the basis functions and the frequency domain feature maps in order to achieve efficient computation. This leads to significant memory overhead and limits the viable input resolution. Both constraints limit the spectral methods' accuracy on real world 360° images. Finally, unlike any of the above prior work [3, 7, 11, 12, 15, 36, 45], our KTN can transfer across different source CNNs with the same architecture to perform new tasks without re-training; all other methods require training a new model for each task.

CNNs with geometric transformations For perspective images, too, there is interest in encoding geometric transformations in CNN architectures. Spatial transformer networks [20] transform the feature map into a canonical view to achieve transformation invariance. Active convolution [21] and deformable convolution [13] model geomet-

ric transformations using the receptive field of the kernel. While these methods account for geometric transformations in the input data, they are not suitable for 360° images because the transformation is *location* dependent rather than *content* dependent in 360° images. Furthermore, all of them model only geometric transformation and ignore the correlation between different channels in the feature map. In contrast, our method captures the properties of 360° images and the cross channel correlation in the features.

3. Approach

In this section, we introduce the Kernel Transformer Network for transferring convolutions to 360° images. We first introduce the KTN module, which can replace the ordinary convolution operation in vanilla CNNs. We then describe the architecture and objective function of KTN. Finally, we discuss the difference between KTN and existing methods for learning CNNs on 360° data.

3.1. KTN for Spherical Convolution

Our KTN can be considered as an generalization of ordinary convolutions in CNNs. In the convolution layers of vanilla CNNs, the same kernel is applied to the entire input feature map to generate the output feature map. The assumption underlying the convolution operation is that the feature patterns, i.e., the kernels, are translation invariant and should remain the same over the entire feature map. This assumption, however, does not hold in 360° images. A 360° image is defined by the visual content projected on the sphere centered at the camera's optical center. To represent the image in digital format, the sphere has to be unwrapped into a 2D pixel array, e.g., with equirectangular projection or cubemaps. Because all sphere-to-plane projections introduce distortion, the feature patterns are not translation invariant in the pixel space, and ordinary CNNs trained for perspective images do not perform well on 360° images.

To overcome this challenge, we propose the Kernel Transformer Network, which can generate kernels that account for the distortion. Assume an input feature map $I \in \mathbf{R}^{H\times W\times C}$ and a source kernel $K \in \mathbf{R}^{k\times k\times C}$ defined in undistorted images (i.e., perspective projection). Instead of applying the source kernel directly

$$F[x,y] = \Sigma_{i,j} K[i,j] * I[x-i, y-j], \tag{1}$$

we learn the KTN (f) that generates different kernels for different distortions:

$$K_\Omega = f(K, \Omega) \tag{2}$$

$$F[x,y] = \Sigma_{i,j} K_\Omega[i,j] * I[x-i, y-j] \tag{3}$$

where the distortion is parameterized by Ω. Because the distortion in 360° images is location dependent, we can define Ω as a function on the sphere

$$\Omega = g(\theta, \phi), \tag{4}$$

where θ and ϕ are the polar and azimuthal angle in spherical coordinates, respectively. Given the KTNs and the new definition of convolution, our approach permits applying an ordinary CNN to 360° images by replacing the convolution operation in Eq. 1 with Eq. 3.

KTNs make it possible to take a CNN trained for some target task (recognition, detection, segmentation, etc.) on ordinary perspective images and apply it directly to 360 panoramas. Critically, KTNs do so without using any annotated 360° images. Furthermore, as we will see below, once trained for a given architecture (e.g., VGG), the same KTN is applicable for a *new* task using that architecture without retraining the KTN. For example, we could train the KTN according to a VGG network trained for ImageNet classification, then apply the same KTN to transfer a VGG network trained for Pascal object detection; with the same KTN, both tasks can be translated to 360° images.

3.2. KTN Architecture

In this work, we consider 360° images that are unwrapped into 2D rectangular images using equirectangular projection. Equirectangular projection is the most popular format for 360° images and is part of the 360° video compression standard [8]. The main benefit of equirectangular projection for KTNs is that the distortion depends only on the polar angle. Because the polar angle has an one-to-one correspondence with the image row ($y{=}\theta H/\pi$) in the equirectangular projection pixel space, the distortion can be parameterized easily using $\Omega = g(\theta, \phi) = y$. Furthermore, we can generate one kernel and apply it to the entire row instead of generating one kernel for each location, which leads to more efficient computation.

A KTN instance is based on a given CNN architecture. There are two basic requirements for the KTN module. First, it has to be lightweight in terms of both model size and computational cost. A large KTN module would incur a significant overhead in both memory and computation, which would limit the resolution of input 360° images during both training and test time. Because 360° images by nature require a higher resolution representation in order to capture the same level of detail compared with ordinary images, the accuracy of the model would degrade significantly if we were forced to use lower resolution inputs.

Second, KTNs need to generate output kernels with variable size, because the appropriate kernel shape may vary in a single 360° image. A common way to generalize convolution kernels on the 2D plane to 360° images is to define the kernels on the tangent plane of the sphere. As a result, the receptive field of the kernel on the 360° image is the back projection of the receptive field on the tangent plane, which varies at different polar angles [12,36,45]. While one could address this naively by always generating the kernels in the largest possible size, doing so would incur significant overhead in both computation and memory.

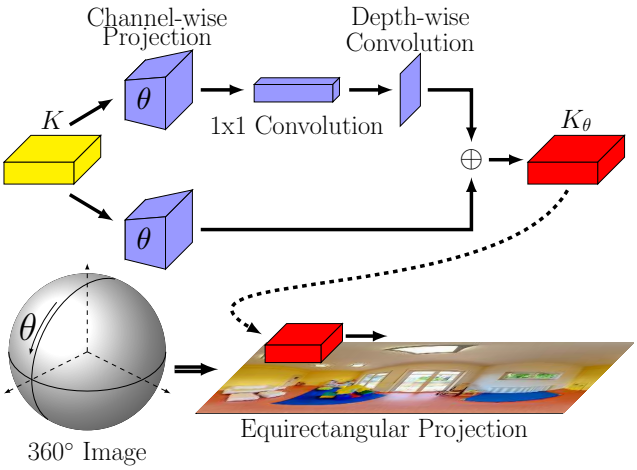

Figure 2: KTN consists of row dependent channel-wise projections that resize the kernel to the target size and depth separable convolution blocks. It takes a source kernel K and θ as input and generates an output kernel K_Ω. K_Ω is then applied to the 360° image in its equirectangular projection at row $y{=}\theta H/\pi$. The transformation accounts for the distortion in equirectangular projection, while maintaining cross-channel interactions.

We address the first requirement (size and cost) by employing depthwise separable convolutions [9, 18] within the KTN. Instead of learning 3D (i.e., height×width×channels) kernels, KTN alternates between *pointwise* convolution that captures cross-channel correlation and *depthwise* convolution that captures spatial correlation. Using the same 3x3 depthwise convolutions as in MobileNet [18], the computation cost is about 8 to 9 times less than standard convolution. Furthermore, the model size overhead for KTN is roughly $1/k^2$ of the source kernels, where most of the parameters are in the 1x1 convolution. The size overhead turns out to be necessary, because cross channel correlation is captured only by the 1x1 convolution in KTN, and removing it reduces the final spherical convolution accuracy significantly.

To address the second requirement (variable-sized kernels), we learn a row dependent depthwise projection to resize the source kernel. The projection consists of h projection matrices P_i, for $i \in [1, h]$, where h is the number of rows in the 360° image. Let $r_i = h_i \times w_i$ be the target kernel receptive field at row i. The projection matrix has the size $P_i \in \mathbf{R}^{r_i \times k^2}$, which projects the source kernel into the target size. Similar to the depthwise convolution, we perform channel-wise projection to reduce the model size.

The complete architecture for KTN is in Fig. 2. We use a Residual Network [17]-like architecture. For both the residual and shortcut branches, we first apply the row dependent projection to resize the kernel to the target size. The residual branch then applies depthwise separable convolution twice. Our depthwise separable convolution block consists of ReLU-pointwise conv-ReLU-depthwise conv. This design removes the batch normalization used in MobileNet to reduce the model size and memory consumption. The two branches are added together to generate the output kernel, which is then applied to a 360° feature map as in Eq. 3. Note that while the KTN can be applied to different kernels, the structure of a KTN depends on P_i, which is determined by the receptive field of the source kernel. Therefore, we need one KTN for each layer of a source CNN.

3.3. KTN Objective and Training Process

Having introduced the KTN module and how to apply it for CNNs on 360° images, we now describe the KTN objective function and training process. The goal of the KTN is to adapt the source kernel to the 360° domain. Therefore, we train the model to reproduce the outputs of the source kernels. Let $F^l \in \mathbf{R}^{H \times W \times C^l}$ and $F^{l+1} \in \mathbf{R}^{H \times W \times C^{l+1}}$ be the feature maps generated by the l-th and $(l{+}1)$-th layer of a source CNN respectively. Our goal is to minimize the difference between the feature map generated by the source kernels K^l and that generated by the KTN module:

$$\mathcal{L} = \|F^{l+1} - f^l(K^l, \Omega) * F^l\|^2 \tag{5}$$

for any 360° image. Note that during training the feature maps F^l are not generated by applying the source CNN directly on the equirectangular projection of the 360° images. Instead, for each point (x, y) in the 360° image, we project the image content to the tangent plane of the sphere at

$$(\theta, \phi) = (\frac{\pi \times y}{H}, \frac{2\pi \times x}{W}) \tag{6}$$

and apply the source CNN on the tangent plane. This ensures that the target training values are accurately computed on undistorted image content. $F^l[x, y]$ is defined as the l-th layer outputs generated by the source CNN at the point of tangency. Our objective function is similar to that of SphConv [36], but, importantly, we optimize the model over the entire feature map instead of on a single polar angle in order to factor the kernel itself out of the KTN weights.

The objective function depends only on the source pretrained CNN and does not require any annotated data for training. In fact, it does not require image data specific to the target task, because the loss is defined over 360° images. In practice, we sample arbitrary 360° images for training regardless of the source CNN. For example, in experiments we train a KTN on YouTube video frames and then apply it for a Pascal object detection task. Our goal is to fully reproduce the behavior of the source kernel. Therefore, even if the training images do not contain the same objects, scenes, etc. as are seen in the target task, the KTN should still minimize the loss in Eq. 5. Although KTN takes only the source kernels and θ as input, the exact transformation f may depend on all the feature maps $F^l, F^{l-1}, \ldots, F^1$ to resolve the error introduced by non-linearities. Our KTN learns the important components of those transformations from data.

Table 1: Comparison of different approaches. EQUIRECTANGULAR and CUBEMAP refer to applying the given CNN directly to the equirectangular and cubemap projection, respectively. Supervised training means that the method requires annotated 360 images. The model size is the size for a single layer, where c, k, H refer to the number of channels, kernel size, and input resolution (bandwidth) respectively. Note that $c \sim H \gg k$ for real images and source CNNs, and we keep only the leading term for each method.

	Translation Invariance	Rotation Invariance	Supervised Training	Model Size	Transferable Across Models
EQUIRECTANGULAR	No	No	No	c^2k^2	No
CUBEMAP	No	No	No	c^2k^2	No
S^2CNN [11]	Yes	Yes	Yes	c^2H	No
SPHERICAL CNN [15]	Yes	Yes	Yes	c^2H	No
SPHERICAL U-NET [45]	Yes	No	Yes	c^2k^2	No
SPHERENET [12]	Yes	No	Yes	c^2k^2	No
SPHCONV [36]	Yes	No	No	c^2k^2H	No
KTN	Yes	No	No	$c^2k^2 + c^2$	Yes

KTN's transferability across source kernels is analogous to the generalizability of visual features across natural images. In general, the more visual diversity in the unlabeled training data, the more accurately we can expect the KTN to be trained. While one could replace all convolution layers in a CNN with KTNs and train the entire model end-to-end using annotated 360° data, we believe that Eq. 5 is a stronger condition while also enjoying the advantage of bypassing any annotated training data.

3.4. Discussion

Compared to existing methods for convolution for 360° images, the main benefits of KTN are its *compactness* and *transferability*. The information required to solve the target task is encoded in the source kernel, which is fed into the KTN as an *input* rather than part of the model. As a result, the same KTN can be applied to another CNN having the same base architecture but trained for a different target task. In other words, without additional training, the same KTN model can be used to solve multiple vision tasks on 360° images by replacing the source kernels, provided that the source CNNs for each task have the same base architecture.

Most related to our work is the spherical convolution approach (SphConv) [36]. SphConv learns the kernels adapted to the distortion in equirectangular projection. Instead of learning the transformation function f in Eq. 2, SphConv learns K_Ω directly, and hence must learn one K_Ω for every different row of the equirectangular image. While SphConv should be more accurate than KTN theoretically (i.e., removing any limitations on memory and training time and data) our experimental results show that the two methods perform similarly in terms of accuracy. Furthermore, the number of parameters in SphConv is hundreds of times larger than KTN, which makes SphConv much more difficult to train and deploy. The difference in model size becomes even more significant when there are multiple models to be evaluated: the same KTN can apply to multiple source CNNs and thus incurs only constant overhead,

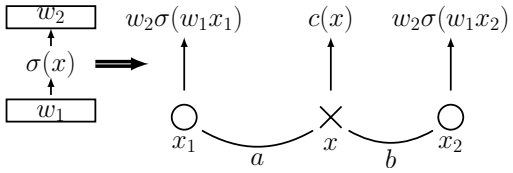

Figure 3: Beyond the first CNN layer, the feature interpolation assumption in SphereNet [12] yields only approximated results. See text for details.

whereas SphConv must fully retrain and store a new model for each source CNN. For example, if we want to apply five different VGG-based CNNs to 360° images, SphConv will take 29×5=145GB of space, while KTN takes only 56×5+14=294MB (cf. Sec. 4.3). In addition, since SphConv trains K_Ω for a single source kernel K, the model does not generalize to different source CNNs.

SphereNet [12] formulates the transformation function f using the sphere-to-tangent-plane image projection. While the projection transformation leads to an analytical solution for f, it implicitly assumes that CNN feature maps can be interpolated like pixels. This assumption is only true for the first layer in a network because of non-linear activation functions used in modern CNNs between convolution layers. Consider a two layer 1D convolution with a kernel of size 1, as sketched in Fig. 3. If we interpolate the pixel first and apply the kernels, the output of at location x is

$$c(x) = w_2 \times \sigma(w_1(ax_1 + bx_2)). \tag{7}$$

However, if we apply the kernels and then interpolate the features, the result is

$$c(x) = aw_2 \times \sigma(w_1x_1) + bw_2 \times \sigma(w_1x_2). \tag{8}$$

These two values are not equal because σ is non-linear, and the error will propagate as the network becomes deeper. The interpolated feature can at most be an approximation for the exact feature. Our experimental results show that a projection transformation for f leads to sub-optimal performance.

Finally, other methods attempt to reduce distortion by unwrapping a single 360° image into multiple images using perspective projection locally [3,7], e.g., with cubemap projection. It is non-trivial to define convolution across multiple image planes, where two cube faces meet. Prior work addresses this problem by "cube-padding" the feature maps using output from adjacent image planes [3, 7], but experimental results indicate that the resultant features are not accurate enough and degrade the accuracy. The reason is that the same object may have different appearance on different tangent planes, especially when the field-of-view is large and introduces significant perspective distortion. Alternatively, one could sample the tangent planes densely and apply convolution on each tangent plane independently, but doing so incurs unrealistic computational overhead [37].

Table 1 summarizes the tradeoffs between existing spherical convolution models. In short, KTN is distinct from all others in its ability to transfer to new tasks without any labeled data. Furthermore, KTN has the favorable properties of a highly compact model and the ability to preserve orientation-specific features (typically desirable for recognition and other high-level tasks).

4. Experiments

We evaluate KTN on multiple datasets and multiple source models. The goal is to 1) validate the accuracy of KTN as compared to other methods for learning CNNs on 360° images, 2) demonstrate KTN's ability to generalize to novel source models, and 3) examine KTN's memory and computation overhead compared to existing techniques.

Datasets Our experiments make use of both unannotated 360° videos and 360° images with annotation.

Spherical MNIST is constructed from the MNIST dataset by back projecting the digits into equirectangular projection with 160×80 resolution. The digit labels are used to train the source CNN (recognition model), but they are *not* used to train the KTN. Classification accuracy on the 360°-ified test set is used as the evaluation metric.

Pano2Vid is a real world 360° video dataset [39]. We sample frames from non-overlapping videos for training and testing, and the frames are resized to 640×320 resolution. The models are trained to reproduce the convolution outputs of the source model, so no labels are required for training. The root-mean-square error (RMSE) of the final convolution outputs is used as the evaluation metric.

Pascal VOC 2007 is a perspective image dataset with object annotations. We backproject the object bounding boxes to equirectangular projection with 640×320 resolution. Following [36], we use the accuracy of the detector network in Faster R-CNN on the validation set as the evaluation metric. This dataset is used for evaluation only.

Source Models For *Spherical MNIST*, we train the source CNN on the MNIST training set. The model consists of three convolution layers followed by one fully connected layer. Each convolution layer consists of 5x5Conv-MaxPool-ReLU, and the number of kernels is 32, 64, and 128, respectively. For *Pano2Vid* and *Pascal VOC*, we take off-the-shelf Faster R-CNN [34] models with VGG architecture [35] as the source model. The Faster R-CNN is trained on Pascal VOC if not mentioned specifically. Source models are not fine-tuned on 360° data in any form.

Baselines We compare to the following existing methods:

- EQUIRECTANGULAR—Apply ordinary CNNs on the 360° image in its equirectangular projection.
- CUBEMAP—Apply ordinary CNNs on the 360° image in its cubemap projection.
- S^2CNN [11]—We train S^2CNN using the authors' implementation. For *Pano2Vid* and *Pascal VOC*, we reduce the input resolution to 64×64 due to memory limits (see Supp). We add a linear read-out layer at the end of the model to generate the final feature map.
- SPHERICAL CNN [15]—We train SPHERICAL CNN using the authors' implementation. Again, the resolution of input is scaled down to 80×80 due to memory limits for *Pano2Vid* and *Pascal VOC*.
- SPHERICAL U-NET [45]—We use the spherical convolution layer in Spherical U-Net to replace ordinary convolution in CNN. Input resolution is reduced to 160×80 for *Pano2Vid* and *Pascal VOC* due to memory limits.
- SPHERENET [12]—We implement SPHERENET using row dependent channel-wise projection.[2] We derive the weights of the projection matrices using the feature projection operation and train the source kernels. For the *Pano2Vid* dataset, we train each layer independently using the same objective as KTN due to memory limits.
- SPHCONV [36]—We use the authors' implementation.
- PROJECTED—Similar to SPHERENET, except that it uses the source kernels without training.

The network architecture for EQUIRECTANGULAR and CUBEMAP is the same as the source model. For all methods, the number of layers and kernels are the same as the source model.

Note that the resolution reductions specified above were necessary to even run those baseline models on the non-MNIST datasets, even with state-of-the-art GPUs. All experiments were run on NVIDIA V100 GPU with 16GB memory—the largest in any generally available GPU today. Therefore, the restriction is truly imposed by the latest hardware technology. Compatible with these limits, the resolution in the authors' own reported results is restricted to 60 × 60 [11], 64 × 64 [15], or 150 × 300 [45]. On the *SphericalMNIST* dataset, all methods use the exact same

[2]The authors' code and data were not available at the time of publication.

Table 2: Model accuracy.

	MNIST (Acc.↑)	*Pano2Vid* (RMSE ↓)	*Pascal VOC* (Acc.↑)
EQUIRECTANGULAR	95.24	3.44	41.63
CUBEMAP	68.53	3.57	49.29
S^2CNN [11]	95.79	2.37	4.32
SPHERICAL CNN [15]	97.48	2.36	6.06
SPHERICAL U-NET [45]	98.43	2.54	24.98
SPHERENET [12]	87.20	2.46	46.68
SPHCONV [36]	**98.72**	**1.50**	63.54
PROJECTED	10.70	4.24	6.15
KTN	97.94	1.53	**69.48**

Figure 4: KTN object detection examples on *Pano2Vid*. See Supp. for detection examples on *Pascal VOC*.

image resolution. The fact that KTN scales to higher resolutions is precisely one of its technical advantages, which we demonstrate on the other datasets.

For *Spherical MNIST*, the baselines are trained to predict the digit projected to the sphere except SPHCONV. SPHCONV and our KTN are trained to reproduce the conv3 outputs of the source model. For *Pano2Vid*, all methods are trained to reproduce the conv5_3 outputs.

Please see Supp. file for additional details.

4.1. Model Accuracy

Table 2 summarizes the methods' CNN accuracy on all three 360° datasets. KTN performs on par with the best baseline method (SPHCONV) on *Spherical MNIST*. The result verifies that KTN can transfer the source kernels to the entire sphere by learning to reproduce the feature maps, and it can match the accuracy of existing models trained with annotated 360° images.

KTN and SPHCONV perform significantly better than the other baselines on the high resolution datasets, i.e., *Pano2Vid* and *Pascal VOC*. S^2CNN, SPHERICAL CNN, and SPHERICAL U-NET suffer from their memory constraints, which as discussed above restricts them to lower resolution inputs. Their accuracy is significantly worse on realistic full resolution datasets. These models cannot take higher resolution inputs even after using model parallelism over four GPUs with a total of 64GB of memory. Although EQUIRECTANGULAR and CUBEMAP are trained and applied on the full resolution inputs, they do not account for the distortion in 360° images and yield lower accuracy. Finally, the performance of PROJECTED and SPHERENET suggests that the transformation f cannot be modeled by a tangent plane-to-sphere projection. Although SPHERENET shows that the performance can be significantly improved by training the source kernels on 360° images, the accuracy is still worse than KTN because feature interpolation introduces error. The error accumulates across layers, as discussed in Sec. 3.4, which substantially degrades the accuracy when applying a deep CNN. Note that the number of learnable parameters in KTN is much smaller than that in SPHERENET, but it still achieves a much higher accuracy.

Interestingly, although SPHCONV performs better in RMSE on *Pano2Vid*, KTN peforms better in terms of object classification accuracy on *Pascal VOC*. We attribute this to KTN's inherent generalizability. SPHCONV has a larger number of parameters, and the kernels at different θ are trained independently. In contrast, the parameters in KTN are shared across different θ and thus trained with richer information. Therefore, SPHCONV is more prone to overfit the training loss, which is to minimize the RMSE for both models. Furthermore, our KTN has a significant compactness advantage over SPHCONV, as discussed above.

Similarly, although SPHERICAL U-NET and SPHERENET perform slightly worse than S^2CNN and SPHERICAL CNN on *Pano2Vid*, they are significantly better than those baselines on *Pascal VOC*. This result reinforces the practical limitations of imposing rotation invariance. S^2CNN and SPHERICAL CNN require full rotation invariance; the results show that orientation information is in fact important in tasks like object recognition. Thus, the additional rotational invariance constraince limits the expressiveness of the kernels and degrades the performance of S^2CNN and SPHERICAL CNN. Furthermore, the kernels in S^2CNN and SPHERICAL CNN may span the entire sphere, whereas spatial locality in kernels has proven important in CNNs for visual recognition.

Fig. 4 shows example outputs of KTN with a Faster R-CNN source model. The detector successfully detects objects despite the distortion. On the other hand, KTN can fail when a very close object cannot be captured in the field-of-view of perspective images.

4.2. Transferability

Next, we evaluate the transferability of KTN across different source models on *Pano2Vid*. In particular, we evaluate whether KTNs trained with a Faster R-CNN that is trained on COCO can be applied to another Faster R-CNN (both using VGG architecture) that is trained on Pascal VOC and vice versa. We denote KTN trained on a different source CNN than it is being tested on as KTN-TRANSFER and KTN otherwise.

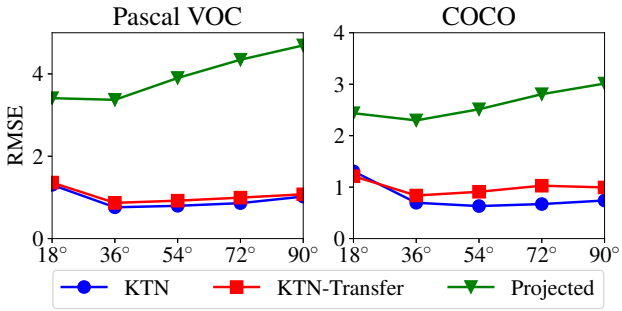

Figure 5: Model transferability. The title indicates the source CNN being tested. KTN performs almost identically regardless of the source network it is trained on. The results show we can learn a single KTN and apply it to other source CNNs with the same architecture, even if that source model is trained for a different task.

Fig. 5 shows the results. The accuracy of KTN-TRANSFER is almost identical to KTN. The results demonstrate that KTN indeed learns a task-independent transformation and can be applied to different source models with the same base architecture. None of the existing models [11, 12, 15, 36, 45] are equipped to perform this kind of transfer, because they learn fixed kernels for a specific task in some form. Hence, the PROJECTED baseline is the only baseline shown in Fig. 5. Although PROJECTED can be applied to any source CNN without training, the performance is significantly worse than KTN. Again, the results indicate that a projection operation is not sufficient to model the required transformation f. The proposed KTN is the first approach to spherical convolution that translates across models without requiring labeled 360° images or retraining. We also perform the same experiments between VGG trained for ImageNet classification and Faster R-CNN trained for Pascal object detection, and the results are similar. See Supp.

4.3. Size and Speed

Finally, we compare the overhead introduced by KTN versus that required by the baseline methods. In particular, we measure the model size and speed for the convolution layers in the VGG architecture. For the model size, we compute the total size of the parameters using 32-bit floating point numbers for the weights. While there exist algorithms that compress neural networks, they are equally applicable for all methods. For the speed, we measure the average processing time (I/O excluded) of an image for computing the conv5_3 outputs. All methods are evaluated on a dedicated AWS p3.8xlarge instance. Because the model size for SPHCONV is 29GB and cannot fit in GPU memory (16GB), it is run on CPUs. Other methods are run on GPUs.

Fig. 6 shows the results. We can see that the model size of KTN is very similar to EQUIRECTANGULAR, CUBEMAP and PROJECTED. In fact, it is only 25% (14MB) larger than

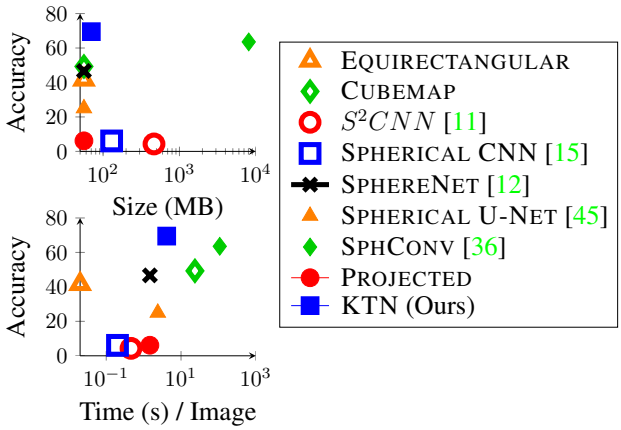

Figure 6: Model size (top) and speed (bottom) vs. accuracy for VGG. KTN is orders of magnitude smaller than SPHCONV, and it is similarly or more compact as all other models, while being significantly more accurate.

the source CNN. At the same time, KTN achieves a much better accuracy compared with all the models that have a comparable size. Compared with SPHCONV, KTN not only achieves a higher accuracy but is also orders of magnitude smaller. Similarly, S^2CNN and SPHERICAL CNN increase model size by 131% and 727% while performing worse in terms of accuracy. Note that we do not include parameters that can be computed analytically, such as the bases for S^2CNN and the projection matrices for SPHERENET, though in practice they also add further memory overhead for those baselines.

On the other hand, the computational cost of KTN is naturally much higher than EQUIRECTANGULAR. The latter only needs to run the source CNN on an equirectangular image, whereas the convolution kernels are generated at run time for KTN. However, as all the results show, KTN is much more accurate. Furthermore, KTN is 26 times faster than SPHCONV, since the smaller model size allows the model to be evaluated on GPU.

5. Conclusion

We propose the Kernel Transformer Network for transfering CNNs from perspective images to 360° images. KTN learns a function that transforms a kernel to account for the distortion in the equirectangular projection of 360° images. The same KTN model can transfer to multiple source CNNs with the same architecture, significantly streamlining the process of visual recognition for 360° images. Our results show KTN outperforms existing methods while providing superior scalability and transferability.

Acknowledgement. We thank Carlos Esteves for the help on SPHERICAL CNN experiments. This research is supported in part by NSF IIS-1514118, an AWS gift, a Google PhD Fellowship, and a Google Faculty Research Award.

References

[1] David Newman Adeel Abbas. A novel projection for omni-directional video. In *Proc.SPIE 10396*, 2017. 2

[2] Brent Ayrey and Christopher Wong. Introducing facebook 360 for gear vr. https://newsroom.fb.com/news/2017/03/introducing-facebook-360-for-gear-vr/, March 2017. 1

[3] Wouter Boomsma and Jes Frellsen. Spherical convolutions and their application in molecular modelling. In *NIPS*, 2017. 2, 6

[4] Chip Brown. Bringing pixels front and center in VR video. https://www.blog.google/products/google-vr/bringing-pixels-front-and-center-vr-video/, March 2017. 2

[5] Che-Han Chang, Min-Chun Hu, Wen-Huang Cheng, and Yung-Yu Chuang. Rectangling stereographic projection for wide-angle image visualization. In *ICCV*, 2013. 2

[6] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *ECCV*, 2018. 1

[7] Hsien-Tzu Cheng, Chun-Hung Chao, Jin-Dong Dong, Hao-Kai Wen, Tyng-Luh Liu, and Min Sun. Cube padding for weakly-supervised saliency prediction in 360° videos. In *CVPR*, 2018. 2, 6

[8] Byeongdoo Choi, Ye-Kui Wang, and Miska M. Hannuksela. Wd on iso/iec 23000-20 omnidirectional media application format. ISO/IEC JTC1/SC29/WG11, 2017. 3

[9] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *CVPR*, 2017. 4

[10] Shih-Han Chou, Yi-Chun Chen, Kuo-Hao Zeng, Hou-Ning Hu, Jianlong Fu, and Min Sun. Self-view grounding given a narrated 360° video. In *AAAI*, 2018. 1, 2

[11] Taco Cohen, Mario Geiger, Jonas Köhler, and Max Welling. Spherical cnns. In *ICLR*, 2018. 1, 2, 5, 6, 7, 8

[12] Benjamin Coors, Alexandru Paul Condurache, and Andreas Geiger. Spherenet: Learning spherical representations for detection and classification in omnidirectional images. In *ECCV*, 2018. 1, 2, 3, 5, 6, 7, 8

[13] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. In *ICCV*, 2017. 2

[14] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: a large-scale hierarchical image database. In *CVPR*, 2009. 1

[15] Carlos Esteves, Christine Allen-Blanchette, Ameesh Makadia, and Kostas Daniilidis. Learning so(3) equivariant representations with spherical cnns. In *ECCV*, 2018. 1, 2, 5, 6, 7, 8

[16] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. Convolutional two-stream network fusion for video action recognition. In *CVPR*, 2016. 1

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 4

[18] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 4

[19] Hou-Ning Hu, Yen-Chen Lin, Ming-Yu Liu, Hsien-Tzu Cheng, Yung-Ju Chang, and Min Sun. Deep 360 pilot: Learning a deep agent for piloting through 360° sports video. In *CVPR*, 2017. 1, 2

[20] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *Advances in Neural Information Processing Systems*, pages 2017–2025, 2015. 2

[21] Yunho Jeon and Junmo Kim. Active convolution: Learning the shape of convolution for image classification. In *CVPR*, 2017. 2

[22] Mostafa Kamali, Atsuhiko Banno, Jean-Charles Bazin, In So Kweon, and Katsushi Ikeuchi. Stabilizing omnidirectional videos using 3d structure and spherical image warping. In *IAPR MVA*, 2011. 2

[23] Shunichi Kasahara, Shohei Nagai, and Jun Rekimoto. First person omnidirectional video: System design and implications for immersive experience. In *ACM TVX*, 2015. 2

[24] Renata Khasanova and Pascal Frossard. Graph-based classification of omnidirectional images. In *ICCV Workshops*, 2017. 2

[25] Yeong Won Kim, Chang-Ryeol Lee, Dae-Yong Cho, Yong Hoon Kwon, Hyeok-Jae Choi, and Kuk-Jin Yoon. Automatic content-aware projection for 360° videos. In *ICCV*, 2017. 2

[26] Johannes Kopf. 360° video stabilization. *ACM Transactions on Graphics (TOG)*, 35(6):195, 2016. 2

[27] Evgeny Kuzyakov and David Pio. Under the hood: Building 360 video. https://code.facebook.com/posts/1638767863078802/under-the-hood-building-360-video/, October 2015. 2

[28] Evgeny Kuzyakov and David Pio. Next-generation video encoding techniques for 360 video and VR. https://code.facebook.com/posts/1126354007399553/next-generation-video-encoding-techniques-for-360-video-and-vr/, January 2016. 2

[29] Wei-Sheng Lai, Yujia Huang, Neel Joshi, Chris Buehler, Ming-Hsuan Yang, and Sing Bing Kang. Semantic-driven generation of hyperlapse from 360° video. *IEEE Transactions on Visualization and Computer Graphics*, PP(99):1–1, 2017. 1, 2

[30] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*, 2014. 1

[31] Yen-Chen Lin, Yung-Ju Chang, Hou-Ning Hu, Hsien-Tzu Cheng, Chi-Wen Huang, and Min Sun. Tell me where to look: Investigating ways for assisting focus in 360 video. In *CHI*, 2017. 2

[32] Yung-Ta Lin, Yi-Chi Liao, Shan-Yuan Teng, Yu-Ju Chung, Liwei Chan, and Bing-Yu Chen. Outside-in: Visualizing out-of-sight regions-of-interest in a 360° video using spatial picture-in-picture previews. In *UIST*, 2017. 2

[33] Amy Pavel, Björn Hartmann, and Maneesh Agrawala. Shot orientation controls for interactive cinematography with 360 video. In *UIST*, 2017. 2

[34] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NIPS*, 2015. 6

[35] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. 1, 2, 6

[36] Yu-Chuan Su and Kristen Grauman. Learning spherical convolution for fast features from 360° imagery. In *NIPS*, 2017. 1, 2, 3, 4, 5, 6, 7, 8

[37] Yu-Chuan Su and Kristen Grauman. Making 360° video watchable in 2d: Learning videography for click free viewing. In *CVPR*, 2017. 1, 2, 6

[38] Yu-Chuan Su and Kristen Grauman. Learning compressible 360° video isomers. In *CVPR*, 2018. 2

[39] Yu-Chuan Su, Dinesh Jayaraman, and Kristen Grauman. Pano2vid: Automatic cinematography for watching 360° videos. In *ACCV*, 2016. 1, 2, 6

[40] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In *ICCV*, 2015. 1

[41] Ville Ukonaho. Global 360 camera sales forecast by segment: 2016 to 2022. https://www.strategyanalytics.com/access-services/devices/mobile-phones/emerging-devices/market-data/report-detail/global-360-camera-sales-forecast-by-segment-2016-to-2022, March 2017. 1

[42] Youngjae Yu, Sangho Lee, Joonil Na, Jaeyun Kang, and Gunhee Kim. A deep ranking model for spatio-temporal highlight detection from a 360° video. In *AAAI*, 2018. 1, 2

[43] Lihi Zelnik-Manor, Gabriele Peters, and Pietro Perona. Squaring the circle in panoramas. In *ICCV*, 2005. 2

[44] Yinda Zhang, Shuran Song, Ping Tan, and Jianxiong Xiao. Panocontext: A whole-room 3d context model for panoramic scene understanding. In *ECCV*, 2014. 2

[45] Ziheng Zhang, Yanyu Xu, Jingyi Yu, and Shenghua Gao. Saliency detection in 360° videos. In *ECCV*, 2018. 1, 2, 3, 5, 6, 7, 8