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

The supplementary materials consist of:

A Complete architecture of KTN
B Experimental details
C Model transferability experiment on *Pascal VOC*
D Comparison of model accuracy versus depth
E Discussion of multiple projections baselines
F Additional qualitative detection examples

1. KTN Architecture

In this section, we show the complete architecture of KTN. Fig. 1 shows how to apply KTN for spherical convolution. For each layer $l \in \{1, \cdots, L\}$ of the source CNN, we learn a function f^l that transforms the source kernel K^l to $K^l_{\theta_i}$ for every $\theta \in [0, \pi]$. The output kernel $K^l_{\theta_i}$ is then applied to the 360° equirectangular image at the corresponding row of θ_i. We find that it is unnecessary to generate one kernel for each row in the equirectangular projection, because spherical convolution kernels for adjacent rows are usually similar. In practice, we share the same kernel every five rows to reduce the computational cost and model size. Fig. 2 shows the full architecture of KTN. KTN uses a ResNet-like architecture. For both branches, it uses a row dependent channel-wise projection to resize the kernel to the target size. The residual branch then applies two depth separable convolution blocks before adding the output with that of the shortcut branch. Each depth separable convolution block consists of ReLU-pointwise conv-ReLU-depthwise conv.

To compute the target kernel size at a given polar angle, we first back project the receptive field of the source kernel to equirectangular projection. The minimum bounding box centered at the polar angle that can cover the receptive field on equirectangular projection is then selected as the target kernel shape. Note that we restrict the kernel height and width to be an odd number to ensure that the kernel is defined on the equirectangular pixel space. Because the size of the back projected receptive field may grow rapidly and span the entire image, we restrict the actual kernel width and height to be less then 65 pixels and dilate the kernel to increase the effective receptive field if necessary.

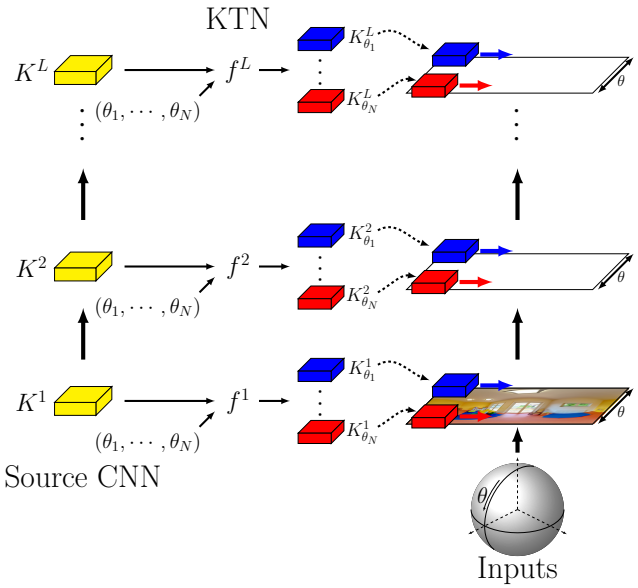

Figure 1: Application of spherical convolution using KTN.

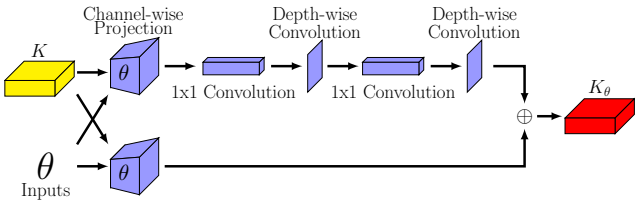

Figure 2: Full architecture of KTN.

For Spherical Faster R-CNN, we define the bounding boxes on the tangent plane: we project the features to the tangent plane and apply the RPN and detector networks there. The implementation is the same as SPHCONV [7].

2. Experimental Details

In this section, we describe additional experimental details that could not fit in the main paper.

2.1. Datasets

The following is an expanded version of the dataset descriptions in the main text.

Spherical MNIST is constructed from the MNIST dataset

by backprojecting the digits into equirectangular projection. The resolution of the resultant 360° image is 160×80, and the digit covers a 65.5° field-of-view (FOV). For the training set, we project each digit to a random polar angle $\theta\in[0,\pi]$. For the test set, we project each digit to nine different polar angles $\theta=8^\circ, 16^\circ, \ldots, 72^\circ$, which results in a test set that is nine times larger. Note that we do not rotate the digit itself because digits are oriented by definition (e.g. 6 versus 9). All baselines are trained to predict the digit label on the *Spherical MNIST* training set except SPHCONV, which does not require such labels. Both KTN and SPHCONV are trained to re-produce the top-most convolution output (conv3). Classification accuracy averaged across θ is used as the evaluation metric.

Pano2Vid is a real world 360° video dataset [8]. It contains 86 videos from four categories: "Hiking," "Parade," "Soccer," and "Mountain Climbing." Following [7], we sample 1,056 frames from the first three categories for training and 168 frames from the last category for testing, and the frames are resized to 640×320 resolution. The root-mean-square error (RMSE) over the final convolution outputs is used as the evaluation metric.

Pascal VOC is a perspective image dataset with object annotations. Similar to *Spherical MNIST*, we backproject the object bounding boxes to equirectangular projection but with 640×320 resolution. Each bounding box is projected to different polar angles $\theta\in\{18^\circ, 36^\circ, 54^\circ, 72^\circ, 90^\circ\}$ and covers a 65.5° FOV. Because the perspective images do not cover the full 360° FOV, regions outside the FOV of the original image are zero-padded (black). This dataset is used for evaluation only. Following the experiment setting of the Faster R-CNN [6] source model, we evaluate all methods on the validation set of Pascal VOC 2007. We use the accuracy of the detector network in Faster R-CNN as the evaluation metric. The ground truth bounding box is used for ROI-pooling during evaluation for all methods.

2.2. Baselines

In this section, we expand on the implementation details of each baseline method. We keep the number of layers and kernels the same for all methods. For the *Spherical MNIST* dataset, the models consist of three convolution layers followed by a max-pooling over the spatial dimensions and a fully connected layer. The convolution layers have 32, 64, and 128 kernels respectively, and the resolution of the feature map is reduced by a factor of two using max-pooling after each convolution layer. For the *Pano2Vid* and *Pascal VOC* datasets, the models have the same number of layers and kernels as the VGG16 architecture. Following SPHCONV [7], we remove the max-pooling operation in the network and use dilated convolution with factor of two in the conv5 layers to increase the receptive field. The differences between different methods are in the convolution and pooling operations as described below.

- EQUIRECTANGULAR—Apply ordinary CNNs on the 360° image in its equirectangular projection.
- CUBEMAP—Apply ordinary CNNs on the 360° image in its cubemap projection, with cube padding [1]. For the PANO2VID and PASCAL VOC datasets, the conv5_3 feature map is re-projected to equirectangular projection as the final output.
- S^2CNN [2]—We use the S2Convolution and SO3Convolution in the authors' implementation[1] for convolution. S2Convolution is applied in the first convolution layer, and SO3Convolution is used for the other layers. The default near identity grid is used for both S2 and SO3 convolution. Furthermore, we reduce the feature map resolution by reducing the output bandwidth instead of using max-pooling following the authors' implementation. The input resolution is 80×80 for *Spherical MNIST* and 64×64 for *Pano2Vid* and *Pascal VOC*. For *Spherical MNIST*, we use SO(3) integration instead of max-pooling to reduce the final feature map. For *Pano2Vid* and *Pascal VOC*, because the output of SO3Convolution is a 3D feature map, we add a 1x1 convolution layer on top of the conv5_3 output to generate a 2D feature map. The feature map is then resized to 640×320 as the final output. We reduce the output bandwidth in conv2_2 and conv3_3 and distribute the model to four NVIDIA V100 GPUs using model parallelism due to the GPU memory limit.
- SPHERICAL CNN [4]—We use the sphconv module in the authors' implementation[2] for convolution. Similar to S^2CNN, we replace max-pooling with spectral pooling. Furthermore, we apply batch normalization in each convolution layer following the example code. The input resolution is 80×80 for all datasets. For the *Pano2Vid* and *Pascal VOC* dataset, we reduce the output bandwidth in conv4_1 and conv5_1 due to the memory limit. The conv5_3 feature map is resized to 640×320 as the final output.
- SPHERICAL U-NET [9]—We use the SphericalConv module in Spherical U-Net[3] for convolution. We apply batch normalization and set the kernel size to 8×4 following the authors' example. For the *Pano2Vid* and *Pascal VOC* dataset, the input is resized to 160×80 due to memory limit, and the conv5_3 feature map is resized to 640×320 as the final output. The model is distributed to four NVIDIA V100 GPUs using model parallelism.
- SPHERENET [3]—We implement the SPHERENET model using row dependent channel-wise projection. The authors' code and data were unavailable at the time of submission. Because feature projection is the weighted sum of the features, the projection weights can be com-

[1] https://github.com/jonas-koehler/s2cnn

[2] https://github.com/daniilidis-group/spherical-cnn

[3] https://github.com/xuyanyu-shh/Saliency-detection-in-360-video

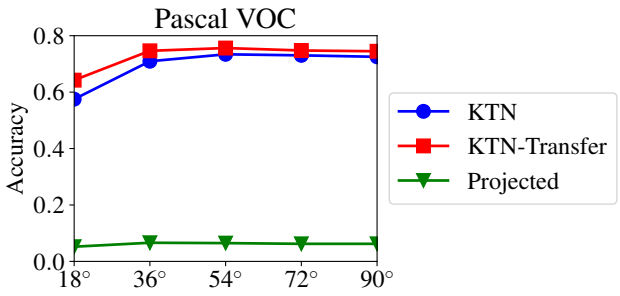

Figure 3: Model transferability on *Pascal VOC*.

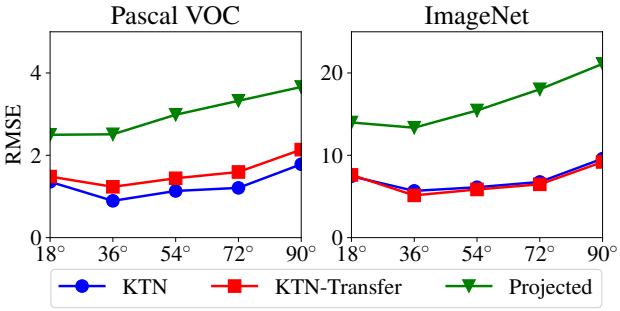

Figure 4: Model transferability of ImageNet trained VGG.

bined with the kernel weights as a single kernel. We derive the weights of the channel-wise projection using the feature projection operation and train the source kernels. For the *Pano2Vid* dataset, we train each layer independently using the same objective function as KTN because the entire model cannot fit in GPU memory.

- SPHCONV [7]—We use the authors' implementation[4]. Because the model is too large to fit into GPU memory even for evaluation, it is run on CPUs.[5]
- PROJECTED—Assuming that the kernel transformation f can be modeled using the tangent plane-to-sphere projection, we derive the analytic solution for the kernels K_θ using bilinear interpolation.

Note that the aspect ratio for the inputs is 1:1 for S^2CNN and SPHERICAL CNN. This is the requirement of the methods, so we reduce the resolution along the azimuthal angle. The input aspect ratio for all other methods is 2:1 following the common format of 360° images.

2.3. Training Details

We train all the methods using ADAM [5] for 40 epochs. The learning rate is initialized to 1.0×10^{-3} and is decreased by a factor of 10 after 20 epochs. We also apply L2 regularization with weight 5×10^{-4}. For *Pano2Vid*, the batch size is set to one for S^2CNN, two for *Spherical CNN*, and four for all other methods, which is again limited by the memory. For *Spherical MNIST*, the batch size is set to 64 for all methods except S^2CNN, which uses a batch size of 16. The weights are randomly initialized using a normal distribution with standard deviation 0.01. The training time for KTN on *Pano2Vid* is about a week using six AWS p3.8xlarge instances with V100 GPUs.

[4]https://github.com/sammy-su/Spherical-Convolution

[5]For the other baselines, testing is still possible with GPUs.

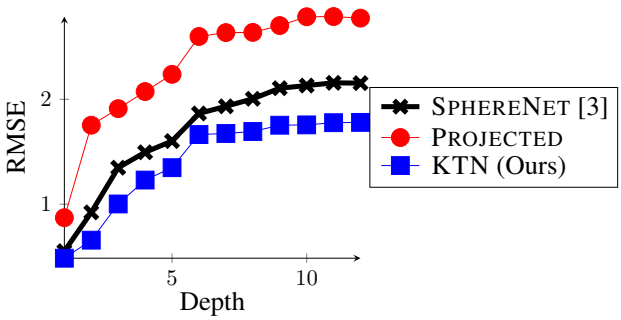

Figure 5: Model accuracy at different layers.

3. Transferability on *Pascal VOC*

As noted in the main paper, in this section, we evaluate the transferability of KTN on *Pascal VOC*. In particular, we measure whether the KTN model trained on a VGG source model can be applied to Faster R-CNN to perform object detection. The result is in Fig. 3. Again, KTN performs almost identical regardless of the source model on which it is trained. We also evaluate the transferability between VGG trained for ImageNet classification and Faster R-CNN trained for Pascal object detection. The result is in Fig. 4. The results are consistent with that in Sec. 4.3 of the main paper and verifies that KTN is transferable across source CNNs with the same architecture.

4. Model Accuracy versus Depth

As discussed in the main paper, the interpolation assumption made by SPHERENET [3] and the PROJECTED baselines is problematic, particularly at deeper layers as errors accumulate. Hence, we compare the accuracy of SPHERENET [3], PROJECTED, and KTN with different network depths. We change the network depth by feeding in the ground truth value of the intermediate layer and compare the RMSE of conv5_3 outputs. The experiment is performed on *Pano2Vid* using Faster R-CNN source model.

The results are in Fig. 5. Not surprisingly, the error increases as the model depth increases for all methods. More importantly, the gap between KTN and the other methods increases as the network becomes deeper. The results suggest that the error of interpolated features increases as the number of non-linearities increases and is consistent with the analysis in Sec. 3.4 in the main paper.

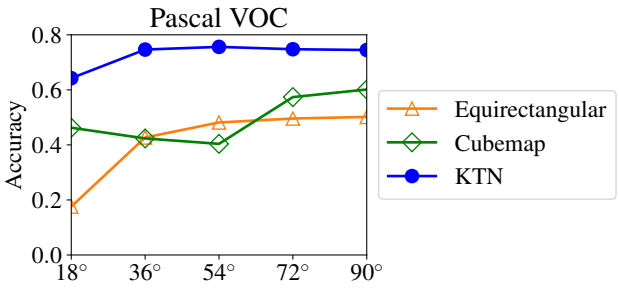

Figure 6: Model accuracy of projection based baselines.

5. Multiple Projections Baseline

Fig. 6 shows that the worst accuracy of KTN (at θ=18°) outperforms the best accuracy of EQUIRECTANGULAR and CUBEMAP (at θ=90°). While a possible method for improving the performance of the projection based methods (i.e. EQUIRECTANGULAR and CUBEMAP) is to aggregate the detection results from multiple projections to reduce the effect of distortion, the results suggest that EQUIRECTANGULAR and CUBEMAP is less accurate then KTN even if they are always evaluated on the less distorted region. This implies that KTN will always be more accurate than EQUIRECTANGULAR and CUBEMAP no matter how many different projections we sample. Furthermore, evaluating the model on multiple projections increases the computational cost and introduces the problem of how to combine detection results, which is non-trivial especially in dense prediction problems such as depth prediction.

6. Object Detection Examples

In this section, we show additional object detection examples. Fig. 8 and Fig. 9 show object detection examples on the *Pano2Vd* and *Pascal VOC* dataset, respectively. Notice how KTN can detect the distorted objects by translating the source CNN appropriately to the spherical data.

Fig. 7 show failure examples on the *Pano2Vid* dataset. In the first example, the model fails to capture the entire human body and returns two positive detections instead of one. This is caused by the fact that our method cannot handle close objects that cannot be captured by the FOV of perspective images. In the second example, the model fails on the top view of the person because a top view is very rare in ordinary images. The result indicates that the data distribution is different in 360° images and perspective images. The performance of the model may be further improved if we can train the source CNNs on 360° images.

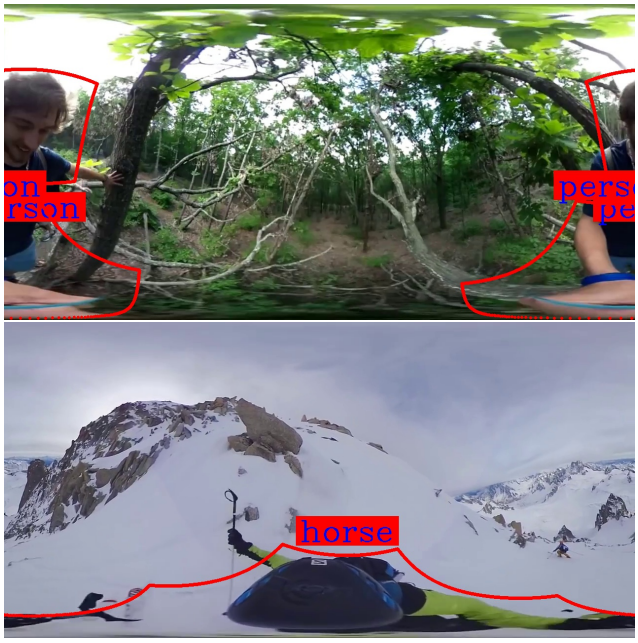

Figure 7: Failure cases on *Pano2Vid*.

References

[1] 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

[2] Taco Cohen, Mario Geiger, Jonas Köhler, and Max Welling. Spherical cnns. In *ICLR*, 2018. 2

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

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

[5] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 3

[6] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NIPS*, 2015. 2

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

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

[9] Ziheng Zhang, Yanyu Xu, Jingyi Yu, and Shenghua Gao. Saliency detection in 360° videos. In *ECCV*, 2018. 2

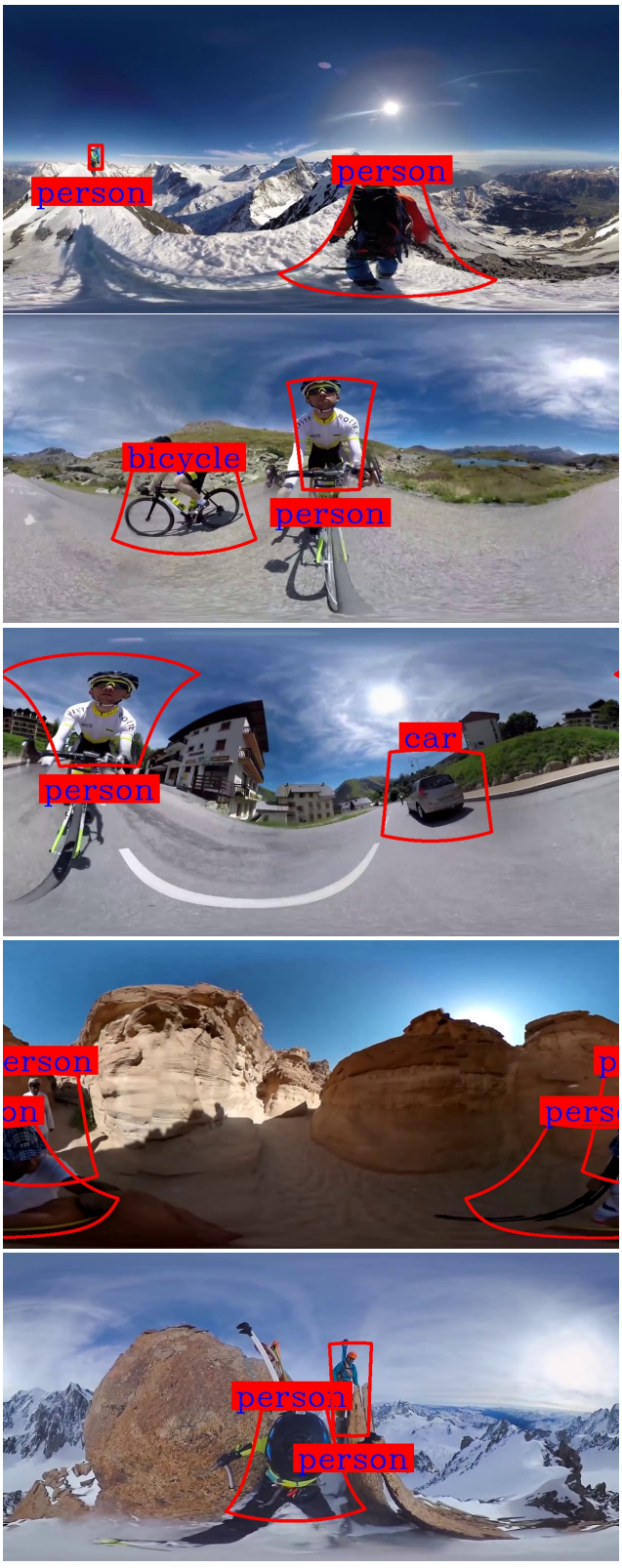

Figure 8: Object detection examples on *Pano2Vid*.

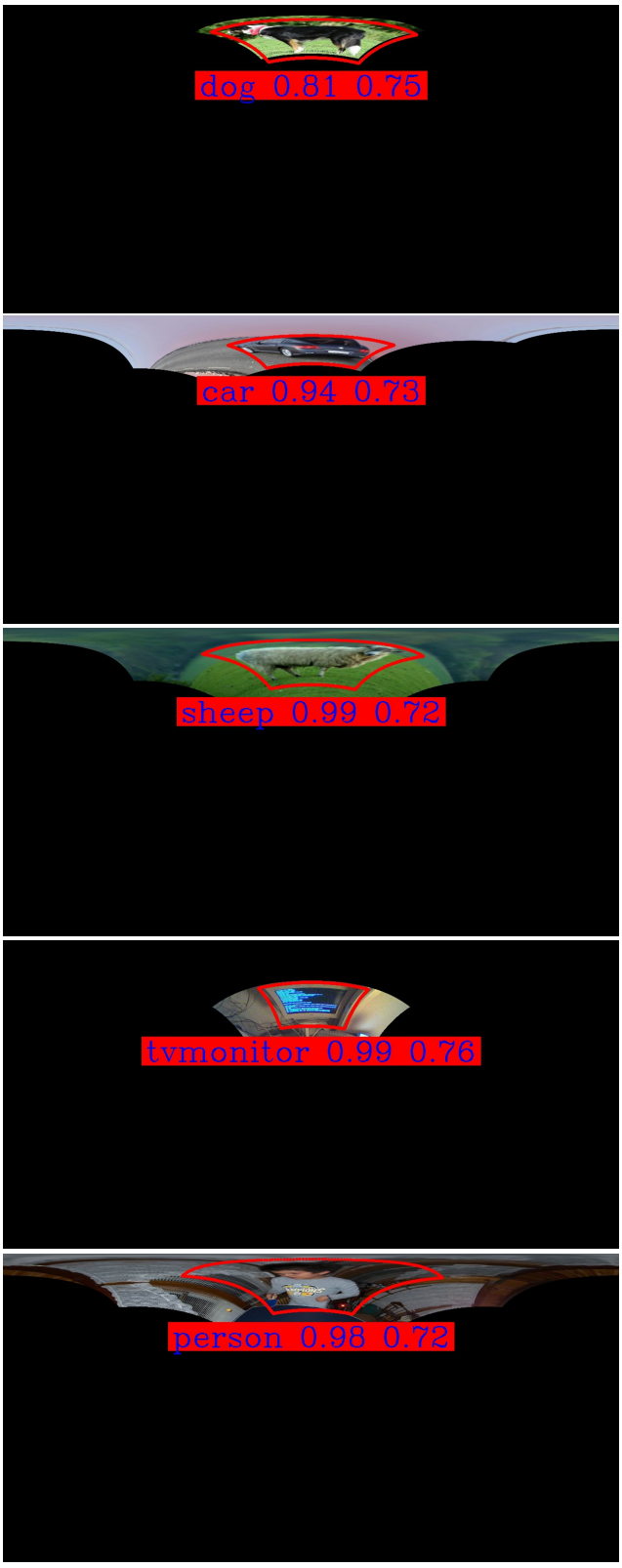

Figure 9: Object detection examples on 360-ified *Pascal VOC images*.