The Dissertation Committee for Yu-Chuan Su
certifies that this is the approved version of the following dissertation:

Learning for 360° Video Compression, Recognition, and Display

Committee:

Kristen Grauman, Supervisor

Noah Snavely

Qixing Huang

Scott Niekum

Learning for 360° Video Compression, Recognition, and Display

by

Yu-Chuan Su

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

August 2019

Dedicated to my family.

Acknowledgments

I would like to express my very great appreciation to my advisor Kristen Grauman. Her passion for research has always been inspiring and pushes me to achieve a higher standard. I have learned how to be a good researcher and how to convey my research effectively from her. Without her guidance and support, all my achievements during PhD would not have been possible. She is truly a great mentor and is a perfect role model for a researcher.

I would also like to thank all my thesis committee members Prof. Noah Snavely, Prof. Scott Niekum, and Prof. Qixing Huang for their insightful comments on this thesis. Their feedback helped to make my research more complete and strengthen this thesis.

My sincere thank also goes to my labmates. I learn a lot from my current and former labmates, including Chao-Yeh, Suyog, Dinesh, Aron, Bo, Danna, Wei-Lin, Ruohan, Antonino, Ziad, Tushar, and Santhosh. I really enjoy the time we spent together during the conferences, in the reading group, in the board game night, and many others. I especially want to thank Chao-Yeh for all the suggestions and advice even before I started my PhD. My life as a PhD student would have been much more difficult without his help. I also want to thank Dinesh for the inspiring discussions in research and his brilliant idea that led me into the research presented in this thesis.

I am thankful to Prof. Winston Hsu, Prof. Chih-Jen Lin, Prof. Hsuan-Tien Lin, Dr. Yan-Ying Chen, Dr. Yin-Hsi Kuo, and Guan-Long Wu. Without their help, it would be impossible for me to have the opportunity to pursue my PhD at UT Austin and to receive fellowships from the university and Google. Thanks also to Sudheendra Vijayanarasimhan and David Ross for mentoring my internship at Google, and my friends Cho-Jui, Si Si, Kai-Yang, Hsiang-Fu, Kai, En-Hsu, Pei-Chi, Yi-Hung, Wei-Ju, Yi-Hsuan, Yi-Shan, Yao-Lun, Hsin-Yu, Chao-Yuan, Chia-Chen, Liang-Chieh, Ting, Jyh-Jing, Chien-Hung, Bor-Chun, Yu-Hsueh, Min-Hua, Tsung-Pu, and Jung-Wei for sharing all the joyable memory throughout my PhD.

Finally, this thesis would not have been possible without the unconditional support and love from my family: my father San-Leng Su, my mother Chiu-Yun Shen, and my sister Yu-Wen Su. I am grateful to my parents for allowing me to pursue my goal freely and being by my side during the most depressfull moments, and my sister for always being so positive and encouraging for my decision. There's nothing that can describe my gratitude, and I dedicate this thesis to my family.

Learning for 360° Video Compression, Recognition, and Display

Publication No. _____

Yu-Chuan Su, Ph.D.
The University of Texas at Austin, 2019

Supervisor: Kristen Grauman

360° cameras are a core building block of the Virtual Reality (VR) and Augmented Reality (AR) technology that bridges the real and digital worlds. It allows us to build virtual environments for VR/AR applications from the real world easily by capturing the entire visual world surrounding the camera simultaneously. With the rapid growth of VR/AR technology, the availability and popularity of 360° cameras are also growing faster than ever. People now create, share, and watch 360° content in their everyday life, and the amount of 360° content is increasing rapidly.

While 360° cameras offer tremendous new possibilities in various domains, they also introduce new technical challenges. These challenges span over the entire 360° video production pipeline, ranging all the way from video capturing to high level applications. For example, the 360° field-of-view makes it difficult to display the content to users, and the distortion in the planar pro-

jection degrades the performance of both the compression and visual recognition algorithms. Many of the challenges remain unsolved or even unexplored, which prohibits people from exploiting the full potential of the new media. This leads to a dire need for a more mature 360° video production pipeline like those for traditional media.

To this end, my thesis targets three fundamental challenges in 360° production—video compression, visual recognition, and 360° video display. Because a proper compressed format is the foundation of all video technologies and applications, I begin with improving 360° video compression. It has been shown that existing video codecs do not perform well on 360° video, and 360° video compression standard is under rapid development. Complementary to the ongoing progress in 360° video compression standards, I propose to exploit the orientation of the 360° video projection for a better compression rate. The method explores a new dimension in video compression and is compatible with existing compression technologies. It reduces video sizes to allow easy storage and transmission of 360° videos.

Besides being able to collect and distribute 360° content, another prerequisite for building advanced applications on top of the new media is the ability to analyze the visual content. Therefore, I next study visual recognition on 360° imagery. I propose a general approach that transfers an existing Convolutional Neural Network (CNN) trained on perspective images to 360° imagery. It allows us to transfer knowledge from perspective images to 360° images, including both the network architecture and training data. Compared

with existing strategies for applying existing CNN models on 360° data, the method sacrifices neither accuracy nor efficiency and does not need any additional annotation effort. The method allows us to perform visual recognition on the new format given an existing CNN model with zero manual labor.

After building the basis for 360° video applications, I finally tackle one of the most important applications of 360° video: displaying the video content to users. The common interface for 360° video display requires human viewers to actively control "where" to look while watching the video. This task is non-trivial, and a poor navigation will lead to a sub-optimal user experience. I propose to address this problem by controlling the viewing direction automatically and formulate it as an automatic videography problem in 360° video. I further propose a data-driven approach that finds important content in 360° videos and controls the viewing direction to focus on the discovered content. The method takes the burden of choosing "where to look" off the human viewer and provides an easier and better viewing experience.

Combining the proposed solutions, my research builds a basic pipeline for 360° video production ranging from video storage, processing, to display. This pipeline can also serve as the basis for other 360° video applications such as 360° video editing and will allow both the content creators and technology developers to further explore the possibility of the new media.

Table of Contents

Acknowledgments **v**

Abstract **vii**

List of Tables **xiii**

List of Figures **xiv**

Chapter 1. Introduction **1**

1.1 360° Video Compression 4

1.2 Visual Recognition in 360° Imagery 5

1.3 360° Video Display 8

Chapter 2. Related Work **11**

2.1 360° Image Projection 11

2.2 Video Compression 13

2.2.1 360° Video Compression 13

2.2.2 Deep Learning for Image/Video Compression 14

2.3 Convolutional Neural Networks with Geometric Transformations 15

2.4 Convolutional Neural Networks on Spherical Data 16

2.5 Knowledge Distillation 17

2.6 Automatic Video Editing and Generation 18

2.6.1 Video Summarization 19

2.6.2 Video Retargeting 19

2.6.3 Virtual Cinematography 20

2.6.4 Video Saliency 20

2.6.5 Virtual Cinematography in 360° Video 21

Chapter 3. Learning Compressible 360° Video Isomers **22**
3.1 360° Video Isomers Analysis 25
3.1.1 Cubemap Preparation 27
3.1.2 Cubemap Analysis 30
3.2 Approach for Exploiting 360° Video Isomers 39
3.3 Experiments 44
3.3.1 Dataset 45
3.3.2 Evaluation Metrics 46
3.3.3 Baselines 47
3.3.4 Implementation Details 47
3.3.5 Results 48
3.4 Conclusion 57

Chapter 4. Learning Spherical Convolution **59**
4.1 Spherical Convolution Definition 62
4.2 Approach for Learning Spherical Convolution 64
4.2.1 Network Structure 64
4.2.2 Training Process 68
4.3 Experiments 70
4.3.1 Dataset 70
4.3.2 Evaluation Metrics 71
4.3.3 Baselines 72
4.3.4 Implementation Details 74
4.3.5 Results 75
4.4 Conclusion 81

Chapter 5. Kernel Transformer Networks for Compact Spherical Convolution **84**
5.1 Kernel Transformer Network Definition 87
5.2 Approach for Learning Kernel Transformer Network 90
5.2.1 KTN Architecture 90
5.2.2 KTN Objective and Training Process 94
5.2.3 Spherical Faster R-CNN 95

5.2.4 Discussion 99
5.3 Experiments 102
5.3.1 Dataset 103
5.3.2 Source Models 104
5.3.3 Baselines 104
5.3.4 Results 108
5.4 Conclusion 117

Chapter 6. Learning 360° Video Display **119**
6.1 Pano2Vid Definition 122
6.2 Approach for Solving Pano2Vid 123
6.2.1 Capture-worthiness Score 124
6.2.2 Trajectory Search 128
6.3 Experiments 136
6.3.1 Dataset 136
6.3.2 Evaluation Metrics 140
6.3.3 Baselines 143
6.3.4 Results 144
6.4 Conclusion 150

Chapter 7. Future Work **151**
7.1 User Aware 360° Video Compression 151
7.2 Learning Automatic Cinematography from User Control 152

Chapter 8. Conclusion **154**

Bibliography **156**

Vita **173**

List of Tables

3.1 Achievable video size reduction through rotation 35
3.2 Correlation of isomer size across video formats 37
3.3 Correlation of isomer size across bitrate 38
3.4 Video size reduction of our approach 48
3.5 Video size reduction for lossy video compression 50
3.6 Transferability across video formats 54
3.7 JVET test condition results 56

5.1 Spherical convolution approaches comparison 102
5.2 Spherical convolution approaches accuracy 108

6.1 Pano2Vid results 144
6.2 HumanEdit consistency. 148

List of Figures

1.1 Targeted challenges 2

3.1 360° video isomer 23
3.2 Cubemap format transformation 29
3.3 Isomer size distribution 31
3.4 Isomer size distribution of a single clip 32
3.5 Explanations for isomer size difference 33
3.6 Examples for isomer size difference 33
3.7 Achievable size reduction at different compression rate 36
3.8 Model for isomer size prediction 39
3.9 Proposed 360° video compression pipeline 43
3.10 JVET test sequences 44
3.11 Absolute video size reduction 49
3.12 360° video isomer qualitative examples 51
3.13 360° video isomer failure examples 53
3.14 Ω^{min} distribution 55

4.1 Existing strategies for applying CNN to 360° images 60
4.2 SphConv objective 63
4.3 Inverse perspective projection examples 65
4.4 Heuristic for SphConv kernel height 66
4.5 Spherical convolution 67
4.6 SphConv output error 75
4.7 SphConv computational cost 76
4.8 SphConv kernels visualization 78
4.9 SphConv object detection accuracy 79
4.10 SphConv object detection IoU 80
4.11 SphConv object detection examples 82

4.12 SphConv failure cases 83

5.1 Kernel transformer network 85
5.2 KTN vs. SphConv 86
5.3 KTN transferability illustion 87
5.4 KTN architecture 92
5.5 Spherical NMS 96
5.6 Spherical Faster R-CNN 98
5.7 Feature interpolation error 100
5.8 KTN accuracy across θ 110
5.9 Spherical convolution accuracy at different layers 111
5.10 KTN object detection examples 113
5.11 KTN transferability 114
5.12 KTN transferability for VGG 115
5.13 KTN model size and speed 116

6.1 Pano2Vid definition 120
6.2 AutoCam outline 124
6.3 Example glimpses scored by AutoCam 127
6.4 Coarse-to-fine camera trajectory search 131
6.5 Diverse trajectory search 135
6.6 Pano2Vid example frames 137
6.7 HumanEdit interface 138
6.8 AutoCam baselines 143
6.9 AutoCam qualitative result 146
6.10 AutoCam multiple solution example 147
6.11 AutoCam computational cost 149

Chapter 1

Introduction

360° cameras are gaining popularity as part of the virtual reality (VR) and augmented reality (AR) technologies, and will also be increasingly influential for wearable cameras, autonomous mobile robots, and video-based security applications. Unlike a traditional perspective camera, which samples a limited field-of-view (FOV) of the 3D scene projected onto a 2D plane, a 360° camera captures the entire viewing sphere surrounding its optical center, providing a complete picture of the visual world—an omnidirectional field of view. A videographer no longer has to determine which direction to capture in the scene, and a human video consumer has the freedom to explore the visual content based on her interest, without being severely restricted by choices made by the videographer. As such, viewing 360° imagery provides a more immersive experience of the visual content compared to traditional media.

Following the rising trend of VR/AR, the sales of 360° cameras are expected to grow by 1500% from 2016 to 2022 [115]. Foreseeing the tremendous opportunities in 360° video, many companies are investing in it. For example, Facebook and YouTube have offered 360° content support since 2015. Facebook users have since uploaded more than one million 360° videos [8], and

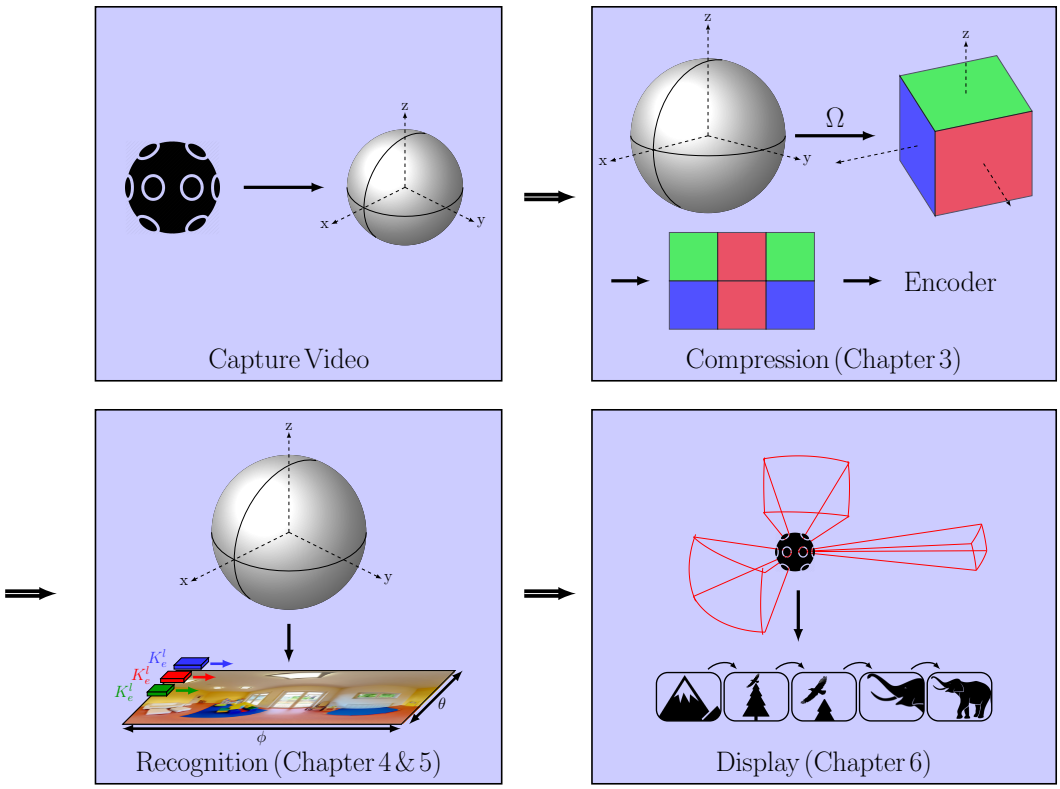


Figure 1.1: Targeted challenges in the 360° video production pipeline. For 360° video compression, I propose to improve the compression rate by exploiting the orientation of cubemap projection (Chapter 3). For visual recognition in 360° imagery, I propose a general approach that transfers Convolution Neural Networks trained on perspective images to 360° images (Chapter 4 and Chapter 5). For 360° video display, I propose to learn virtual camera control from Web videos to help users determine where to look in the video (Chapter 6). The proposed methods help to build a more mature 360° video production pipeline.

YouTube plans to bring 360° videos to even broader platforms (TV, gaming consoles). 360° editing tools are now available in popular video editors such as PowerDirector and Premiere Pro. All together, these efforts make 360° video production and distribution easier and more prevalent than ever.

The rapidly growing number of 360° content incurs an unprecedented need for technologies to handle the new media. However, it remains relatively unexplored in various aspects. The difference between 360° videos and traditional videos introduces many new challenges throughout the video production pipeline, starting from low-level image capture and compression, mid-level visual content analysis and processing, to high-level video editing and display. The research community has just started to explore these challenges. My thesis focuses on three essential problems in the pipeline: 360° *video compression*, *visual recognition in* 360° *imagery*, and 360° *video display*. The solutions for these challenges form the basis for a 360° video production pipeline. See Fig. 1.1.

Throughout the thesis, I assume complete 360° imagery as the input. The 360° imagery is defined as the visual content projected onto the unit sphere centered at the 360° camera optical center. Although most 360° cameras consist of multiple cameras with limited FOV (i.e. FOV $< 360°$), they usually convert the raw data into common 360° imagery before exporting, and most 360° content being distributed is "360°". Also, assuming 360° input instead of raw camera data allows us to focus on the key challenges and enables the methods to generalize across camera models. The following sections briefly introduce these challenges and overview the proposed solutions. Technical details and evaluation will be introduced in the following chapters.

1.1 360° Video Compression

A compressed video format is the foundation of all video technologies, and 360° video is no exception. Without proper compression, all 360° video applications will have very limited scale in terms of both data and users. Therefore, I begin with the challenge of 360° video compression. Evidence has shown that existing video codecs are sub-optimal for 360° videos, and 360° specific video formats are under rapid development [5,22,33,43]. The common strategy for compressing 360° video is to project the panoramic image into a rectangular image and then encode the resulting rectangular video using off-the-shelf video codecs [5, 22, 43, 71]. Therefore, the focus has been finding a proper projection that transforms a 360° frame into a rectangular image that will have a high compression rate. The most common projection for existing 360° videos is equirectangular projection due to its popularity in other applications such as cartography, but it is not very friendly for existing video codecs. Alternatively, cubemap projection and its variants have been shown to improve the compression rate by up to 25% and is the preferable format for future standards [12, 70, 86].

While the content of 360° video is defined on the sphere and is equivalent under rotation, the cubemap projection is defined by both the video content and the orientation of the cube. Therefore, there exists multiple possible cubemap projections representing the very same video content. Our key observation is that these projections are not equivalent for the video compression algorithm—some orientations are more compressible than others using

existing video codecs. This is because some cube orientations better preserve the properties of perspective image in the unwrapped cubemap, which leads to higher compression rates. We perform a detailed analysis to verify that the orientation of the cubemap projection is important for the ultimate video size. The results across three popular codecs show scope for reducing video sizes by up to 77% through rotation, with an average of more than 8% over all videos.

We further introduce an approach to exploit this unique property of 360° video for a better compression. The proposed method predicts the cube orientation that will yield the maximal compression rate. Given video clips in their original encoding, a Convolutional Neural Network (CNN) learns the association between a clip's visual content and its compressibility at different rotations of a cubemap projection. Given a novel video, the model efficiently infers the most compressible direction in one shot, without repeated rendering and compression of the source video. This approach achieves 78% the compression rate of the optimal orientation while requiring less than 0.6% of the computation of a search based solution. Furthermore, the approach explores a new dimension in video compression and is compatible with ongoing efforts in video compression algorithms and formats, both 360° specific and generally. This part of my thesis first appears in [106] and is described in Chapter 3.

1.2 Visual Recognition in 360° Imagery

While a better compression algorithm allows us to collect and distribute 360° videos more easily and benefit many applications of 360° video, many po-

tential applications would be impossible if we were unable to perform visual recognition on the new format. In fact, extracting visual features has become part of the standard pipeline in media sharing sites [59]. However, most existing computer vision algorithms and models are designed specifically for traditional perspective images. Applying these models on 360° images naively leads to significant performance drop, because the underlying projections are different and have different geometric properties. In other words, the models for perspective images will suffer from the distortion in 360° image projections. On the other hand, rebuilding visual recognition algorithms and models on 360° data is inefficient since the underlying tasks remain the same. Therefore, a general approach that can transfer existing visual recognition models trained on perspective images to 360° images efficiently is highly desirable. In particular, we focus on transferring a CNN model, given that CNNs are arguably the most powerful tools in computer vision today [15,30,45,80,92,114]. These CNN models encode both the knowledge of the model design and the massive training data for the target task obtained from perspective images.

There exist two common strategies for applying existing CNN models on 360° imagery. The first one projects the entire 360° image into a rectangular one, usually using equirectangular projection, and then applies the CNN on the resulting 2D image. The problem is that any sphere-to-plane projection introduces distortion, and the resulting convolutions are inaccurate. The second strategy repeatedly projects the visual content to the tangent planes of the sphere and then applies the CNN on the tangent planes. While this may

yield a more accurate result given dense enough tangent plane sampling, it introduces very high computational cost due to the reprojection process. An alternative solution is to re-train the CNN models on 360° images. However, this requires re-collecting the labeled training data in 360° format, which is very expensive and time consuming. Moreover, how to collect annotations such as object bounding boxes or segmentations in 360° format itself is a non-trivial problem.

To address this problem, we propose to *learn* a spherical convolutional network that translates a planar source CNN to process 360° imagery directly in its equirectangular projection. The approach learns to reproduce the flat filter outputs on 360° data, sensitive to the varying distortion effects across the viewing sphere. The key benefits are 1) efficient feature extraction for 360° images and video, and 2) the ability to leverage powerful pre-trained networks researchers have carefully honed (together with massive labeled image training sets) for perspective images. To enable the translation, we propose a systematic procedure to adjust the network structure from the source network in order to account for the distortions. Furthermore, we propose a kernel-wise pre-training procedure that greatly reduces the computational resources required during training and significantly accelerates the training process. We validate the approach compared to several alternative methods in terms of both raw CNN output accuracy as well as applying a state-of-the-art "flat" object detector to 360° data. The proposed method yields the most accurate results while saving orders of magnitude in computation versus the existing

exact reprojection solution. This part of my thesis first appears in [104] and is described in Chapter 4.

Despite the accuracy and efficiency, spherical convolution introduces a new technical challenge—the model size for a spherical convolution network is orders of magnitude larger than an ordinary CNN. In fact, it may even be larger than the GPU memory size, which makes it difficult to deploy spherical convolution networks in practice. To overcome this challenge, we further propose to learn a transformation function that can generate the spherical convolution network from a planar CNN. By avoid learning and storing the spherical convolution network explicitly, the proposed method can greatly reduce the model size and make the method more tractable on current hardware. Furthermore, because the transformation function takes the planar CNN as input, the same transformation function can be applied to different planar CNNs to perform different visual recognition tasks on 360° images. The resulting approach, kernel transformer network, allows us to transfer any existing computer vision algorithm implemented in a CNN to 360° images and perform visual recognition on 360° data as long as the model is available in flat data. This part of my thesis first appears in [107] and is described in Chapter 5.

1.3 360° Video Display

The solutions for the previous two challenges serve as the foundation for many applications of 360° video. I next turn to one of the most important applications of 360° video—displaying the video to human viewers. While

the larger FOV of 360° videos is usually considered as a benefit, it becomes a challenge when it comes to display because there is no single best way to display a 360° FOV to users. Displaying the entire 360° panoramic video inevitably introduces distortion. Moreover, watching a video with a FOV larger than that of human vision is unintuitive and difficult. The current trend is to display only a small portion of the content and allow the user to determine where to look in the 360° video axis. This incurs another problem for the human viewer—where and what should the users look at in the 360° video? This is a non-trivial task, because users have no information beyond the current FOV and may miss important content in the video. Awkward interfaces to navigate the video and uninformed control may lead to suboptimal viewing experiences. A better way to display 360° video is therefore very important, especially for user generated videos that lack professional editing.

To address this difficulty, we define "Pano2Vid", a new computer vision problem. The task is to design an algorithm to automatically control the pose and motion of a virtual normal field-of-view (NFOV) camera within an input 360° video. The output is the NFOV video captured by this virtual camera. Camera control must be optimized to produce video that could conceivably have been captured by a human observer equipped with a *real* NFOV camera. A successful Pano2Vid solution would therefore take the burden of choosing "where to look" off both the videographer and the end viewer: the videographer could enjoy the moment without consciously directing her camera, while the end viewer could watch intelligently-chosen portions of the video in the

familiar NFOV format.

We propose an algorithm that solves the Pano2Vid problem in a data driven approach and does not require human labor. The algorithm first learns a discriminative model of human-captured NFOV web videos. It then uses this model to identify candidate viewpoints and events of interest to capture in the 360° video, before finally stitching them together through optimal camera motions using a dynamic programming formulation for presentation to human viewers. Because we are the first to address the problem, we compile a dataset of 360° videos downloaded from the web, together with human-selected NFOV camera trajectories. We also define multiple evaluation metrics that measure how close a Pano2Vid algorithm's output videos are to human-generated NFOV videos. Experiment results show that the proposed algorithm is able to generate viewing paths that look like human captured video and provide an easier and more effective way to present the 360° video to human viewers. This part of my thesis first appears in [105, 108] and is described in Chapter 6.

Our proposed methods improve the 360° video production pipeline by addressing three fundamental challenges. The new pipeline provides a better compression of 360° video for easier storage and distribution, allows accurate yet efficient analysis of 360° video content, and enables a more efficient presentation of the 360° video to human viewers. For the rest of the thesis, Chapter 2 provide the literature survey for related research. Chapter 3 through 6 give a detailed description for the proposed solutions and evaluation. Chapter 7 discusses the future directions for my research.

Chapter 2

Related Work

In this chapter, I will review research related to my thesis. First, I will review literature presenting sphere-to-plane projection (Sec. 2.1). This is the foundation for 360° video and all the following research. Next, I will review works related to video compression (Sec. 2.2), which is related to the first challenge I tackle. I will then review works studying CNNs with geometric transformations (Sec. 2.3), CNNs on spherical data (Sec. 2.4) and knowledge distillation (Sec. 2.5). These works relate to my research on visual recognition in 360° imagery. Finally, I review works related to automatic video editing (Sec. 2.6), which is related to the proposed approach for solving the 360° video display problem. I will also highlight the difference between the related works and my research in each section.

2.1 360° Image Projection

360° image projection has long been studied in the field of cartography. As famously proven by Gauss, no single projection can project a sphere to a plane without introducing some kind of distortion. Therefore, many different projections are proposed, each designed to preserve certain properties such as

distance, area, direction, etc. [102]. For example, the popular equirectangular projection preserves the distance along longitude circles. Various projection models have also been developed to improve perceived quality for 360° images. Prior work [122] studies how to select or combine the projections for a better display, and others develop new projection methods to minimize visual artifacts [14, 65].

360° image projection has also been the focus for 360° video standard format development. Cubemap is adopted as one of the two presentations for 360° video in the MPEG Omnidirectional MediA Format (OMAF) [86], and major 360° video sharing sites such as YouTube and Facebook have turned to this new format [12, 70]. Cubemaps can improve the compression rate by 25% compared to equirectangular projection, which suffers from redundant pixels and distorted motions [71]. The Rotated Sphere Projection is an alternative to cubemap projection with fewer discontinuous boundaries [3].

Although it is possible to tackle the challenges I address in my thesis by designing new projections, we decide to build the methods on top of existing projections that are popular and comply with the future standard. This ensures that the proposed methods will be compliant with real 360° data. Also, most of the proposed methods are either independent of the projection or generic to different projections.

2.2 Video Compression

There exists a long history for research in video compression. In this section, I focus on those that are directly related to my research. I first review recent efforts on 360° video standard development. Because the method we propose for 360° video compression is a learning-based method and relies on a CNN model, I next review recent works on applying deep learning for image and video compression.

2.2.1 360° Video Compression

360° video has sparked initial interest in new video compression techniques. A Call for Evidence last year for a meeting on video standards [117] calls attention to the need for compression techniques specific to 360° video, and responses indicate that substantial improvement can be achieved in test cases [5, 22, 33, 43]. The result leads to the development of a new standard format for 360° video, and the first international standard was published in 2019. For video streaming, some work studies the value in devoting more bits to the region of 360° content currently viewed by the user [103, 111]. However, they require the *current* viewing direction of the user and reduce the video quality beyond the user's field of view. In contrast, our method does not know where the user will look and encodes the entire video with the same quality.

2.2.2 Deep Learning for Image/Video Compression

Recent work investigates ways to improve image compression using deep neural networks. The most common approach is to improve predictive coding using either a feed-forward network [4, 67, 83, 93, 99] or recurrent neural network (RNN) [60,112,113]. The concept can also be extended to video compression [16, 81, 94, 99, 116, 118]. Other approaches include allocating bitrate dynamically using a learned model [75], replacing frequency transformation with learned CNN transformation [77], etc. Based on the success of deep learning based compression, recent works also study how to improve entropy coding [61, 84] in the learning based compression framework. While we also study video compression using a CNN (cf. Chapter 3), we are the first to study 360° video compression, and—CNN or otherwise—the first to exploit spherical video orientation to improve compression rates. Our idea is orthogonal to existing video compression algorithms, which could be combined with our approach without any modification to further improve performance.

Whereas these efforts aim for the next generation in video compression standards, our method for improving 360° compression is compatible with existing video formats and can be applied directly without any modification of existing video codecs. Furthermore, our idea is orthogonal to these ongoing progress in video compression, which could be combined with our approach to further improve performance.

2.3 Convolutional Neural Networks with Geometric Transformations

Learning convolution on 360° imagery essentially needs to handle the geometric distortion introduced by projection. Therefore, this section reviews works studying CNN structures that incorporate geometric transformations. There is an increasing interest in generalizing convolution in CNNs to handle geometric transformations or deformations. Spatial transformer networks (STNs) [54] represent a geometric transformation as a sampling layer and predict the transformation parameters based on input data. Active convolution [57] learns the kernel shape together with the weights for a more general receptive field, and deformable convolution [25] goes one step further by predicting the receptive field location.

These methods, however, are not suitable for convolution on 360° imagery because their assumption on the geometric transformation does not align well with properties of 360° images. STNs assume the transformation is invertible such that the subsequent convolution can be performed on data without the transformation. This is not possible in 360° images because it requires a projection that introduces no distortion. Active and deformable convolution are too restrictive for convolution on 360° images because they assume a fixed kernel size and weight. In contrast, the proposed spherical convolution (cf. Chapter 4) adapts the kernel size and weight based on the transformation to achieve better accuracy. Furthermore, spherical convolution exploits problem-specific geometric information for efficient training and testing.

2.4 Convolutional Neural Networks on Spherical Data

In the last two years, several methods develop new spherical CNN models. Some design CNN architectures that account for the distortion in 360° images [24,126]. Following our spherical convolution [104], both SphereNet [24] and Spherical U-Net [126] focus on enabling weight sharing in order to reduce the model size. SphereNet [24] defines the kernels on the tangent plane and projects features to the tangent planes before applying the kernels. Similarly, Spherical U-Net [126] defines the kernels on the sphere and resamples the kernels on the grid points for every location in the equirectangular projection. However, both of them 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 kernel transformer network [107] (cf. Chapter 5) learns a transformation that considers both spatial and cross-channel correlation and achieves a higher recognition accuracy.

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 [62] such that the outputs are rotation invariant. Another approach transforms both the feature maps and kernels into the spectral domain and applies convolution there [23, 29]. 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, some works try to improve model accuracy by training vanilla CNNs on the cubemap projection, which introduces less distortion [11,17], but the model still suffers from cubemap distortion and discontinuities and has sub-optimal accuracy for tasks such as object detection. Moreover, unlike any of the above prior work [11,17,23,24,29,62,126], the proposed kernel transformer network 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.

2.5 Knowledge Distillation

The proposed approach for transferring CNN models to 360° imagery is largely motivated by knowledge distillation. Knowledge distillation aims to learn a new model given existing model(s) [9,13,49,95]. Rather than optimize an objective function on annotated data, it learns the new model that can reproduce the behavior of the existing model, by minimizing the difference between their outputs. Most prior work explores distillation for model compression [9, 13, 49, 95]. For example, a deep network can be distilled into a shallower [9] or thinner [95] one, or an ensemble can be compressed to a single model [49]. In contrast, our goal is to learn *across* domains, namely to link networks on images with different projection models, rather than to compress

the model.

Limited work considers distillation for transfer [40, 87]. In particular, unlabeled target-source paired data can help learn a CNN for a domain lacking labeled instances (e.g., RGB vs. depth images) [40], and multi-task policies can be learned to simulate action value distributions of expert policies [87]. Learning spherical convolution can also be seen as a form of transfer, though for a novel task motivated strongly by image processing complexity as well as supervision costs. Different from any of the above, we show how to adapt the network structure to account for geometric transformations caused by different projections. Also, whereas most prior work uses only the final output for supervision, we use the intermediate representation of the target network as both input and target output to enable kernel-wise pre-training.

2.6 Automatic Video Editing and Generation

We propose to solve the challenge of 360° video display by learning automatic cinematography, which performs automatic video editing on 360° video. In this section, I first review three common automatic video editing problems. Both video summarization and retargeting share goals similar to the proposed Pano2Vid problem by an reducing input video to its essential portion; virtual cinematography controls a virtual camera to generate new output videos like our approach. Next, I review works in visual saliency, which helps automatic video editing by predicting the important region in the video for human viewers. Finally, I briefly review follow up works of our proposed

approach for 360° video display.

2.6.1 Video Summarization

Video summarization aims to generate a concise representation for a video by removing temporal redundancy while preserving the important events [38,41,42,64,74,89,91,109,120]. The goal is different from ours (cf. Chapter 6) in the sense that video summarization selects content temporally whereas Pano2Vid selects content spatially. Also, the outputs of our algorithm are continuous videos that look as if they were captured by a hand-held camera in the scene, whereas the output of a video summarization algorithm is usually keyframes or concatenated disjoint video clips.

Some efforts also address *multi-video* summarization [6, 26, 32], where the objective is to select, at each time instant, video feed from one camera among many to include in a summary video. The input cameras are human-directed, whether stationary or dynamic [6]. In contrast, we deal with a single hand-held 360° camera, which is not intentionally directed to point anywhere.

2.6.2 Video Retargeting

Video retargeting adapts a source video by cropping and scaling to better fit the target display while minimizing the information loss [7,55,63,68, 78,96]. Both retargeting and our algorithm select portions of the original video to display to the user, but retargeting takes an already well-edited video as input and tries to generate a new version that conveys the same information.

In contrast, our input 360° video is not pre-edited, and the goal is to generate multiple outputs that convey different information. Also, Pano2Vid entails a more severe reduction in spatial extent, e.g., compared to retargeting a 2D video from the Web to display nicely on a mobile device.

2.6.3 Virtual Cinematography

Most existing work on virtual cinematography studies virtual camera control in virtual (computer graphics) environments [21, 28, 47, 85] or else a specialized domain such as lecture videos [31, 98, 110]. Aside from camera control, some prior works also study automatic editing of raw materials like videos or photos [35, 36, 48]. The goal is to generate an effective video presentation automatically to reduce human labor filming or editing. Existing approaches typically rely on heuristics that encode popular cinematographic rules. Pano2Vid differs from the above in that it takes unrestricted real 360° videos as input. Furthermore, the approach *learns* the cinematography tendencies directly from Web videos.

2.6.4 Video Saliency

Saliency studies visual content that attracts viewers' attention [44, 52, 53,79,97,123], where attention is usually measured by gaze fixations under free viewing settings. Although the research originated in static images, there is increasing work that studies video saliency [53,97,123]. Both video saliency and Pano2Vid try to predict spatial locations in videos. However, saliency targets

locations that are eye-catching in 2D image coordinates whereas Pano2Vid (cf. Chapter 6) predicts directions in spherical coordinates that videographers would try to capture with cameras. Also, saliency usually depends on local image content whereas Pano2Vid depends on the content and composition of the entire FOV.

2.6.5 Virtual Cinematography in 360° Video

After we proposed to solve the 360° video display problem using automatic videography, several other works proposed alternative solutions for the problem [20, 51, 72]. These methods rely on object information to guide the virtual camera control, using either object proposals [51], semantic segmentations [72], or object detection results [20]. They also take user input such as preferred object(s) or narrative to improve view selection. The design implicitly assumes that users prefer to focus on objects in 360° videos. While this is true in some examples, we decided not to make this assumption in order to handle more generic scenarios. Also, some of the methods perform object detection/segmentation on equirectangular projection using an off-the-shelf model [20, 72], which cannot handle regions with severe distortion in the equirectangular projection. In contrast, our method always processes visual content in its perspective projection and does not suffer from distortion.

Chapter 3

Learning Compressible 360° Video Isomers

A compressed video format is the core of all video technologies and is the basis for all video related applications, ranging from video capture, storage, processing to distribution. Without adequate compression, all of the above suffer. 360° video is no exception. In this chapter, I study how to improve 360° video compression by exploring a new dimension in video compression—the orientation of the 360° video projection[1].

Thus far, the focus for 360° video compression is to find a proper projection that transforms a 360° frame into a rectangular planar image that will have a high compression rate. A current favorite is to project the sphere to a *cubemap* and unwrap the cube into a planar image [12, 70, 86] (see Fig. 3.2). Cubemaps can improve the compression rate by up to 25% compared to the previously popular equirectangular projection [71].

One unique property of 360° video is that each spherical video has an *infinite number of equivalents related by a rotation.* Therefore, each 360° video could be transformed into multiple possible cubemaps by changing the orien-

[1]The work in this chapter was originally published in: "Learning Compressible 360° Video Isomers," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition 2018 [106]. The authors are Yu-Chuan Su and Kristen Grauman.

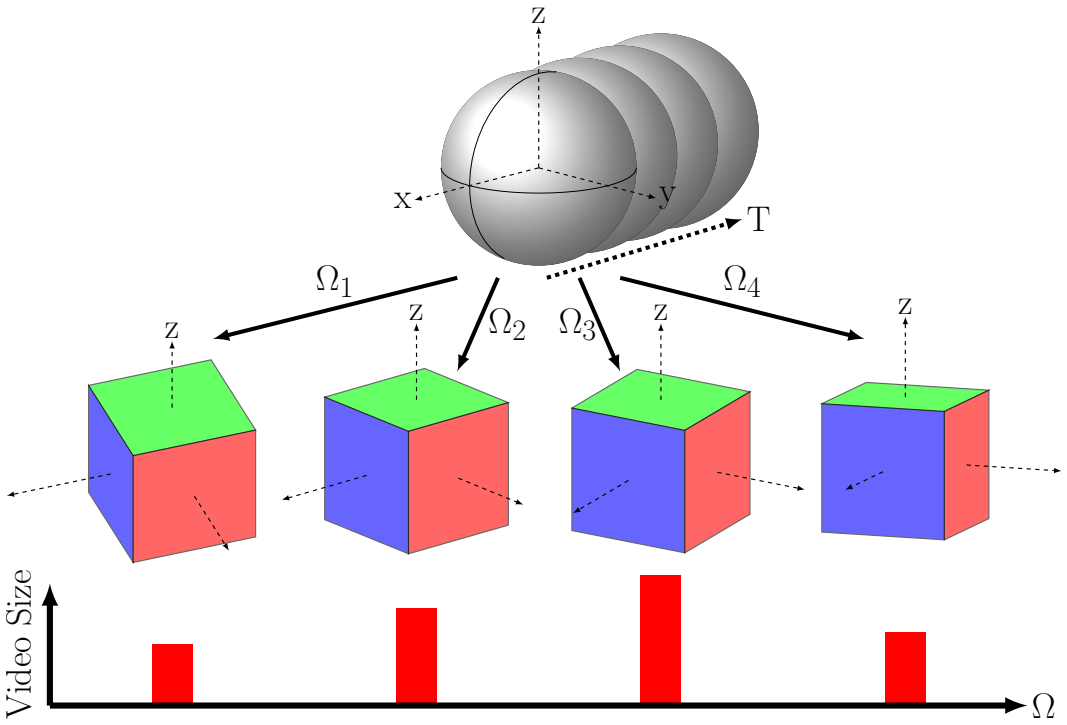


Figure 3.1: Our approach learns to automatically rotate the 360° video axis before storing the video in cubemap format. While the 360° videos are equivalent under rotation ("isomers"), the bitstreams are not because of the video compression procedures. Our approach analyzes the video's visual content to predict its most compressible isomer.

tation of the cube, yet all of them represent the very same video content. I will refer to these content-equivalent rotations as 360° *isomers*.[2] The isomers, however, are *not* equivalents in terms of compression. Different isomers interact differently with a given compression algorithm and so yield different compression rates (See Fig. 3.1). This is because the unwrapped cubemap is

[2]Strictly speaking isomers are equivalent only theoretically, because pixels are discretely sampled and rotating a cubemap requires interpolating the pixels. Nevertheless, as long as the pixel density, i.e. video resolution, is high enough, the information delta is negligible.

not a homogenous perspective image. Therefore, some of the properties that current compression algorithms exploit in perspective images do not hold. For example, while the content is smooth and continuous in perspective images, this need not be true along an inter-face boundary in an unwrapped cubemap. The discontinuity can introduce artificial high frequency signals and large abrupt motions, both of which harm the compression rate (cf. Sec. 3.1.2 and Fig. 3.5). In short, my key insight is that the compression rate of a 360° video will depend on the orientation of the cubemap it is projected on.

I propose a learning-based approach to predict—from the video's visual content itself—the cubemap orientation that will minimize the video size [106]. First, I demonstrate empirically that the orientation of a cubemap does influence the compression rate, and the difference is not an artifact of a specific encoder but a general property over a variety of popular video formats. Based on that observation, I propose to automatically re-orient the cubemap for every group of pictures (GOP).[3] A naive solution would enumerate each possible orientation, compress the GOP, and pick the one with the lowest encoded bitstream size. However, doing so would incur substantial overhead during compression, prohibitively costly for many settings. Instead, I propose to render the GOP for a *single* orientation after predicting the optimal orientation from the video clip rendered in its canonical orientation. Given encoded videos in a fixed orientation, we train a CNN that takes both the segmentation contours and motion vectors in the encoded bitstream and predicts the orientation

[3]A collection of successive pictures within a coded video stream.

that will yield the minimum video size. By avoiding rendering and encoding the video clip in all possible orientations, our approach greatly reduces the computational cost and strikes a balance between speed and compression rate.

The key benefit of our approach is a higher compression rate for 360° video that requires only to re-render the cubemap. In particular, the approach does not require changing the video format nor the compression algorithm, which makes it fully compatible with any existing video codec. This is especially important in the realm of video compression, because a new video format often takes years to standardize and deploy, and so changing the bitstream format would incur very high overhead. The only additional information that our method needs to encode is the selected orientation of each GOP, which can easily be encoded as meta data (and is part of the standard format [18]).

3.1 360° Video Isomers Analysis

Our goal is to develop a computationally efficient method that exploits a cubemap's orientation for better compression rates. In this section, I perform a detailed analysis on the correlation between the encoded video size and cubemap orientation. The intent is to verify that orientation is indeed important for 360° video compression.

First I briefly review fundamental video compression concepts, which will help in understanding where our idea has leverage. Modern video compression standards divide a video into a series of GOPs, which can be decoded independently to allow fast seeking and error recovery. Each GOP starts with

an *I-frame*, or intra-coded picture, which is encoded independently of other frames like a static image. Other frames are encoded as inter-coded pictures, and are divided into rectangular blocks. The encoder finds a reference block in previous frames for each block that minimizes their difference. Instead of encoding the pixels directly, the encoder encodes the relative location of the reference block, i.e., the *motion vector*, and the residual between the current and reference block. This inter-frame prediction allows encoders to exploit temporal redundancy in the video. Note that the encoder has the freedom to fall back to intra-coding mode for blocks in an inter-coded frame if no reference block is found.

Just like static image compression, the encoder performs transform coding by transforming the pixels in I-frames and residuals in inter-coded frames into the frequency domain and encoding the coefficients. The transformation improves the compression rate because high frequency signals are usually few in natural images, and many coefficients will be zero. To further reduce the video size, video compression formats also exploit spatial redundancy through intra-prediction, which predicts values to be encoded using adjacent values that are previously encoded. The encoder will encode only the residual between the prediction and real value. This applies to both the motion vectors' and transformed coefficients' encoding. Most of the residuals will be small and can be encoded efficiently using entropy coding. For a more complete survey, see [90].

3.1.1 Cubemap Preparation

To study the correlation between cubemap orientation and compression rate, we collect a 360° video dataset from YouTube. Existing datasets [51,108] contain videos with arbitrary quality, many with compression artifacts that could bias the result. Instead, we collect only high quality videos using the 4K filter in YouTube search. We use the keyword "360 video" together with the 360° filter to search for videos and manually filter out those consisting of static images or CG videos. The dataset covers a variety of video content and recording situations, including but not limited to aerial, underwater, sports, animal, news, and event videos, and the camera can be either static or moving. We download the videos in equirectangular projection with 3,840 pixels width encoded in H264 high profile. The dataset contains 80 videos with 4.2 hours total length.

We next transcode the video into cubemap format and extract the video size for each orientation. Because it is impossible to enumerate all possible cubemap orientations over time, we subsample both the GOP length and cubemap orientations in our analysis.

Our analysis is divided into two parts. In the first part, we examine whether the orientation of cubemap representation affects compression rate using lossless video compression. Here we focus on lossless video compression to reduce the number of factors in the analysis and therefore the computational cost. Similarly, we use a fixed two second GOP length, which results in 7,436 video clips without overlap. Although common video codecs have

the flexibility to adjust the GOP length within a given range, e.g., between 25-250 frames (1-10 seconds) in the default x264 encoder or 1-2 seconds in Blu-ray videos [10], they usually use the longest GOP length unless a scene cut is detected. Therefore, a fixed GOP length should not affect the analysis significantly.

In the second part of the analysis, we examine whether the result of the first part generalizes to more general video compression settings. In particular, we relax the constraints of 1) lossless video compression and 2) fixed two-second GOPs. We focus on these two factors because they are the most common factors that may change in applications: the compression rate is controlled by the user to meet the requirement of a specific application, and the GOP length is controlled by the encoder to improve the compression rate. We sample three different GOP lengths—one, two, four seconds—every second in the video, so the video clips may overlap. Because lossy video compression analysis considers more factors and incurs higher computational cost, we use only the first 60 seconds from each video in the analysis, which leads to 13,920 video clips.

For each clip, we sample the cubemap orientation

$$\Omega = (\phi, \theta) \in \Phi \times \Theta \tag{3.1}$$

with different ϕ and θ in $\Theta = \Phi = \{-45°, -40°, \cdots, 45°\}$, i.e., every $5°$ between $[-45°,\ 45°]$. This yields $|\Phi \times \Theta| = 361$ different orientations. We restrict the orientation within $90°$ because of the rotational symmetry along each axis.

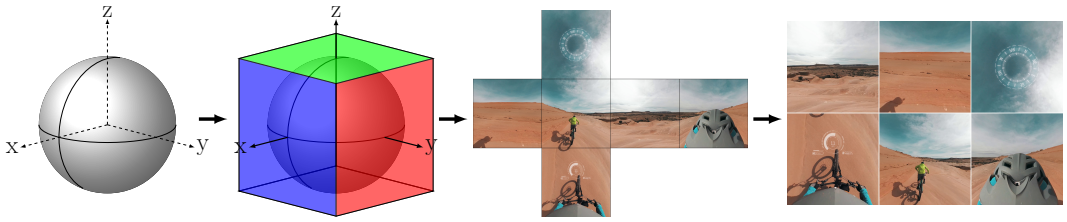


Figure 3.2: Cubemap format transformation. The 360° video is first projected to a cube enclosing the unit sphere and then unwrapped into 6 faces. The 6 faces are re-arranged to form a rectangular picture to fit video compression standards (2×3 frame on the right).

For each orientation, we transform the video into cubemap format with 960 pixel resolution for each face. Fig. 3.2 illustrates the transformation. The video is then encoded using off-the-shelf encoders before extracting the compressed bitstream size.

For lossless video compression, we render the cubemaps using the transform360 filter[4] in FFMPEG released by Facebook. We then encode the video into three popular formats—H264 using x264[5], HEVC using x265[6], and VP9 using libvpx[7]. Among them, H264 is currently the most common video format. HEVC, also known as H265, is the successor of H264 and is the latest video compression standard. VP9 is a competitor of HEVC developed by Google and is most popular in web applications.

For lossy video compression, we render the video using 360Lib [1] and compress the video into H264 format using x264 with five different video qual-

[4] https://github.com/facebook/transform360
[5] https://www.videolan.org/developers/x264.html
[6] http://x265.org
[7] https://chromium.googlesource.com/webm/libvpx/

ities. The video qualities are controlled such that the resulting bitrates are roughly equally spaced in log scale, i.e., bitrate $\in \{B, B/4, B/16, B/64, B/256\}$, where B is the bitrate for lossless compression. This is controlled by the constant rate factor (CRF) parameter in the x264 encoder. We restrict the initial analysis to H264 format in lossy video compression due to the computational cost, but our results below extend to multiple encoders and both lossless and lossy compression.

Note that we use popular open source tools for both cubemap rendering and video compression to ensure that they are well optimized and tested. This way any size changes we observe can be taken as common in 360° video production instead of an artifact of our implementation.

3.1.2 Cubemap Analysis

Next we investigate how much and why the orientation of an isomer matters for compressibility. If not mentioned specifically, all the results are obtained from H264 fromat.

Video size distribution w.r.t. Ω We first show the video size distribution with respect to Ω. We compute the *normalized clip size*

$$\tilde{S}_\Omega = 100 \times \frac{S_\Omega - S_{\Omega^{min}}}{S_{\Omega^{max}} - S_{\Omega^{min}}} \tag{3.2}$$

for every Ω and cluster the size distribution of each clip using K-Means. S_Ω is the encoded bitstream size with orientation Ω and $\Omega^{max}/\Omega^{min}$ corresponds

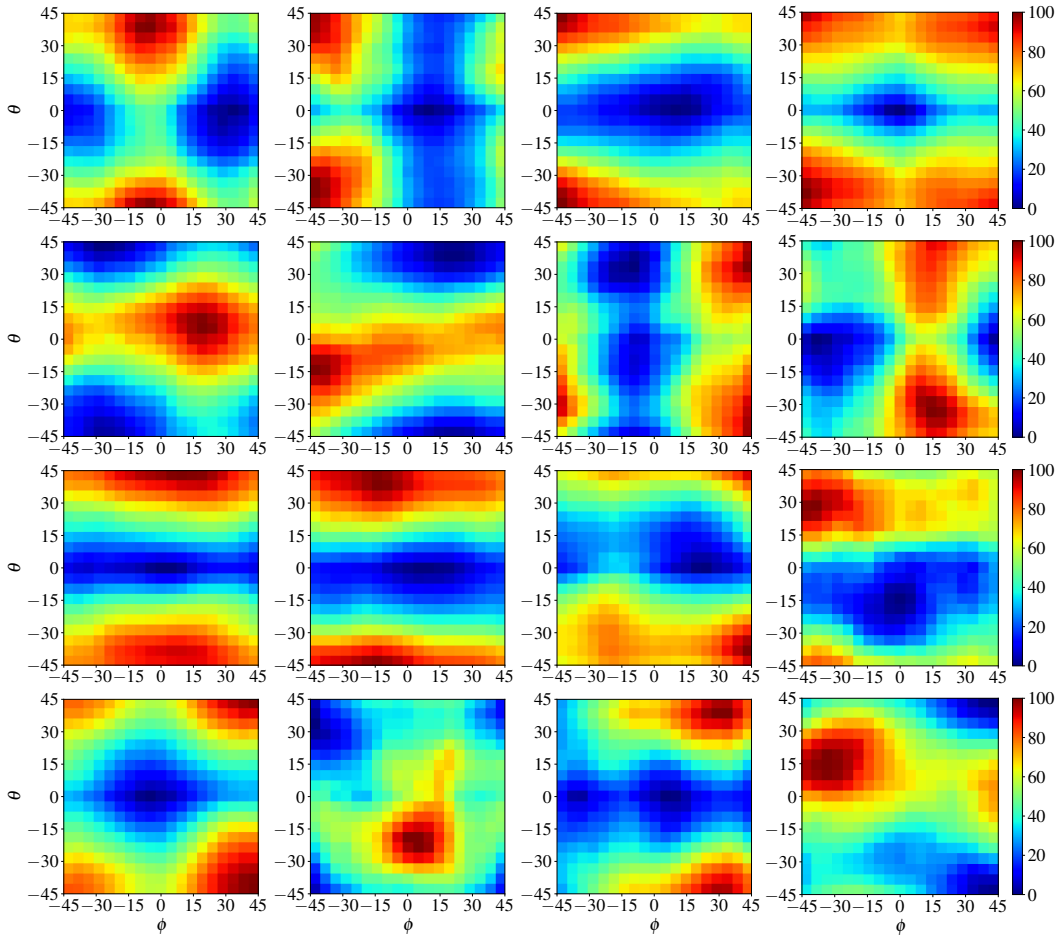


Figure 3.3: Relative clip size distribution w.r.t. Ω. We cluster the distribution into 16 clusters and show the cluster centers.

to the orientation with maximum/minimum bitstream size. Each cluster is represented by the nearest neighbor to the center.

Fig. 3.3 shows the results. We can see Ω^{min} lies on or near θ=0° in about half of the clusters. In general, this corresponds to orienting the cubemap perpendicular to the ground such that the top face captures the sky and the

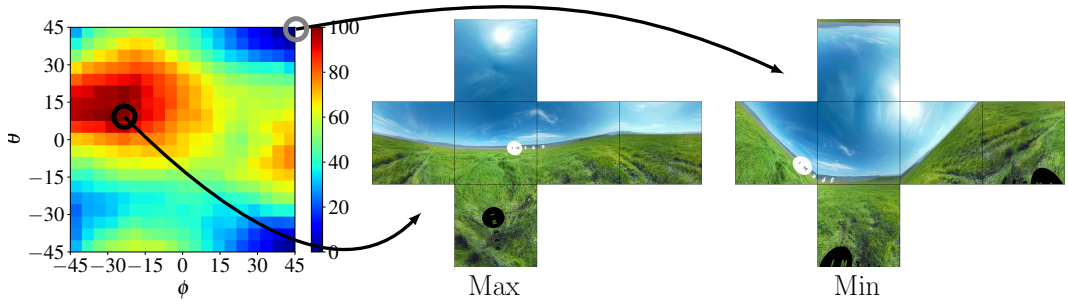
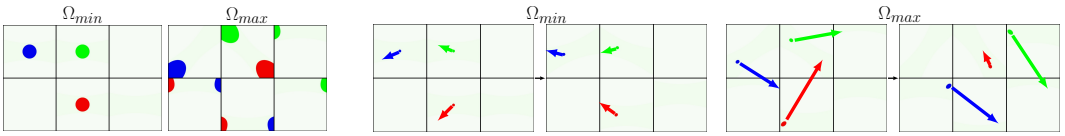


Figure 3.4: Size distribution for one particular video. We show the cubemaps corresponding to $\Omega_{max}/\Omega_{min}$, which are its worst (max) and best (min) orientations for compressibility.

bottom face captures the camera and ground. See Fig. 3.2 for example. The top and bottom faces tend to have smaller motion within the faces in these orientations, and the compression rate is higher because the problem reduces from compressing six dynamic pictures to four dynamic pictures plus two near static pictures. However, θ=0° is not best for every clip, and there are multiple modes visible in Fig. 3.3. For example, the minimum size occurs at θ=ϕ=45° in Fig. 3.4. The non-trivial distribution of video size shows that the orientation of isomers indeed matters for video compression.

Reasons for the compression rate difference Why does the video size depend on Ω? The fundamental reason is that all the video compression formats are designed for perspective images and heavily exploit the image properties. The unwrapped cubemap format is a perspective image only locally within each of the six faces. The cubemap projection introduces perspective distortion near the face boundaries and artificial discontinuities across face



(a) Content discontinuity. (b) Motion discontinuity.

Figure 3.5: Explanations for why different Ω have different compression rate, shown for good (Ω_{min}) and bad (Ω_{max}) rotations. (a) From a static picture perspective, some Ω introduce content discontinuity and reduce spatial redundancy. (b) From a dynamic picture perspective, some Ω make the motion more disordered and break the temporal redundancy.

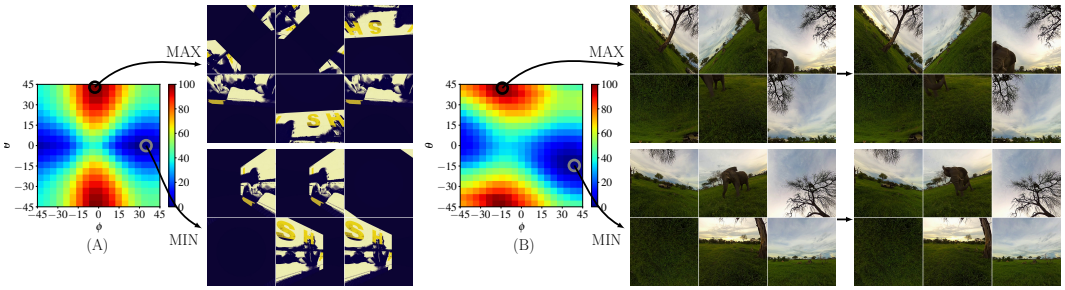


Figure 3.6: Real examples for the explanations in Fig. 3.5. Example (A) shows content discontinuity introduced by rotation. Example (B) shows motion discontinuity. The encoder fails to find reference blocks in this example, and the number of intra-coded blocks increases.

boundaries, both of which make the cubemap significantly different from perspective images and can degrade the compression rate. Because the degradation is content dependent, different orientations result in different compression rates.

More specifically, the reasons for the compression rate difference can be divided into two parts: static content and dynamic motion. From the static image perspective, artificial edges may be introduced if continuous patterns fall on the face boundary. See Fig. 3.5 (a) and Fig. 3.6 for examples. The

edges introduce additional high frequency signals and reduce the efficiency of transform coding. Furthermore, the single continuous patch is divided into multiple patches that are dispersed to multiple locations in the image. This reduces the spatial redundancy and breaks the intra-prediction.

From the dynamic video perspective, the face boundaries may introduce abrupt jumps in the motion. If an object moves across the boundary, it may be teleported to a distant location on the image. See Fig. 3.5 (b) and Fig. 3.6 for examples. The abrupt motion makes it difficult to find the reference block during encoding, and the encoder may fall back to intra-coding mode which is much less efficient. Even if the encoder successfully finds the reference block, the motion vectors would have very different magnitudes and directions compared to those within the faces, which breaks intra-prediction. Finally, because the perspective distortion is location dependent, the same pattern will be distorted differently when it falls on different faces, and the residual of inter-frame prediction may increase. The analysis applies similarly across all formats, which makes sense, since their compression strategies are broadly similar.

Achievable video size reduction We next examine the size reduction we can achieve by optimally changing the cubemap orientation. In particular, we compute the *reduction*

$$r = 100 \times \frac{S_{\Omega^{max}} - S_{\Omega^{min}}}{S_{\Omega^{max}}}. \tag{3.3}$$

		H264	HEVC	VP9
Video r (%)	Avg.	8.43 ± 2.43	8.11 ± 2.03	7.83 ± 2.34
	Range	[4.34, 15.18]	[4.58, 13.67]	[3.80, 14.72]
Clip r (%)	Avg.	10.37 ± 8.79	8.88 ± 8.23	9.78 ± 8.62
	Range	[1.08, 76.93]	[1.40, 74.95]	[1.70, 75.84]

Table 3.1: Achievable video size reduction through rotation for lossless compression in three different formats. Clip-level reductions are per GOP (fixed in length); video-level reductions are aggregated over all clips in a video (and hence vary in length). We can reduce the video size by up to 77% by optimally changing the cubemap orientation.

We start with lossless video compression. Table 3.1 shows the results. The average video size reduction $\overline{r}$ is 8.43% for H264, which means that we can reduce the overall 360° video size by more than 8% by rotating the video axis. This corresponds to a 2GB reduction in our 80 video database and would scale to 25.3TB for a 1M video database. The range of r for each clip using H264 is $[1.08, 76.93]$, which indicates that the compression rate is strongly content dependent, and the size reduction can be up to 77% for a single video if we allow the encoder to re-orient the 360° video. If we restrict the rotation to ϕ and fix $\theta = 0°$, $\overline{r}$ drops to 2.35%. This result suggests that it is important to allow rotation along both axes. Table. 3.1 also shows the outcomes are similar across the three formats, which makes sense, since their compression strategies are broadly similar.

We next extend the analysis to lossy video compression. Fig. 3.7 shows the results for H264. Clearly, the correlation between cubemap orientation and bitstream size is not specific to lossless video compression or a single

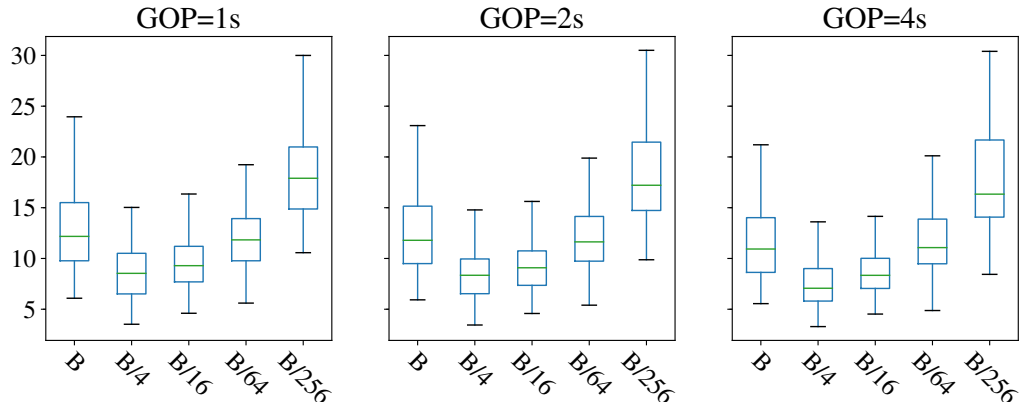


Figure 3.7: Achievable video size reduction through rotation for lossy compression at different compression rates and GOP lengths for H264. B refers to the bitrate for lossless video compression.

video compression setting. Instead, it is a general property of the cubemap representation and video codecs over different compression rates and GOP lengths. We can also see that the size reduction is not sensitive to the GOP length. Although the effect of orientation varies across different compression rates, the average size reduction is between 8-18%, which is consistent with the results in lossless compression. Note that the size reduction for lossy video compression in Fig. 3.7 (bitrate=B) is larger than that in Table 3.1. The reason is that Fig. 3.7 is computed over the first 60 seconds of the video. Because some of the videos contain title slides at the beginning—which are more sensitive to cubemap orientation—the size reduction tends to be more significant in the first 60 seconds of the videos.

Video size correlation across formats Finally, we verify the correlation between video size and orientation is a generic property of the cubemap rep-

Encoders	H264 / H265	H264 / VP9	H265 / VP9
Avg. ρ	0.8757	0.9533	0.8423

Table 3.2: The correlation of relative video sizes across video formats. The high correlation indicates that the dependency between video size and Ω is common across formats.

resentation and the common structure of video compression algorithm, as opposed to an artifact of a specific video codec. We compare the size reduction that can be achieved through rotation under different video compression settings and their correlations. Recall that Table 3.1 shows the bitstream sizes using different video encoders in lossless video compression. That result clearly shows that the dependency between the compression rate and Ω is a common property across current video compression formats. This is further verified in Table 3.2, where we observe a high correlation between the relative video size, i.e.,

$$S'_\Omega = S_\Omega - S_{0,0}, \tag{3.4}$$

of different encoders.

Next we consider the correlations for lossy video compression for different compression rates (Table 3.3). The results show that the bitstream sizes across different compression settings are highly correlated, especially when the video qualities are not significantly different (i.e., from lossless compression to the lowest possible quality). Note that the bitrates in the analysis cover almost the entire range of available bitrates in the x264 encoder, and the common bitrate in practical applications is between $[B/7, B/25]$ according to

Bitrate	B	B/4	B/16	B/64	B/256
B	1.00	0.93	0.83	0.69	0.45
B/4	0.93	1.00	0.90	0.73	0.44
B/16	0.83	0.90	1.00	0.87	0.55
B/64	0.69	0.73	0.87	1.00	0.75
B/256	0.45	0.44	0.55	0.75	1.00
Avg.	0.78	0.80	0.83	0.81	0.64

Table 3.3: The correlation of relative video sizes across compression rate.

the FFMPEG documents.

The high correlation between the bitstream sizes across different video compression settings not only verifies that the correlation between bitstream sizes and orientation is a general property of the cubemap representation, it also suggests that we may be able to predict the bitstream size of other compression settings having trained with one particular setting. This implies that if we can learn a model that predicts the bitstream size of one compression setting (e.g., GOP length 2 and bitrate B/16), the same model may predict the bitstream size of other compression settings as well. We empirically verify the transferability of the prediction model in the experiments (Sec. 3.3). This is very important for video compression: there is a large number of potential settings users may choose, and it would be impractical to optimize the cubemap orientation for each possible compression setting independently.

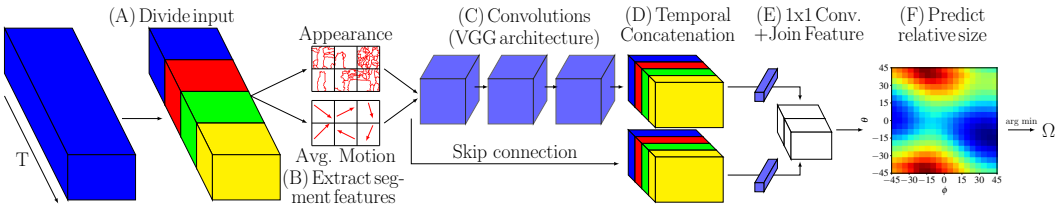


Figure 3.8: Our model takes a video clip as input and predicts Ω^{min} as output. (A) It first divides the video into 4 segments temporally and (B) extracts appearance and motion features from each segment. (C) It then concatenates the appearance and motion feature maps and feeds them into a CNN. (D) The model concatenates the outputs of each segment together and joins the output with the input feature map using skip connections to form the video feature. (F) It then learns a regression model that predicts the relative video size S'_Ω for all Ω and takes the minimum one as the predicted optimally compressible isomer.

3.2 Approach for Exploiting 360° Video Isomers

In this section, I introduce our approach for improving 360° video compression rates by predicting the most compressible isomer. Given a 360° video clip, the goal is to identify Ω^{min} to minimize the video size. A naive solution is to render and compress the video for all possible angles Ω and compare their sizes. While this guarantees the optimal solution, it introduces a significant computational overhead, i.e., 360 times more computation than encoding the video with a fixed Ω. For example, it takes more than 15 seconds to encode one single clip using the default x264 encoder on a 48 core machine with Intel Xeon E5-2697 processors, which corresponds to $15s \times 360 \approx 1.5$ hours for one clip if we were to enumerate all Ω. Moreover, the computational cost will grow quadratically if we allow more fine-grained control. Therefore, enumerating Ω is not practical.

Instead, we propose to predict Ω^{min} from the raw input without re-rendering the video. Given the input video in cubemap format, we extract both motion and appearance features (details below) and feed them into a CNN that predicts the video size S_Ω for each Ω, and the final prediction of the model is

$$\Omega^{min} = \arg\min_{\Omega} S_\Omega. \tag{3.5}$$

See Fig. 3.8. The computational cost remains roughly the same as transcoding the video because the prediction takes less than a second, which is orders of magnitude shorter than encoding the video and thus negligible. Since no predictor will generalize perfectly, there is a chance of decreasing the compression rate in some cases. However, experimental results show that it yields very good results and strikes a balance between computation time and video size. Furthermore, due to our two-pass design (defined below, cf. Fig. 3.9), if the input orientation of the video proves to encode to a smaller sized file than the one encoded using our method's predicted angle, we can simply return that one, i.e., "do no harm".

Because the goal is to find Ω^{min} for a given video clip, exact prediction of S_Ω is not necessary. Instead, the model predicts the relative video size S'_Ω from Eq. 3.4. The value S'_Ω is scaled to $[0, 100]$ over the entire dataset to facilitate training. We treat it as a regression problem and learn a model that predicts 361 real values using L2 loss as the objective function. Note that we do not predict S_Ω in Eq. 3.2 because it would amplify the loss for clips with smaller size, which may be harmful for the absolute size reduction.

The model first divides the input video into 4 equal length segments. For each segment, it extracts the appearance and motion features for each frame and average them over the segment. For appearance features, we segment the frame into regions using SLIC [2] and take the segmentation contour map as a feature. The segmentation contour represents edges in the frame, which imply object boundaries and high frequency signals that take more bits in video compression.

For motion features, we take the motion vectors directly from the input video stream encoding, as opposed to computing optical flow. The motion vectors are readily available in the input and thus this saves computation. Furthermore, motion vectors provide more direct information about the encoder. Specifically, we sample one motion vector every 8 pixels and take both the forward and backward motion vectors as the feature. Because each motion vector consists of both spatial and temporal displacement, this results in a 6-dimensional feature. For regions without a motion vector, we simply pad 0 for the input regardless of the encoding mode. We concatenate the appearance and motion features to construct a feature map with depth 7. Because the motion feature map has lower resolution than the video frame, we downscale the appearance feature map by 8 to match the spatial resolution. The input resolution of each face of the cube map is therefore $960/8 = 160$ pixels.

The feature maps for each segment are then fed into a CNN and concatenated together as the video feature. We use the VGG architecture [101] except that we increase the number of input channels in the first convolu-

tion layer. Because fine details are important in video compression, we use skip connections to combine low level information with high level features, following models for image segmentation [80]. In particular, we combine the input feature map and final convolution output as the segment feature after performing 1x1 convolution to reduce the dimension to 4 and 64 respectively. The video feature is then fed into a fully-connected layer with 361 outputs as the regression model. Note that we remove the fully-connected layers in the VGG architecture to keep the spatial resolution for the regression model and reduce model size.

Aside from predicting S_Ω, in preliminary research we tried other objective functions such as regression for Ω^{min} directly or predicting Ω^{min} from the 361 possible Ω with 361-way classification, but none of them perform as well as the proposed approach. Regressing Ω^{min} often falls back to predicting $(\theta, \phi) = (0, 0)$ because the distribution is symmetric. Treating the problem as 361-way classification has very poor accuracy, i.e., slightly better than random ($\approx$5%), because the amount of training data is small and imbalanced. We also examined different input features. For motion features, we tried 3D convolution instead of explicitly feeding the motion information as input, but 3D convolution performs 4$-$30% worse than 2D convolution despite having a higher computational cost. For appearance features, we tried raw pixels with various network architectures but find that segmentation contours consistently perform better.

Based on the prediction model, we propose a two-pass compression

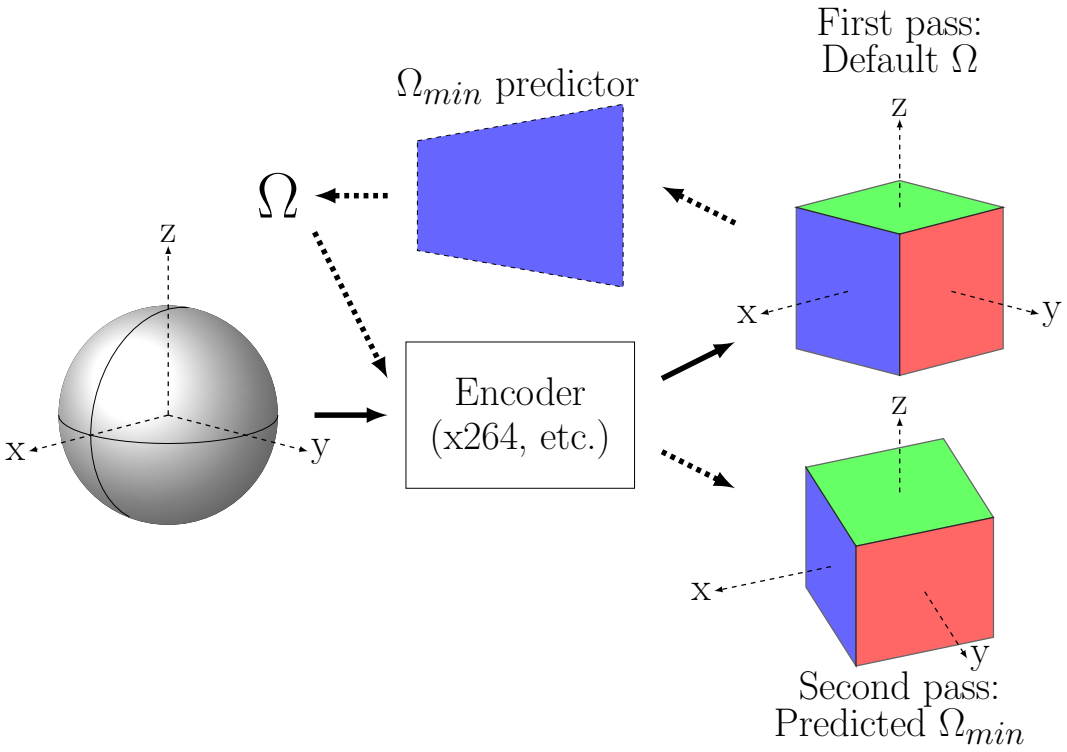


Figure 3.9: We propose a two-pass compression pipeline. The second pass optimizes the compression rate by taking the orientation predicted from the output bitstream of the first pass. The dashed line indicates the additional steps in the new pipeline.

pipeline for 360° videos. Given a 360° video, we first compress it in the default orientation using any standard encoder that will unwrap the video into a cubemap and encode it. Next, we feed the cubemap representation into the prediction model, which predicts Ω^{min} given the video content. As mentioned before, the cubemap representation provides the motion vectors as motion features readily, which reduces the computational cost of the entire pipeline. Finally, we feed Ω^{min} and the 360° video back to the same encoder to perform the second pass of compression, where the output cubemap representation has

Figure 3.10: Example frames from JVET test sequences. The images are from *ChairliftRide*, *Gaslamp*, *Harbor*, *KiteFlite*, *SkateboardInLot*, and *Trolley* sequences, respectively, from top left to bottom right.

orientation Ω^{min} and achieves a better compression rate. See Fig. 3.9.

Note that the two-pass compression pipeline is very common in video compression, especially when one wants to encode the video with a constant bitrate. Thus, our proposed pipeline can be considered as an extension of the original two-pass compression pipeline where we introduce an additional bitstream analysis. Also, because the new pipeline does not make any bitstream level changes on the compression algorithm, it can be combined with existing video compression techniques easily.

3.3 Experiments

To evaluate our method, we compute the size reduction it achieves on high resolution 360° video datasets with real and dynamic 360° videos.

3.3.1 Dataset

We evaluate the prediction model on both the YouTube 360° video dataset introduced in Sec. 3.1 and the common test sequences introduced by the Joint Video Exploration Team (JVET), a key compression benchmark. For the YouTube 360° video dataset, we use all 80 videos with full length as in the previous analysis. For the JVET test sequences, we use the six 8K sequences introduced in the JVET common test conditions. All the sequences contain 300 frames with 30 fps frame rate, and the resolution is 8192×4096 in equirectangular projection with YUV420 pixel format. The name of the sequences are in Table 3.7, which provides a high level description of the video content, and Fig. 3.10 shows example frames from each sequence.

An important benefit of the JVET test set is that it provides raw video input without any compression. The raw format avoids compression artifacts in the source video, which may affect the analysis of compression algorithms. Because most use cases for video compression take the raw or high quality video as input, evaluating the algorithm on the raw format provides an accurate estimate of performance. Furthermore, these sequences are widely used in the development of 360° video compression, so the results may be compared with other ongoing work.

To train and test the model, we divide the YouTube 360° video dataset into 4 folds, each containing 20 videos. Three are used for training, and the other is used for testing. We report the average result over 4 folds as the final performance. For the JVET test sequences, we use the models trained on the

YouTube 360° video dataset directly without modification.

3.3.2 Evaluation Metrics

We use two metrics: normalized size reduction and Bjøntegaard-Delta bitrate (BD-rate). We compare each method using the normalized size reduction $\tilde{r} = 1 - \tilde{S}$ for each video. Specifically, we compute the largest full-video size by choosing Ω^{max} for every clip and sum the clip sizes. Similarly, we compute the minimum video size. Given the predicted orientation for each clip, we compute the video size when rotating the cubemap by the predicted orientation. The result indicates the fraction of reduction the method achieves compared to the optimal result.

Besides the normalized size reduction, we also evaluate the performance using the BD-rate, which is the most common evaluation metric for video compression. Because there is a trade-off between bitrate and video quality in video compression and it is hard to require a fixed quality or bitrate for a compression algorithm, BD-rate evaluates the performance of a compression algorithm over the entire bitrate versus video quality curve instead on a single point. In particular, it computes the average bitrate difference between two codecs in percent over all possible video qualities, namely by fitting a third order polynomial over discrete sample points. We use Peak-Signal-to-Noise-Ratio (PSNR) for the quality metric, which is the most common quality metric in video compression algorithm. Following the ongoing works for 360° video compression, we also evaluate the Weighted-to-Spherically-Uniform PSNR (WS-PSNR). Be-

cause the pixels in equirectangular projection are not uniformly distributed on the sphere, WS-PSNR reweights the quality metric so it is uniform over the entire sphere.

3.3.3 Baselines

To our knowledge, no prior work studies how to predict the cubemap orientation for better compression. Thus, we compare our method with the following two heuristics:

- RANDOM — Randomly rotate the cubemap to one of the 361 orientations. This represents the compression rate when we have no knowledge about the video orientation.
- CENTER — Use the orientation provided by the videographer. This is a strong prior, usually corresponding to the direction of the videographer's gaze or movement and lying on the horizon of the world coordinate.

3.3.4 Implementation Details

We initialize the weights using an ImageNet pre-trained VGG model provided by the authors [101]. For the first layer, we replicate the weights of the original network to increase the number of input channels. Weights that are not in the original model are randomly initialized using Xavier initialization [37]. We train the model using ADAM [66] for 4,000 iterations with batch size 64 parallelized to 16 GPUs. The base learning rate is initialized to 1.0×10^{-3} and is decreased by a factor of 10 after 2,000 iterations. We also

	H264	HEVC	VP9
RANDOM	50.75	51.62	51.20
CENTER	74.35	63.34	72.92
OURS	82.10	79.10	81.55

Table 3.4: Size reduction of each method. The range is $[0, 100]$, the higher the better.

apply L_2 regularization with the weight set to 5.0×10^{-4} and use dropout for the fully-connected layers with ratio 0.5. For SLIC, we segment each face of the cubemap independently into 256 superpixels with compactness m=1. The low compactness value leads to more emphasis on the color proximity in the superpixels.

3.3.5 Results

If not mentioned specifically, the results are obtained from the YouTube 360° video dataset.

Video size reduction We first examine the size reduction the proposed method achieves. Table 3.4 shows the results for lossless video compression. The proposed method performs better than the baselines in all video formats by $7\% - 16\%$. The improvement over the baseline is largest in HEVC, which indicates that the advantage of our approach will become more significant as HEVC gradually replaces H264. Interestingly, the CENTER baseline performs particularly worse in HEVC. The reason is that HEVC allows the encoder to achieve good compression rates in more diversed situations, so the distribution

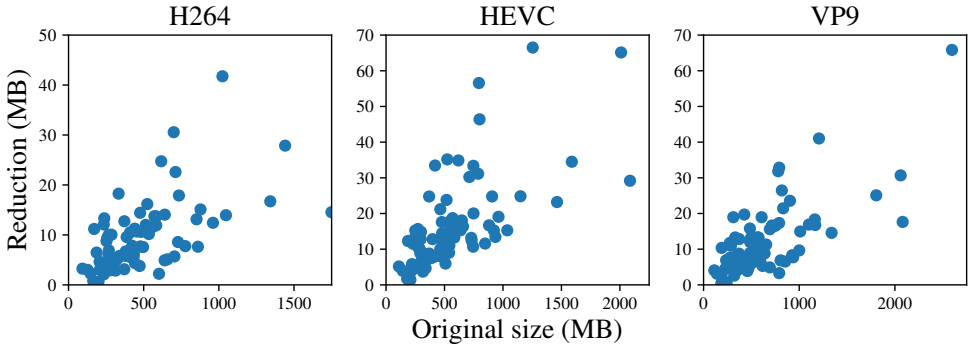


Figure 3.11: Absolute size reduction (MB) of each video. Each point represents the input video size vs. size reduction relative to CENTER achieved by our model.

of Ω^{min} becomes more dispersed. The result further shows the value in considering cubemap orientation during compression as more advanced video codecs are used. While there remains a 20% room for improvement compared to the optimal result (as ascertained by enumerating Ω), our approach is significantly faster and takes less than 0.6% the computation.

Fig. 3.11 shows the absolute file size reduction for each video. Because the video size depends very much on the video content and length and is hard to compare across examples, we show the reduction versus the original video size. The size reduction by our method, though dependent on the video content, is roughly linear in the original video size. Note that the original videos are encoded with orientation $\Omega_{0,0}$.

Table 3.5 shows the corresponding results for lossy video compression. The model (OURS) is trained and evaluated in the same video compression setting, i.e., the same GOP length and bitrate. Again, the prediction model

		Bitrate				
	GOP	B	B/4	B/16	B/64	B/256
Random	1s	43.61	50.30	50.65	50.62	53.90
	2s	43.10	49.98	51.20	50.91	53.80
	4s	42.64	49.56	51.02	50.97	53.96
Center	1s	95.95	63.64	55.73	62.81	67.64
	2s	96.17	64.16	54.32	62.05	66.87
	4s	96.52	63.72	52.60	61.40	66.51
Ours	1s	96.14	73.84	71.74	73.28	74.80
	2s	96.62	73.66	70.11	74.15	76.69
	4s	96.73	75.34	71.01	74.46	75.09
Ours (Single Model)	1s	96.53	74.77	71.43	73.86	75.71
	2s	96.75	74.99	71.25	74.15	75.97
	4s	96.90	74.91	70.82	74.28	76.09

Table 3.5: Video size reduction for lossy video compression (higher is better). Ours shows the reduction when the model is trained and evaluated on the same compression setting, while Ours (Single Model) shows the result using a model trained on a single compression setting. Note that Ours requires training $3\times5=15$ models, whereas Ours (Single Model) requires training only 1.

performs $10-20\%$ better than the Center baseline and $20-25\%$ better than Random. On the other hand, the absolute performance in lossy video compression is worse than that in lossless video compression. Our hypothesis is that the video encoders have a higher degree of freedom in lossy compression, so the final bitstream size is less predictable than that in lossless compression.

Fig. 3.12 shows example prediction results. Our approach performs well despite the diversity in the video content and recording situation. The complexity in the content would make it hard to design a simple rule-based

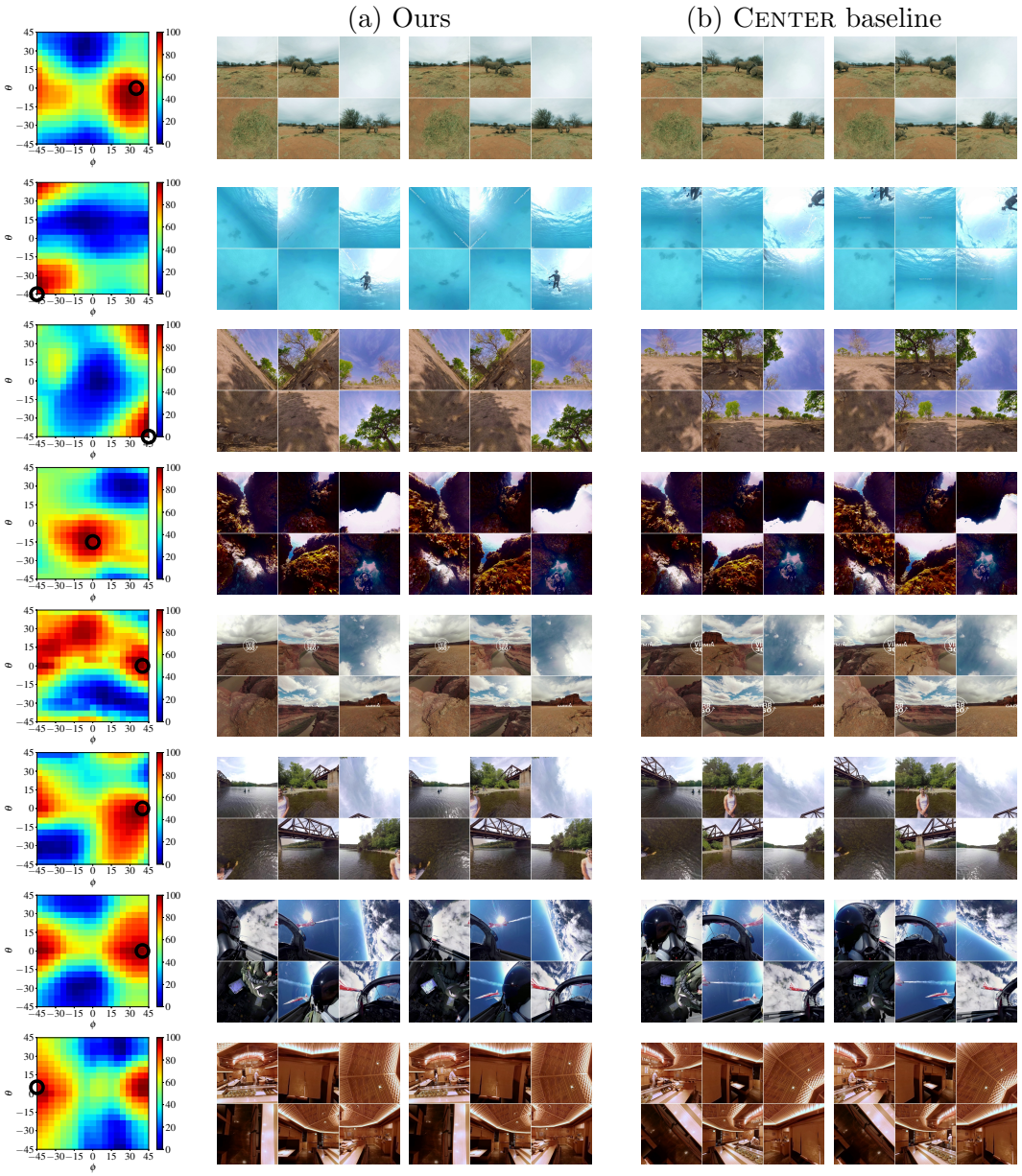


Figure 3.12: Qualitative examples. The heatmap shows the actual normalized reductions for all angles. Black circle shows the predicted result, which is rendered in the second and third figures. The fourth and fifth figures show the CENTER baseline. The second and fourth figures show the first frame of the clip, and the third and fifth figures are the last frame. Best viewed in color.

method to predict Ω^{min} (such as analyzing the continuity in Fig. 3.6); a learning based method is necessary. We can see that even small objects can affect the compression rate, such as the rhinos in the first example and the diver in the second example. The pattern doesn't even have to correspond to a real object like the blank region in the fourth example or the logo in the fifth example. The fifth example also shows how the file size is affected by multiple factors jointly. The distribution would be symmetric with respect to θ=0 if the file size only depended on the logo, but the sky and cloud lead to the additional mode at the top middle. We also see that the continuity of foreground objects is not the only factor that matters from the sixth and seventh example; the person in the sixth example and the pilot in the seventh example lie on the face boundary in the optimal orientation. The result suggests that heuristics based on object location, either automatic or manual, do not solve the problem.

Fig. 3.13 shows failure examples. In the first example, we can see the best compression rate occurs when the coral is continuous, while our method fails because it decides to keep the sun light (round white pattern) continuous. In the second and third example, the video size tends to be smaller when the horizon falls on the face diagonal, possibly because it is more friendly for intra-prediction in compression. Our method doesn't learn this tendency, so it fails to predict the optimal θ and only predicts the correct ϕ.

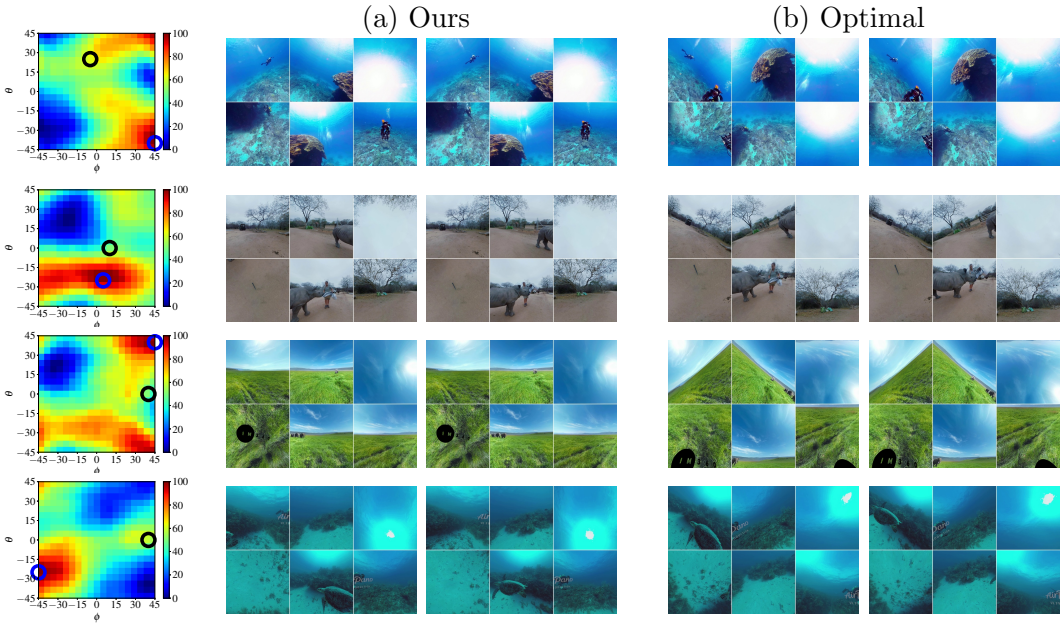


Figure 3.13: Failure cases. Blue circle shows true minimum and is rendered in the fourth and fifth figures. The two figures are the first and last frame of the clip.

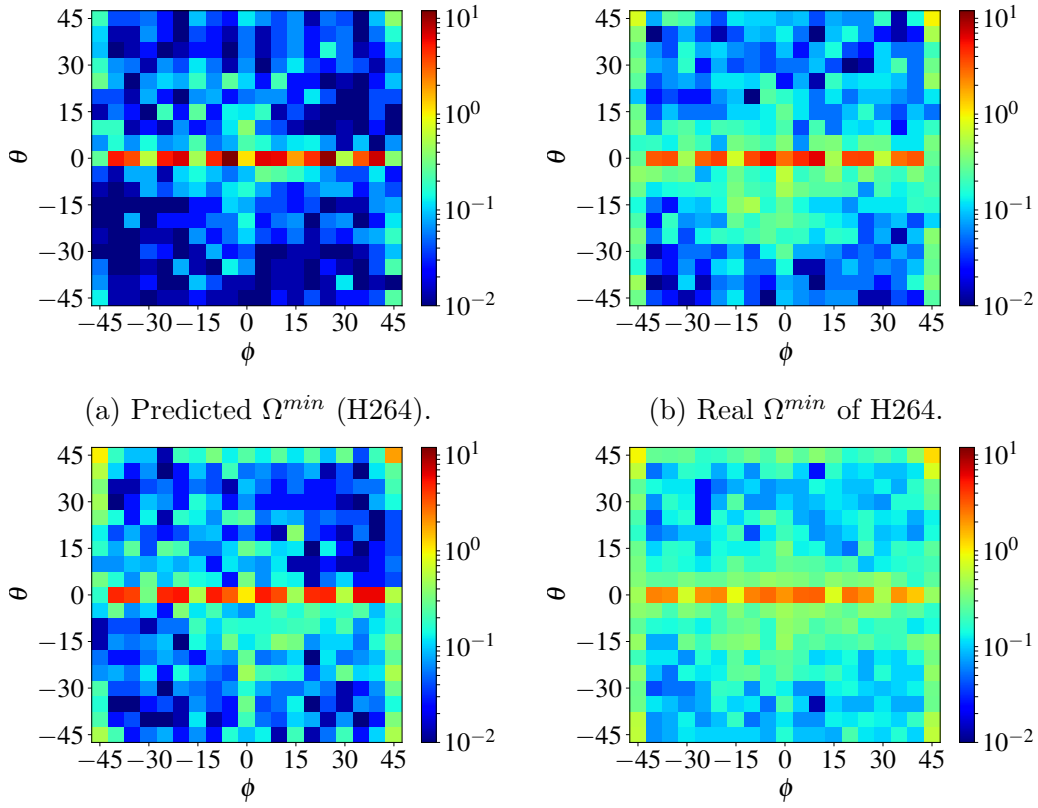
Model transferability We next examine whether the model can be transferred across video formats, e.g. can the model trained on H264 videos improve the compression rate of HEVC videos? First, we evaluate the transferability across different encoders with lossless video compression. Table 3.6 shows the results. Overall, the results show that our approach is capable of generalizing across video formats given common features. We find that the model trained on H264 is less transferable, while the models trained on HEVC and VP9 perform fairly well on H264. In particular, the model trained on HEVC performs the best across all formats. The reasons are twofold. First, the models trained on HEVC and VP9 focus on the appearance feature which is common across all formats. Second, the predictions generated by models trained on H264 is

H264		HEVC		VP9	
HEVC	VP9	H264	VP9	H264	HEVC
70.82	78.17	85.79	84.61	83.19	75.16

Table 3.6: Transferability across video formats. We show the size reduction of our approach. Top row indicates training source, second row is test sources.

more biased. See Fig. 3.14. The predicted Ω^{min} tend to be more concentrated around θ=0 than the real Ω^{min}. Because the distribution of Ω^{min} is more concentrated in H264, so is the prediction of Ω^{min} by the model trained on H264. We hypothesize that the distribution difference is the result of the flexibility of the encoder, and our method works better with more modern codecs like HEVC.

Next we examine the transferability across the GOP length and compression rate. The model is trained with bitstreams encoded using x264 with GOP length 2 and bitrate B/16. We choose these compression settings based on the best correlation between bitstream sizes from our previous analysis (cf. Table 3.3). The size reduction achieved by this *single* prediction model is indicated by Ours (Single Model) in Table 3.5. The performance of the model is almost identical to Ours, which trains one model for each setting independently. In other words, Ours (Single Model) requires only one model but achieves similar performance to Ours, which requires 15 models, one per GOP $\times$ bitrate combination. Note that in practice, Ours may require even more models because of the number of different settings the encoder may support. Ours (Single Model) is therefore more practical than Ours in real



(a) Predicted Ω^{min} (H264).

(b) Real Ω^{min} of H264.

(c) Predicted Ω^{min} (HEVC).

(d) Real Ω^{min} of HEVC.

Figure 3.14: Distribution of Ω^{min} (%). Predictions are on H264 videos with different training data.

video codecs. The result, together with that in Table 3.6, is encouraging because it suggests we can apply the same prediction model for all compression settings without a significant loss in terms of the achievable compression rate.

Bjøntegaard-Delta Rate for Lossy Compression Finally, we compute the BD-rate the proposed method achieves. For JVET test sequences, we report the BD-rate for each video following the convention of the JVET common

	PSNR Y	PSNR U	PSNR V	WS-PSNR Y	WS-PSNR U	WS-PSNR V
Trolley	−0.74%	−1.07%	−1.11%	−1.31%	−1.22%	−1.27%
SkateboardingInLot	3.53%	−26.48%	−16.87%	2.44%	−30.49%	−7.58%
Chairlift	−0.38%	−0.19%	−0.45%	−0.13%	−0.12%	−0.22%
KiteFlite	0.79%	0.15%	0.39%	0.41%	0.08%	0.88%
Harbor	2.13%	−0.11%	0.01%	2.66%	−0.13%	−0.10%
Gaslamp	−3.22%	−1.53%	−2.91%	−4.46%	−1.54%	−2.34%
YouTube (avg.)	1.37%	−2.14%	−2.33%	1.18%	−1.38%	−1.11%

Table 3.7: Bitrate reduction on JVET test sequences and the YouTube 360° video dataset. Refer to Sec. 3.3.2 for metric definitions.

test condition. For the YouTube videos, we report the average result over all 80 videos. BD-rate makes a head-to-head comparison between two compression algorithms; we compare our method to CENTER, given that it performs better than RANDOM in previous analysis and is consistent with the common test condition adopted by JVET.

Table 3.7 shows the results. Our method improves the compression rate in four of the test videos, and the improvement in bitrate is between −0.10% to −30.49%. Note that for *SkateboardingInLot*, although the bitrate increases for the Y (luma) channel, the bitrate reduction is more significant in the color channels so the overall compression rate improves. The result further verifies that we can improve the compression rate of 360° video by optimizing the orientation for cubemap representation, and the range of improvement heavily depends on the video content.

3.4 Conclusion

In this chapter, I introduced how to improve 360° video compression by rotating the video content. I first perform a detailed analysis to verify that the orientation of cubemap projection matters for compression. The analysis across three popular codecs shows scope for reducing video sizes by up to 77% through rotation, with an average of more than 8% over all videos. I then propose an approach that predicts the optimal orientation given the video in a single orientation. It achieves 78% the compression rate of the optimal orientation while requiring less than 0.6% of the computation of a search-based solution (fraction of a second vs. 1.5 hours per GOP).

The proposed approach explores an untapped dimension for 360° video compression and is compatible with ongoing progress in video compression. However, it ignores the interactive nature of 360° videos and is evaluated on objective metrics over the entire video. Because users may interact with the 360° video and focus on a small FOV at a time, the subjective quality may not always align well with the objective metrics because it is determined purely by the content within the FOV. In Chapter 7, I discuss future work plans to exploit user interactions for better 360° video compression. Another limitation of the proposed method is the latency. While being much more efficient than the exhaustive search solution, the latency induced by the two-pass compression pipeline still makes it unsuitable for applications such as video streaming. In order to reduce the latency, we need to avoid the two-pass pipeline and compress the 360° video in the preferable orientation directly.

A proper compression algorithm makes various applications of 360° video possible by allowing us to store, transmit, and distribute this new media easily. On the other hand, simply distributing the raw 360° video is not enough for many potential applications, and further visual content analysis is necessary in order to build more advanced applications. For example, face recognition may be very helpful for video conference applications, and semantic features can help to build a more effective 360° video display as I will show in Chapter 6. Therefore, the next chapter will introduce the challenges for applying existing computer vision models on 360° imagery and our solution for these challenges.

Chapter 4

Learning Spherical Convolution

Many visual data applications rely on visual content analysis. For example, object detection and segmentation are fundamental steps for current visual search systems. In fact, extracting visual features is now part of the standard pipeline in media distributing sites [59]. The same applies for 360° data. As I will show in the Chapter 6, sementic features can help to improve the viewing experience of 360° video. Therefore, performing visual recognition on 360° content becomes a practical need and attracts the attention of both researchers and application developers[1].

Arguably the most powerful tools in computer vision today are CNNs. CNNs are responsible for state-of-the-art results across a wide range of vision problems, including image recognition [46,127], object detection [34,92], image and video segmentation [45,56,80], and action detection [30,100]. Furthermore, significant research effort over the past few years (and really decades [73]) has led to well-honed CNN architectures that, when trained with massive labeled image datasets [27], produce "pre-trained" networks broadly useful as

[1]The work in this chapter was originally published in: "Learning Spherical Convolution for Fast Features from 360° Imagery", in Proceedings of the Advances in Neural Information Processing Systems 2017 [104]. The authors are Yu-Chuan Su and Kristen Grauman.

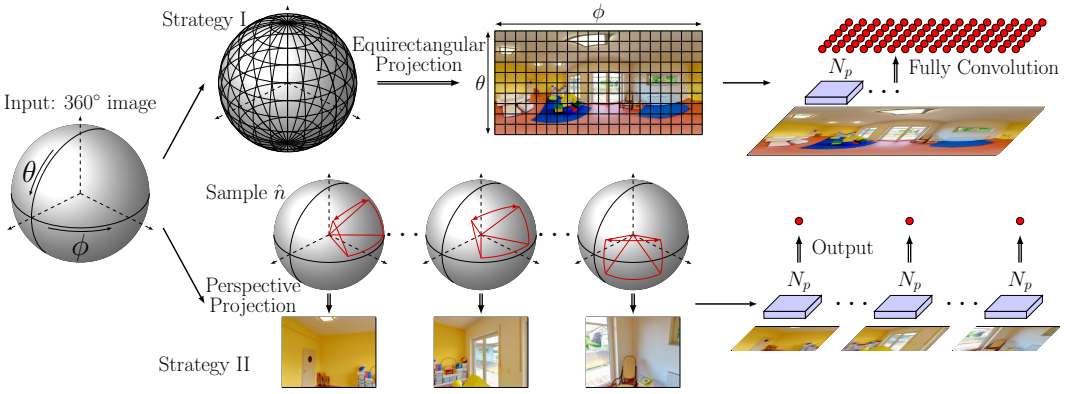


Figure 4.1: Two existing strategies for applying CNNs to 360° images. **Top:** The first strategy unwraps the 360° input into a single planar image using a global projection (most commonly equirectangular projection), then applies the CNN on the distorted planar image. **Bottom:** The second strategy samples multiple tangent planar projections to obtain multiple perspective images, to which the CNN is applied independently to obtain local results for the original 360° image. Strategy I is fast but inaccurate; Strategy II is accurate but slow. The proposed approach learns to replicate flat filters on spherical imagery, offering both speed and accuracy.

feature extractors for new problems. Indeed such networks are widely adopted as off-the-shelf feature extractors for other algorithms and applications (cf., VGG [101], ResNet [46], and AlexNet [69] for images; C3D [114] for video).

However, thus far, powerful CNN models are awkward if not off limits in practice for 360° imagery. The problem is that the underlying projection models of current CNNs and 360° data are different. Both the existing CNN filters *and* the expensive training data that produced them are "flat"', i.e., the product of perspective projection to a plane. In contrast, a 360° image is projected onto the unit sphere surrounding the camera's optical center.

To address this discrepancy, there are two common, though flawed, ap-

proaches. In the first, the 360° image is projected to a planar one, e.g., with equirectangular projection, then the CNN is applied to the resulting 2D image [51, 72] (see Fig. 4.1, top). However, any sphere-to-plane projection introduces distortion, making the resulting convolutions inaccurate. In the second existing strategy, the 360° image is repeatedly projected to tangent planes around the sphere, each of which is then fed to the CNN [105, 108, 119, 125] (Fig. 4.1, bottom). In the extreme of sampling every tangent plane, this solution is exact and therefore accurate. However, it suffers from very high computational cost. Not only does it incur the cost of rendering each planar view, but also it prevents amortization of convolutions: the intermediate representation cannot be shared across perspective images because they are projected to different planes.

I propose a learning-based solution that, unlike the existing strategies, sacrifices neither accuracy nor efficiency. The main idea is to learn a CNN that processes a 360° image in its equirectangular projection (fast) but mimics the "flat" filter responses that an existing network would produce on all tangent plane projections for the original spherical image (accurate). Because convolutions are indexed by spherical coordinates, I refer to our method as *spherical convolution* (SphConv). I develop a systematic procedure to adjust the network structure in order to account for distortions. Furthermore, I propose a kernel-wise pre-training procedure which significantly accelerates the training process.

In addition to providing fast general feature extraction for 360° imagery,

my approach provides a bridge from 360° content to existing heavily supervised datasets dedicated to perspective images. In particular, training requires no new annotations—only the source CNN model (e.g., VGG [101] pre-trained on millions of labeled images) and an arbitrary collection of unlabeled 360° images.

4.1 Spherical Convolution Definition

We first define the objective for spherical convolution. Let I_s be the input spherical image, and let I_e be the corresponding flat RGB image in equirectangular projection. We define the perspective projection operator $\mathcal{P}$ which projects an α-degree FOV from I_s to W pixels on the the tangent plane $\hat{n}$:

$$\mathcal{P}(I_s, \hat{n}) = I_p \in \mathbb{I}^{W \times W \times 3}. \tag{4.1}$$

The projection operator is characterized by the pixel size

$$\Delta_p \theta = \alpha / W \tag{4.2}$$

in I_p, and I_p denotes the resulting perspective image. Note that we assume $\Delta\theta = \Delta\phi$ following common digital imagery.

Given a source network[2] N_p trained on perspective images I_p with receptive field $R \times R$, we define the output on I_s at $\hat{n} = (\theta, \phi)$ as

$$N_p(I_s)[\theta, \phi] = N_p(\mathcal{P}(I_s, (\theta, \phi))), \tag{4.3}$$

[2]e.g., N_p could be AlexNet [69] or VGG [101] pre-trained for a large-scale recognition task.

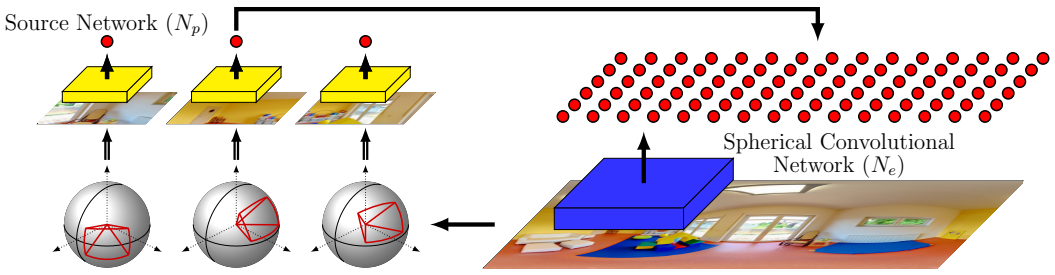


Figure 4.2: The objective for a spherical convolution network (SphConv) is to mimic the output of a source model on the perspective projection while taking equirectangular projection as input. This can be considered as a knowledge distillation across different projections.

where w.l.o.g. we assume $W = R$ for simplicity. The goal is to learn a spherical convolution network N_e that takes an equirectangular map I_e as input and, for every image position (x, y), produces as output the results of applying the perspective projection network to the corresponding tangent plane for spherical image I_s:

$$N_e(I_e)[x, y] \approx N_p(I_s)[\theta, \phi], \qquad \forall (x, y) \in D^e, \tag{4.4}$$

where

$$D^e = \{0, 1, \cdots, W_e - 1\} \times \{0, 1, \cdots, H_e - 1\}. \tag{4.5}$$

Note the image coordinate (x, y) in equirectangular projection and spherical coordinate (θ, ϕ) are linearly mapped by

$$(\theta, \phi) = (\frac{\pi y}{H_e}, \frac{2\pi x}{W_e}). \tag{4.6}$$

See Fig. 4.2.

This can be seen as a domain adaptation problem where we want to transfer the model from the domain of I_p to that of I_e. However, unlike typical

domain adaptation problems, the difference between I_p and I_e is characterized by a geometric projection transformation rather than a shift in data distribution. Note that the training data to learn N_e requires no manual annotations: it consists of arbitrary 360° images coupled with the "true" N_p outputs computed by exhaustive planar reprojections, i.e., evaluating the rhs of Eq. 4.3 for every (θ, ϕ). Furthermore, at *test* time, only a single equirectangular projection of the entire 360° input will be computed using N_e to obtain the dense (inferred) N_p outputs, which would otherwise require multiple projections and evaluations of N_p.

4.2 Approach for Learning Spherical Convolution

This section describes how we learn the spherical convolution network. We exploit the domain knowledge for spherical convolution to facilitate learning. Sec. 4.2.1 introduces how to adapt the structure from the source network, and Sec. 4.2.2 presents the training process.

4.2.1 Network Structure

The main challenge for transferring N_p to N_e is the distortion introduced by equirectangular projection. The distortion is location dependent—a $k \times k$ square in perspective projection will not be a square in the equirectangular projection, and its shape and size will depend on the polar angle θ. See Fig. 4.3. The convolution kernel should transform accordingly. The proposed approach 1) adjusts the shape of the convolution kernel to account for the

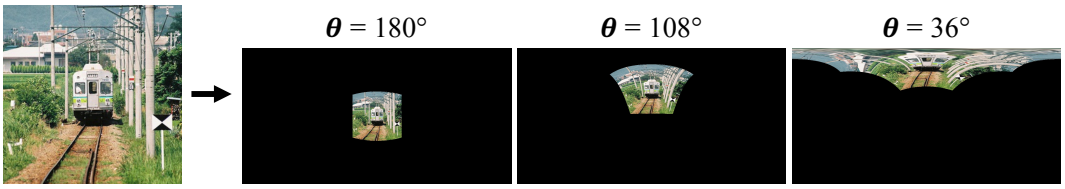


Figure 4.3: Inverse perspective projections $\mathcal{P}^{-1}$ to equirectangular projections at different polar angles θ. The same square image will distort to different sizes and shapes depending on θ. Because equirectangular projection unwraps the 180° longitude, a line will be split into two if it passes through the 180° longitude, which causes the double curve in $\theta = 36°$.

distortion, in particular the content expansion, and 2) reduces the number of max-pooling layers to match the pixel sizes in N_e and N_p, as I detail next.

We adapt the architecture of N_e from N_p using the following heuristic. The goal is to ensure each kernel receives enough information from the input in order to compute the target output. First, we untie the weight of convolution kernels at different θ by learning one kernel K_e^y for each output row y. Next, we adjust the shape of K_e^y such that it covers the receptive field of the original kernel. We consider $K_e^y \in N_e$ to cover $K_p \in N_p$ if more than 95% of pixels in the receptive field of K_p are also in the receptive field of K_e in I_e. The receptive field of K_p in I_e is obtained by backprojecting the $R \times R$ grid to $\hat{n} = (\theta, 0)$ using $\mathcal{P}^{-1}$, where the center of the grid aligns on $\hat{n}$. K_e should be large enough to cover K_p, but it should also be as small as possible to avoid overfitting. Therefore, we optimize the shape of $K_e^{l,y}$ for layer l as follows. The shape of $K_e^{l,y}$ is initialized as 3×3. We first adjust the height k_h and increase k_h by 2 until the height of the receptive field is larger than that of K_p in I_e. We then adjust the width k_w similar to k_h. Furthermore, we restrict the kernel

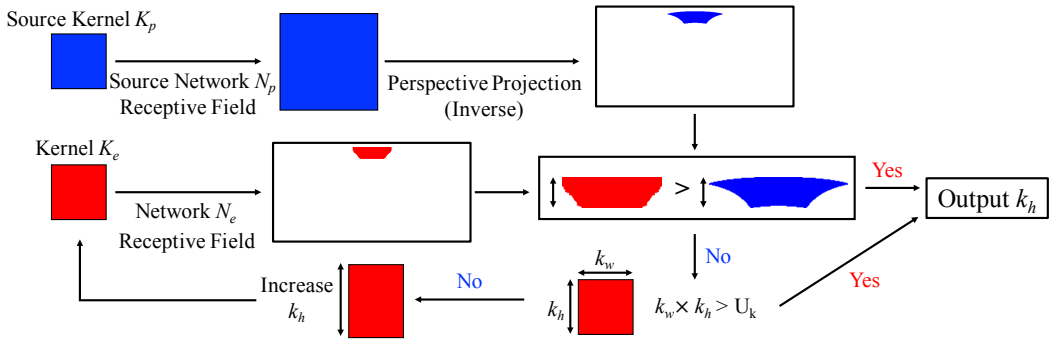


Figure 4.4: Method to select the kernel height k_h. We project the receptive field of the source kernel to equirectangular projection I_e and increase k_h until it is taller than the source kernel in I_e. The kernel width k_w is determined using the same procedure after k_h is set. We restrict the kernel size $k_w \times k_h$ by an upper bound U_k.

size $k_h \times k_w$ to be smaller than an upper bound U_k. See Fig. 4.4. Because the receptive field of K_e^l depends on K_e^{l-1}, we search for the kernel size starting from the bottom layer.

It is important to relax the kernel from being square to being rectangular, because equirectangular projection will expand content horizontally near the poles of the sphere (see Fig. 4.3). If we restrict the kernel to be square, the receptive field of K_e can easily be taller but narrower than that of K_p which leads to overfitting. It is also important to restrict the kernel size, otherwise the kernel can grow wide rapidly near the poles and eventually cover the entire row. Although cutting off the kernel size may lead to information loss, the loss is not significant in practice because pixels in equirectangular projection do not distribute on the unit sphere uniformly; they are denser near the pole, and the pixels are by nature redundant in the region where the kernel size expands

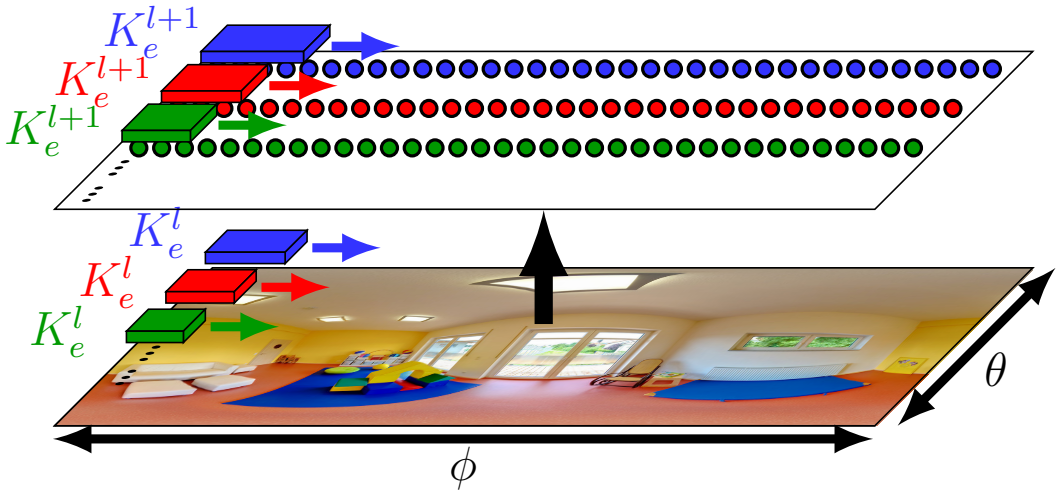


Figure 4.5: Spherical convolution. The kernel weight in spherical convolution is tied only along each row of the equirectangular image (i.e., ϕ), and each kernel convolves along the row to generate 1D output. Note that the kernel size differs at different rows and layers, and it expands near the top and bottom of the image.

dramatically.

Besides adjusting the kernel sizes, we also adjust the number of pooling layers to match the pixel size $\Delta\theta$ in N_e and N_p. We define $\Delta\theta_e = 180°/H_e$ and restrict $W_e = 2H_e$ to ensure $\Delta\theta_e = \Delta\phi_e$. Because max-pooling introduces shift invariance up to k_w pixels in the image, which corresponds to $k_w \times \Delta\theta$ degrees on the unit sphere, the physical meaning of max-pooling depends on the pixel size. Since the pixel size is usually larger in I_e and max-pooling increases the pixel size by a factor of k_w, we remove the pooling layer in N_e if $\Delta\theta_e \geq \Delta\theta_p$.

Fig. 4.5 illustrates how spherical convolution differs from ordinary CNN. Note that we approximate one layer in N_p by one layer in N_e, so the number of layers and output channels in each layer is exactly the same as the source

network. However, this does not have to be the case. For example, we could use two or more layers to approximate each layer in N_p. Although doing so may improve accuracy, it would also introduce significant overhead, so we stick with the one-to-one mapping.

4.2.2 Training Process

Given the goal in Eq. 4.4 and the architecture described in Sec. 4.2.1, we would like to learn the network N_e by minimizing the L_2 loss

$$E[(N_e(I_e) - N_p(I_s))^2]. \tag{4.7}$$

However, the network converges slowly, possibly due to the large number of parameters. Instead, we propose a kernel-wise pre-training process that disassembles the network and initially learns each kernel independently.

To perform kernel-wise pre-training, we further require N_e to generate the same intermediate representation as N_p in all layers l:

$$N_e^l(I_e)[x, y] \approx N_p^l(I_s)[\theta, \phi] \quad \forall l \in N_e. \tag{4.8}$$

Given Eq. 4.8, every layer $l \in N_e$ is independent of each other. In fact, every kernel is independent and can be learned separately. We learn each kernel by taking the "ground truth" value of the previous layer $N_p^{l-1}(I_s)$ as input and minimizing the L_2 loss

$$E[(N_e^l(I_e) - N_p^l(I_s))^2], \tag{4.9}$$

except for the first layer. Note that N_p^l refers to the convolution output of layer l before applying any non-linear operation, e.g. ReLU, max-pooling, etc. It is

important to learn the target value before applying ReLU because it provides more information. we combine the non-linear operation with K_e^{l+1} during kernel-wise pre-training, and we use dilated convolution [121] to increase the receptive field size instead of performing max-pooling on the input feature map.

For the first convolution layer, we derive the analytic solution directly. The projection operator $\mathcal{P}$ is linear in the pixels in equirectangular projection:

$$\mathcal{P}(I_s, \hat{n})[x, y] = \sum_{ij} c_{ij} I_e[i, j], \tag{4.10}$$

for coefficients c_{ij} from, e.g., bilinear interpolation. Because convolution is a weighted sum of input pixels

$$K_p * I_p = \sum_{xy} w_{xy} I_p[x, y], \tag{4.11}$$

we can combine the weight w_{xy} and interpolation coefficient c_{ij} as a single convolution operator:

$$K_p^1 * I_s[\theta, \phi] = \sum_{xy} w_{xy} \sum_{ij} c_{ij} I_e[i, j] = \sum_{ij} \Big(\sum_{xy} w_{xy} c_{ij} \Big) I_e[i, j] = K_e^1 * I_e. \tag{4.12}$$

The output value of N_e^1 will be exact and requires no learning. Of course, the same is not possible for $l > 1$ because of the non-linear operations between layers. See sec. 5.2.4 for more detailed discussion.

After kernel-wise pre-training, we can further fine-tune the network *jointly* across layers and kernels by minimizing the L_2 loss of the final output.

Because the pre-trained kernels cannot fully recover the intermediate representation, fine-tuning can help to adjust the weights to account for residual errors. We ignore the constraint introduced in Eq. 4.8 when performing fine-tuning. Although Eq. 4.8 is necessary for kernel-wise pre-training, it restricts the expressive power of N_e and degrades the performance if we only care about the final output.

4.3 Experiments

To evaluate our approach, we consider both the accuracy of its convolutions as well as its applicability for object detections in 360° data. We use the VGG architecture[3] and the Faster R-CNN [92] model as our source network N_p. We learn a network N_e to produce the topmost (conv5_3) convolution output.

4.3.1 Dataset

We use two datasets: Pano2Vid for training, and Pano2Vid and Pascal VOC for testing.

Pano2Vid We sample frames from the 360° videos in the Pano2Vid dataset [108] for both training and testing. The dataset consists of 86 videos crawled from YouTube using four keywords: "Hiking," "Mountain Climbing," "Parade," and "Soccer". We sample frames at 0.05fps to obtain 1,056 frames for train-

[3] `https://github.com/rbgirshick/py-faster-rcnn`

ing and 168 frames for testing. We use "Mountain Climbing" for testing and others for training, so the training and testing frames are from disjoint videos. Because the supervision is on a per pixel basis, this corresponds to $N \times W_e \times H_e \approx 250M$ (non i.i.d.) samples. See Sec. 6.3.1 for more information about the dataset. Note that most object categories targeted by the Faster R-CNN detector do not appear in Pano2Vid, meaning that our experiments test the content-independence of my approach.

Pascal VOC Because the source model was originally trained and evaluated on Pascal VOC 2007, we "360-ify" it to evaluate the object detector application. We test with the 4,952 images in Pascal VOC 2007 validation set, which contain 12,032 bounding boxes. We transform them to equirectangular images as if they originated from a 360° camera. In particular, each object bounding box is backprojected to 3 different scales $\{0.5R, 1.0R, 1.5R\}$ and 5 different polar angles $\theta \in \{36°, 72°, 108°, 144°, 180°\}$ on the 360° image sphere using the inverse perspective projection, where R is the resolution of the source network's receptive field. Regions outside the bounding box are zero-padded. Backprojection allows us to evaluate the performance at different levels of distortion in the equirectangular projection.

4.3.2 Evaluation Metrics

We generate the output widely used in the literature (conv5_3) and evaluate it with the following metrics.

Network output error The first metric measures the difference between $N_e(I_e)$ and $N_p(I_s)$. We report the root-mean-square error (RMSE) over all pixels and channels. For Pascal VOC, we measure the error over the receptive field of the detector network.

Detector network performance The second metric measures the performance of the detector network in Faster R-CNN using multi-class classification accuracy. We replace the ROI-pooling in Faster R-CNN by pooling over the bounding box in I_e. Note that the bounding box is backprojected to equirectangular projection and is no longer a square region.

Proposal network performance The last metric evaluates the proposal network in Faster R-CNN using average Intersection-over-Union (IoU). For each bounding box centered at $\hat{n}$, we project the conv5_3 output to the tangent plane $\hat{n}$ using $\mathcal{P}$ and apply the proposal network at the center of the bounding box on the tangent plane. Given the predicted proposals, we compute the IoUs between foreground proposals and the bounding box and take the maximum. The IoU is set to 0 if there is no foreground proposal.

4.3.3 Baselines

We compare our method with the following baselines.

- EXACT — Compute the true target value $N_p(I_s)[\theta, \phi]$ for every pixel. This serves as an upper bound in performance and does not consider the compu-

tational cost.

- DIRECT — Apply N_p on I_e directly. We replace max-pooling with dilated convolution to produce a full resolution output. This is Strategy I in Fig. 4.1 and is used in 360° video analysis [51, 72].

- INTERP — Compute $N_p(I_s)[\theta, \phi]$ every S-pixels and interpolate the values for the others. We set S such that the computational cost is roughly the same as our SPHCONV. This is a more efficient variant of Strategy II in Fig. 4.1.

- PERSPECT — Project I_s onto a cube map [70] and then apply N_p on each face of the cube, which is a perspective image with 90° FOV. The result is backprojected to I_e to obtain the feature on I_e. We use W=960 for the cube map resolution so $\Delta\theta$ is roughly the same as I_p. This is a second variant of Strategy II in Fig. 4.1 used in PanoContext [125].

We also evaluate three variants of the proposed SPHCONV:

- OPTSPHCONV — To compute the output for each layer l, OPTSPHCONV computes the exact output for layer $l-1$ using $N_p(I_s)$ then applies spherical convolution for layer l. OPTSPHCONV serves as an upper bound for our approach, where it avoids accumulating any error across layers.

- SPHCONV-PRE — Uses the weights from kernel-wise pre-training directly without fine-tuning.

- SphConv — The full spherical convolution with joint fine-tuning of all layers.

4.3.4 Implementation Details

We set the resolution of I_e to 640×320. For the projection operator $\mathcal{P}$, we map α=65.5° to W=640 pixels following SUN360 [119]. The pixel size is therefore $\Delta\theta_e$=360°/640 for I_e and $\Delta\theta_p$=65.5°/640 for I_p. Accordingly, we remove the first three max-pooling layers so N_e has only one max-pooling layer following conv4_3. The kernel size upper bound U_k=7 × 7 following the max kernel size in VGG.

We train the network using ADAM [66]. For pre-training, we use the batch size of 256 and initialize the learning rate to 0.01. For layers without batch normalization, we train the kernel for 16,000 iterations and decrease the learning rate by 10 every 4,000 iterations. For layers with batch normalization, we train for 4,000 iterations and decrease the learning rate every 1,000 iterations. For fine-tuning, we first fine-tune the network on conv3_3 for 12,000 iterations with batch size of 1. The learning rate is set to 1e-5 and is divided by 10 after 6,000 iterations. We then fine-tune the network on conv5_3 for 2,048 iterations. The learning rate is initialized to 1e-4 and is divided by 10 after 1,024 iterations. We do not insert batch normalization in conv1_2 to conv3_3 because we empirically find that it increases the training error.

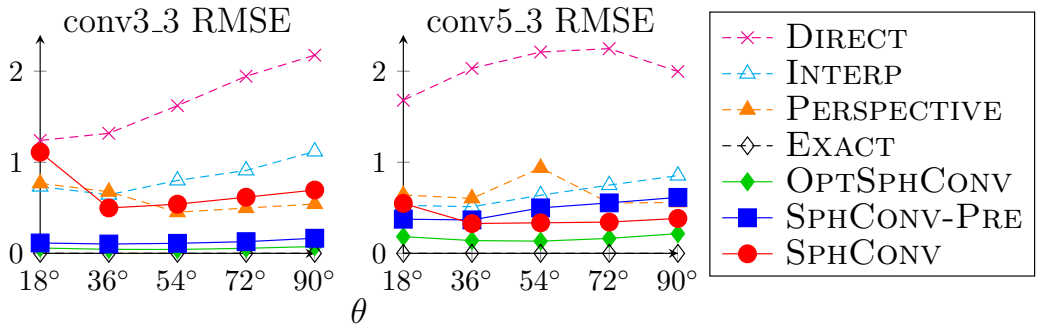


Figure 4.6: Network output error on Pano2Vid; lower is better. Note the error of Exact is 0 by definition. SphConv's convolutions are much closer to the exact solution than the baselines'.

4.3.5 Results

Convolution output accuracy Fig. 4.6 shows the output error of layers conv3_3 and conv5_3 on the Pano2Vid [108] dataset. The error is normalized by that of the mean predictor. We evaluate the error at 5 polar angles θ uniformly sampled from the northern hemisphere, since error is roughly symmetric with the equator.

First we discuss the three variants of our method. OptSphConv performs the best in all layers and θ, validating our main idea of spherical convolution. It performs particularly well in the lower layers, because the receptive field is larger in higher layers and the distortion becomes more significant. Overall, SphConv-Pre performs the second best, but as to be expected, the gap with OptConv becomes larger in higher layers because of error propagation. SphConv outperforms SphConv-Pre in conv5_3 at the cost of larger error in lower layers (as seen here for conv3_3). It also has larger error at θ=18°

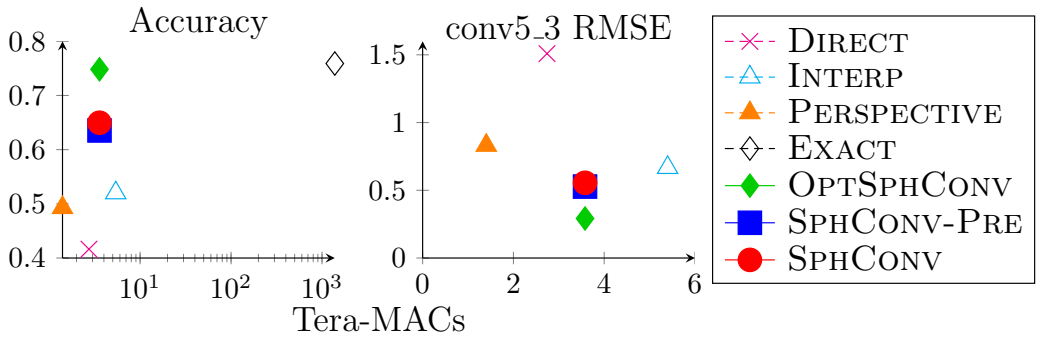


Figure 4.7: Computational cost vs. accuracy on Pascal VOC. Our approach yields accuracy closest to the exact solution while requiring orders of magnitude less computation time (left plot). SphConv's cost is similar to the other approximations tested (right plot). Plot titles indicate the y-labels, and error is measured by root-mean-square-error (RMSE).

for two possible reasons. First, the learning curve indicates that the network learns more slowly near the pole, possibly because the receptive field is larger and the pixels degenerate. Second, we optimize the joint L_2 loss, which may trade the error near the pole with that at the center.

Comparing to the baselines, we see that SphConv achieves lowest errors. Direct performs the worst among all methods, underscoring that convolutions on the flattened sphere—though fast—are inadequate. Interp performs better than Direct, and the error decreases in higher layers. This is because the receptive field is larger in the higher layers, so the S-pixel shift in I_e causes relatively smaller changes in the receptive field and therefore the network output. Perspective performs similarly in different layers and outperforms Interp in lower layers. The error of Perspective is particularly large at θ=54°, which is close to the boundary of the perspective image and

has larger perspective distortion.

Computational cost Fig. 4.7 shows the accuracy vs. cost tradeoff. Computational cost is measured by the number of Multiply-Accumulate (MAC) operations. The leftmost plot shows cost on a log scale. Here we see that EXACT—whose outputs we wish to replicate—is about 400 times slower than SPHCONV, and SPHCONV approaches EXACT's detector accuracy much better than all baselines. The second plot shows that SPHCONV is about 34% faster than INTERP (while performing better in all metrics). PERSPECTIVE is the fastest among all methods and is 60% faster than SPHCONV, followed by DIRECT which is 23% faster than SPHCONV. However, both baselines are noticeably inferior in accuracy compared to SPHCONV.

SphConv kernel visualization To visualize what SPHCONV has learned, we learn the first layer of the AlexNet [69] model provided by the Caffe package [58] and examine the resulting kernels. Fig. 4.8 shows the original kernel K_p and the corresponding kernels K_e at different polar angles θ. K_e is usually the re-scaled version of K_p, but the weights are often amplified because multiple pixels in K_p fall to the same pixel in K_e like the second example. We also observe situations where the high frequency signal in the kernel is reduced, like the third example, possibly because the kernel is smaller. Note that we learn the first convolution layer for visualization purposes only, since $l = 1$ (only) has an analytic solution (cf. Sec 4.2.2).

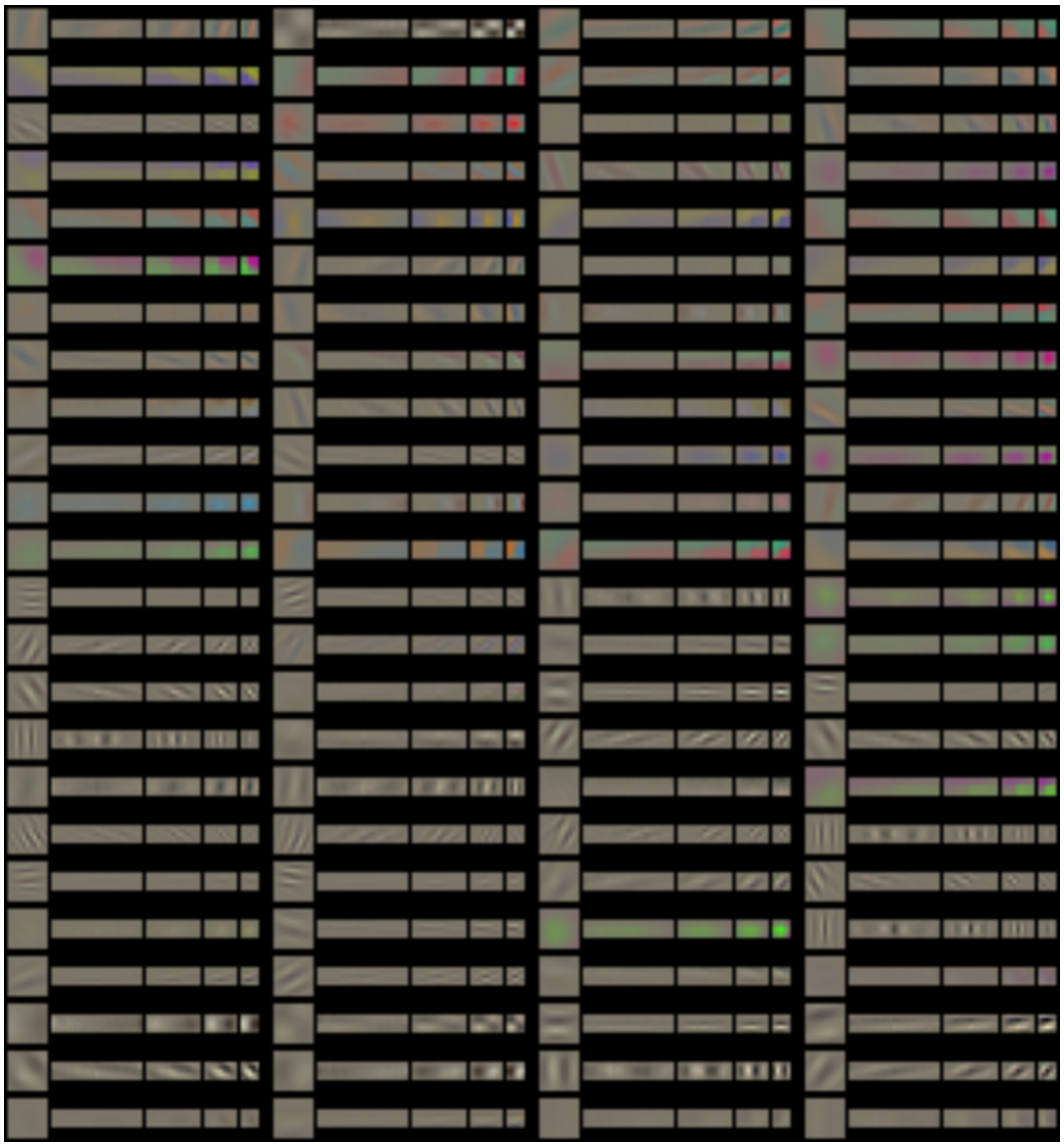


Figure 4.8: AlexNet conv1 kernels (left squares) and their corresponding four SphConv-Pre kernels at $\theta \in \{9°, 18°, 36°, 72°\}$ (left to right).

Object detection accuracy Having established SphConv provides accurate and efficient N_e convolutions, we now examine how important that

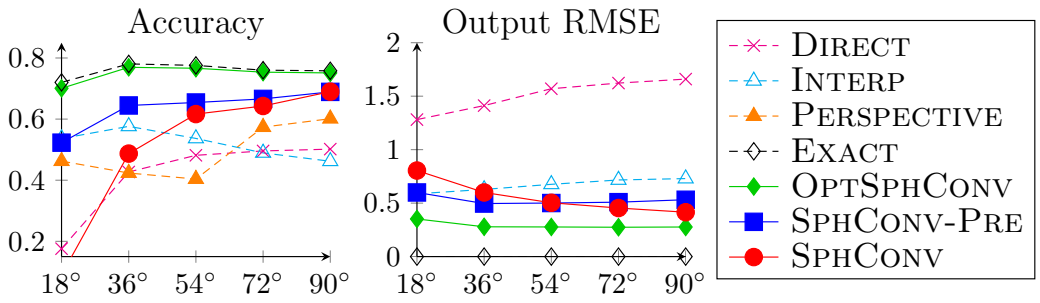


Figure 4.9: Faster R-CNN object detection accuracy on a 360° version of Pascal VOC across polar angles θ. Best viewed in color.

accuracy is to object detection on 360° inputs. Fig. 4.9 shows the result of the Faster R-CNN detector network on Pascal VOC in 360° format. OptSphConv performs almost as well as Exact. The performance degrades in SphConv-Pre because of error accumulation, but it still significantly outperforms Direct and is better than Interp and Perspective in most regions. Although joint training (SphConv) improves the output error near the equator, the error is larger near the pole which degrades the detector performance. Note that the receptive field of the detector network spans multiple rows, so the error is the weighted sum of the error at different rows. The result, together with Fig. 4.6, suggest that SphConv reduces the conv5_3 error in parts of the receptive field but increases it at the other parts. The detector network needs accurate conv5_3 features throughout the receptive field in order to generate good predictions.

Direct again performs the worst. In particular, the performance drops significantly at θ=18°, showing that it is sensitive to the distortion. In contrast, Interp performs better near the pole because the samples are denser

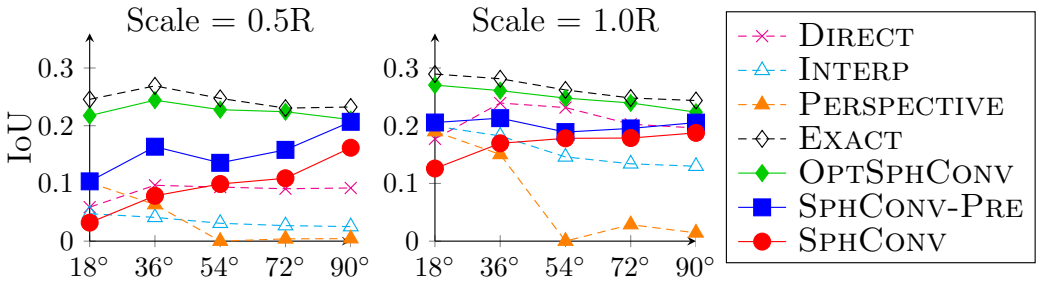


Figure 4.10: Faster R-CNN object detection average IoU on a 360° version of Pascal VOC across polar angles θ. R refers to the receptive field of N_p.

on the unit sphere. In fact, Interp should converge to Exact at the pole. Perspective outperforms Interp near the equator but is worse in other regions. Note that $\theta \in \{18°, 36°\}$ falls on the top face, and $\theta = 54°$ is near the border of the face. The result suggests that Perspective is still sensitive to the polar angle, and it performs the best when the object is near the center of the faces where the perspective distortion is small.

Object proposal accuracy Fig. 4.10 shows the performance of the object proposal network for two scales. Interestingly, the result is different from the detector network. OptSphConv still performs almost the same as Exact, and SphConv-Pre performs better than baselines. However, Direct now outperforms other baselines, suggesting that the proposal network is not as sensitive as the detector network to the distortion introduced by equirectangular projection. The performance of the methods is similar when the object is larger (right plot), even though the output error is significantly different. The only exception is Perspective, which performs poorly for $\theta \in \{54°, 72°, 90°\}$

regardless of the object scale. It again suggests that objectness is sensitive to the perspective image being sampled.

Fig. 4.11 shows examples of objects successfully detected by our approach in spite of severe distortions. Also see Fig. 4.12 for failure examples.

4.4 Conclusion

In this chapter, I proposed to learn spherical convolutions for 360° images. Because the solution entails a new form of distillation across camera projection models, it can transfer existing CNN models trained on large "flat" images to 360° images without any annotation effort. Compared to current practices for applying CNN models on 360° images/video, spherical convolution benefits efficiency by avoiding performing multiple perspective projections, and it benefits accuracy by adapting kernels to the distortions in equirectangular projection. Results on two datasets demonstrate how it successfully transfers state-of-the-art vision models from the realm of limited FOV 2D imagery into the realm of omnidirectional data.

One practical limitation of SphConv is the model size. Because it unties the kernel weights along θ, the model size grows linearly with the equirectangular image height. The model size can easily grow to tens of gigabytes as the image resolution increases. A more compact model would be necessary in order to incorporate SphConv into standard framework. In the next chapter, I will introduce an approach that can reduce the spherical convolution network size and make the model more tractable on current hardware.

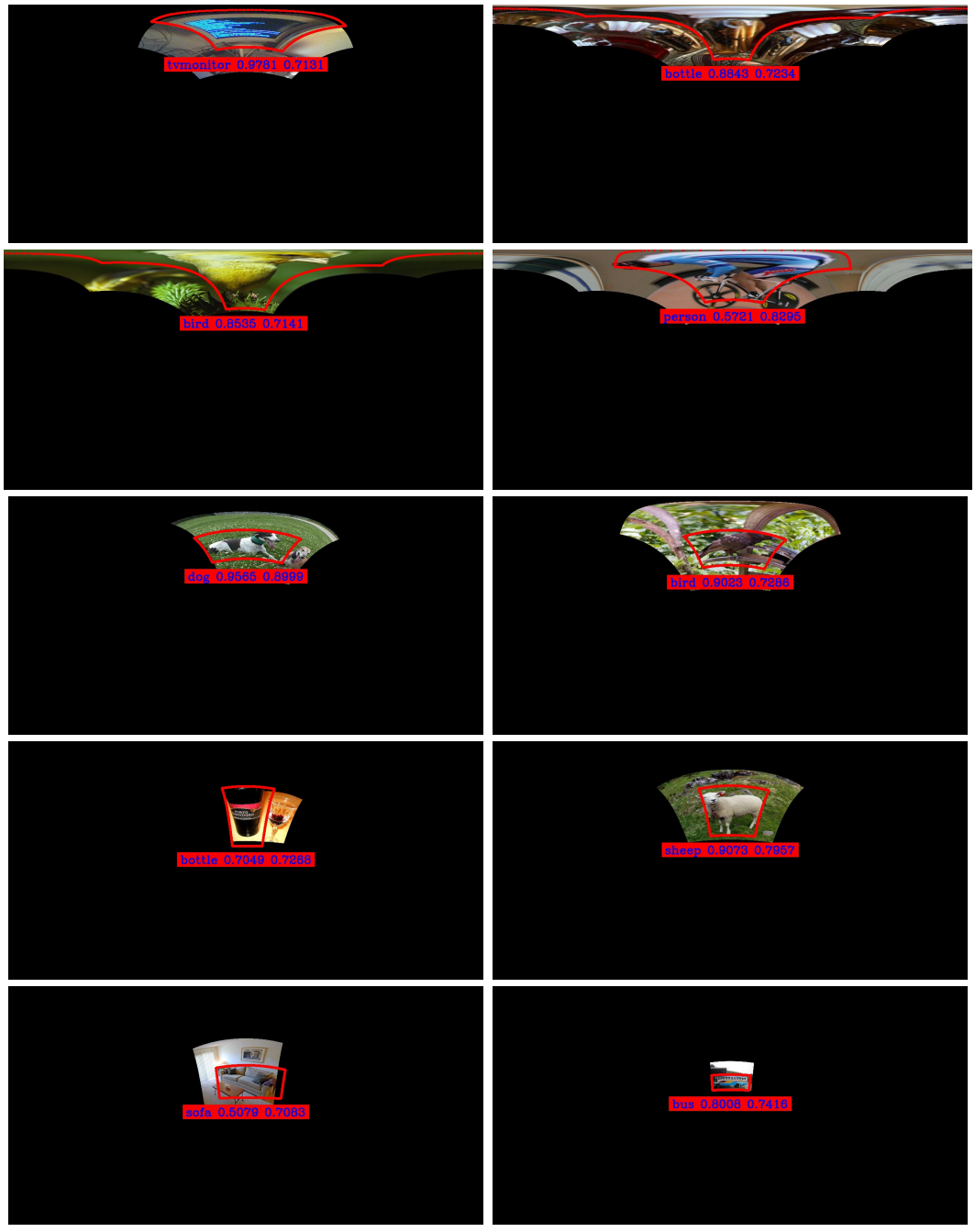


Figure 4.11: Object detection examples on 360° Pascal VOC test images. Text gives predicted label, multi-class probability, and IoU, resp. SphConv successfully detects objects undergoing severe distortion, some of which are barely recognizable even for a human viewer.

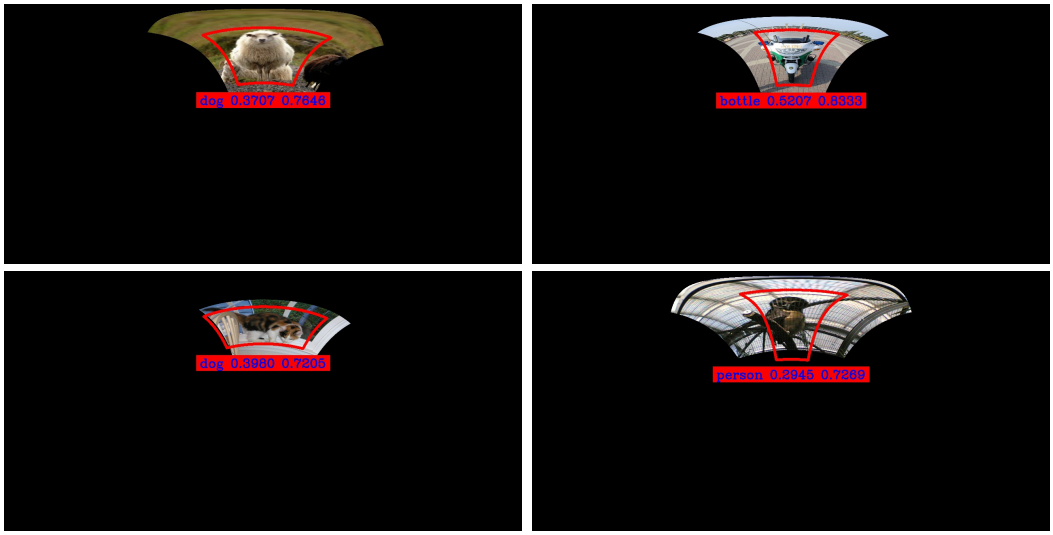


Figure 4.12: Failure cases for SphConv.

Chapter 5

Kernel Transformer Networks for Compact Spherical Convolution

In the previous chapter, I introduced the spherical convolution network for visual recognition on 360° images. While SphConv is both accurate and computationally efficient, it suffers from significant model bloat. For example, the size of a VGG network will grow from 56MB to 29GB in SphConv when the model takes a 640×320 input resolution. The model size is even larger than the memory of the latest GPU in 2019, which makes the model hard to deploy in practice[1].

Since our initial SphConv work, several other methods have been proposed for learning a new CNN on 360° images. Broadly speaking, they pursue one of two approaches. The first approach adapts the kernels on the sphere, resampling the kernels or projecting their tangent plane features [24, 126]. While allowing kernel sharing and hence smaller models, this approach degrades accuracy—especially for deeper networks—due to an implicit interpolation assumption as I will explain below. The second approach defines convo-

[1]The work in this chapter was originally published in: "Kernel Transformer Networks for Compact Spherical Convolution," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition 2019 [107]. The authors are Yu-Chuan Su and Kristen Grauman.

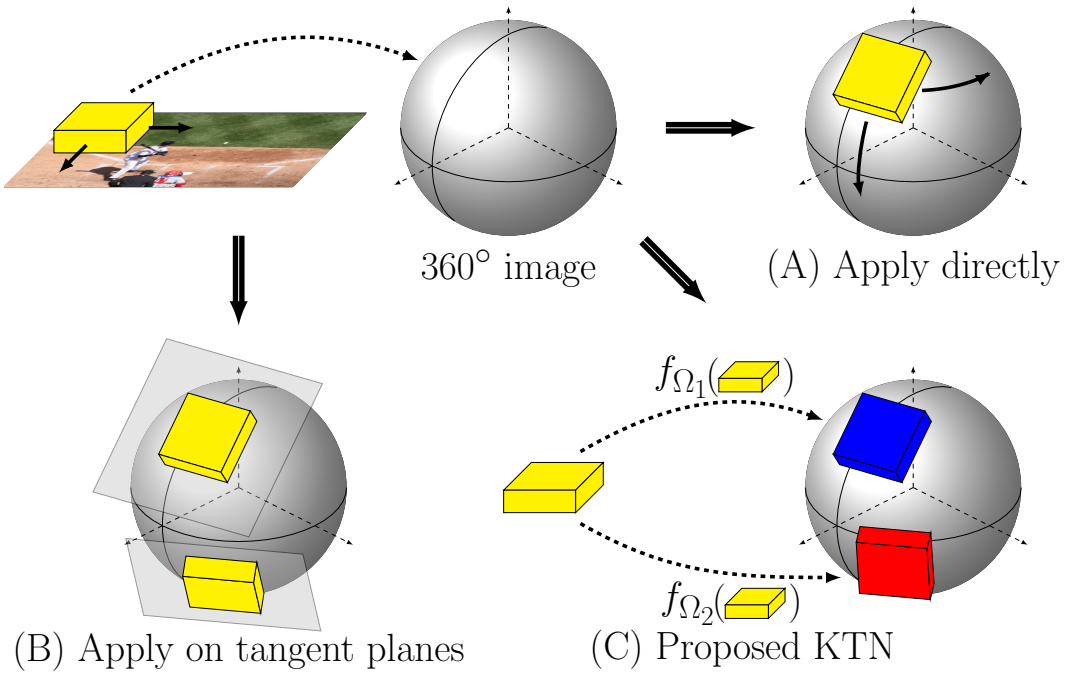


Figure 5.1: The goal is to transfer CNNs trained on planar images to 360° images. As introduced in Chapter 4, common approaches either (A) apply CNNs directly on the equirectangular projection of a 360° image or (B) project the content to tangent planes and apply the models on the tangent planes. In contrast, kernel transformer network (KTN) adapts the kernels in CNNs to account for the distortion in 360° images.

lution in the spectral domain [23, 29], which has significant memory overhead and thus far limited applicability to real-world data. Moreover, all of the above require retraining to handle a new recognition task.

In light of these shortcomings, I propose the kernel transformer network (KTN). KTN is built upon SphConv introduced in the previous chapter. However, instead of learning the kernels in the equirectangular projection pixel space directly, KTN learns a *function* that takes a kernel in the source CNN as input and generates the corresponding spherical convolution kernels

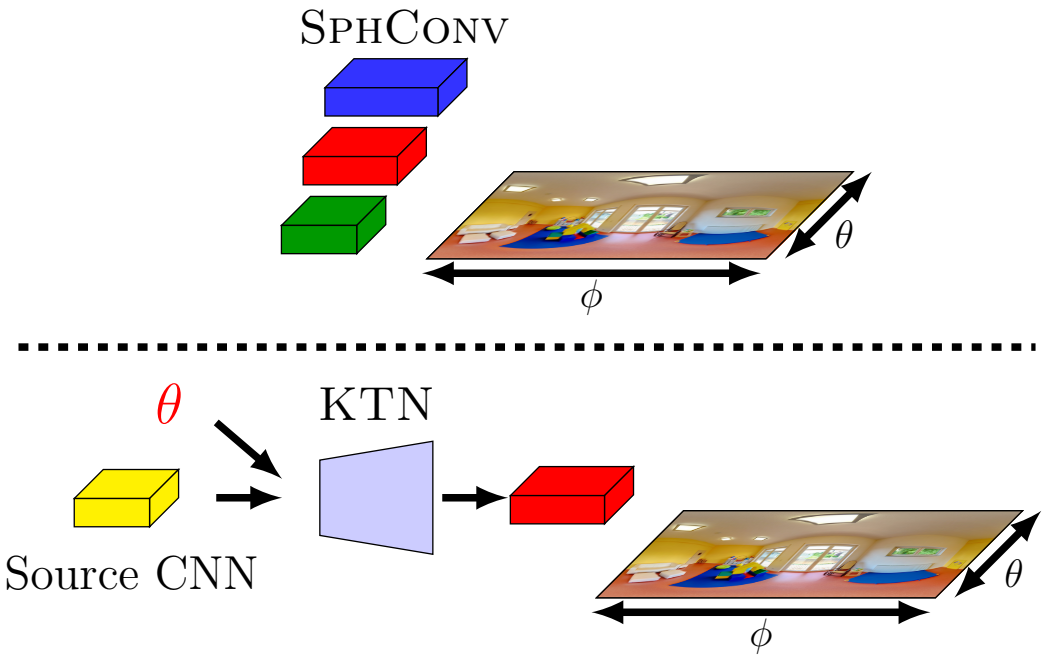


Figure 5.2: KTN vs. SphConv. The proposed KTN improves upon the SphConv introduced in Chapter 4 by learning a function f that generates the spherical convolution kernels from the source kernel. The generator formulation leads to a *transferable* and more *compact* model compared with SphConv.

as output. See Fig. 5.1 (C) and Fig. 5.2. The function accounts for the distortion in 360° images, returning different transformations depending on both the polar angle θ and the source kernel. Following SphConv, the transformation function 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 the function-based design. Once trained for a

Figure 5.3: KTN is transferable across different source models. The same KTN can take different source CNNs as input in order to perform different visual recognition task, given that the source CNNs have the same architecture.

base architecture, the same KTN can transfer multiple source CNNs to 360° images. For example, having trained a KTN for VGG [101] on ImageNet classification, we can transfer the same KTN to run a VGG-based Pascal VOC object detector on 360° panoramas. See Fig. 5.3. This is possible because the KTN takes the source CNN as input rather than embed the CNN kernels into its own parameters (unlike [23, 24, 29, 104, 126]). 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%). Compared with SphConv, KTN avoids the need of learning and storing the spherical convolution kernels explicitly and reduces the model size by orders of magnitude. The smaller model size enables implementing spherical convolution on GPU and makes the method more usable.

5.1 Kernel Transformer Network Definition

In this section, I introduce the concept of a kernel transformer network for transferring convolutions to 360° images. KTN can be considered as a

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. However, as discussed in the previous chapter, this assumption 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

$$F[x, y] = \Sigma_{i,j} K[i, j] * I[x - i, y - j], \tag{5.1}$$

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

$$K_\Omega = f(K, \Omega) \tag{5.2}$$

$$F[x, y] = \Sigma_{i,j} K_\Omega[i, j] * I[x - i, y - j] \tag{5.3}$$

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

$$\Omega = g(\theta, \phi), \tag{5.4}$$

where θ and ϕ are the polar and azimuthal angle in spherical coordinates, respectively. Given the KTNs and the new definition of convolution, the proposed approach permits applying an ordinary CNN to 360° images by replacing the convolution operation in Eq. 5.1 with Eq. 5.3.

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 VOC object detection; with the same KTN, both tasks can be translated to 360° images. Whereas this transfer is not possible with SphConv (Chapter 4), it is possible in KTN because KTN factorizes the spherical convolution kernels into the source kernel and the transformations.

5.2 Approach for Learning Kernel Transformer Network

This section describes the implementation of KTN for spherical convolution. I first describe the architecture and objective function of KTN. Next, I will further discuss the difference between KTN and existing methods for learning CNNs on 360° data.

5.2.1 KTN Architecture

Following SphConv, KTN considers 360° images that are unwrapped into 2D rectangular images using equirectangular projection. The main benefit of equirectangular projection for KTNs is that the distortion depends only on the polar angle. Because the polar angle has a 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 [24,104,126]. 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.

We address the first requirement (size and cost) by employing depthwise separable convolutions [19, 50] 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 [50], 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

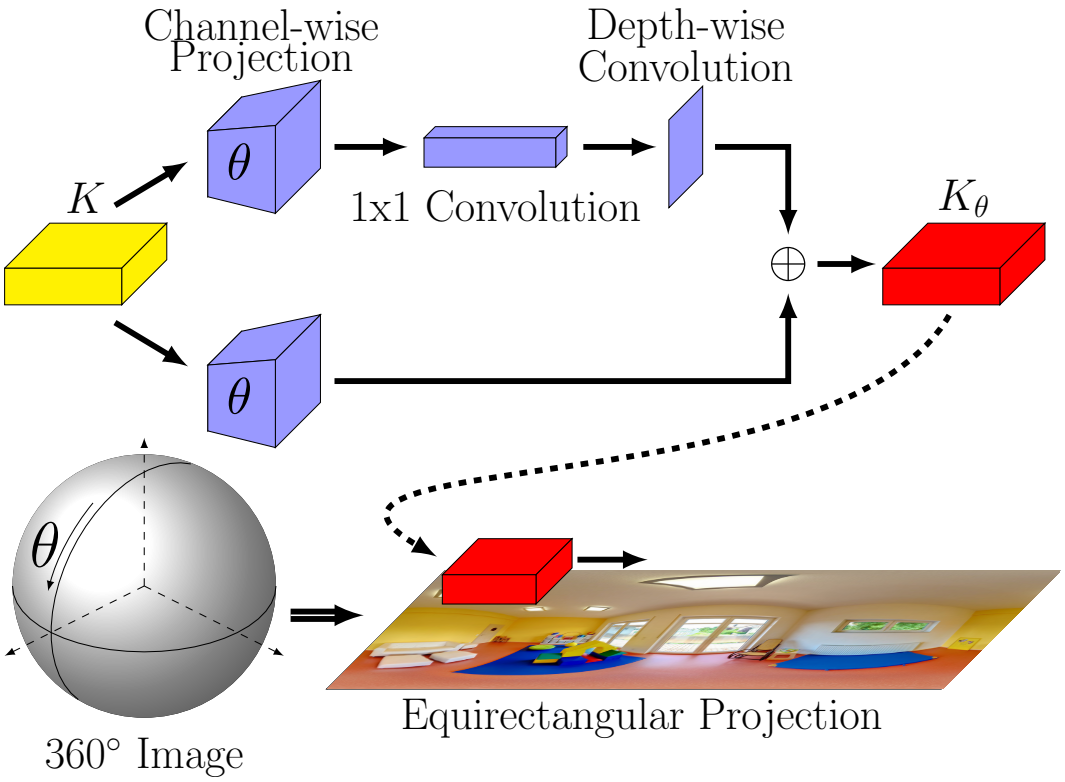


Figure 5.4: KTN consists of row dependent channel-wise projections that resize the kernel to the target size and depth separable convolution blocks. It takes a source kernel K and θ as input and generates an output kernel K_Ω. K_Ω is then applied to the 360° image in its equirectangular projection at row $y{=}\theta H/\pi$. The transformation accounts for the distortion in equirectangular projection, while maintaining cross-channel interactions.

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, KTN performs channel-wise projection to reduce the model size.

The complete architecture for KTN is in Fig. 5.4. It adopts a Residual Network [46]-like architecture. For both the residual and shortcut branches, the model first applies the row dependent projection to resize the kernel to the target size. The residual branch then applies depthwise separable convolution twice. The 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. 5.3.

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. 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. 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.

5.2.2 KTN Objective and Training Process

Having introduced the KTN module and how to apply it for CNNs on 360° images, I 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. The goal is to minimize the difference between the feature map generated by the source kernels K^l and that generated by the KTN module:

$$\mathcal{L} = \|F^{l+1} - f^l(K^l, \Omega) * F^l\|^2 \tag{5.5}$$

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

$$(\theta, \phi) = (\frac{\pi \times y}{H}, \frac{2\pi \times x}{W}) \tag{5.6}$$

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. The objective function is similar to that of SphConv (Chapter 4), but KTN optimizes 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 any 360° images. The 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. 5.5. 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. KTN learns the important components of those transformations from data. 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, Eq. 5.5 is a stronger condition while also enjoying the advantage of bypassing any annotated training data.

5.2.3 Spherical Faster R-CNN

Using KTNs, I next introduce how to implement a spherical Faster R-CNN model [92] based on spherical convolution. Faster R-CNN is a two-stage object detection model that has been widely adopted in various applications. It first uses a backbone CNN (e.g., VGG, ResNet) to extract a feature map from the input image. It then solves the object detection problem on top of

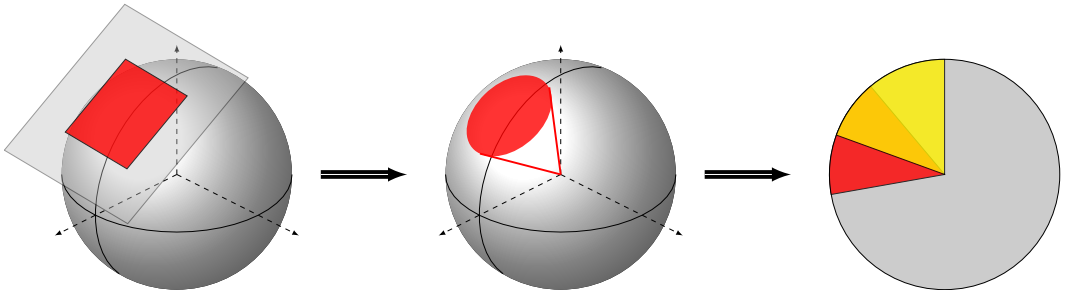


Figure 5.5: Spherical non-maximum suppression. We approximate rectangular proposals using cones on the sphere and then cast the cone overlap problem into a sector overlap problem, which can be computed more efficiently than computing the exact overlap between proposals.

the feature map with two sequential sub-tasks, i.e., object proposal generation and object classification. A region proposal network (RPN) is first applied on the feature map to generate candidate bounding boxes. Because the RPN may predict multiple proposals for the same object, non-maximum suppression (NMS) is then applied to remove redundant proposals. An object detector network then uses a region-of-interest (ROI) pooling to generate the features for each object proposal from the same input feature map as the region proposal network. It then predicts the object category and refines the bounding box for each candidate proposal.

In our Spherical Faster R-CNN implementation, we replace the backbone CNN with our spherical convolution network using KTN. While KTN transfers kernels and therefore the features from planar images to spherical images, it does not transfer the proposals and ROI pooling, which are defined on 2D planes. Instead of redefining the object proposals and ROI pooling operation, we project the feature map onto tangent planes before applying the

region proposal and detector networks on the projected feature maps, which allows us to reuse existing models directly without any change. This is particularly important when we want to detect objects in different scales, where we can simply change the FOV of the projection operation while leaving the rest of the model the same. We use nearest neighbor interpolation for the projection operation and precompute the indices in order to compute the projected features efficiently. While feature projection introduces error as we will discuss in Sec. 5.2.4, the error is tolerable because we perform feature projection only once—as opposed to accumulating projection errors across layers of a convolutional network.

While the RPN generates a separate set of proposals for each tangent plane, the proposals defined on different tangent planes may overlap when back projected onto the sphere. Therefore, instead of applying the detection on each tangent plane independently, we perform NMS over the proposals from all tangent planes before applying the detector network. However, because the bounding boxes on different tangent planes are not well aligned on a single 2D plane, computing the exact overlap is inefficient and computationally infeasible. To improve the computational efficiency, we propose a spherical NMS that uses an approximate overlap between the proposals in NMS. In practice, we approximate each rectangular proposal using a cone on the sphere. The center of the cone co-located with the center of the bounding box and the radius of the cone is

$$FOV_c = \frac{FOV_x + FOV_y}{2}, \tag{5.7}$$

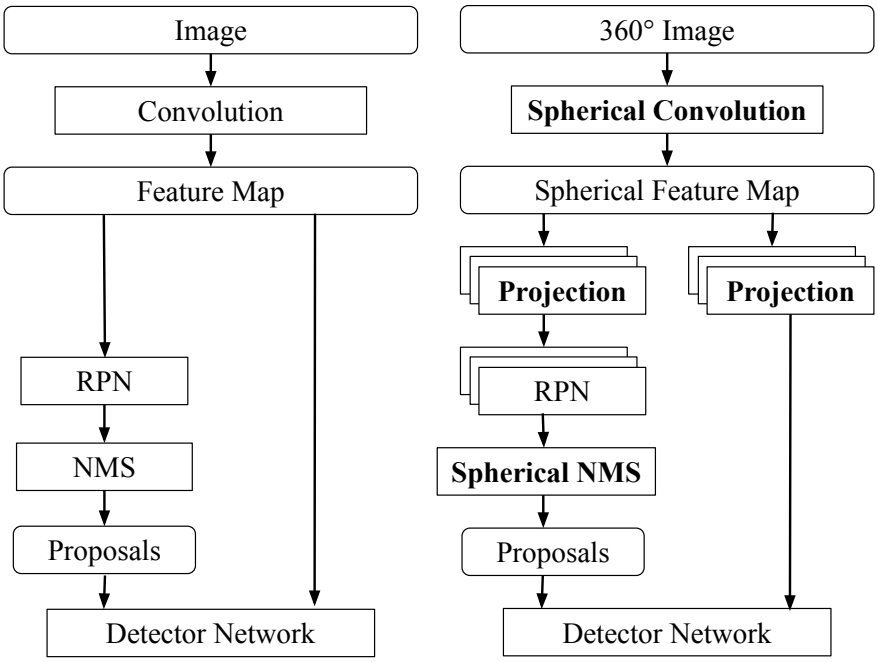


Figure 5.6: Comparison between ordinary Faster R-CNN and spherical Faster R-CNN. The spherical Faster R-CNN model 1) replaces the ordinary CNN with a KTN spherical convolution network for feature extraction, 2) projects features to tangent planes before applying the region proposal network (RPN) and object detector network, and 3) replaces non-maximum suppression (NMS) with spherical NMS introduced in Fig. 5.5.

where FOV_x and FOV_y correspond to the horizontal and vertical FOV of the proposal respectively. To compute the overlap between two cones, we cast the cone to one dimension and reduce the problem to the overlap between two circular sectors. See Fig. 5.5. This approximation allows us to compute the overlap and therefore NMS more efficiently.

To recap, we translate the Faster R-CNN object detector to spherical data using KTN as follows: first, we replace the backbone CNN with a spher-

ical convolutional network to extract spherical feature map. We then project the features to tangent planes and apply the RPN on each tangent plane. Redundant proposals from all tangent planes are removed using a spherical NMS process, before the proposals are fed into the detector network to generate the final object prediction. See Fig. 5.6 for the full pipeline.

5.2.4 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.

Compared with KTN, SphConv learns the kernels adapted to the distortion in equirectangular projection. Instead of learning the transformation function f in Eq. 5.2, 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) experimental results show that the two methods perform similarly in terms of accuracy. Furthermore, the

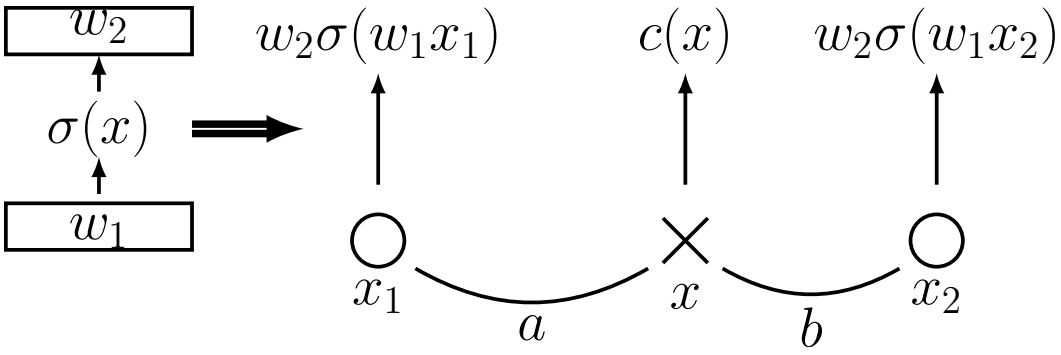


Figure 5.7: Beyond the first CNN layer, the feature interpolation assumption in SphereNet [24] yields only approximated results. See text for details.

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, 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. 5.3.4). In addition, since SphConv trains K_Ω for a single source kernel K, the model does not generalize to different source CNNs.

SphereNet [24] 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. 5.7. If we interpolate the pixel first and apply the kernels, the output of at location x is

$$c(x) = w_2 \times \sigma(w_1(ax_1 + bx_2)). \tag{5.8}$$

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

$$c(x) = aw_2 \times \sigma(w_1 x_1) + bw_2 \times \sigma(w_1 x_2). \tag{5.9}$$

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. 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 [11, 17], 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 [11,17], 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

Table 5.1: 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 [23]	Yes	Yes	Yes	c^2H	No
SPHERICAL CNN [29]	Yes	Yes	Yes	c^2H	No
SPHERICAL U-NET [126]	Yes	No	Yes	c^2k^2	No
SPHERENET [24]	Yes	No	Yes	c^2k^2	No
SPHCONV	Yes	No	No	c^2k^2H	No
KTN	Yes	No	No	$c^2k^2 + c^2$	Yes

computational overhead [105].

Table 5.1 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).

5.3 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.

5.3.1 Dataset

The 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 [108]. 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 is a perspective image dataset with object annotations. We backproject the object bounding boxes to equirectangular projection with 640×320 resolution. Following SphConv (Chapter 4), we use the accuracy of the detector network in Faster R-CNN on the validation set as the evaluation metric instead of evaluating the full spherical Faster R-CNN model. This

dataset is used for evaluation only. Note that the settings for Pano2Vid and Pascal VOC are the same as that in Chapter 4. Please refer to Sec. 4.3 for details.

5.3.2 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 [92] models with VGG architecture [101] 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.

5.3.3 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, with cube padding [17]. For the Pano2Vid and Pascal VOC datasets, the conv5_3 feature map is re-projected to equirectangular projection as the final output.

- S^2CNN [23]—We use the S2Convolution and SO3Convolution in the authors' implementation[2] 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 [29]—We use the sphconv module in the authors' implementation[3] 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

[2] `https://github.com/jonas-koehler/s2cnn`
[3] `https://github.com/daniilidis-group/spherical-cnn`

conv5_3 feature map is resized to 640×320 as the final output.

- Spherical U-Net [126]—We use the SphericalConv module in Spherical U-Net[4] 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 [24]—We implement the SphereNet model using row dependent channel-wise projection. The authors' code and data were unavailable at the time of this work. Because feature projection is the weighted sum of the features, the projection weights can be combined 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.

- 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.

- SphConv—Because the model is too large to fit into GPU memory even for evaluation, it is run on CPUs.[5]

[4] `https://github.com/xuyanyu-shh/Saliency-detection-in-360-video`
[5] For the other baselines, testing is still possible with GPUs.

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. 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 at the time of publication. 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 [23], 64×64 [29], or 150×300 [126]. On the Spherical MNIST dataset, all methods use the exact same 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 KTN are trained to reproduce the conv3 outputs of the source model. For Pano2Vid, all methods are trained to reproduce the conv5_3 outputs.

Table 5.2: 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 [23]	95.79	2.37	4.32
Spherical CNN [29]	97.48	2.36	6.06
Spherical U-Net [126]	98.43	2.54	24.98
SphereNet [24]	87.20	2.46	46.68
Projected	10.70	4.24	6.15
SphConv	**98.72**	**1.50**	63.54
KTN	97.94	1.53	**69.48**

5.3.4 Results

Model Accuracy Table 5.2 summarizes the methods' CNN accuracy on all three 360° datasets. SphConv performs the best on Spherical MNIST, and the performance of KTN is on par with SphConv. 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 Equirectangu-

LAR 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. 5.2.4, 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 performs 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, 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

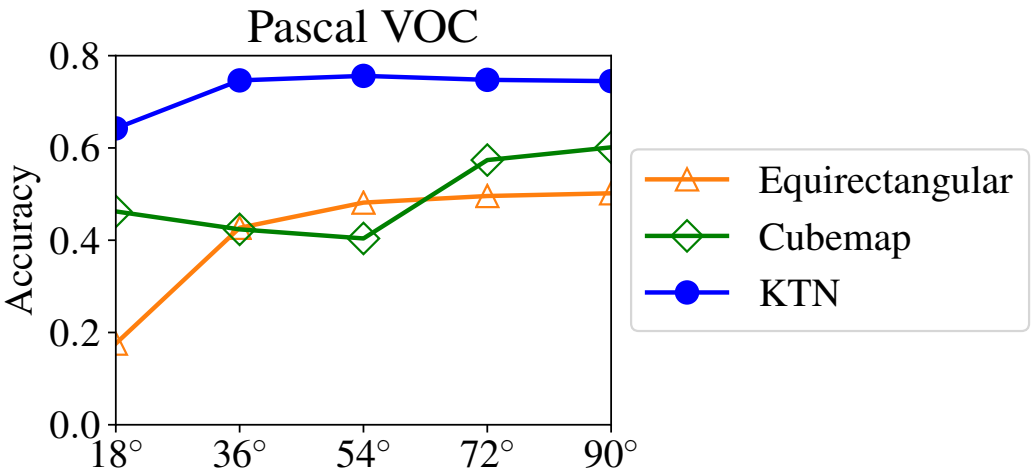


Figure 5.8: Model accuracy across θ of projection based baselines.

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 constraint 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. 5.8 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

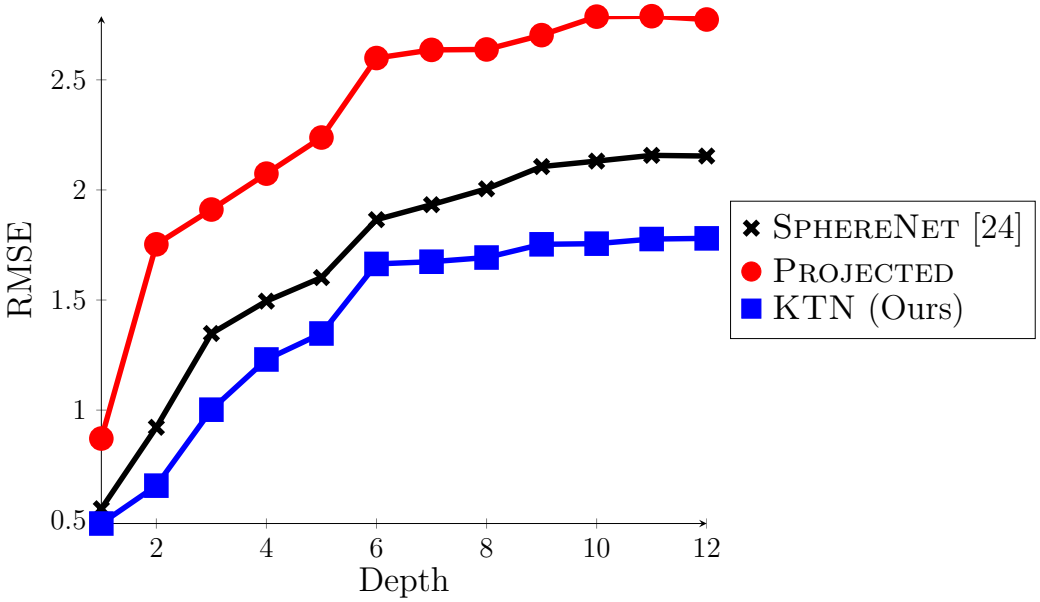


Figure 5.9: Model accuracy at different layers.

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.

As discussed previously, the interpolation assumption made by SphereNet [24] and the Projected baselines is problematic, particularly at deeper layers as errors accumulate. Hence, we compare the accuracy of SphereNet [24], 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. 5.9. 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. 5.2.4.

Fig. 5.10 shows example outputs of KTN with a Faster R-CNN source model. The detector successfully detects objects despite the distortion. On the other hand, because SphConv and KTN aim to reproduce the behavior of source CNNs, their accuracy are bounded by the source model. For example, the spherical Faster R-CNN model fails to detect the person in the middle in the bottom right image in Fig. 5.10, because the source model does not recognize a person from the top view. Note that the outputs in Fig. 5.10 are generated by the full spherical Faster R-CNN model, while previous evaluations use the ground truth bounding boxes following the experiment settings in Chapter 4.

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

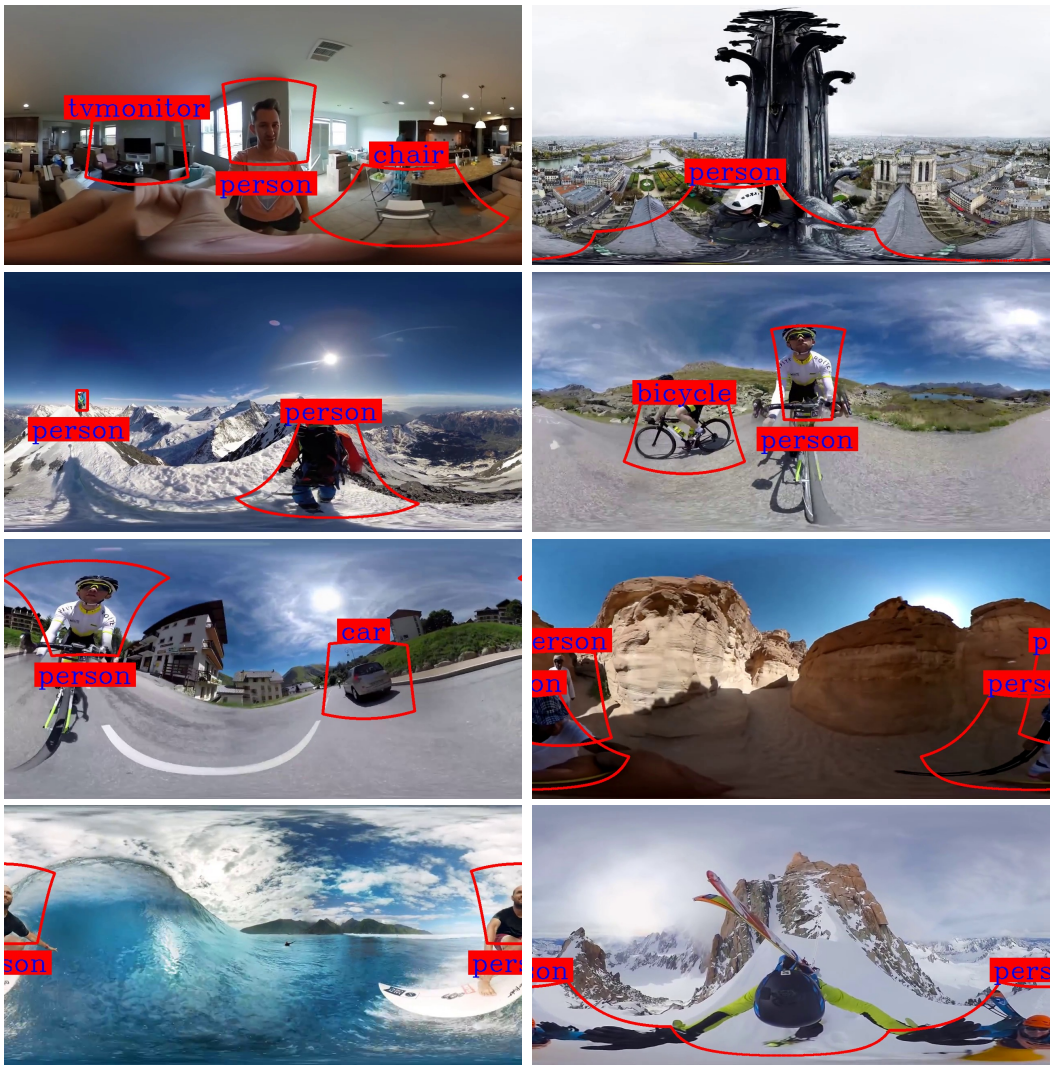


Figure 5.10: KTN object detection examples on Pano2Vid.

it is being tested on as KTN-TRANSFER and KTN otherwise.

Fig. 5.11 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

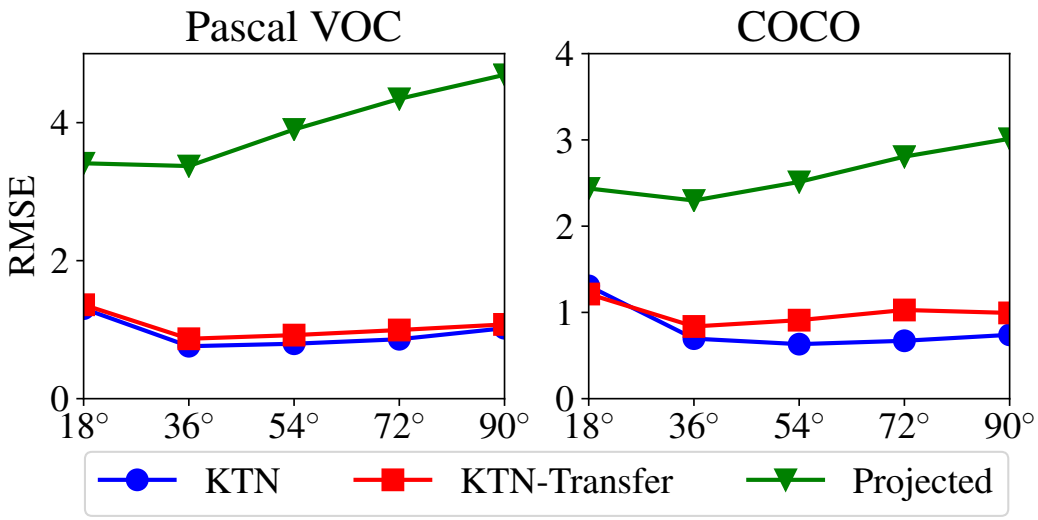


Figure 5.11: Model transferability. The title indicates the source CNN being tested. KTN performs almost identically regardless of the source network it is trained on. The results show we can learn a single KTN and apply it to other source CNNs with the same architecture, even if that source model is trained for a different task.

with the same base architecture. None of the existing models [23, 24, 29, 104, 126] 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. 5.11. 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. For experiments between VGG trained for ImageNet classification and Faster R-CNN trained for Pascal object

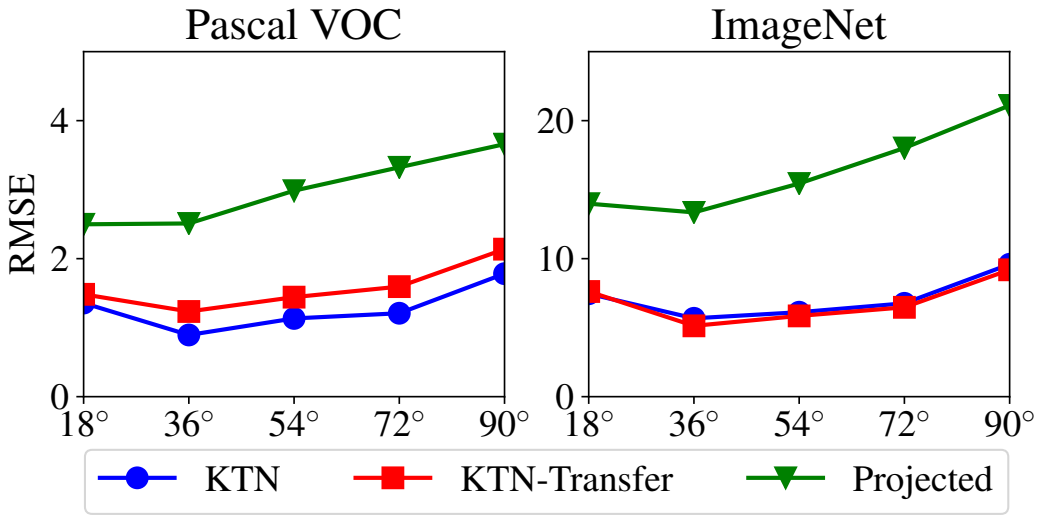


Figure 5.12: Model transferability of ImageNet trained VGG.

detection, and the results are similar. See Fig. 5.12.

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.

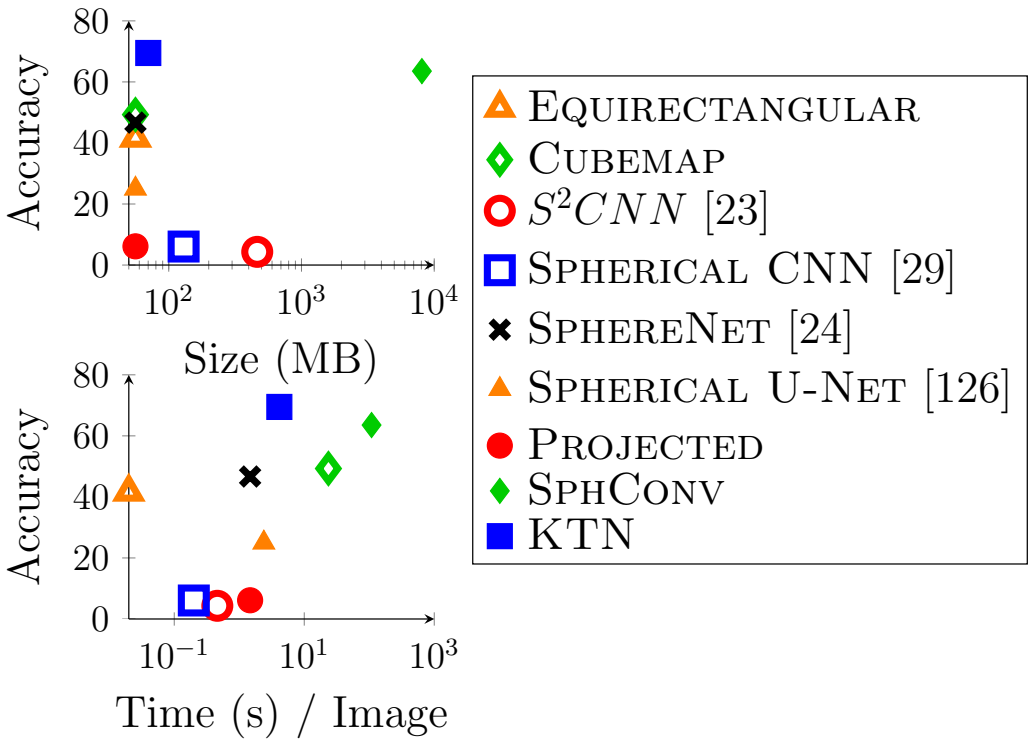


Figure 5.13: Model size (top) and speed (bottom) vs. accuracy for VGG. KTN is orders of magnitude smaller than SphConv, and it is similarly or more compact as all other models, while being significantly more accurate.

Fig. 5.13 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 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.

5.4 Conclusion

I proposed 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. Experimental results show that KTN outperforms existing methods while providing superior scalability and transferability.

One limitation of KTN is that it cannot handle very close objects that span a large FOV. Because the goal of our spherical convolution is to reproduce the behavior of models trained on perspective images, the capability and performance of the model is bounded by the source CNN. However, perspective cameras can only capture a small portion of a very close object in the

FOV, and very close objects are usually not available in the training data of the source CNN. Therefore, even though 360° images offer a much wider FOV, spherical convolution inherits the limitations of the source model and may not recognize very close large objects. More generally speaking, SphConv and KTN ignores the domain shift in terms of the visual content, so the performance may be supotimal compared with a model that has been trained on massive 360° images.

Another limitation of KTN is that it only takes equirectangular projection as input. While equirectangular projection is the most popular format for 360° images and videos, there exist other formats for 360° video as introduced in Chapter 3. The current model is designed specifically for equirectangular projection and may not perform well on other formats such as cubemap projection, despite the fact that the definition of kernel transformer network is generic to the projection. A more generic and flexible architecture is necessary in order to apply KTN to other data formats.

Spherical convolution and kernel transformer network allow us to perform various CNN based visual analysis algorithms on 360° data easily. They enable many potential applications of 360° videos which require advanced computer vision algorithm in the pipeline. In the next Chapter, I will introduce one particular example application—a more effective method to display 360° video.

Chapter 6

Learning 360° Video Display

Having introduced an approach for 360° video compression and enabling accurate and efficient visual recognition on 360° data, we are now better prepared to build high level applications for 360° video. One of the most important applications for this new media is to display the video to human viewers, especially for general consumers. Therefore, the next challenge I tackle is 360° video display[1].

A 360° camera captures the entire visual world as observable from its optical center. The increased FOV of 360° cameras affords exciting new ways to record and experience visual content. A videographer no longer has to determine which direction to capture in the scene, freeing her to experience the moment rather than the act of recording a video. Meanwhile, a human video consumer has the freedom to explore the visual content based on her interest, without being severely restricted by choices made by the videographer. For

[1]The work in this chapter was originally published in: "Pano2Vid: Automatic Cinematography for Watching 360° Videos," in Proceedings of the Asian Conference on Computer Vision 2018 [108] by Yu-Chuan Su, Dinesh Jayaraman, and Kristen Grauman and "Making 360° Video Watchable in 2D: Learning Videography for Click Free Viewing," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition 2017 [105] by Yu-Chuan Su and Kristen Grauman.

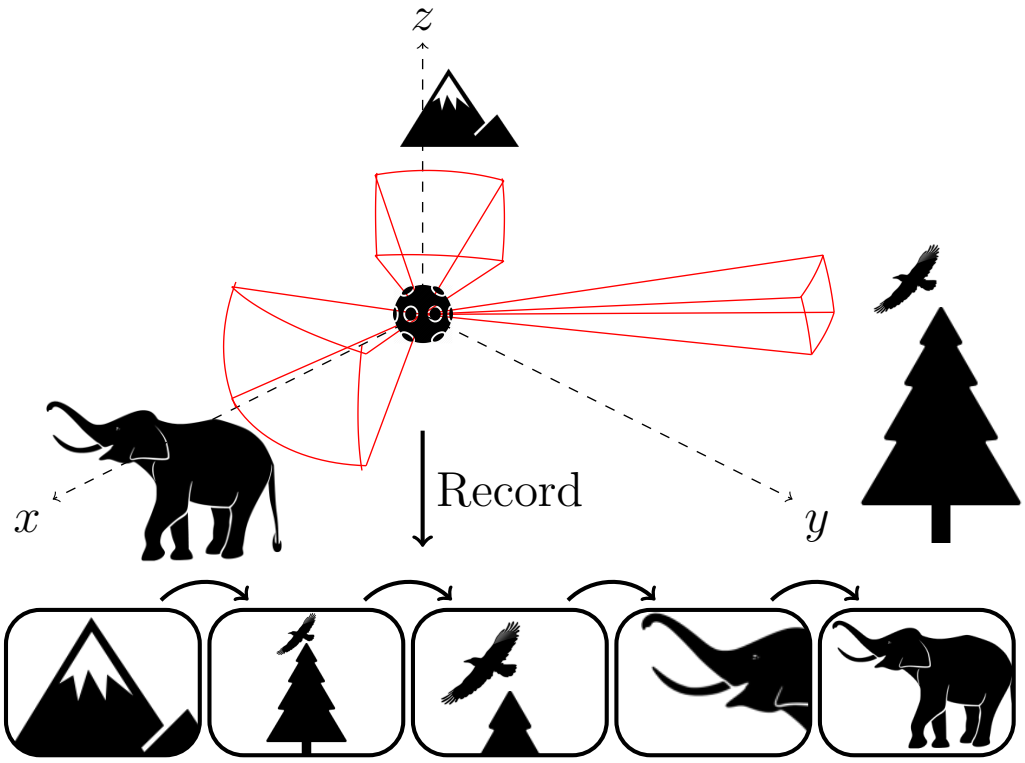


Figure 6.1: The goal of Pano2Vid is to control the direction and field of view of a virtual camera within a 360° video in order to record a video that looks as if it were captured by a human videographer.

example, a news correspondent can traverse a war zone without consciously considering how to portray the scene, and subsequent viewers will still have an immersive experience about the tragedy and witness events in more detail than the videographer may even be able to attend to in the moment.

On the other hand, the medium also introduces new challenges. Foremost, it largely transfers the choice of "where to look" from the videographer to the viewer. This makes 360° video hard to view effectively, since a human viewer must now somehow make the "where to look" choice and convey it to

a video player in real time. These choices determine the content seen by the viewer and thus the user experience. Because the viewer has no information about the content beyond the current FOV, it may be difficult to determine where to look, e.g., a 360° video viewer can easily fail to notice that there is something approaching the camera from the opposite direction. In fact, the viewer may have to watch the video multiple times in order to find a proper way to control the virtual camera that navigates through the content of interest. While 360° videos may alternatively be displayed in their entirety using equirectangular projection, the unfamiliar format and distortion make such video hard to watch.

To address this difficulty, I define "Pano2Vid", a new computer vision problem (see Fig 6.1) [105, 108]. The task is to design an algorithm to automatically control the pose, motion, and focal length of a virtual normal field-of-view (NFOV) camera within an input 360° video. The output of the system is the NFOV video captured by this virtual camera. Camera control must be optimized to produce video that could conceivably have been captured by a human observer equipped with a *real* NFOV camera. A successful Pano2Vid system would therefore take the burden of choosing "where to look" off both the videographer and the end viewer: the videographer could enjoy the moment without consciously directing her camera, while the viewer could watch intelligently-chosen portions of the video in the familiar NFOV format.

I propose the AutoCam algorithm to solve the Pano2Vid problem in a data driven approach that does not require human labor. The algorithm

first learns a discriminative model of human-captured NFOV web videos. The NFOV videos are crawled automatically from the web and do not need human annotations. It then uses this model to identify candidate viewpoints & focal lengths and events of interest to capture in the 360° video, before finally stitching them together through optimal camera motions for presentation to human viewers. I propose a dynamic programing based solution for the camera trajectory search, which generates a set of plausible output videos for each input 360° video. The trajectory search algorithm is designed to encourage the diversity between different output videos while reducing the overall computational cost. The users can therefore control what to see by choosing between different output videos.

In the rest of chapter, I will first provide a formal definition for the Pano2Vid problem (Sec. 6.1). I will then introduce our method (Sec. 6.2), before finally evaluating the proposed method on real 360° video (Sec. 6.3).

6.1 Pano2Vid Definition

We define the Pano2Vid task of automatic videography for 360° videos as follows. Given a dynamic panoramic 360° video, the goal is to produce "natural-looking" NFOV video. NFOV indicates the horizontal FOV of common zoom lens, i.e. from standard lens (46.4°) to ultra wide angle lens (104.3°).

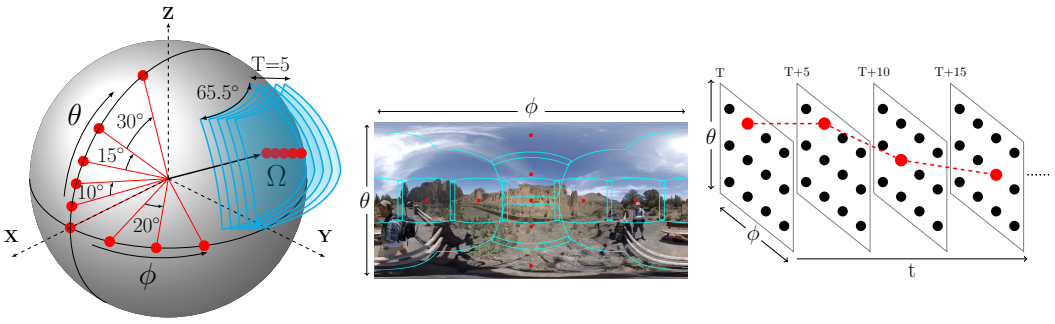
Broadly, a natural-looking NFOV video is one which is indistinguishable from human-captured NFOV video (henceforth "HumanCam"). Our ideal video output should be such that it conceivably could have been captured by

a human videographer equipped with an NFOV camera whose optical center coincides exactly with that of the 360° camera, with the objective of best presenting the event(s) in the scene. The NFOV video is difined by both the camera trajectory, i.e., the time sequence of the camera's principal axis directions, and focal length. To solve the Pano2Vid problem, a system must determine a NFOV camera trajectory through the 360° video to carve it into a HumanCam-like NFOV video.

6.2 Approach for Solving Pano2Vid

I now present AutoCam, my approach to solve the Pano2Vid task. The input to the system is an arbitrary 360° video, and the output is a natural looking NFOV video extracted from it.

AutoCam works in two steps. First, it evaluates all virtual NFOV spatio-temporal glimpses (ST-glimpses) sampled from the 360° video for their "capture-worthiness"—their likelihood of appearing in HumanCam NFOV video. Next, it selects virtual NFOV camera trajectories, prioritizing both 1) high-scoring ST-glimpses from the first step, and 2) smooth human-like camera movements. AutoCam is fully automatic and does not require any human input. Furthermore, as we will see next, the proposed learning approach is unsupervised—it learns a model of human-captured NFOV video simply by watching clips people upload to YouTube.



(a) Sample ST-glimpses and score capture-worthiness. (b) Stitch glimpses.

Figure 6.2: AutoCam first samples and scores the capture-worthiness of ST-glimpses. It then jointly selects a glimpse for each time step and stitches them together to form the output NFOV video. Best viewed in color.

6.2.1 Capture-worthiness Score

The first stage aims to find content that is likely to be captured by human videographers. We achieve this by scoring the capture-worthiness of candidate ST-glimpses sampled from the 360° video. An ST-glimpse is a five-second NFOV video clip recorded from the 360° video by directing the camera to a fixed direction in the 360° camera axes. One such glimpse is depicted as the blue stack of frame excerpts on the surface of the sphere in Fig 6.2a. These are not rectangular regions in the equirectangular projection (Fig 6.2a, right) so they are projected into NFOV videos before processing. We sample candidate ST-glimpses at 18 azimuthal angles and 11 polar angles every five seconds:

$$\begin{aligned} \theta \in \quad & \Theta = \{0, \pm 10, \pm 20, \pm 30, \pm 45, \pm 75\}, \\ \phi \in \quad & \Phi = \{0, 20, \ldots, 340\}, \\ t \in \quad & T = \{0s, 5s, \ldots, L - 5s\}, \end{aligned} \tag{6.1}$$

where L is the video length.

To achieve the effect of zooming, we further sample ST-glimpses with different focal lengths (f). Zooming is the technique of changing f of the lens, which is equivalent to changing the FOV of the camera because they are related by

$$\text{FOV} = 2\arctan(\frac{d}{2f}) \tag{6.2}$$

where d is the horizontal sensor size and is a constant for the camera. Assume the focal length for the 65.5° FOV is f^0, we sample ST-glimpses with three different focal lengths

$$f \in F = \{0.5f^0, f^0, 1.5f^0\}, \tag{6.3}$$

which results in FOV$\in\{104.3°, 65.5°, 46.4°\}$ respectively. The 104.3° FOV corresponds to an ultra wide angle lens and is the largest FOV commonly used in photography. The 65.5° and 46.4° FOV cover the range of standard lenses. Each candidate ST-glimpse is therefore defined by the camera principal axis (θ, ϕ) direction, focal length f, and time in the video t:

$$\Omega_{t,\theta,\phi,f} \equiv (\theta_t, \phi_t, f_t) \in \Theta \times \Phi \times F. \tag{6.4}$$

Our approach then learns to score capture-worthiness from HumanCam data. We expect capture-worthiness to rely on two main facets: content and composition. The *content* captured by human videographers is naturally very diverse. For example, in a mountain climbing video, people may consider capturing the recorder and his companion as well as a beautiful scene such as the sunrise as being equally important. Similarly, in a soccer video, a player

dribbling and a goalkeeper blocking the ball may both be capture-worthy. Our approach accounts for this diversity both by learning from a wide array of NFOV HumanCam clips and by targeting related domains via the keyword query data collection described above. The *composition* in HumanCam data is a meta-cue, largely independent of semantic content, that involves the framing effects chosen by a human videographer. For example, an ST-glimpse that captures only the bottom half of a human face is not capture-worthy, while a framing that captures the full face is; a composition for outdoor scenes may tend to put the horizon towards the middle of the frame, etc.

Rather than attempt to characterize capture-worthiness through rules, AutoCam *learns* a data-driven model. We make the following hypotheses: 1) the majority of content in HumanCam NFOV videos were considered capture-worthy by their respective videographers, and 2) most random ST-glimpses would *not* be capture-worthy. Based on these hypotheses, we train a capture-worthiness classifier. Specifically, we divide each HumanCam video into non-overlapping 5-second clips, to be used as positives, following 1) above. Next, *all* candidate ST-glimpses extracted from (disjoint) 360° videos are treated as negatives, per hypothesis 2) above. Due to the weak nature of this supervision, both positives and negatives may have some label noise.

To represent each ST-glimpse and each 5s HumanCam clip, we use off-the-shelf convolutional 3D features (C3D) [114]. C3D is a generic video feature based on 3D (spatial+temporal) convolution that captures appearance and motion information in a single vector representation, and is known

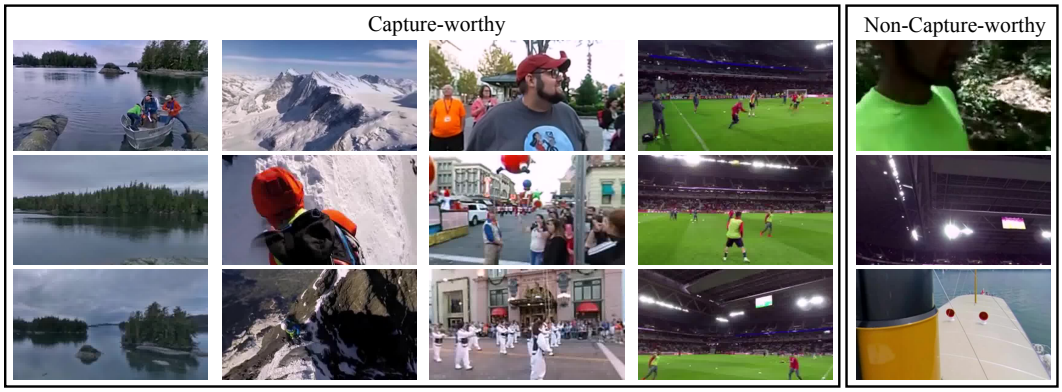


Figure 6.3: Example glimpses scored by AutoCam. Left 4 columns are glimpses considered capture-worthy by our method; each column is from the same time step in the same video. Right column shows non-capture-worthy glimpses.

to be useful for recognition tasks. We use a leave-one-video-out protocol to train one capture-worthiness classifier for each 360° video. Both the positive and negative training samples are from videos returned by the same keyword query term as the test video, and we sub-sample the 360° videos so that the total number of negatives is twice that of positives. We use logistic regression classifiers; positive class probability estimates of ST-glimpses from the left-out video are now treated as their capture-worthiness scores.

Fig 6.3 shows examples of "capture-worthy" and "non-capture-worthy" glimpses as predicted by our system. We see that there may be multiple capture-worthy glimpses at the same moment, and both the content and composition are important for capture-worthiness.

6.2.2 Trajectory Search

After obtaining the capture-worthiness score of each candidate ST-glimpse, we construct a camera trajectory by finding a path over the ST-glimpses that maximizes the *aggregate* capture-worthiness score, while simultaneously producing human-like smooth camera motions. A naive solution would be to choose the glimpse with the maximum score at each step. This trajectory would capture the maximum aggregate capture-worthiness, but the resultant NFOV video may have large/shaky unnatural camera motions. For example, when two ST-glimpses located in opposite directions on the viewing sphere have high capture-worthiness scores, such a naive solution would end up switching between these two directions at every time-step, producing unpleasant and even physically impossible camera movements.

Instead, to construct a trajectory with more human-like camera operation, we introduce a *smooth motion* prior when selecting the ST-glimpse at each time step. The prior prefers trajectories that are stable over those that jump abruptly between directions. For the example described above, the smooth prior would suppress trajectories that switch between the two directions constantly and promote those that focus on one direction for a longer amount of time. In practice, we realize the smooth motion prior by restricting the trajectory from choosing an ST-glimpse that is displaced from the previous ST-glimpse by more than a threshold ϵ in both longitude and latitude, i.e.

$$|\Delta\Omega|_\theta = |\theta_t - \theta_{t-1}| \leq \epsilon_\theta,\ |\Delta\Omega|_\phi = |\phi_t - \phi_{t-1}| \leq \epsilon_\phi. \tag{6.5}$$

We also restrict the change in focal length between consecutive ST-glimpses:

$$|\Delta\Omega|_f = |f_t - f_{t-1}| \leq 0.5 f^0, \tag{6.6}$$

which is a prior saying that human videographers tend to use gradual changes in zoom.

Given 1) the capture-worthiness scores of all candidate ST-glimpses and 2) the smoothness constraint in Eq. 6.5 and Eq. 6.6, we next introduce our algorithm for constructing camera trajectories.

Dynamic programming solution The problem of finding the trajectories with maximum aggregate capture-worthiness scores can be reduced to a shortest path problem. Let $C(\Omega_{t,\theta,\phi,f})$ be the capture-worthiness score of the ST-glimpse at time t with viewpoint (θ, ϕ) and focal length f. We construct a 3D lattice per time slice, where each node corresponds to an ST-glimpse at a given Ω. The edges in the lattice connect ST-glimpses from time step t to $t+1$, and the weight for an edge is defined by:

$$E\left(\Omega_{t,\theta,\phi,f}, \Omega_{t+1,\theta',\phi',f'}\right) = \begin{cases} -C(\Omega_{t+1,\theta',\phi',f'}), & |\Omega_{t,\theta,\phi,f} - \Omega_{t+1,\theta',\phi',f'}| \leq \epsilon \\ \infty, & \text{otherwise,} \end{cases} \tag{6.7}$$

where the difference above is shorthand for the smoothness constraint in Eq. 6.5 and Eq. 6.6.

The solution to the shortest path problem over this graph now corresponds to camera trajectories with maximum aggregate capture-worthiness. This solution can be efficiently computed using dynamic programming. See

Algorithm 1 Camera trajectory selection

$C \leftarrow$ Capture-worthiness scores
$\epsilon \leftarrow$ Valid camera motion
for all θ, ϕ, f **do**
  $Accum[\Omega_{1,\theta,\phi,f}] \leftarrow C[\Omega_{1,\theta,\phi,f}]$
end for
for $t \leftarrow 2, T$ **do**
  for all θ, ϕ, f **do**
    $\Omega_{t-1,\theta',\phi',f'} \leftarrow \arg\max_{\theta',\phi',f'} Accum[\Omega_{t-1,\theta',\phi',f'}]$
      $s.t.\ |\Omega_{t,\theta,\phi,f} - \Omega_{t-1,\theta',\phi',f}| \leq \epsilon$
    $Accum[\Omega_{t,\theta,\phi,f}] \leftarrow Accum[\Omega_{t-1,\theta',\phi',f'}] + C[\Omega_{t,\theta,\phi,f}]$
    $TraceBack[\Omega_{t,\theta,\phi,f}] \leftarrow \Omega_{t-1,\theta',\phi',f'}$
  end for
end for
$\Omega \leftarrow \arg\max_{\theta,\phi,f} Accum[\Omega_{T,\theta,\phi,f}]$
for $t \leftarrow T, 1$ **do**
  $Traj[t] \leftarrow \Omega$
  $\Omega \leftarrow TraceBack[\Omega]$
end for

pseudocode in Alg 1. At this point, the optimal trajectory indicated by this solution is "discrete" in the sense that it makes jumps between discrete directions after each 5-second time-step. To smooth over these jumps, we linearly interpolate the trajectories between the discrete time instants, so that the final camera motion trajectories output by AutoCam are continuous.

Coarse-to-fine solution While the dynamic programming solution can be solved efficiently, it requires that $C[\Omega]$ is known for all Ω. However, the computational bottleneck for the algorithm is in fact estimating the capture-worthiness score of each ST-glimpse because it adopts Strategy II discussed in Chapter 4 (Fig. 4.1). Therefore, it has to first render the 360° ST-glimpse

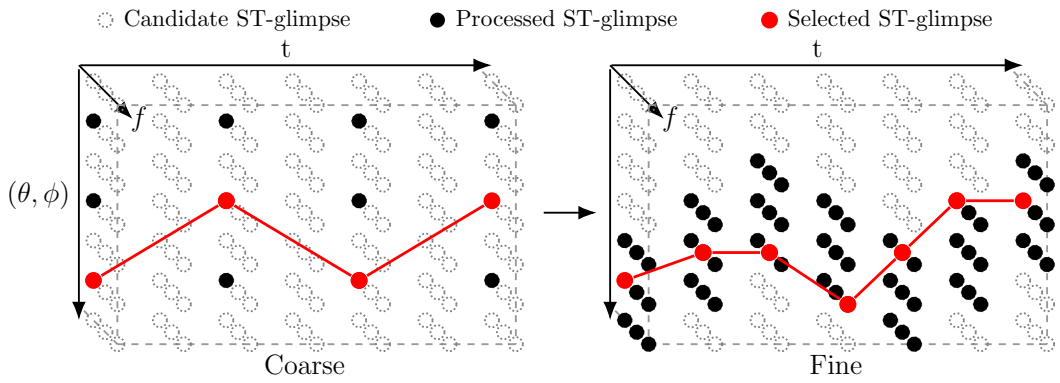


Figure 6.4: Coarse-to-fine camera trajectory search. We first construct the trajectory on a coarse sample of ST-glimpses and then refine it on a dense sample of ST-glimpses around the trajectory. It reduces computational cost by avoiding processing all candidate ST-glimpses.

into NFOV video and then extract the C3D feature, both of which are computationally intensive. Even if we assume we can render the NFOV video and extract C3D features in real time, the processing time would be orders of magnitude longer than the input video length due to the large number of candidate ST-glimpses, i.e. several hours to process even a 1 minute 360° video. The most straightforward solution would be to downsample the number of candidate ST-glimpses. However, this would lead to coarser camera control and a degradation in the quality of the output videos. In other words, the computational overhead not only makes the algorithm slow but it also restricts the granularity of virtual camera control.

Instead, we propose a coarse-to-fine approach that preserves the effective number of candidate ST-glimpses while reducing the number of ST-glimpses that require computation of the capture-worthiness score. The basic

idea is to first construct the trajectory over coarsely sampled ST-glimpses and then refine the solution over densely sampled ST-glimpses centered around the initial trajectory. See Fig. 6.4. Furthermore, to keep the total cost sublinear in the number of available focal lengths, we construct the coarse trajectory with a single focal length and enable zooming only when refining the solution. Because we process ST-glimpses densely only in a small portion of the video, the total number of capture-worthiness scores required by the algorithm decreases. The proposed coarse-to-fine approach is based on the observation that the capture-worthiness scores of neighbor ST-glimpses are positively correlated, and the optimal trajectory in densely sampled ST-glimpses leads to a candidate solution in coarsely sampled ST-glimpses. Although the solution is not guaranteed to be the same as that of dynamic programming over all candidate ST-glimpses, empirical results verify that it perform well.

We start the algorithm by sampling ST-glimpses at

$$\begin{aligned} \theta \in &\quad \Theta' = \{\pm 10, \pm 30, \pm 75\}, \\ \phi \in &\quad \Phi' = \{0, 40, \ldots, 320\}, \\ t \in &\quad T' = \{0s, 10s, \ldots, L - 10s\}, \\ f \in &\quad F' = \{0.5f^0\}. \end{aligned} \tag{6.8}$$

We use the focal length $f = 0.5f^0$ which corresponds to the largest FOV so the visual content of these initial ST-glimpses cover that of other focal lengths at the same direction. Eq. 6.8 downsamples the candidate ST-glimpses by a factor of two from Eq. 6.1, so the number of capture-worthiness scores that

need to be computed is only

$$\frac{|\Theta' \times \Phi' \times T'|}{|\Theta \times \Phi \times T|} \times \frac{1}{|F|} \approx 4.5\% \tag{6.9}$$

the number of candidate ST-glimpses. We solve for the trajectory using dynamic programming, but set the smoothness constraint to 2ϵ to account for the coarser samples

$$|\Delta\Omega|_\theta = |\theta_t - \theta_{t-1}| \leq 2\epsilon,\ |\Delta\Omega|_\phi = |\phi_t - \phi_{t-1}| \leq 2\epsilon. \tag{6.10}$$

Denoting the ST-glimpses selected by the trajectory as

$$\Omega^0_{t,\theta,\phi} \equiv (\theta^0_t, \phi^0_t) \tag{6.11}$$

for $t \in T'$, we then interpolate the ST-glimpses for $t \in T \setminus T' = \{5s, 15s, \ldots\}$ to obtain the full trajectory.

To refine the trajectory, we sample ST-glimpses

$$\Omega^1_{t,\theta,\phi,f} = (\theta^1_t, \phi^1_t, f^1_t) \tag{6.12}$$

that are adjacent to the original trajectory in direction

$$|\theta^1_t - \theta^0_t| \leq \epsilon_\theta,\ |\phi^1_t - \phi^0_t| \leq \epsilon_\phi \tag{6.13}$$

following Eq. 6.1 and 6.3. We then solve the same trajectory search problem over the sampled ST-glimpses using dynamic programming with the smoothness constraints in Eq. 6.5 and Eq. 6.6. The number of candidate ST-glimpses is greatly reduced by the adjacency constraint in Eq. 6.13 and is only 5% of all candidate ST-glimpses.

Diverse trajectory search I next introduce how to expand that single best solution to a set of *diverse* plausible outputs. The motivation to generate a diverse set of output videos stems from the fact that, by definition, there may be multiple valid Pano2Vid solutions for each 360° video. For example, one might capture a soccer game by tracking the ball or by focusing on a particular player. Both of them will lead to a plausible presentation for the game and should be a valid output. In fact, when we ask human editors to manually extract NFOV videos from 360° data, the outputs for any two editors on the same source video have only about 47% overlap on average. In addition, many applications of Pano2Vid would prefer a set of candidate solutions instead of a single output. An editing aid system would be more useful if the editor has the freedom to choose from different reasonable algorithm-provided initializations. Similarly, a 360° video player that allows the viewers to choose from different NFOV video presentations is likely to achieve a better user experience because the viewers can decide what to see based on their preferences.

It is difficult to encourage diversity in a single pass of dynamic programming, because all the potential solutions are constructed concurrently and the distances between them are hard to control. Instead, we generate trajectories iteratively and encourage diversity by imposing the minimum distance constraint between trajectories generated in different iterations. To realize the constraint, we sample a time window and forbid the trajectories of the current iteration from selecting the same ST-glimpses as the solution of previous iterations in the window. Therefore, the length of the time window determines the

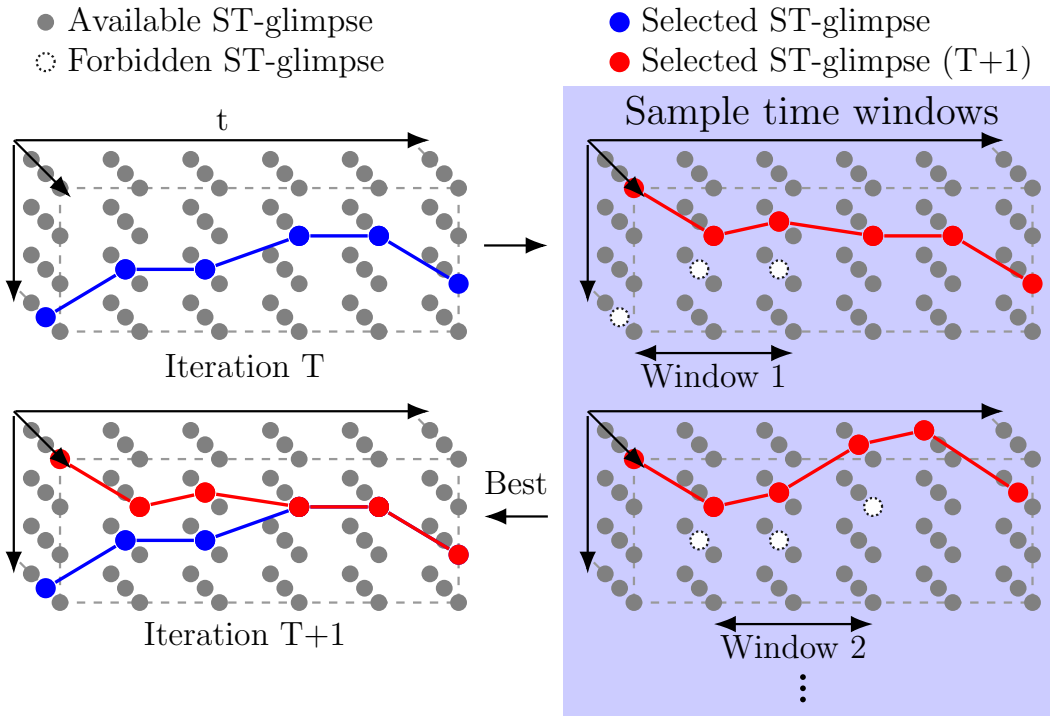


Figure 6.5: Diverse trajectory search generates trajectories iteratively. In each iteration, we construct multiple trajectory search problems by sampling time windows and removing previously selected ST-glimpses in the window from the search space. We solve all the problems and take the best solution as the output trajectory.

minimum distance between the solutions of different iterations. We sample the time window at multiple temporal locations and construct the optimal trajectory for each window. We take the best trajectory among them in terms of accumulated capture-worthiness score as the solution of the current iteration. This avoids critical glimpses being excluded from the solution space even if it is selected by previous trajectories. See Fig. 6.5.

In practice, we set the length of the time window to 10% the original

video length and sample 20 different windows distributed evenly over time. Once the length and location of the time window is specified, the optimal trajectory can be found using dynamic programming on a modified shortest path problem where the ST-glimpses selected by previous solutions in the window are removed from the search space. To improve computational efficiency, we divide the unit sphere into 6 regions (by 3 azimuthal angles and 2 polar angles) and generate an output per region by finding the best trajectory ending in the region. This leads to 6 trajectories per iteration. The iteration ends after K-trajectories have been generated.

6.3 Experiments

We evaluate the proposed AutoCam algorithm on unconstrained real world 360° videos. Because we are the first to study the Pano2Vid problem, we also design two sets of evaluation metrics and three baselines for the problem.

6.3.1 Dataset

360° videos We collect 360° videos from YouTube using the keywords "Soccer," "Mountain Climbing," "Parade," and "Hiking." These terms were selected to have 1) a large number of relevant 360° video results, 2) dynamic activity, i.e., spatio-temporal *events*, rather than just static scenes, and 3) possibly multiple regions/events of interest at the same time. For each query term, we download the top 100 videos sorted by relevance and filter out any that are not truly 360° videos (e.g., animations, slide shows of panoramas,

Figure 6.6: Examples frames from the Pano2Vid 360° video dataset. The videos are under the keywords "Soccer," "Mountain Climbing," "Parade," and "Hiking," for each row respectively.

restricted FOV) or have poor lighting, resolution, or stitching quality. This yields a Pano2Vid test set of 86 total 360° videos with a combined length of 7.3 hours. See Fig. 6.6 for example video frames. Note that this is the same

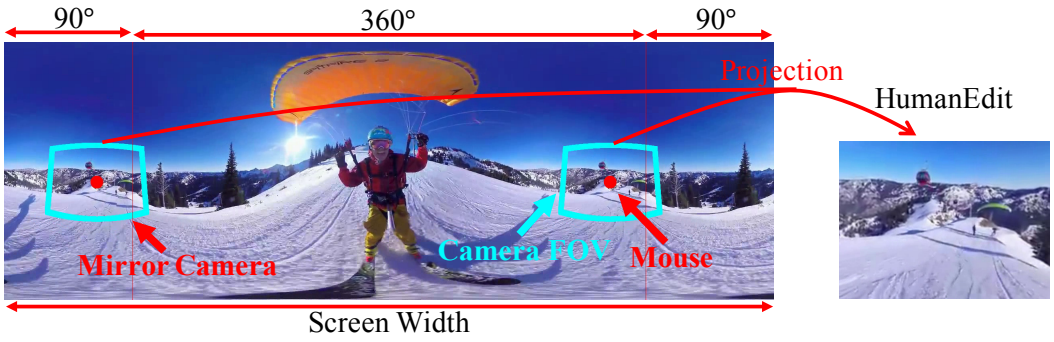


Figure 6.7: HumanEdit interface. We display the 360° video in equirectangular projection and ask annotators to direct the camera using the mouse. The NFOV video is rendered and displayed to the annotator offline. Best viewed in color.

Pano2Vid dataset used in Chapter 4.

HumanCam NFOV videos In both the learning stage of AutoCam (Sec 6.2.1) and the proposed evaluation methods (Sec. 6.3.2), we need a model for HumanCam. We collect a large diverse set of HumanCam NFOV videos from YouTube using the same query terms as above and imposing a per-video max length of 4 minutes. For each query term, we collect about 2,000 videos, yielding a final HumanCam set of 9,171 videos totalling 343 hours.

Human annotation collection To collect human editors' annotations, we ask multiple annotators to watch the 360° test videos and generate the camera trajectories from them. I next describe the annotation collection process.

Fig 6.7 shows the HumanEdit annotation interface. We display the entire 360° video in equirectangular projection. Annotators are instructed

to move a cursor to direct a virtual NFOV camera. Virtual NFOV frame boundaries are backprojected onto the display (shown in cyan) in real time as the camera is moved. The editors can also control the focal length of the virtual camera. The available focal lengths are the same as those available to the algorithm, and the interface will switch to the next available focal length when the editor presses the button for zoom in/out.

We design the interface to mitigate problems due to discontinuities at the edges. First, we extend the panoramic strip by 90° on the left and right as shown in Fig 6.7. The cursor may now smoothly move over the 360° boundaries to mimic camera motion in the real world. Second, when passing over these boundaries, content is duplicated, and so is the cursor position and frame boundary rendering. When passing over an edge of this extended strip, the cursor is repositioned to the duplicated position that is already on-screen by this time.

For each 360° video, we ask the editors to watch the full video in equirectangular projection first to familiarize themselves with the content. Next, we ask them to annotate *four* camera trajectories per video. For each of the four passes, we pan the panoramic strip by the angle of $[0°, 180°, 0°, 180°]$ to force the editors to consider the trajectories from different points of view. Finally, for the first two trajectories of the first two videos annotated by each editor, we render and show the output video to the editor right after the annotation to help him understand what the resulting video will look like. We collect HumanEdit data for 40 videos, each of them annotated by 3 editors.

Overall, we collect 480 trajectories totaling 717.2 minutes of video, and roughly 18 hours of annotation time.

6.3.2 Evaluation Metrics

Next I present evaluation metrics for the Pano2Vid problem. A good metric must measure how close a Pano2Vid algorithm's output videos are to human-generated NFOV video, while simultaneously being reproducible for easy benchmarking in future work. We devise two main criteria as described in the following two paragraphs.

HumanCam-based metrics The first criterion measures whether the output videos look like human-captured NFOV video (HumanCam). The more indistinguishable the algorithm outputs are from HumanCam, the better the algorithm. There are three metrics:

- **Distinguishability** quantifies if it is possible to tell the algorithm output apart from HumanCam. For a fully successful Pano2Vid algorithm, these sets would be entirely indistinguishable. This method can be considered as an automatic Turing test that is based on feature statistics instead of human perception; it is also motivated by the adversarial network framework [39] where the objective of the generative model is to disguise the discriminative model. We measure distinguishability using 5-fold cross validation performance of a discriminative classifier trained with HumanCam videos as positives, and algorithmically generated videos as negatives. Training and

testing negatives in each split are generated from disjoint sets of 360° video. *Higher* error is better.

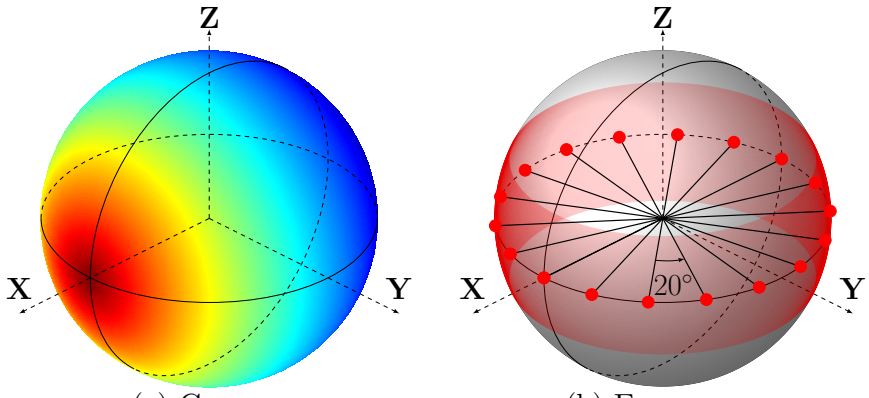
- **HumanCam-likeness** measures the relative distance from algorithm outputs to HumanCam data in a semantic feature space. Once again a classifier is trained on HumanCam videos as positives, but this time with *all* algorithm-generated videos as negatives. Similar to exemplar SVM [82], each algorithm-generated video is assigned a ranking based on its distance from the decision boundary (i.e. HumanCam-likeness), using a leave-one-360°-video-out training and testing scheme. We rank all Pano2Vid algorithms for each 360° video and compare their normalized mean rank; *lower* is better. We use classification score rather than raw feature distance because we are only interested in the factors that distinguish Pano2Vid and HumanCam. Since this metric depends on the relative comparison of all methods, it requires the output of all methods to be available during evaluation.

- **Transferability** measures how well semantic classifiers trained on HumanCam videos transfer to algorithm-generated videos, and vice versa. A similar method is used to evaluate automatic image colorization in [124]. We take the four search keywords as labels to learn a multi-class classifier on one domain and measure transferability using the test error on the other domain. The more transferable the classifiers are, the more similar the HumanCam and algorithm outputs are.

We use logistic regression classifiers in all metrics.

HumanEdit-based metrics Whereas the above metrics score output videos by their resemblance to human-captured videos in general, the HumanEdit metric measures the similarity between algorithm-generated camera trajectories and the manually created trajectories in the same 360° video. The more similar they are, the better the algorithm. This metric captures the subjective preference of human editors but is easily reproducible, in contrast to one-off user studies.

In particular, we compute the **overlap** of the camera FOV in each frame. The overlap is approximated by $\max(1 - \frac{2\Delta\Omega}{FOV_H + FOV_A}, 0)$, where FOV_H and FOV_A correspond to the FOV of algorithm and human controlled camera respectively. We report overlap results under two pooling strategies: 1) **Trajectory pooling**: Each Pano2Vid trajectory is compared to its best-matched HumanEdit trajectory. Frame-wise overlap to each human trajectory are first averaged. Each Pano2Vid output is then assigned a score corresponding to the minimum of its average overlap to HumanEdit trajectories. Trajectory pooling rewards Pano2Vid outputs that are similar to at least one HumanEdit trajectory *over the whole video*, and 2) **Frame pooling**: This pooling method rewards Pano2Vid outputs that are similar to different HumanEdit tracks in different portions of the video. First, each frame is assigned a score based on its minimum frame-wise overlap to a HumanEdit trajectory. Now, we simply average this over all frames to produce the "frame overlap" score for that trajectory. Frame pooling rewards Pano2Vid outputs that are similar to any HumanEdit trajectory at each frame.



(a) CENTER (b) EYE-LEVEL

Figure 6.8: Baseline illustration. CENTER generates random trajectories biased toward the 360° video center. EYE-LEVEL generates static trajectories on the equator.

6.3.3 Baselines

We compare the following methods in the experiments.

- SALIENCY — replace the capture-worthiness scores in AUTOCAM by saliency scores.[2] The saliency is computed by a popular method [44] over the 360° video frame in equirectangular projection.

- CENTER — random trajectories biased toward the "center" of the 360° video axes ($\theta = \phi = 0°$). The bias accounts for the fact that user-generated 360° videos often contain interesting content close to the center, possibly because the 360° camera design allows the users to use it as if it were a NFOV camera. We sample the camera direction for the next time-step from

[2]We also considered a saliency baseline that permits zoom like our method, but it fared worse than all others.

Table 6.1: Pano2Vid performance: HumanCam-based and HumanEdit-based metrics. The arrows in column 3 indicate whether lower scores are better (⇓), or higher scores (⇑). AutoCam significantly outperforms the baselines, and the relative improvements over the best performing baseline are up to 124.4%.

			Center	Eye-level	Saliency	AutoCam w/o Zoom	AutoCam
Distinguishability	Error rate (%)	⇑	1.93	4.03	7.70	12.05	**17.28**
HumanCam-Likeness	Mean Rank	⇓	0.659	0.707	0.612	0.522	**0.267**
Transferability	Human → Auto	⇑	0.582	**0.607**	0.597	0.517	0.591
	Auto → Human		0.526	0.552	0.549	0.584	**0.617**
Overlap	Trajectory	⇑	0.271	0.335	0.359	0.343	**0.442**
	Frame		0.498	0.555	0.580	0.530	**0.630**

a Gaussian centered around the current direction, which starts from the center. See Fig. 6.8a.

- Eye-level — static trajectories that place the virtual camera on the equator ($\theta = 0°$). The equator usually corresponds to eye-level in 360° video where most interesting events happen. We sample the azimuthal angle ϕ every 20° for 18 different camera directions. See Fig. 6.8b.

6.3.4 Results

Output quality First we quantify the quality of the algorithm-generated videos using the HumanCam-/HumanEdit-based metrics. We take the top K=20 outputs from each method.

Table 6.1 shows the results. AutoCam algorithm significantly outperforms all other methods. Compared to the best performing baseline (Saliency), it improves Distinguishability by 124.4% and ranks 34.5% better on average in the HumanCam-Likeness. We also see a 23% improvement in the Trajectory Overlap metric. Fig. 6.9 and 6.10 show qualitative examples. In particular,

Fig. 6.10 shows the importance of generating multiple trajectories in the algorithm. See our project webpage for example videos[3].

Note how the zooming capability improves the output video quality. The Distinguishability drops by 30% if we allow only a single focal length in the AutoCam algorithm. Interestingly, the camera zooms out (i.e. FOV>65.5°) more often than it zooms in. In fact, 76% of the ST-glimpses selected by AutoCam have 104.3° FOV, and editors select 104.3° FOV in 55% of the HumanEdit data. These results suggest that a larger FOV is preferable when viewing 360° videos, possibly because the object of interest is usually closer to the camera. We use 65.5° FOV for AutoCam w/o Zoom, which is the most common FOV for ordinary cameras.

Table 6.1 also shows that the Center and Eye-level baselines perform poorly, indicating that hand coded heuristics based on prior knowledge are not enough and a content-dependent method is necessary. Eye-level performs better than Center, which reflects the fact that Eye-level is a generic prior while the Center prior only holds when the 360° camera is asymmetric and the user uses it as an ordinary camera. Although Saliency is content-dependent, it captures content that attracts gaze, which appears to be a poor proxy for the Pano2Vid task. It underperforms AutoCam in all metrics except the Human → Auto transferability.

[3] `http://vision.cs.utexas.edu/projects/watchable360`

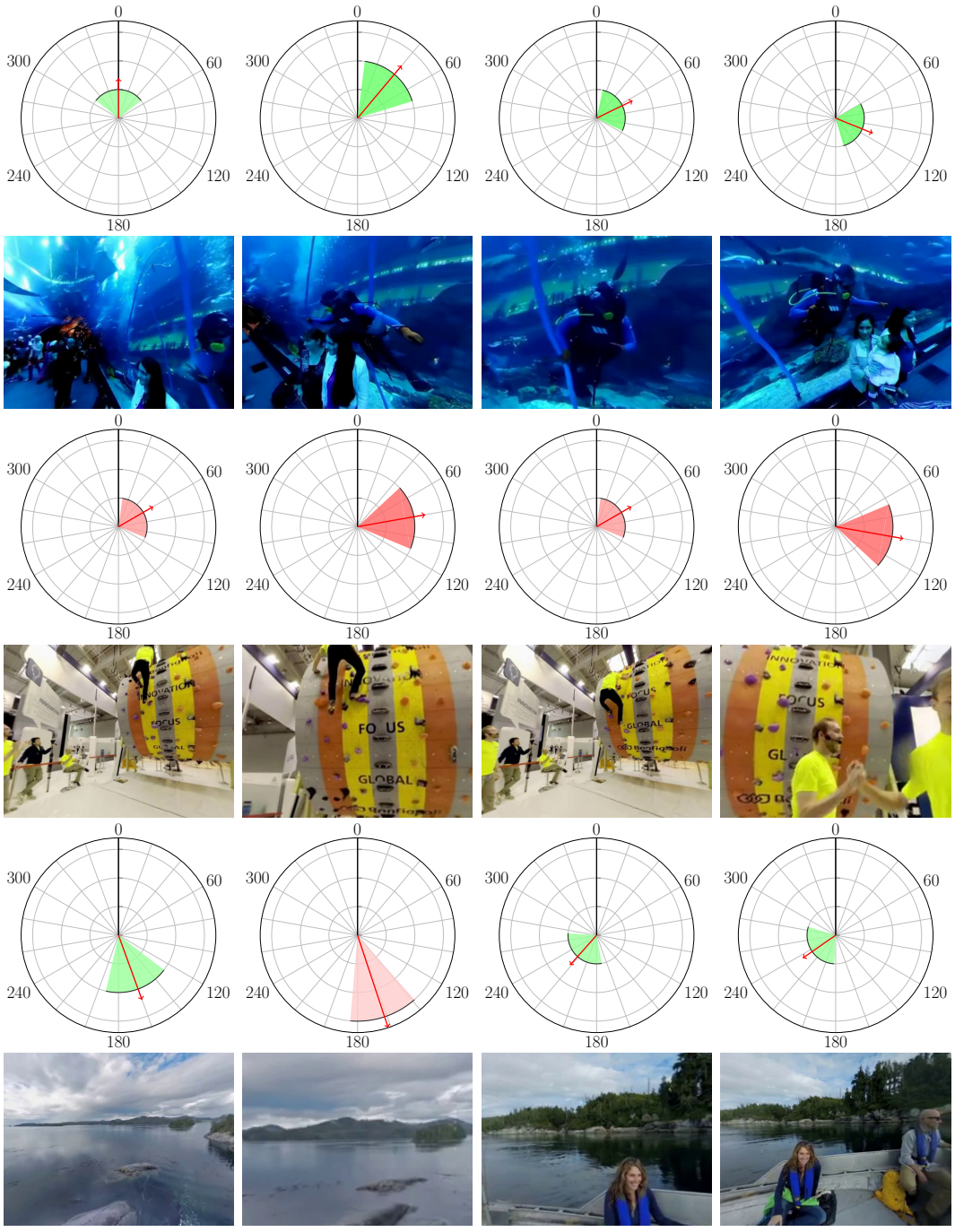


Figure 6.9: Example AutoCam outputs and the corresponding camera poses. The circular sector shows the camera FOV and azimuthal angle, and the color shows the polar angle. Red/green indicates the angle is greater/smaller than 0, and more saturated color indicates larger value.

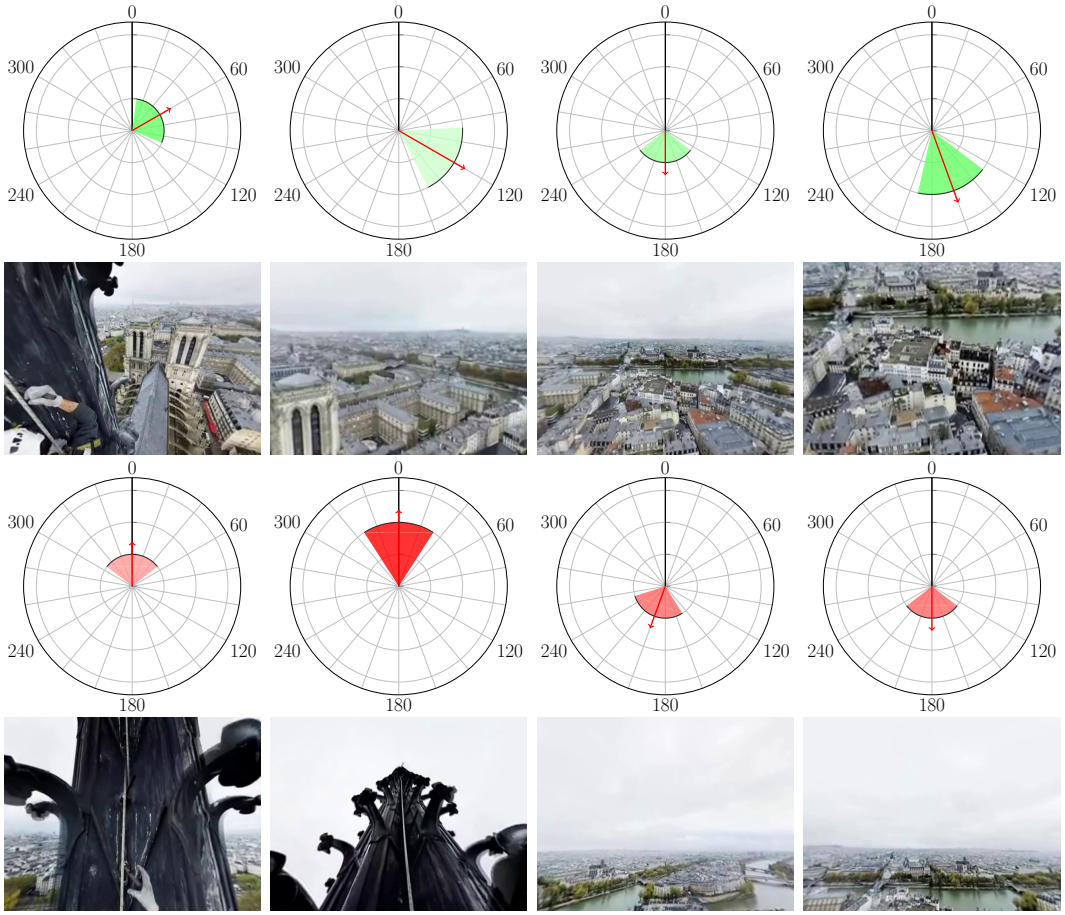


Figure 6.10: Two trajectories extracted from the same 360° video. Our algorithm presents the same scene in different manners.

Although AutoCam outperforms all baselines, we notice that the learned capture-worthiness is unable to capture preferences induced by context. For example, the algorithm fails to concentrate on family members in a family video. Also, the smoothness constraint may be too strong in some scenes where the camera is unable to adjust to rapidly changing content.

Table 6.2: HumanEdit consistency.

Overlap	Trajectory	0.560
	Frame	0.782

HumanEdit agreement We also measure the consistency between trajectories annotated by different annotators to provide a reference for how well the algorithms perform. Table 6.2 shows the results. AutoCam reaches 80% the overlap of inter annotator agreement, suggesting that it not only performs better than the baslines but also provides a reasonable coverage for content that captures annotators' attention. Still, there is still a significant room for improvement. Note that the average overlap between human trajectories is only 56%, while the frame overlap is much higher. These differences indicate that there is more than one natural trajectory for each 360° video, and different annotators may pick different trajectories. Still, with $> 78\%$ overlap at any given moment, we see that there is often something in the 360° video that catches everyone's eye; different trajectories arise because people choose to navigate through them in different manner. Overall, this analysis justifies our design to ask each annotator to annotate multiple trajectories and underscores the need for metrics that take the multiple trajectories into account, as we propose.

Computational cost Fig. 6.11 shows the computational cost versus output quality, measured by HumanCam Distinguishability and HumanEdit Trajectory Overlap. We measure computational cost by the average processing time

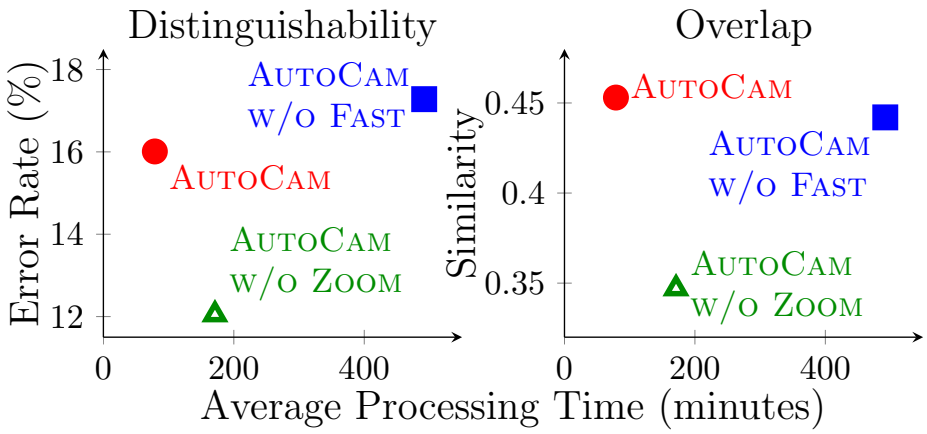


Figure 6.11: Computational cost vs. output quality for our method and AutoCam [108]. Computational cost is measured by the average processing time per 1 minute of input 360° video. Quality is measured by the Distinguishability and Trajectory Overlap; higher is better (⇑) for both metrics.

for 1 minute of 360° video. The time is measured on a machine equipped with Intel Xeon E5-2697 v2 processors (24 cores) and one GeForce GTX Titan Black GPU (including I/O). The coarse-to-fine trajectory search reduces the computational cost by 84% while performing similarly in Trajectory Overlap and only 6% worse in Distinguishability. Comparing the trajectories generated by the two methods, the coarse-to-fine approach tends to favor the largest FOV and ignore trajectories that select the smallest FOV throughout the video, because the initial trajectories are constructed with the largest FOV. This may cause the degradation in Distinguishability due to the fact that the 104.3° FOV introduces distortion in output frames. AutoCam takes less than 50% of the computation of AutoCam w/o Zoom yet is more accurate in all metrics. Although further optimization on the implementation may reduce the

processing time, the relative cost will remain the same, as it is linear in the number of ST-glimpses processed.

6.4 Conclusion

In this chapter, I proposed an approach for more effective 360° video display. The idea is to cast the FOV control problem as a virtual cinematography problem in 360° video, and I propose a data driven approach to solve the virtual cinematography problem. My proposed method generates proper NFOV videos from the raw 360° video like a human editor, so the viewers can watch the "edited" videos just as they would watch traditional NFOV videos.

One important factor ignored by the AutoCam algorithm is the interactive nature of 360° video display. The algorithm does not consider the situation where the user actively controls the FOV while watching the video. While effort has been made to improve interactive 360° video display with better human-computer interaction design [76, 88], these methods take static camera trajectories as reference and cannot consider the user control as feedback signal for better FOV control or suggestion. Also, the AutoCam algorithm assumes offline processing of the 360° video and does not work for an online 360° video stream. Further study is necessary in order to adapt the proposed method for different 360° video display settings, as will be discussed in Chapter 7.

Chapter 7

Future Work

In this chapter, I introduce future directions for my research on 360° media. My current research addresses existing challenges in 360° video processing and provides the basis for more advanced applications. Based on the results, I aim to explore the new possibilities enabled by the new media in the future. In particular, I believe it is important to take user interaction into consideration, because the most significant benefit of 360° video is the capability of providing a more interactive video watching experience.

7.1 User Aware 360° Video Compression

Video compression algorithms are most commonly evaluated with objective metrics that consider the quality over the entire video. However, the objective metrics do not always align with subjective metrics because users do not perceive the entire video equally. This is particularly true in 360° videos—viewers only watch a small portion of the video, and content beyond the FOV will not affect the perceived quality. Therefore, it is possible to achieve a better subjective quality by allocating higher bitrate to salient regions. The saliency may come directly from user interaction or from model prediction,

and it may depend on the specific application of the video. This suggests building a better compression algorithm that takes both the application and user interaction into consideration. I am interested in building a user aware compression algorithm for 360° videos that achieves a better subjective quality under the same bitrate.

7.2 Learning Automatic Cinematography from User Control

Existing algorithms for automatic cinematography usually rely on heuristics to encode popular idioms from cinematography [21, 28, 31, 47, 85, 110] and are applicable in limited scenes. AutoCam is the first to learn cinematographic tendencies from data. However, it learns from arbitrary web videos which provide only indirect and noisy supervision. In contrast, user interaction in 360° video provides a more direct supervision for automatic cinematography. It also provides richer information including saliency of visual content, cinematography idioms, and preference of users. I would like to explore the possibility of learning automatic cinematography from user control signals in 360° video in order to learn a better AutoCam model.

I am also interested in adapting AutoCam to the streaming and interactive scenarios. In the streaming scenario, the algorithm has no access to the video content beyond current time step. A Reinforcement Learning (RL) solution fits well in the streaming scenario, as RL models usually consider only observations made by the agent so far. Furthermore, an RL model has

the potential to learn to construct the camera trajectories and may perform better than a hand crafted dynamic programming solution. In the interactive scenario, the user may actively control the virtual camera to deviate from the optimal trajectory. I would like to study how to adapt the algorithm to consider the user interaction for better automatic cinematography.

Chapter 8

Conclusion

My thesis addresses several problems encountered during 360° video production. In particular, I study three fundamental challenges for 360° video—compression of 360° video, recognition on 360° imagery, and 360° video display. For 360° video compression, I propose to exploit the orientation of cubemap projection to improve the compression rate. My analysis shows that the orientation of the 360° video representation affects the compression algorithm, and I propose an efficient method to exploit the orientation for 360° video compression. The proposed method is compatible with existing codecs and ongoing progress for video compression, and the better compression rate allows us to store and distribute 360° video more easily. For recognition on 360° imagery, I propose a general approach to transfer CNN models trained on perspective images to 360° images. The proposed method allows accurate and efficient visual recognition on 360° images using off-the-shelf CNNs. Furthermore, because the proposed method does not require any many labor during training and is transferable across CNNs for different visual recognition tasks, it greatly reduce the cost for learning visual recognition models on 360° images. For 360° video display, I propose to cast the virtual camera control problem in 360° video player as a virtual videography problem. I further introduce a data

driven solution for the problem that can learn to control the virtual camera in 360° video from arbitrary human captured ordinary videos. The proposed method can transform a 360° video into easy-to-watch ordinary videos like a human editor automatically and allows users to watch 360° videos without additional effort. All together, the proposed solutions provide a more mature 360° video production pipeline. It will serve as the basis for various 360° video applications and allow both technology developers and content creators to further explore the possibility of the new media.

Bibliography

[1] Algorithm descriptions of projection format conversion and video quality metrics in 360lib. JVET-F1003, 2017.

[2] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Süsstrunk. Slic superpixels compared to state-of-the-art superpixel methods. *IEEE transactions on pattern analysis and machine intelligence*, 34(11):2274–2282, 2012.

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

[4] M. Akbari, J. Liang, and J. Han. Dsslic: Deep semantic segmentation-based layered image compression. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019.

[5] E. Alshina, K. Choi, V. Zakharchenko, S. N. Akula, A. Dsouza, C. Pujara, K. K. Ramkumaar, and A. Singh. Samsung's response to joint cfe on video compression with capability beyond hevc (360 category). JVET-G0025, 2017.

[6] I. Arev, H. S. Park, Y. Sheikh, J. Hodgins, and A. Shamir. Automatic editing of footage from multiple social cameras. In *SIGGRAPH*, 2014.

[7] Shai Avidan and Ariel Shamir. Seam carving for content-aware image resizing. In *ACM TOG*, 2007.

[8] 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.

[9] Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *NeurIPS*, 2014.

[10] Blu-ray Disc Association. White paper blu-ray disc read-only format coding constraints on hevc video streams for bd-rom version 3.0, June 2015.

[11] Wouter Boomsma and Jes Frellsen. Spherical convolutions and their application in molecular modelling. In *NeurIPS*, 2017.

[12] 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.

[13] Cristian Buciluǎ, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *ACM SIGKDD*, 2006.

[14] C-H. Chang, M-C. Hu, W-H. Cheng, and Y-Y. Chuang. Rectangling stereographic projection for wide-angle image visualization. In *ICCV*, 2013.

[15] 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.

[16] Z. Chen, T. He, X. Jin, and F. Wu. Learning for video compression. *IEEE Transactions on Circuits and Systems for Video Technology*, pages 1–1, 2019.

[17] 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.

[18] B. Choi, Y.-K. Wang, and Miska M. Hannuksela. Wd on iso/iec 23000-20 omnidirectional media application format. ISO/IEC JTC1/SC29/WG11, 2017.

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

[20] 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. *arXiv preprint arXiv:1711.08664*, 2017.

[21] David B Christianson, Sean E Anderson, Li-wei He, David H Salesin, Daniel S Weld, and Michael F Cohen. Declarative camera control for automatic cinematography. In *AAAI/IAAI, Vol. 1*, 1996.

[22] M. Coban, G. Van der Auwera, and M. Karczewicz. Qualcomms response to joint cfe in 360-degree video category. JVET-G0023, 2017.

[23] Taco Cohen, Mario Geiger, and Max Welling. Convolutional networks for spherical signals. *arXiv preprint arXiv:1709.04893*, 2017.

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

[25] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. *arXiv preprint arXiv:1703.06211*, 2017.

[26] K. Dale, E. Shechtman, S. Avidan, and H. Pfister. Multi-video browsing and summarization. In *CVPR*, 2012.

[27] J. Deng, W. Dong, R. Socher, L. Li, and L. Fei-Fei. Imagenet: a large-scale hierarchical image database. In *CVPR*, 2009.

[28] David K Elson and Mark O Riedl. A lightweight intelligent virtual cinematography system for machinima production. In *AIIDE*, 2007.

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

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

[31] Jonathan Foote and Don Kimber. Flycam: Practical panoramic video and automatic camera control. In *ICME*, 2000.

[32] Yanwei Fu, Yanwen Guo, Yanshu Zhu, Feng Liu, Chuanming Song, and Zhi-Hua Zhou. Multi-view video summarization. In *IEEE TOM*, 2010.

[33] A. Gabriel and E. Thomas. Polyphase subsampling applied to 360-degree video sequences in the context of the joint call for evidence on video compression. JVET-G0026, 2017.

[34] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014.

[35] Michael Gleicher and James Masanz. Towards virtual videography. In *Proceedings of the eighth ACM international conference on Multimedia*, 2000.

[36] Michael L Gleicher, Rachel M Heck, and Michael N Wallick. A framework for virtual videography. In *Proceedings of the 2nd international symposium on Smart graphics*, 2002.

[37] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 2010.

[38] Boqing Gong, Wei-Lun Chao, Kristen Grauman, and Fei Sha. Diverse sequential subset selection for supervised video summarization. In *NeurIPS*, 2014.

[39] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NeurIPS*, 2014.

[40] Saurabh Gupta, Judy Hoffman, and Jitendra Malik. Cross modal distillation for supervision transfer. In *CVPR*, 2016.

[41] Michael Gygli, Helmut Grabner, Hayko Riemenschneider, and Luc Van Gool. Creating summaries from user videos. In *ECCV*, 2014.

[42] Michael Gygli, Helmut Grabner, and Luc Van Gool. Video summarization by learning submodular mixtures of objectives. In *CVPR*, 2015.

[43] P. Hanhart, X. Xiu, F. Duanmu, Y. He, and Y. Ye. Interdigitals response to the 360 video category in joint call for evidence on video compression with capability beyond hevc. JVET-G0024, 2017.

[44] Jonathan Harel, Christof Koch, and Pietro Perona. Graph-based visual saliency. In *NeurIPS*, 2006.

[45] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *ICCV*, 2017.

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

[47] Li-wei He, Michael F Cohen, and David H Salesin. The virtual cinematographer: a paradigm for automatic real-time camera control and directing. In *ACM CGI*, 1996.

[48] Rachel Heck, Michael Wallick, and Michael Gleicher. Virtual videography. *ACM Trans. Multimedia Comput. Commun. Appl.*, 3(1), February 2007.

[49] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[50] 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.

[51] 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.

[52] L. Itti and P. Baldi. Bayesian surprise attracts human attention. *Vision Research*, 49(10):1295–1306, 2009.

[53] L. Itti, N. Dhavale, and F. Pighin. Realistic avatar eye and head animation using a neurobiological model of visual attention. In *Proc. SPIE*

48th Annual International Symposium on Optical Science and Technology, 2003.

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

[55] E. Jain, Y. Sheikh, A. Shamir, and J. Hodgins. Gaze-driven video re-editing. In *SIGGRAPH*, 2015.

[56] S. Jain, B. Xiong, and K. Grauman. Fusionseg: Learning to combine motion and appearance for fully automatic segmentation of generic objects in video. In *CVPR*, 2017.

[57] Yunho Jeon and Junmo Kim. Active convolution: Learning the shape of convolution for image classification. *arXiv preprint arXiv:1703.09076*, 2017.

[58] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *ACM MM*, 2014.

[59] Yushi Jing, David Liu, Dmitry Kislyuk, Andrew Zhai, Jiajing Xu, Jeff Donahue, and Sarah Tavel. Visual search at pinterest. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2015.

[60] Nick Johnston, Damien Vincent, David Minnen, Michele Covell, Saurabh Singh, Troy Chinen, Sung Jin Hwang, Joel Shor, and George Toderici. Improved lossy image compression with priming and spatially adaptive bit rates for recurrent networks. *arXiv preprint arXiv:1703.10114*, 2017.

[61] Seunghyun Cho Jooyoung Lee and Seung-Kwon Beack. Context-adaptive entropy model for end-to-end optimized image compression. In *ICLR*, 2019.

[62] Renata Khasanova and Pascal Frossard. Graph-based classification of omnidirectional images. *arXiv preprint arXiv:1707.08301*, 2017.

[63] Paween Khoenkaw and Punpiti Piamsa-Nga. Automatic pan-and-scan algorithm for heterogeneous displays. In *Springer MTA*, 2015.

[64] Aditya Khosla, Raffay Hamid, Chih-Jen Lin, and Neel Sundaresan. Large-scale video summarization using web-image priors. In *CVPR*, 2013.

[65] Y. Kim, C-R. Lee, D-Y. Cho, Y. Kwon, H-J. Choi, and K-J. Yoon. Automatic content-aware projection for 360° videos. In *ICCV*, 2017.

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

[67] Jan P. Klopp, Yu-Chiang Frank Wang, Shao-Yi Chien, and Liang-Gee Chen. Learning a code-space predictor by exploiting intra-image-dependencies. In *BMVC*, 2018.

[68] Philipp Krähenbühl, Manuel Lang, Alexander Hornung, and Markus Gross. A system for retargeting of streaming video. In *ACM TOG*, 2009.

[69] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, 2012.

[70] 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.

[71] 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.

[72] 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.

[73] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Proc. of the IEEE*, 1998.

[74] Yong Jae Lee, Joydeep Ghosh, and Kristen Grauman. Discovering important people and objects for egocentric video summarization. In *CVPR*, 2012.

[75] Mu Li, Wangmeng Zuo, Shuhang Gu, Debin Zhao, and David Zhang. Learning convolutional networks for content-weighted image compression. *arXiv preprint arXiv:1703.10553*, 2017.

[76] 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.

[77] Dong Liu, Haichuan Ma, Zhiwei Xiong, and Feng Wu. Cnn-based dct-like transform for image compression. In *MMM*, 2018.

[78] Feng Liu and Michael Gleicher. Video retargeting: automating pan and scan. In *ACM MM*, 2006.

[79] Tie Liu, Zejian Yuan, Jian Sun, Jingdong Wang, Nanning Zheng, Xiaoou Tang, and Heung-Yeung Shum. Learning to detect a salient object. In *PAMI*, 2011.

[80] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015.

[81] Guo Lu, Wanli Ouyang, Dong Xu, Xiaoyun Zhang, Chunlei Cai, and Zhiyong Gao. Dvc: An end-to-end deep video compression framework. In *CVPR*, 2019.

[82] Tomasz Malisiewicz, Abhinav Gupta, and Alexei A. Efros. Ensemble of exemplar-svms for object detection and beyond. In *ICCV*, 2011.

[83] Fabian Mentzer, Eirikur Agustsson, Michael Tschannen, Radu Timofte, and Luc Van Gool. Practical full resolution learned lossless image compression. In *CVPR*, 2019.

[84] Fabian Mentzer, Eirikur Agustsson, Michael Tschannen, Radu Timofte, and Luc Van Gool. Conditional probability models for deep image compression. In *CVPR*, 2018.

[85] Peter Mindek, Ladislav Čmolík, Ivan Viola, Eduard Gröller, and Stefan Bruckner. Automatized summarization of multiplayer games. In *ACM CCG*, 2015.

[86] Moving Picture Experts Group. Point cloud compression mpeg evaluates responses to call for proposal and kicks off its technical work [press release]. `https://mpeg.chiariglione.org/meetings/120`, October 2017.

[87] Emilio Parisotto, Jimmy Ba, and Ruslan Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning. In *ICLR*, 2016.

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

[89] Danila Potapov, Matthijs Douze, Zaid Harchaoui, and Cordelia Schmid. Category-specific video summarization. In *ECCV*, 2014.

[90] K. R. Rao, Do Nyeon Kim, and Jae Jeong Hwang. *Video Coding Standards: AVS China, H.264/MPEG-4 PART 10, HEVC, VP6, DIRAC and VC-1*. Springer Netherlands, 2014.

[91] A. Rav-Acha, Y. Pritch, and S. Peleg. Making a long video short: Dynamic video synopsis. In *CVPR*, 2006.

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

[93] Oren Rippel and Lubomir Bourdev. Real-time adaptive image compression. In *ICML*, 2017.

[94] Oren Rippel, Sanjay Nair, Carissa Lew, Steve Branson, Alexander G Anderson, and Lubomir Bourdev. Learned video compression. *arXiv preprint arXiv:1811.06981*, 2018.

[95] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. In *ICLR*, 2015.

[96] Michael Rubinstein, Ariel Shamir, and Shai Avidan. Improved seam carving for video retargeting. In *ACM TOG*, 2008.

[97] D. Rudoy, D.B. Goldman, E. Shechtman, and L. Zelnik-Manor. Learning video saliency from human gaze using candidate selection. In *CVPR*, 2013.

[98] Yong Rui, Anoop Gupta, Jonathan Grudin, and Liwei He. Automating lecture capture and broadcast: technology and videography. *Multimedia Systems*, 10(1):3–15, 2004.

[99] Shibani Santurkar, David Budden, and Nir Shavit. Generative compression. *arXiv preprint arXiv:1703.01467*, 2017.

[100] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In *Advances in neural information processing systems*, pages 568–576, 2014.

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

[102] John Parr Snyder. *Map projections–A working manual*, volume 1395. US Government Printing Office, 1987.

[103] K. K. Sreedhar, A. Aminlou, M. M. Hannuksela, and M. Gabbouj. Viewport-adaptive encoding and streaming of 360-degree video for virtual reality applications. In *IEEE ISM*, 2016.

[104] Yu-Chuan Su and Kristen Grauman. Learning spherical convolution for fast features from 360° imagery. In *NeurIPS*, 2017.

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

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

[107] Yu-Chuan Su and Kristen Grauman. Kernel transformer networks for compact spherical convolution. In *CVPR*, 2019.

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

[109] Min Sun, Ali Farhadi, and Steve Seitz. Ranking domain-specific highlights by analyzing edited videos. In *ECCV*, 2014.

[110] Xinding Sun, Jonathan Foote, Don Kimber, and BS Manjunath. Region of interest extraction and virtual camera control based on panoramic video capturing. In *IEEE TOM*, 2005.

[111] Y. Snchez, R. Skupin, and T. Schierl. Compressed domain video processing for tile based panoramic streaming using hevc. In *ICIP*, 2015.

[112] George Toderici, Sean M O'Malley, Sung Jin Hwang, Damien Vincent, David Minnen, Shumeet Baluja, Michele Covell, and Rahul Sukthankar. Variable rate image compression with recurrent neural networks. In *ICLR*, 2016.

[113] George Toderici, Damien Vincent, Nick Johnston, Sung Jin Hwang, David Minnen, Joel Shor, and Michele Covell. Full resolution image compression with recurrent neural networks. In *CVPR*, 2017.

[114] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri. Learning spatiotemporal features with 3d convolutional networks. In *ICCV*, 2015.

[115] 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.

[116] L. Wang, A. Fiandrotti, A. Purica, G. Valenzise, and M. Cagnazzo. Enhancing hevc spatial prediction by context-based learning. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019.

[117] M. Wien, V. Baroncini, J. Boyce, A. Segall, and T. Suzuki. Joint call for evidence on video compression with capability beyond hevc. JVET-F1002, 2017.

[118] Chao-Yuan Wu, Nayan Singhal, and Philipp Krähenbühl. Video compression through image interpolation. In *ECCV*, 2018.

[119] Jianxiong Xiao, Krista A Ehinger, Aude Oliva, and Antonio Torralba. Recognizing scene viewpoint using panoramic place representation. In *CVPR*, 2012.

[120] Bo Xiong and Kristen Grauman. Detecting snap points in egocentric video with a web photo prior. In *ECCV*, 2014.

[121] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.

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

[123] Yun Zhai and Mubarak Shah. Visual attention detection in video sequences using spatiotemporal cues. In *ACM MM*, 2006.

[124] Richard Zhang, Phillip Isola, and Alexei A Efros. Colorful image colorization. In *arXiv preprint arXiv:1603.08511*, 2016.

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

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

[127] Bolei Zhou, Agata Lapedriza, Jianxiong Xiao, Antonio Torralba, and Aude Oliva. Learning deep features for scene recognition using places database. In *NeurIPS*, 2014.

Vita

Yu-Chuan Su was born in Taipei, Taiwan. He received his high school diploma from Taipei Municipal Jianguo High School in 2006. He then entered National Taiwan University, where he received the Bachelor of Science degree in Physics and Computer Science & Information Engineering in 2011. After serving for a year in Taiwan Navy, he returned to National Taiwan University and received the Master of Science degree in Computer Science & Information Engineering in 2014 (advisor: Prof. Winston Hsu). In 2014, he entered the Graduate School at the University of Texas at Austin and joined the Computer Vision Group headed by Prof. Kristen Grauman as a graduate research assistant. He is the recipient of the Google PhD fellowship in 2017, IPPR best thesis award in 2015, and the best application paper award in ACCV 2016.

Permanent contact: ycsu@utexas.edu

This dissertation was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.