Deep Neural Networks for 3D Processing and High-Dimensional Filtering

Hang Su
University of Massachusetts Amherst

DEEP NEURAL NETWORKS FOR 3D PROCESSING AND HIGH-DIMENSIONAL FILTERING

A Dissertation Presented

by

HANG SU

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 2020

College of Information and Computer Sciences

DEEP NEURAL NETWORKS FOR 3D PROCESSING AND HIGH-DIMENSIONAL FILTERING

A Dissertation Presented

by

HANG SU

Approved as to style and content by:

Erik Learned-Miller, Chair

Subhransu Maji, Member

Evangelos Kalogerakis, Member

Patrick Flaherty, Outside Member

James Allan, Chair of the Faculty
College of Information and Computer Sciences

ACKNOWLEDGMENTS

My Ph.D. is a six-and-a-half-year journey in which I encountered and had close interactions with many of the most important people in my life. To all of you supported me through this journey, including those highlighted below and many others, I would like to express my most sincere gratitude.

First, I would like to extend thanks to my advisor Prof. Erik Learned-Miller for the impeccable mentoring and guidance I received from him. He taught me the fundamental elements of conducting proper research and how to recognize and propose original research ideas. I would like to thank Prof. Subhransu Maji and Prof. Vangelis Kalogerakis for their contributions to the many pieces of work included in this dissertation. It has always been a pleasure having conversations with them, getting inspiring and insightful comments and ideas. I would also like to thank my outside committee member Prof. Patrick Flaherty for giving thoughtful feedback from a fresh perspective that contributed to the final outcome of this dissertation.

Next, I would like to thank my internship mentors at NVIDIA Research, Varun Jampani and Deqing Sun. The two summers with them were delightful and rewarding. They taught me the art of conducting research in an industry setting, which helps me significantly in planning my career path. I am indebted to both of them.

Finally, I would like to thank my parents and my wife. After years of continuous and unconditional trust and support, it is easy to take them for granted, but I know I can't. Pursuing a Ph.D. is, in many ways, a long-term investment, and they give me all the comfort and encouragement I need to stick to the plan and reap the final rewards. They truly define what family means. Thank you!

ABSTRACT

DEEP NEURAL NETWORKS FOR 3D PROCESSING AND HIGH-DIMENSIONAL FILTERING

MAY 2020

HANG SU, B.Sc., PEKING UNIVERSITY

Sc.M., BROWN UNIVERSITY

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Erik Learned-Miller

Deep neural networks (DNN) have seen tremendous success in the past few years, advancing state of the art in many AI areas by significant margins. Part of the success can be attributed to the wide adoption of convolutional filters. These filters can effectively capture the invariance in data, leading to faster training and more compact representations, and at the same can leverage efficient parallel implementations on modern hardware. Since convolution operates on regularly structured grids, it is a particularly good fit for texts and images where there are inherent rigid 1D or 2D structures. However, extending DNNs to 3D or higher-dimensional spaces is non-trivial, because data in such spaces often lack regular structure, and the curse of dimensionality can also adversely impact performance in multiple ways.

In this dissertation, we present several new types of neural network operations and architectures for data in 3D and higher-dimensional spaces and demonstrate how we can mitigate these issues while retaining the favorable properties of 2D convolutions.

First, we investigate view-based representations for 3D shape recognition. We show that a collection of 2D views can be highly informative, and we can adapt standard 2D DNNs with a simple pooling strategy to recognize objects based on their appearances from multiple viewing angles with unprecedented accuracies. Our next study makes a connection between 3D point cloud processing and sparse high-dimensional filtering. The resulting representation is highly efficient and flexible, and enables native 3D operations as well as joint 2D-3D reasoning. Finally, we show that high-dimensional filtering is also a powerful tool for content-adaptive image filtering. We demonstrate its utility in computer vision applications where preserving sharp details in output is critical, including joint upsampling and semantic segmentation.

TABLE OF CONTENTS

Page

ACKNOWLEDGMENTS . . . . **iv**

ABSTRACT . . . . **v**

LIST OF TABLES . . . . **xi**

LIST OF FIGURES . . . . **xiii**

CHAPTER

1. INTRODUCTION **1**

1.1 Deep Neural Networks 3
1.2 Multi-View Representations 4
1.3 Representation Learning for Point Clouds 6
1.4 Content-Adaptive Convolutions 8
1.5 Dissertation Outline 12

2. LITERATURE REVIEW **14**

2.1 Convolutional Neural Networks 14
2.2 3D Shape Descriptors 15
2.3 Deep Neural Networks for 3D Data 16

2.3.1 Voxel networks 16
2.3.2 Point cloud networks 17
2.3.3 Non-Euclidean networks 18

3. 3D SHAPE RECOGNITION WITH MULTI-VIEW CONVOLUTIONAL NEURAL NETWORKS **20**

3.1 Multi-View Convolutional Neural Networks 20

3.1.1 Multi-view input representations 22
3.1.2 Recognition with multi-view representations 23

3.1.2.1 Image descriptors 23
3.1.2.2 3D shape classification 24
3.1.2.3 3D shape retrieval 24

3.1.3 Learning to aggregate views with Multi-View CNN 25
3.1.4 Low-rank Mahalanobis metric 26

3.2 Experiments 26

3.2.1 3D shape classification and retrieval 26

3.2.1.1 ModelNet40 26
3.2.1.2 ShapeNetCore55 30

3.2.2 Sketch recognition: jittering revisited 32
3.2.3 Sketch-based 3D shape retrieval 35

3.3 Conclusions 36

4. 3D POINT CLOUD PROCESSING WITH SPARSE LATTICE NETWORKS 42

4.1 Bilateral Convolution Layer 42

4.1.1 Inputs to BCL 43
4.1.2 Processing steps in BCL 44
4.1.3 Point cloud density normalization 45
4.1.4 Properties of BCL 45

4.2 Sparse Lattice Networks 46

4.2.1 3D point cloud processing with SPLATNet$_{\text{3D}}$ 47

4.2.1.1 Network architecture. 48
4.2.1.2 Lattice spaces and their scales. 49

4.2.2 Joint 2D-3D processing with SPLATNet$_{\text{2D-3D}}$ 50

4.3 Experiments 53

4.3.1 RueMonge2014 facade segmentation 53

4.3.1.1 Dataset 54
4.3.1.2 Implementation details 54
4.3.1.3 Results on point cloud labeling 56
4.3.1.4 Results on multi-view image labeling 56

4.3.2 ShapeNet part segmentation 56

4.3.2.1 Dataset 57
4.3.2.2 Implementation details 58
4.3.2.3 Results on 3D object part segmentation 59
4.3.2.4 Filtering in higher-dimensional spaces 62
4.3.2.5 Limitations with the dataset 62

4.4 Conclusions 63

5. PIXEL-ADAPTIVE CONVOLUTIONAL NEURAL NETWORKS 65

5.1 Related Work 68

5.1.1 Content-adaptive filtering 68
5.1.2 Dynamic filter networks 69
5.1.3 Self-attention mechanism 70

5.2 Pixel-Adaptive Convolution 70

5.2.1 Extending spatial convolution to higher-dimensions 70
5.2.2 Pixel-Adaptive Convolution 72
5.2.3 PAC as a generalization of other techniques 72
5.2.4 Variants of PAC 74

5.2.4.1 Transposed PAC 74
5.2.4.2 PAC on sparse inputs 74

5.2.5 Implementation 76

5.3 Joint Upsampling 76

5.3.1 Deep joint upsampling with PAC 76
5.3.2 Joint depth upsampling 78
5.3.3 Joint optical flow upsampling 79

5.4 Conditional Random Fields 81

5.4.1 Fully-connected CRFs 82
5.4.2 Efficient, learnable CRFs with PAC 84

5.4.2.1 PAC-CRF 84
5.4.2.2 Inference 85
5.4.2.3 Interpretation of the PAC-CRF formulation 86
5.4.2.4 Long-range connections with dilated PAC 88

5.4.3 Semantic segmentation with PAC-CRF 88

5.4.3.1 Dataset 89
5.4.3.2 Baselines 89
5.4.3.3 Results 90
5.4.3.4 Limitations 91

5.5 Layer Hot-Swapping with PAC 92

5.5.1 Layer hot-swapping 92
5.5.2 Semantic segmentation with layer hot-swapping 93

5.6 Conclusions 94

6. CONCLUSIONS 95

6.1 Future Directions 96

BIBLIOGRAPHY 98

LIST OF TABLES

Table **Page**

3.1 Classification and retrieval results on the ModelNet40 dataset. On the top are results using state-of-the-art 3D shape descriptors. Our view-based descriptors including Fisher vectors (FV) significantly outperform these even when a single view is available at test time (#Views = 1). When multiple views (#Views=12 or 80) are available at test time, the performance of view-based methods improve significantly. The MVCNN architecture outperforms the view-based methods, especially for retrieval. 27

3.2 Classification results on SketchClean. Fine-tuned CNN models significantly outperform Fisher Vectors [106] by a significant margin. MVCNNs are better than CNN trained with data jittering. The results are shown with two different CNN architectures – VGG-M (row 2-5) and VGG-VD (row 6-9). 33

4.1 Results on facade segmentation. Average IoU scores and approximate runtimes for point cloud labeling and 2D image labeling using different techniques. Runtimes indicate the time taken to segment the entire test data (202 images sequentially for 2D and a point cloud for 3D). 53

4.2 Results on ShapeNet part segmentation showing class average mIoU, instance average mIoU and mIoU scores for all the categories on the task of point cloud labeling using different techniques. 61

5.1 Network architectures for joint depth upsampling. "C" stands for regular CONV, "P" stands for PAC$^{\mathsf{T}}$ (the transposed convolution variant of PAC), and the number after them represents the number of output channels. 78

5.2 Joint depth upsampling. Results (in RMSE) show that our upsampling network consistently outperforms other techniques for different upsampling factors. 80

5.3 Joint optical flow upsampling. End-Point-Error (EPE) showing the improved performance compared with DJF [74]. 81

5.4 Semantic segmentation with PAC-CRF. Validation and test mIoU scores along with the runtimes of different techniques. PAC-CRF results in better improvements than Full-CRF [69] while being faster. PAC-CRF also outperforms Conv-CRF [119] and BCL [57]. Runtimes are averaged over all validation images. 90

5.5 Validation mIoU when using different number of mean-field update steps in PAC-CRF. 91

5.6 FCN hot-swapping CONV with PAC. Validation and test mIoU scores along with runtimes of different techniques. Our simple hot-swapping strategy provides 2 IoU gain on test. Combining with PAC-CRF offers additional improvements. 94

LIST OF FIGURES

Figure **Page**

1.1 Various types of 3D data representations. From left to right: polygon mesh, voxels, point cloud, 2D rendering, depth map. 2

1.2 From point clouds and images to semantics. SPLATNet$_{3D}$ directly takes point cloud as input and predicts labels for each point. SPLATNet$_{2D\text{-}3D}$, on the other hand, jointly processes both point cloud and the corresponding multi-view images for better 2D and 3D predictions. 8

1.3 A same scene under two different weather conditions can benefit from convolutional filters adaptive to the conditions. 9

3.1 Multi-View CNN for 3D shape recognition (illustrated using the 1st camera setup). At test time a 3D shape is rendered from 12 different views and are passed thorough CNN$_1$ to extract view based features. These are then pooled across views and passed through CNN$_2$ to obtain a compact shape descriptor. 21

3.2 Precision-recall curves for various methods for 3D shape retrieval on the ModelNet40 dataset. Our method significantly outperforms the state-of-the-art on this task achieving 80.2% mAP. 28

3.3 Confusion matrix of ModelNet40 classification. 38

3.4 Saliency maps on samples from ModelNet. Top three views with the highest saliency are highlighted in blue and the relative magnitude of gradient energy for each view is shown on top. The saliency maps are computed by back-propagating the gradients of the class score onto the images via the view-pooling layer. Notice that the handles of the dresser are the most discriminative features. (Figures are contrast enhanced for visibility.) 39

3.5 Examples of correctly and wrongly classified hand-drawn sketches. All sketches in each row are classified into the class labeled on top left. False positives are in red, with their ground truth classes labeled on top. 40

3.6 Line-drawing style rendering from 3D shapes. 41

3.7 Sketch-based 3D shape retrieval examples. Top matches are shown for each query, with mistakes highlighted in red. 41

4.1 Bilateral Convolution Layer. *Splat*: BCL first interpolates input features F onto a d_l-dimensional permutohedral lattice defined by the lattice features L at input points. *Convolve*: BCL then does d_l-dimensional convolution over this sparsely populated lattice. *Slice*: The filtered signal is then interpolated back onto the input signal. For illustration, input and output are shown as point cloud and the corresponding segmentation labels. 43

4.2 Illustration of inputs, outputs and network architectures for SPLATNet$_{\text{3D}}$ and SPLATNet$_{\text{2D-3D}}$. 47

4.3 Effect of different lattice spaces and scales. Visualizations for different lattice feature spaces $L = (x, y, z), (x, y, z), (n_x, n_y, n_z)$ along with lattice scales $\Lambda = I_3, 8I_3, I_3$. (n_x, n_y, n_z) refers to point normals. All points falling in the same lattice cell are colored the same. 50

4.4 2D to 3D projection. Illustration of 2D to 3D projection using *splat* and *slice* operations. Given input features of 2D images, pixels are projected onto a 3D permutohedral lattice defined by 3D positional lattice features. The splatted signal is then sliced onto the points of interest in a 3D point cloud. 52

4.5 Sample visual results of SPLATNet$_{\text{3D}}$ and SPLATNet$_{\text{2D-3D}}$ for Facade point cloud labeling. 54

4.6 Sample visual results of SPLATNet$_{\text{2D-3D}}$ for 2D facade segmentation. 57

4.7 Sample visual results of SPLATNet$_{\text{3D}}$ and SPLATNet$_{\text{2D-3D}}$ for ShapeNet part segmentation. 58

4.8 Four types of labeling issues in the ShapeNet Part dataset are shown here. Two examples from the test set are given for each type, where the first row shows the ground-truth labels and the second row shows our predictions with SPLATNet$_{\text{2D-3D}}$. Our predictions appear to be more accurate than the ground truth in some cases (see the skateboard axles in Figure 4.8a and the rocket fins in Figure 4.8b). 64

5.1 Pixel-Adaptive Convolution (PAC) modifies a standard convolution on an input $\mathbf{v}$ by modifying the spatially invariant filter $\mathbf{W}$ with an adapting kernel K. The adapting kernel is constructed using either pre-defined or learned features $\mathbf{f}$. $\otimes$ denotes element-wise multiplication of matrices followed by a summation. Only one output channel is shown for the illustration. 67

5.2 Joint upsampling with PAC. Network architecture showing encoder, guidance and decoder components. Features from the guidance branch are used to adapt $\text{PAC}^{\intercal}$ kernels that are applied on the encoder output resulting in upsampled signal. 77

5.3 Deep joint upsampling. Results of different methods for $16\times$ joint depth upsampling (top row) and $16\times$ joint optical flow upsampling (bottom row). Our method produces results that have more details and are more faithful to the edges in the guidance image. 80

5.4 PAC-CRF. Illustration of inputs, outputs and the operations in each mean-field (MF) step of PAC-CRF inference. Also shown is the coverage of two 5×5 PAC filters, with dilation factors 16 and 64 respectively. 84

5.5 Semantic segmentation with PAC-CRF and PAC-FCN. We show three examples from the validation set. Compared to Full-CRF [69], BCL-CRF [57], and Conv-CRF [119], PAC-CRF can recover finer details faithful to the boundaries in the RGB inputs. 90

5.6 Artifacts in the PAC-CRF predictions. From left to right: RGB image, ground-truth labeling, PAC-CRF output with zoomed-in details. 92

5.7 Layer hot-swapping with PAC. A few layers of a network before (top) and after (bottom) hot-swapping. Three CONV layers are replaced with PAC layers, with adapting features coming from an earlier convolution layer. All the original network weights are retained after the modification. 93

CHAPTER 1
INTRODUCTION

Just like the human visual system, computer vision systems acquire visual inputs, process them, and perform reasoning for various kinds of tasks. Inputs to such systems are predominantly 2-dimensional (2D) signals because they are much easier to come by with all sorts of traditional imaging sensors. Consequently, a fundamental challenge of computer vision has been to draw inferences about the three-dimensional (3D) world given only 2D signals. For example, a quintessential computer vision task, object recognition, is typically formulated as classifying 2D images that depict objects from certain viewpoints. As a result, designing powerful 2D image features have become an important and very active research direction in computer vision, with earlier work focusing on handcrafted feature descriptors, such as SIFT [81] and HOG [27], and more recent approaches using neural networks [71].

What if one has access to the 3D models of the objects of interest? With the 3D models, one can extract 3D features, such as surface curvature or voxel occupancy, instead of relying purely on 2D image features. Another source of complications is that 3D data can come in different forms (Figure 1.1), *e.g.* polygon meshes, voxels, point clouds, 2D renderings, depth maps, and it is sometimes advantageous to convert one representation to another for performance or convenience reasons.

The interests in building recognition models directly from 3D representations have recently emerged due to several reasons:

1. **Applications**. Advances in 3D imaging sensors have brought costs down and made the technology practical for real-world applications. A few notable exam-

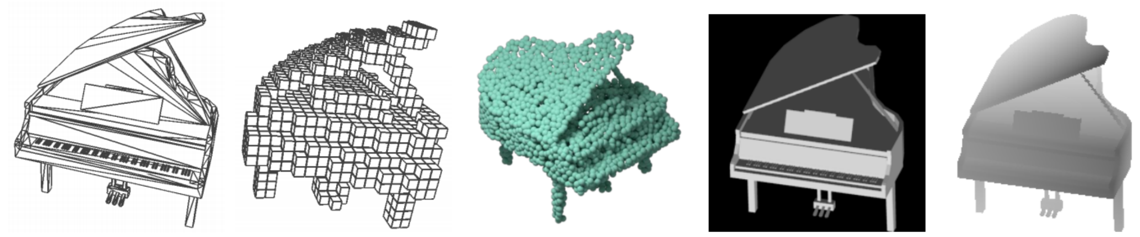


Figure 1.1: Various types of 3D data representations. From left to right: polygon mesh, voxels, point cloud, 2D rendering, depth map.

ples include Microsoft Kinect for motion sensing in gaming and LiDAR sensors used in autonomous vehicles.

2. **Data**. The introduction of large-scale 3D shape repositories, such as 3D Warehouse [1] and TurboSquid [2], as well as academic datasets, such as ModelNet [137] and ScanNet [25], provides means for proper benchmarking and sufficient data for training more complicated machine learning models than before.

3. **Computational cost**. The higher dimensionality usually leads to a larger memory footprint and higher computational cost, especially when the recognition task demands a high resolution in the 3D representations. It is only with the rapid developments in computer hardware such as graphics processing units (GPUs) that 3D recognition approaches start to become feasible.

While a large corpus of shape descriptors has been developed for drawing inferences about 3D objects in both the computer vision and graphics literature (see Section 2.2 for a literature review), they are mostly *hand-engineered* and often do not generalize well beyond their original domains. In this dissertation, we focus on 3D representation learning using deep neural networks (DNNs) and demonstrate that

[1]`https://3dwarehouse.sketchup.com/`

[2]`https://www.turbosquid.com/`

DNNs' advantages over traditional approaches in 2D computer vision tasks can also be carried over to the 3D domain.

1.1 Deep Neural Networks

DNNs are powerful machine learning models and have seen widespread success in recent years. In many artificial intelligence (AI) areas, including computer vision and natural language processing, it is rare nowadays to see tasks where the best solutions do not involve DNNs. However, existing DNN architectures do have limitations when dealing with data that are not well-structured like images or text. In this dissertation, we explore a few such cases around 3D and high-dimensional data, and aim to design new types of neural network operations and architectures to mitigate the issues while retaining the strong performance of DNNs.

A large part of the recent success of DNNs can be attributed to the wide adoption of convolutional operations. Neural networks with such operations are also called convolutional neural networks (CNN). Convolutional filters operate in a sliding-window fashion and can therefore effectively capture the invariance in data, leading to a more efficient use of training data and more compact representations. Another strength of CNNs, due to the prevalent usages in many applications, is that extremely efficient implementations are now available on all sorts of modern hardware making any inventions in this area quickly accessible to a wide audience.

Since convolution operates on regularly structured grids, it is a particularly good fit for texts and images where there are inherent 1D or 2D structures. However, extending CNNs to 3D or higher-dimensional spaces is non-trivial, because data in such spaces often lack regular structure and the curse of dimensionality can also adversely impact performance in many ways. Although raw 3D data are predominantly in the forms of polygon meshes and point clouds (Figure 1.1), many existing techniques first pre-process and project them onto a regular 3D grid through voxelization. In this dissertation, we aim to explore a few different directions, where we utilize multi-view 2D projections (Chapter 3) or directly operate on 3D point clouds (Chapter 4).

1.2 Multi-View Representations

Various kinds of 3D features such as surface curvature or voxel occupancy become available when one has access to 3D models of the objects of interest. While intuitively, it seems logical then to build 3D shape classifiers utilizing those 3D features, in this dissertation we present a seemingly counter-intuitive result – that by building classifiers of 3D shapes from 2D-view renderings of those shapes, we can actually *dramatically outperform* earlier work that builds classifiers directly on the 3D representations. In particular, a CNN trained on a fixed set of rendered views of a 3D shape and only provided with a *single* view at test time can already increase category recognition accuracy by a significant margin over the best earlier models [137] trained on 3D representations. We also show that when more views are available, the recognition performance can further increase.

Such a dramatic result gives additional ammunition to the classical argument that collections of 2D views implicitly contain rich information about 3D structure. Besides, not only do they contain this information about 3D structure, but at least according to our experiments, the 2D views appear to be a *better representation* than the 3D structure itself for certain recognition tasks. The implications are potentially remarkable, given the long history of the debate about how 3D information should be represented in computer vision.

One reason for this result is the relative efficiency of the 2D versus the 3D representations. In particular, while a full resolution 3D representation contains all of the information about an object, in order to use a voxel-based representation in a deep network that can be trained with available samples and in a reasonable amount of time, it would appear that the resolution needs to be significantly reduced. For example, 3D ShapeNets [137] use a coarse representation for shapes, *i.e.* a 30×30×30 grid of binary voxels. In contrast, a single projection of the 3D model of the same input size corresponds to an image of 164×164 pixels (smaller if multiple projections are used). Overall, there is an inherent trade-off between increasing the amount of explicit geometric information (what 3D models are good for) and increasing spatial resolution in appearance (where 2D models are better).

Another advantage of using 2D representations is that we can leverage (i) powerful CNN architectures that have been designed for 2D computer vision tasks and (ii) massive image databases (such as ImageNet [29]) to pre-train the CNNs before they are adapted to 3D tasks. Because images are ubiquitous and large labeled datasets are abundant, we can learn a good deal about generic features for 2D image recognition and then fine-tune to specifics about projections of 3D shapes. While it is possible that some day as much 3D training data will be available, for the time being there is still a significant advantage of 2D representations in this aspect.

Knowing that the simple strategy of classifying views independently already works remarkably well, it seems obvious that using multiple views at the same time should further improve recognition accuracy. A remaining question, though, is how to compile the information from multiple views. The Multi-View CNN architecture we propose in Chapter 3 is an attempt in this direction, offering a simple strategy to compile the information with a single pooling operation. We observe state-of-the-art performance with the resulting descriptors on tasks including 3D object classification, 3D object retrieval using 3D objects, and 3D object retrieval using sketches.

It is worth noting the connection between multi-view representations and the "jittering" process in data augmentation where transformed copies of the data are added during training time to improve invariances to transformations such as rotation or translation. In the context of 3D object recognition, the views can be regarded as jittered copies, and the traditional data augmentation approach aims to make the resulting model *view-invariant*. A Multi-View CNN instead learns to combine the views, and can choose to focus on the more informative views of the object for prediction while ignoring less informative ones. Similar ideas can also be applied to standard image classification tasks as an alternative strategy for traditional data augmentation.

1.3 Representation Learning for Point Clouds

While multi-view representations can be viable choices when high-quality projected views are easy to obtain (*e.g.* rendering views for polygon meshes), they are not well-suited to point clouds. Data obtained with modern 3D sensors such as LiDAR scanners is predominantly in the form of point clouds. Consequent, processing and analysis of point clouds have an important role in applications such as robotic manipulation and autonomous driving where such sensors are widely deployed.

These properties of point clouds make it difficult to use traditional CNN operations and architectures for point cloud processing and analysis. As a result, previous approaches that directly operate on point clouds are dominated by hand-crafted features. A straightforward strategy to use CNNs on point clouds is by first pre-processing a given point cloud into a form that is amenable to standard spatial convolutions. This route leads to two major options, each with notable drawbacks:

- **Using 2D view projections, including multi-view representations, to reduce the task to image-based recognition.** While we propose this as a practical approach for polygon meshes, a few drawbacks limit its effectiveness on points clouds. Unlike polygon meshes, point clouds can be very sparse when rendered in 2D views. The resulting 2D projections have very different appearances and statistics compared to natural images, so the benefit of pre-training on large-scale image datasets such as ImageNet is potentially much less. Another consideration is computational efficiency. Point cloud is a very compact data format, and typically an object is represented by a point cloud of a couple of thousand points and often much fewer (*e.g.* distant objects in LiDAR scans). Directly working with the raw points could thus offer better efficiency and a smaller model footprint compared to multi-view presentations.

- **Converting point clouds into voxel representations.** One can easily apply 3D convolutions and extend existing image recognition architecture to data defined on dense voxel grids. Apart from the trade-off between resolution and computational efficiency discussed earlier in Section 1.2, there are further concerns in this strategy regarding data that originally come in the form of point clouds. Point clouds typically have very unbalanced density distribution spatially. Putting point clouds onto a dense discrete voxel grid means regions with different densities, including sparse areas and even void spaces, all require storage and computation. This not only fails to take the opportunity to leverage the sparsity property for computational benefits, but also throws away useful information encoded in point density. The discretization process also largely discards the precise point locations of the point clouds, which can be an important source of information, especially for tasks requiring fine details.

Recently, a few network architectures [95, 96, 145] have been developed to directly operate on point clouds. One of the main drawbacks of these architectures is that they do not allow a flexible specification of the extent of spatial connectivity across points (*i.e.* filtering neighborhood). Both [95] and [96] use max-pooling to aggregate information across points either globally [95] or in a hierarchical manner [96]. This pooling aggregation may lose valuable surface information because the spatial layouts of points are not explicitly considered.

It is desirable to capture spatial relationships in point clouds through more general convolutional operations while being able to specify filter extents in a flexible manner. With this motivation, we develop SPLATNet (Figure 1.2) described in Chapter 4, a

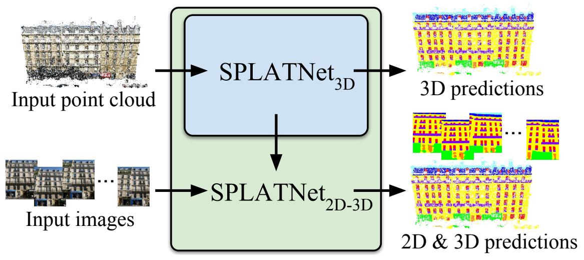


Figure 1.2: From point clouds and images to semantics. SPLATNet$_{3D}$ directly takes point cloud as input and predicts labels for each point. SPLATNet$_{2D\text{-}3D}$, on the other hand, jointly processes both point cloud and the corresponding multi-view images for better 2D and 3D predictions.

new type of neural network architecture for point cloud processing. Some recent work concurrent to this dissertation brings ideas in a similar direction. A more detailed discussion of some of these research work is provided in Chapter 2.

Borrowing ideas from the existing techniques of sparse high-dimensional filtering, the network directly operates on a collection of points represented as a sparse set of samples in a high-dimensional lattice. Naïvely applying convolutions on this lattice scales poorly, both in terms of memory and computational cost, as the size of the lattice increases. Instead, the network uses sparse bilateral convolutional layers (Section 4.1) as building blocks. These layers maintain efficiency by using indexing structures to apply convolutions only on occupied parts of the lattice, and allow flexible specifications of the lattice structure enabling hierarchical and spatially-aware feature learning, as well as joint 2D-3D reasoning. Both point-based and image-based representations can be easily incorporated in a network with such layers and the resulting model can be trained in an end-to-end manner.

1.4 Content-Adaptive Convolutions

The fact that the weights of a convolutional filter are spatially shared brings many of its desirable properties, but it is also a major limitation, as it makes convolutions *content-agnostic*. Once a CNN is trained, the same convolutional filter banks are applied to all the images and all the pixels irrespective of their content. The image content varies substantially across images and pixels. Thus a single trained CNN may not be optimal for all image types (*e.g.* images taken in daylight vs. at night, Figure 1.3) as well as different pixels in an image (*e.g.* sky vs. pedestrian pixels). Ideally, we would like CNN filters to be adaptive to the type of image content, which is not the case with standard CNNs.

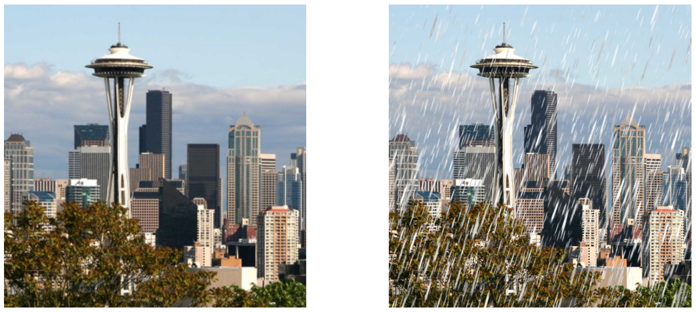


Figure 1.3: A same scene under two different weather conditions can benefit from convolutional filters adaptive to the conditions.

In the last part of the dissertation, we relate sparse high-dimensional filtering used earlier for point cloud processing to *content-adaptive* convolutions. Since pixels of an image are arranged in a dense 2D grid, it is natural to consider their neighborhoods in terms of XY pixel coordinates for convolutional operations. However, that is not the only option, and luckily with proper tools for sparse high-dimensional filtering, any feature coordinates, even high-dimensional ones, can be considered for constructing neighborhood structures to conduct convolutions.

To get an idea why convolutions over other feature spaces make sense, let's take a look at bilateral filters [121] as an example. A bilateral filter is an *edge-preserving* smoothing filter, *i.e.* a smoothing filter that is designed to better preserve the sharp edges in the image. Just like a traditional Gaussian smoothing filter, it produces the new intensity of an image pixel with a weighted average of intensities of its nearby pixels, and the weights come from a Gaussian distribution. The averaging behavior is close to a low-pass filter and reduces noises in the input. Both filters share the definition:

$$I^{\text{filtered}}(p) = \frac{1}{N(x)} \sum_{p' \in \Omega(p)} W(p', p) I(p') \tag{1.1}$$

where p and p' are pixel coordinates, I is the original image, I^{filtered} is the filtered output, Ω is the filtering neighborhood, $W(p', p)$ is the filter weight, and $N(p) = \sum_{p' \in \Omega(x)} W(p', p)$ is a normalization term to ensure that the overall image brightness won't change after the filtering.

For a Gaussian smoothing filter, the weights are simply defined to capture spatial closeness:

$$W(p', p) = \exp\left(-\frac{\|p' - p\|^2}{2\sigma^2}\right) \tag{1.2}$$

where σ is the standard deviation of the Gaussian distribution that is typically manually picked.

The function gives higher weight to nearby pixels than more distant pixels. If a filtering neighborhood $\Omega(p)$ happens to contain a sharp transition of intensity values, pixels across the boundary will still take part in the averaging, resulting a blurrier boundary in the filtered image.

A bilateral filter differs in that it also considers the intensity closeness when assigning weights to the neighboring pixels:

$$W(p', p) = \underbrace{\exp\left(-\frac{\|p' - p\|^2}{2\sigma_s^2}\right)}_{\text{spatial kernel}} \underbrace{\exp\left(-\frac{\|I(p') - I(p)\|^2}{2\sigma_r^2}\right)}_{\text{range kernel}} \tag{1.3}$$

The two components in the weight function, referred to as *spatial kernel* and *range kernel*, depend on Euclidean distances of pixels and intensity differences respectively. Because the filter weights vary according to image content (as opposed to being shared everywhere as in Equation 1.2), bilateral filters can be considered as a type of content-adaptive filter.

The connection to high-dimensional filtering becomes obvious when the two kernels in Equation 1.3 are rewritten as a combined kernel:

$$\begin{aligned} W(p', p) &= \exp\left(-\frac{\|p' - p\|^2}{2\sigma_s^2} - \frac{\|I(p') - I(p)\|^2}{2\sigma_r^2}\right) && (1.4) \\ &= \exp\left(-\frac{1}{2}\left(\frac{x'}{\sigma_s} - \frac{x}{\sigma_s}\right)^2 - \frac{1}{2}\left(\frac{y'}{\sigma_s} - \frac{y}{\sigma_s}\right)^2 - \frac{1}{2}\left(\frac{I(p')}{\sigma_r} - \frac{I(p)}{\sigma_r}\right)^2\right) && (1.5) \\ &= \exp\left(-\frac{1}{2}\|\mathbf{v}_{p'} - \mathbf{v}_p\|^2\right) && (1.6) \end{aligned}$$

where $\mathbf{v}_p$ is a vector containing three elements (assuming image is grayscale): $[x, y, I(p)]^\intercal$.

Equation 1.6 indeed describes bilateral filter as a three-dimensional Gaussian filter for grayscale images. For RGB color images, a bilateral filter can similarly be seen as a five-dimensional Gaussian filter in (x, y, R, G, B) space. Note that in these high-dimensional spaces, the pixels are quite sparse, just like points in a 3D point clouds, and the same computational tools can be applied.

A bilateral filter performs content-adaptive filtering in a very specific and limited manner: it smooths images with filters adapted to local image content so that sharp details are preserved. As we present in the final part of the dissertation (Chapter 5), we explore several directions to extend the idea to design a more general-purpose operation for neural networks, including:

- Replacing the spatial kernel in Equation 1.3 with free-form filters can allow learning filters for purposes beyond smoothing.

- In Equation 1.3, the Gaussian standard deviations σ_s and σ_r are hyperparameters and need to be picked manually, *e.g.* via cross-validation. Allowing them to be learnable parameters would help find the optimal choices more efficiently.

- Bilateral filters use two hand-picked kernels, a spatial kernel and a range kernel. While this design decision is well-motivated for its intended application, ideally, we would want to learn the best features (*i.e.* $\mathbf{v}$ in Equation 1.6) to build a kernel that best suits the diverse end tasks.

- For tasks involving only images, explicitly embedding pixels into a higher-dimensional space and performing convolutions there can be a significant computational overhead. Formulating the operation in a way that can be implemented as a dense 2D operation not only avoids burdensome high-dimensional computations but also allows easier integration with existing image-based neural network architectures.

1.5 Dissertation Outline

The rest of the dissertation is organized as followed. We first provide a literature review over related research topics in Chapter 2. In Chapter 3, we investigate the feasibility of adapting 2D CNNs to 3D recognition tasks to circumvent the difficulties of directly operating on 3D data. We propose a neural network operation that can effectively pool information from multiple 2D views and allows a single CNN to be trained end-to-end incorporating all views simultaneously. Next, we look into the challenging problem of point cloud processing in Chapter 4 and propose a new type of neural network layer based on sparse high-dimensional filtering. With the new layer and several architectures designed around it, we can directly operate on point clouds without voxelization or 2D projections, which incur additional computation burden and bring unavoidable loss of information. The new architectures also make possible efficient joint 2D-3D processing. In Chapter 5, we relate the sparse high-dimensional filtering technique with content-adaptive image filtering, and propose another type of layer that can allow its filters to adapt to image content while still benefit from the spatial-sharing property and the efficiency of traditional convolutional filters. We end the disseratation with conclusions and a discussion of potential future extensions in Chapter 6.

CHAPTER 2
LITERATURE REVIEW

In this chapter, we provide literature review over a few relevant topics. Some additional topics that are specific to only certain chapters are covered later in the respective chapters.

2.1 Convolutional Neural Networks

Much of the work in this dissertation is motivated by the recent success in computer vision tasks using CNNs. AlexNet [71] is an important milestone for image recognition and pioneered many design choices that are still widely adopted today. With a network deeper than ever before, AlexNet achieved image classification performance almost unimaginable at that time on the challenging ImageNet [29] benchmark. Since then, many deeper and more powerful architectures have been proposed, such as VGG [113], ResNet [51], Inception [117], *etc.*, pushing the image classification accuracies even higher.

Besides image classification, CNNs have also seen tremendous success in a wide range of other visual recognition tasks, such as object detection [40, 41, 98], semantic segmentation [20], texture recognition [23], and fine-grained classification [78].

CNNs trained on large datasets such as ImageNet have been shown to learn general-purpose image descriptors useful for other tasks [30, 97, 23], either directly or through a process called *fine-tuning* where smaller-scale further training on the downstream tasks is conducted. This in particular contributes to the success of the Multi-View CNN approach described in Chapter 3.

2.2 3D Shape Descriptors

A large corpus of shape descriptors has been developed for drawing inferences about 3D objects in both the computer vision and graphics literature. Shape descriptors can be classified into two broad categories: *native 3D shape descriptors* that directly work on the native 3D representations of objects, such as polygon meshes, voxel-based discretizations, point clouds, or implicit surfaces, and *view-based descriptors* that describe the shape of a 3D object by "how it looks" in a collection of 2D projections.

With the exception of the recent work of Wu *et al.* [137] which learns shape descriptors from the voxel-based representation of an object through 3D convolutional nets, previous native 3D shape descriptors were largely "hand-designed" according to a particular geometric property of the shape surface or volume. For example, shapes can be represented with histograms or bag-of-features models constructed out of surface normals and curvatures [53], distances, angles, triangle areas or tetrahedra volumes gathered at sampled surface points [89], properties of spherical functions defined in volumetric grids [60], local shape diameters measured at densely sampled surface points [16], heat kernel signatures on polygon meshes [9, 66], or extensions of the SIFT and SURF feature descriptors to 3D voxel grids [64]. Developing classifiers and other supervised machine learning algorithms on top of such 3D shape descriptors poses a number of challenges. First, the size of organized databases with annotated 3D models is rather limited compared to image datasets, *e.g.*, ModelNet contains about 150K shapes (its 40 category benchmark contains about 4K shapes). In contrast, the ImageNet database [29] already includes tens of millions of annotated images. Second, native 3D shape descriptors tend to be very high-dimensional, making classifiers prone to overfitting due to the so-called "curse of dimensionality".

On the other hand view-based descriptors have a number of desirable properties: they are relatively low-dimensional, efficient to evaluate, and robust to 3D shape representation artifacts, such as holes, imperfect polygon mesh tesselations, noisy surfaces. The rendered shape views can also be directly compared with other 2D images, silhouettes or even hand-drawn sketches. An early example of a view-based approach is the work by Murase and Nayar [88] that recognizes objects by matching their appearance in parametric eigenspaces formed by large sets of 2D renderings of 3D models under varying poses and illuminations. Another example, which is particularly popular in computer graphics setups, is the Light Field Descriptor [17] that extracts a set of geometric and Fourier descriptors from object silhouettes rendered from several different viewpoints. Alternatively, the silhouette of an object can be decomposed into parts and then represented by a directed acyclic graph (shock graph) [82]. Cyr and Kimia [24] defined similarity metrics based on curve matching and grouped similar views, called aspect graphs of 3D models [65]. Eitz et al. [32] compared human sketches with line drawings of 3D models produced from several different views based on local Gabor filters, while Schneider et al. [106] proposed using Fisher vectors [92] on SIFT features [81] for representing human sketches of shapes. These descriptors are largely "hand-engineered" and some do not generalize well across different domains.

2.3 Deep Neural Networks for 3D Data

Much attention has been given to 3D representation learning in the past few years following the wide success of deep neural networks in computer vision. As 3D data can come in many different formats (Figure 1.1), researchers have proposed deep neural network operations and architectures specifically designed for common types of 3D data formats.

2.3.1 Voxel networks

Voxel-based methods convert the input 3D shape representation into a 3D volumetric grid. Early voxel-based architectures executed convolution in regular, fixed voxel grids, and were limited to low shape resolutions due to high memory and computation costs [137, 86, 94, 8, 38, 107]. Instead of using fixed grids, more recent approaches pre-process the input shapes into adaptively subdivided, hierarchical grids with denser cells placed near the surface [101, 100, 63, 130, 118]. As a result, they have much lower computational and memory overhead. On the other hand, convolutions are often still executed away from the surface, where most of the shape information resides. An alternative approach is to constrain the execution of volumetric convolutions only along the input sparse set of active voxels of the grid [45].

2.3.2 Point cloud networks

Qi *et al.* [95] pioneered another type of deep networks having the advantage of directly operating on point clouds. The networks learn spatial feature representations for each input point, then the point features are aggregated across the whole point set [95], or hierarchical surface regions [96] through max-pooling. This aggregation may lose surface information since the spatial layout of points is not explicitly considered.

Concurrent to work in this dissertation, a large body of research has developed better leveraging the spatial relationships among local point neighborhoods [73, 54, 138, 132, 120, 136, 83]. Many of these approaches adopt a general notion of convolution, which is defined as:

$$(\mathcal{F} * k)(x_i) = \sum_{x_j \in \mathcal{N}(x_i)} k(x_i, x_j)\mathcal{F}(x_j) \tag{2.1}$$

where x_i and x_j are point coordinates, $\mathcal{N}(x_i)$ is the neighborhood considered for the convolution around x_i, $\mathcal{F}$ contains the point features, and k is the kernel function. Under this definition, these approaches differ in their choices of the neighborhood construction and kernel functions:

- **Neighborhood.** A very common choice to build local point neighborhoods is to construct k-nearest neighbor graph [73, 138, 136, 132]. Other choices include grid cells [54, 83] and radius neighborhoods [120].

- **Kernel function.** The design of kernel function k also sees a great diversity in recent research work. Some researchers advocate using continuous kernel functions to mitigate the issue of irregular spatial layout in point clouds. Multilayer perceptron (MLP) networks is a natural choice for this approach. A MLP can take as inputs the spatial offsets of the neighboring points, $x_j - x_i$, and output the respective weight [136, 73]. More complicated designs can also consider the features at the points as part of the inputs [132]. Some researchers propose to use a parameterized family of geometric functions in place of MLPs for easier optimization [138]. As an alternative to continuous kernel functions, some recent work takes inspiration from the wide success of 2D image convolutions and propose methods to use discrete filters on point clouds [120, 83, 54]. Since kernel weights are defined only on a limited set of discrete locations, the focus of such work is then to properly associate neighboring points with the defined kernel locations. Some networks [120, 83] even possess the capability to *learn* the kernel locations during training, instead of learning only the weights.

The approach presented in this dissertation (Chapter 4) also falls under the general notion of Equation 2.1. We design our networks and make unique choices of the local neighborhoods and kernel functions to achieve a balance in optimal performance, efficiency, and flexibility in applications.

2.3.3 Non-Euclidean networks

An alternative approach is to represent the input surface as a graph (*e.g.*, a polygon mesh or point-based connectivity graph), convert the graph into its spectral representation, then perform convolution in the spectral domain [11, 52, 28, 6].

However, structurally different shapes tend to have largely different spectral bases, and thus lead to poor generalization. Yi *et al*. [141] proposed aligning shape basis functions through a spectral transformer, which, however, requires a robust initialization scheme. Another class of methods embeds the input shapes into 2D parametric domains and then execute convolutions within these domains [114, 84, 34]. However, these embeddings can suffer from spatial distortions or require topologically consistent input shapes. Other methods parameterize the surface into local patches and execute surface-based convolution within these patches [85, 7, 87]. Such non-Euclidean networks have the advantage of being invariant to surface deformations, yet this invariance might not always be desirable in man-made object segmentation and classification tasks where large deformations may change the underlying shape or part functionalities and semantics. We refer to Bronstein *et al*. [10] for an excellent review of spectral, patch- and graph-based methods.

CHAPTER 3

3D SHAPE RECOGNITION WITH MULTI-VIEW CONVOLUTIONAL NEURAL NETWORKS

A longstanding question in computer vision concerns the representation of 3D shapes for recognition: should 3D shapes be represented with descriptors operating on their native 3D formats, such as voxel grid or polygon mesh, or can they be effectively represented with view-based descriptors? In this chapter, we address this question in the context of learning to recognize 3D shapes from a collection of their rendered views on 2D images. We first present a standard CNN architecture trained to recognize the shapes' rendered views independently of each other, and show that a 3D shape can be recognized even from a single view at an accuracy far higher than using state-of-the-art 3D shape descriptors. Recognition rates further increase when multiple views of the shapes are provided. In addition, we present a novel CNN architecture that combines information from multiple views of a 3D shape into a single and compact shape descriptor offering even better recognition performance. The same architecture can be applied to accurately recognize human hand-drawn sketches of shapes. At the end, we conclude that a collection of 2D views can be highly informative for 3D shape recognition and is amenable to emerging CNN architectures and their derivatives.

3.1 Multi-View Convolutional Neural Networks

As discussed above, our focus in this chapter is on developing view-based descriptors for 3D shapes that are trainable, produce informative representations for recognition and retrieval tasks, and are efficient to compute.

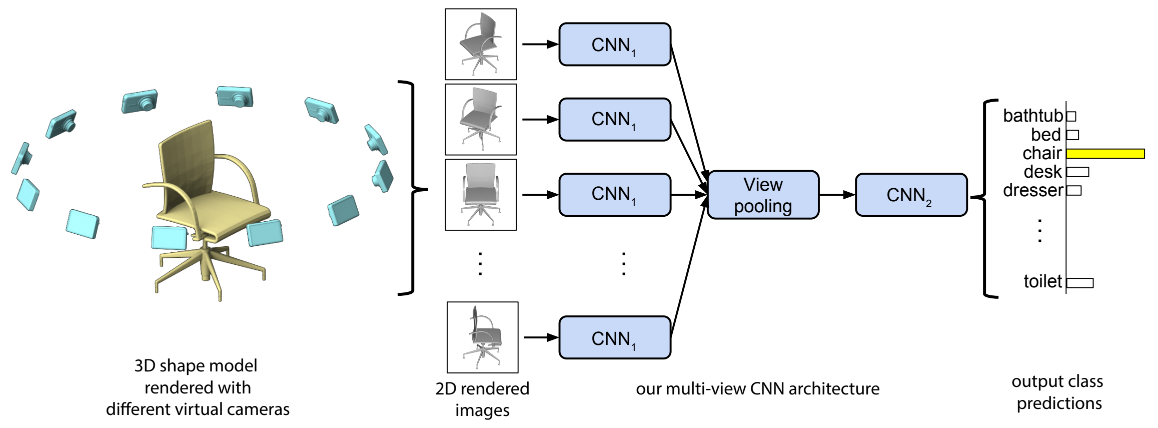


Figure 3.1: Multi-View CNN for 3D shape recognition (illustrated using the 1st camera setup). At test time a 3D shape is rendered from 12 different views and are passed thorough CNN_1 to extract view based features. These are then pooled across views and passed through CNN_2 to obtain a compact shape descriptor.

Our view-based representations start from multiple views of a 3D shape, generated by a rendering engine. A simple way to use multiple views is to generate a 2D image descriptor per each view, and then use the individual descriptors directly for recognition tasks based on some voting or alignment scheme. For example, a naïve approach would be to average the individual descriptors, treating all the views as equally important. Alternatively, if the views are rendered in a reproducible order, one could also concatenate the 2D descriptors of all the views. Unfortunately, aligning a 3D shape to a canonical orientation is hard and sometimes ill-defined. In contrast to the above simple approaches, an aggregated representation combining features from multiple views is more desirable since it yields a single, compact descriptor representing the 3D shape.

Our approach is to learn to combine information from multiple views using a unified CNN architecture that includes a view-pooling layer (Figure 3.1). All the parameters of our CNN architecture are learned discriminatively to produce a single compact descriptor for the 3D shape. Compared to exhaustive pairwise comparisons between single-view representations of 3D shapes, our resulting descriptors can be directly used to compare 3D shapes leading to significantly higher computational efficiency.

3.1.1 Multi-view input representations

3D models in online databases are typically stored as polygon meshes, which are collections of points connected with edges forming faces. To generate rendered views of polygon meshes, we use the Phong reflection model [93]. The mesh polygons are rendered under a perspective projection and the pixel color is determined by interpolating the reflected intensity of the polygon vertices. Shapes are uniformly scaled to fit into the viewing volume.

To create a multi-view shape representation, we need to setup viewpoints (virtual cameras) for rendering each mesh. We experimented with two camera setups. For **the 1$^{\text{st}}$ camera setup**, we assume that the input shapes are upright oriented along a consistent axis (*e.g.*, z-axis). Most models in modern online repositories, such as the 3D Warehouse, satisfy this requirement, and some previous recognition methods also follow the same assumption [137]. In this case, we create 12 rendered views by placing 12 virtual cameras around the mesh every 30 degrees (see Figure 3.1). The cameras are elevated 30 degrees from the ground plane, pointing towards the centroid of the mesh. The centroid is calculated as the weighted average of the mesh face centers, where the weights are the face areas. For **the 2$^{\text{nd}}$ camera setup**, we do not make use of the assumption about consistent upright orientation of shapes. In this case, we render from several more viewpoints since we do not know beforehand which ones yield good representative views of the object. The renderings are generated by placing 20 virtual cameras at the 20 vertices of an icosahedron enclosing the shape. All cameras point towards the centroid of the mesh. Then we generate 4 rendered views from each camera, using 0, 90, 180, 270 degrees rotation along the axis passing through the camera and the object centroid, yielding total 80 views.

We note that using different shading coefficients or illumination models did not affect our output descriptors due to the invariance of the learned filters to illumination changes, as also observed in image-based CNNs [71, 30]. Adding more or different viewpoints is trivial, however, we found that the above camera setups were already enough to achieve high performance. Finally, rendering each mesh from all the viewpoints takes no more than ten milliseconds on modern graphics hardware.

3.1.2 Recognition with multi-view representations

We claim that our multi-view representation contains rich information about 3D shapes and can be applied to various types of tasks. In the first setting, we make use of existing 2D image features directly and produce a descriptor for each view. This is the most straightforward approach to utilize the multi-view representation. However, it results in multiple 2D image descriptors per 3D shape, one per view, which need to be integrated somehow for recognition tasks.

3.1.2.1 Image descriptors

We consider two types of image descriptors for each 2D view: a state-of-the-art "hand-crafted" image descriptor based on Fisher vectors [104] with multi-scale SIFT, as well as CNN activation features [30].

The Fisher vector image descriptor is implemented using VLFeat [128]. For each image multi-scale SIFT descriptors are extracted densely. These are then projected to 80 dimensions with PCA, followed by Fisher vector pooling with a Gaussian mixture model with 64 components, square-root and ℓ_2 normalization.

For our CNN features we use the VGG-M network from [15] which consists of mainly five convolutional layers $\text{conv}_{1,\dots,5}$ followed by three fully connected layers $\text{fc}_{6,\dots,8}$ and a softmax classification layer. The penultimate layer fc_7 (after ReLU non-linearity, 4096-dimensional) is used as image descriptor. The network is pre-trained on ImageNet images from 1k categories, and then fine-tuned on all 2D views of the 3D shapes in training set. As we show in our experiments, fine-tuning improves performance significantly. Both Fisher vectors and CNN features yield very good performance in classification and retrieval compared with popular 3D shape descriptors (*e.g.*, SPH [60], LFD [17]) as well as 3D ShapeNets [137].

3.1.2.2 3D shape classification

We train one-vs-rest linear SVMs (each view is treated as a separate training sample) to classify shapes using their image features. At test time, we simply sum up the SVM decision values over all 12 views and return the class with the highest sum. Alternative approaches, *e.g.*, averaging image descriptors, lead to worse accuracy.

3.1.2.3 3D shape retrieval

A distance or similarity measure is required for retrieval tasks. For shape $\mathbf{x}$ with n_x image descriptors and shape $\mathbf{y}$ with n_y image descriptors, the distance between them is defined in Equation 3.1. Note that the distance between two 2D images is defined as the ℓ_2 distance between their feature vectors, *i.e.* $\|\mathbf{x}_i - \mathbf{y}_j\|_2$.

$$d(\mathbf{x}, \mathbf{y}) = \frac{\sum_j \min_i \|\mathbf{x}_i - \mathbf{y}_j\|_2}{2n_y} + \frac{\sum_i \min_j \|\mathbf{x}_i - \mathbf{y}_j\|_2}{2n_x} \tag{3.1}$$

To interpret this definition, we can first define the distance between a 2D image $\mathbf{x}_i$ and a 3D shape $\mathbf{y}$ as $d(\mathbf{x}_i, \mathbf{y}) = \min_j \|\mathbf{x}_i - \mathbf{y}_j\|_2$. Then given all n_x distances between $\mathbf{x}$'s 2D projections and $\mathbf{y}$, the distance between these two shapes is computed by simple averaging. In Equation 3.1, this idea is applied in both directions to ensure symmetry.

We investigated alternative distance measures, such as minimum distance among all $n_x \cdot n_y$ image pairs and the distance between average image descriptors, but they all led to inferior performance.

3.1.3 Learning to aggregate views with Multi-View CNN

Although having multiple separate descriptors for each 3D shape can be successful for classification and retrieval compared to existing 3D descriptors, it can be inconvenient and inefficient in many cases. For example, in Equation 3.1, we need to compute all $n_x \times n_y$ pairwise distances between images in order to compute distance between two 3D shapes. Simply averaging or concatenating the image descriptors leads to inferior performance. In this section, we focus on the problem of learning to aggregate multiple views in order to synthesize the information from all views into a single, compact 3D shape descriptor.

We design the Multi-View CNN (MVCNN) architecture on top of image-based CNNs (Figure 3.1). Each image in a 3D shape's multi-view representation is passed through the first part of the network (CNN_1) separately, aggregated at a view-pooling layer, and then sent through the remaining part of the network (CNN_2). All branches in the first part of the network share the same parameters in CNN_1. We use element-wise maximum operation across the views in the view-pooling layer. An alternative is element-wise mean operation, but it is not as effective in our experiments. The view-pooling layer can be placed anywhere in the network. We show in our experiments that it should be placed close to the last convolutional layer (conv_5) for optimal classification and retrieval performance. View-pooling layers are closely related to max-pooling layers and maxout layers [42], with the only difference being the dimension that their pooling operations are carried out on. The MVCNN is a directed acyclic graphs and can be trained or fine-tuned using stochastic gradient descent with back-propagation.

Using fc_7 (after ReLU non-linearity) in an MVCNN as an aggregated shape descriptor, we achieve higher performance than using separate image descriptors from an image-based CNN directly, especially in retrieval (62.8% $\rightarrow$ 70.1%). Perhaps more importantly, the aggregated descriptor is readily available for a variety of tasks, *e.g.*, shape classification and retrieval, and offers significant speed-ups against multiple image descriptors.

An MVCNN can also be used as a general framework to integrate perturbed image samples (also known as data jittering). We illustrate this capability of MVCNNs in the context of sketch recognition in Section 3.2.2.

3.1.4 Low-rank Mahalanobis metric

When a MVCNN is fine-tuned for classification, its retrieval performance is not directly optimized. Although we could train it with a different objective function suitable for retrieval, we found that a simpler approach can readily yield a significant retrieval performance boost (see row 12 in Table 3.1). We learn a Mahalanobis metric W that directly projects MVCNN descriptors $\phi \in \mathbb{R}^d$ to $W\phi \in \mathbb{R}^p$, such that the ℓ_2 distances in the projected space are small between shapes of the same category, and large otherwise. We use the large-margin metric learning algorithm and implementation from [111], with $p < d$ to make the final descriptor compact ($p = 128$ in our experiments). The fact that we can readily use metric learning over the output shape descriptor demonstrates another advantage of using MVCNNs.

3.2 Experiments

3.2.1 3D shape classification and retrieval

3.2.1.1 ModelNet40

We first evaluate our shape descriptors on the Princeton ModelNet dataset [125]. ModelNet currently contains 127,915 3D CAD models from 662 categories[1]. A 40-class well-annotated subset containing 12,311 shapes from 40 common categories,

[1]As of 01/20/2020.

Table 3.1: Classification and retrieval results on the ModelNet40 dataset. On the top are results using state-of-the-art 3D shape descriptors. Our view-based descriptors including Fisher vectors (FV) significantly outperform these even when a single view is available at test time (#Views = 1). When multiple views (#Views=12 or 80) are available at test time, the performance of view-based methods improve significantly. The MVCNN architecture outperforms the view-based methods, especially for retrieval.

Method	**Training Config.**			**Test Config.**	**Cla. (Acc.)**	**Ret. (mAP)**
	Pre-train	**Fine-tune**	**#Views**	**#Views**		
(1) SPH [60]	-	-	-	-	68.2%	33.3%
(2) LFD [17]	-	-	-	-	75.5%	40.9%
(3) 3D ShapeNets [137]	ModelNet40	ModelNet40	-	-	77.3%	49.2%
(4) FV	-	ModelNet40	12	1	78.8%	37.5%
(5) FV, 12×	-	ModelNet40	12	12	84.8%	43.9%
(6) CNN	ImageNet1K	-	-	1	83.0%	44.1%
(7) CNN, f.t.	ImageNet1K	ModelNet40	12	1	85.1%	61.7%
(8) CNN, 12×	ImageNet1K	-	-	12	87.5%	49.6%
(9) CNN, f.t.,12×	ImageNet1K	ModelNet40	12	12	88.6%	62.8%
(10) MVCNN, 12×	ImageNet1K	-	-	12	88.1%	49.4%
(11) MVCNN, f.t., 12×	ImageNet1K	ModelNet40	12	12	89.9%	70.1%
(12) MVCNN, f.t.+metric, 12×	ImageNet1K	ModelNet40	12	12	89.5%	**80.2%**
(13) MVCNN, 80×	ImageNet1K	-	80	80	84.3%	36.8%
(14) MVCNN, f.t., 80×	ImageNet1K	ModelNet40	80	80	**90.1%**	70.4%
(15) MVCNN, f.t.+metric, 80×	ImageNet1K	ModelNet40	80	80	**90.1%**	79.5%

* f.t.=fine-tuning, metric=low-rank Mahalanobis metric learning

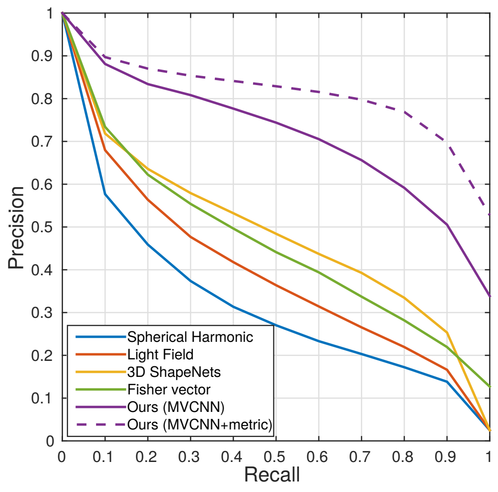


Figure 3.2: Precision-recall curves for various methods for 3D shape retrieval on the ModelNet40 dataset. Our method significantly outperforms the state-of-the-art on this task achieving 80.2% mAP.

ModelNet40, is provided on the ModelNet website. For our experiments, we use the same training and test split of ModelNet40 as in [137][2].

Our shape descriptors are compared against the 3D ShapeNets by Wu *et al.* [137], the Spherical Harmonics descriptor (SPH) by Kazhdan *et al.* [60], the LightField descriptor (LFD) by Chen *et al.* [17], and Fisher vectors extracted on the same rendered views of the shapes used as input to our networks.

Results of shape classification and retrieval on ModelNet are summarized in Table 3.1. Precision-recall curves are provided in Figure 3.2. Remarkably, the Fisher vector baseline with just a single view achieves a classification accuracy of 78.8% outperforming the state-of-the-art learned 3D descriptors (77.3% [137]). When all 12 views of the shape are available at test time (based on our first camera setup), we can also average the predictions over these views. Averaging increases the performance of Fisher vectors to 84.8%. The performance of Fisher vectors further supports our claim

[2]Based on our correspondence with the authors of [137], for each category the first 80 shapes in the "train" folder (or all shapes if there are fewer than 80) should be used for training, while the first 20 shapes in the "test" folder should be used for testing.

that 3D objects can be effectively represented using view-based 2D representations. The trends in performance for shape retrieval are similar.

Using our CNN baseline trained on ImageNet in turn outperforms Fisher vectors by a significant margin. Fine-tuning the CNN on the rendered views of the training shapes of ModelNet40 further improves the performance. By using all 12 views of the shape, its classification accuracy reaches 88.6%, and mean average precision (mAP) for retrieval is also improved to 62.8%.

MVCNN outperforms all state-of-the-art descriptors as well as the Fisher vector and CNN baselines. With fine-tuning on the ModelNet40 training set, our model achieves 89.9% classification accuracy, and 70.1% mAP on retrieval using the first camera setup. If we do not make use of the assumption about consistent upright orientation of shapes (second camera setup), the performance remains still intact, achieving 90.1% classification accuracy and 70.4% retrieval mAP. MVCNN constitutes an absolute gain of 12.8% in classification accuracy compared to the state-of-the-art learned 3D shape descriptor [137] (77.3% $\rightarrow$ 90.1%). Similarly, retrieval mAP is improved by 21.2% (49.2% $\rightarrow$ 70.4%). Finally, learning a low-rank Mahalanobis metric improves retrieval mAP further while classification accuracy remains almost unchanged, and the resulting shape descriptors become more compact ($d = 4096, p = 128$).

Confusion matrix of 3D shape classification on ModelNet40 is given in Figure 3.3. Here MVCNN with fine-tuning on 12 views (row 11 in Table 3.1) is used. Top confusions occur at 1) *flower pot* $\rightarrow$ *plant* (45%), 2) *table* $\rightarrow$ *desk* (32%), 3) *flower pot* $\rightarrow$ *vase* (20%), 4) *plant* $\rightarrow$ *flower* (19%), and 5) *stool* $\rightarrow$ *chair* (15%). Note that these are all very closely related categories. Distinctions between some of these pairs are ambiguous even for humans.

Position of view-pooling. We considered different locations to place the view-pooling layer in the MVCNN architecture.Performance is not very sensitive among the later few layers (conv$_4$–fc$_7$); however any location prior to conv$_4$ decreases classification accuracies significantly. We find conv$_5$ offers slightly better accuracies ($\sim$1%), and thus use it for all our experiments.

Saliency map among views. For each 3D shape S, our multi-view representation consists of a set of K 2D views $\{I_1, I_2 \ldots I_K\}$. We would like to rank pixels in the 2D views w.r.t. their influence on the output score F_c of the network (*e.g.* taken from fc$_8$ layer) for its ground truth class c. Following [112], saliency maps can be defined as the derivatives of F_c w.r.t. the 2D views of the shape:

$$[w_1, w_2 \ldots w_K] = \left[\left.\frac{\partial F_c}{\partial I_1}\right|_S, \left.\frac{\partial F_c}{\partial I_2}\right|_S \ldots \left.\frac{\partial F_c}{\partial I_K}\right|_S \right] \tag{3.2}$$

The magnitude of the derivative indicates changes to which pixels can influence the class score the most. For MVCNN, w in Equation 3.2 can be computed using back-propagation with all the network parameters fixed, and can then be rearranged to form saliency maps for individual views. Examples of saliency maps are shown in Figure 3.4. From the saliency maps, the most important views are usually the canonical views where the object looks the most recognizable. Within each individual saliency map, highly activated regions often point to distinct details (such as the handles of a dresser).

3.2.1.2 ShapeNetCore55

We participated the competition of SHREC'16 track "Large-Scale 3D Shape Retrieval from ShapeNetCore55"[3] and submitted results based on MVCNNs. The ShapeNetCore55 competition dataset contains a total of 51,190 3D models from 55 categories. The dataset provides standard training, validation and test splits, consti-

[3]`http://shapenet.cs.stanford.edu/shrec16/`

tuting 70% (35,765), 10% (5,159) and 20% (10,266) of the dataset respectively. The competition uses two different versions of the dataset: a *non-perturbed* dataset where all 3D models are consistently aligned, and a *perturbed* dataset where each model has been randomly rotated by a uniformly sampled rotation. We use the 1$^{\text{st}}$ camera setup with 12 views described in Section 3.1.1 for the non-perturbed dataset, and the 2$^{\text{nd}}$ camera setup with 80 views for the perturbed dataset.

Evaluation metrics. The submitted results to the competition are evaluated based on a set of standard information retrieval evaluation metrics:

1. Precision and Recall
2. F-score
3. Mean average precision (mAP)
4. Normalized discounted cumulative gain (NDCG)

Each model in the dataset is also assigned a sub-category which indicates a more refined class for the object. There are 204 sub-categories in total. The sub-category label is only considered in NDCG as a rough notion of finer relevance between 3D models.

Data balancing. The 55 categories in ShapeNetCore55 are highly imbalanced. In the training set, the largest category has about 150 times more shapes than the smallest category. The subcategories are even more imbalanced. To perform category balancing, we apply Equation 3.3 to the training class distribution $\mathbf{d}$, and randomly sample a training set for each training epoch according to $\mathbf{d}_{balanced}$. t is a parameter that controls the trade-off between macro-averaged and micro-averaged evaluation metrics. We set t to 0.5 for training the 55-category network and 0.2 for the 204-subcategory network.

$$\mathbf{d}_{balanced}(k) = avg(\mathbf{d}) \cdot (\frac{\mathbf{d}(k)}{avg(\mathbf{d})})^t \tag{3.3}$$

Retrieval with MVCNN. We train two networks for each camera setup: one for 55-way classification and another for 204-way subcategory classification. For each query, we first predict its 55-way class label and construct a retrieval set containing all shapes with the same predicted label. Then we extract features for the query and the targets from the output layer of the 204-way subcategory network (*i.e.* the features are the classification probabilities) and re-rank the results according to their L2 distances to the query. The re-ranking step will not influence precision and recall, and is designed mainly for improving NDCG.

The networks are based on VGG-M with the view-pooling layer placed at fc_7 layer and are initialized with ImageNet-pretrained weights. For ModelNet40, we use the penultimate layer in the network as features together with low-rank Mahalanobis metric learning for dimension reduction. For ShapeNetCore55, however, we use the output layer for optimal performance on the competition.

Our submission obtained the 1st place in the competition on the non-perturbed dataset and the 2nd place on the perturbed dataset. Detailed results can be found in the official competition report [105].

3.2.2 Sketch recognition: jittering revisited

Given the success of our aggregated descriptors on multiple views of a 3D object, it is logical to ask whether aggregating multiple views of a 2D image could also improve performance. Here we show that this is indeed the case by exploring its connection with data jittering in the context of sketch recognition.

Table 3.2: Classification results on SketchClean. Fine-tuned CNN models significantly outperform Fisher Vectors [106] by a significant margin. MVCNNs are better than CNN trained with data jittering. The results are shown with two different CNN architectures – VGG-M (row 2-5) and VGG-VD (row 6-9).

Method	**Aug.**	**Accuracy**
(1) FV [106]	-	79.0%
(2) CNN M	-	77.3%
(3) CNN M, fine-tuned	-	84.0%
(4) CNN M, fine-tuned	6×	85.5%
(5) MVCNN M, fine-tuned	6×	86.3%
(6) CNN VD	-	69.3%
(7) CNN VD, fine-tuned	-	86.3%
(8) CNN VD, fine-tuned	6×	86.0%
(9) MVCNN VD, fine-tuned	6×	**87.2%**
(10) Human performance	n/a	93.0%

Data jittering, or data augmentation, is a method to generate extra samples from a given image. It is the process of perturbing the image by transformations that change its appearance while leaving the high-level information (class label, attributes, *etc.*) intact. Jittering can be applied at training time to augment training samples and to reduce overfitting, or at test time to provide more robust predictions. In particular, several authors [71, 15, 117] have used data jittering to improve the performance of deep representations on 2D image classification tasks. In these applications, jittering at training time usually includes random image translations (implemented as random crops), horizontal reflections, and color perturbations. At test time jittering usually only includes a few crops (*e.g.*, four at the corners, one at the center and their horizontal reflections). We now examine whether we can get more benefit out of jittered views of an image by using the same feature aggregation scheme we developed for recognizing 3D shapes.

The human sketch dataset [31] contains 20,000 hand-drawn sketches of 250 object categories such as airplanes, apples, bridges, *etc.* The accuracy of humans in recognizing these hand-drawings is only 73% because of the low quality of some sketches. Schneider and Tuytelaars [106] cleaned up the dataset by removing instances and categories that humans find hard to recognize. This cleaned dataset (SketchClean) contains 160 categories, on which humans can achieve 93% recognition accuracy. Using SIFT Fisher vectors with spatial pyramid pooling and linear SVMs, Schneider and Tuytelaars achieved 68.9% recognition accuracy on the original dataset and 79.0% on the SketchClean dataset. We split the SketchClean dataset randomly into training, validation and test set,[4] and report classification accuracy on the test set in Table 3.2.

With an off-the-shelf CNN (VGG-M from [15]), we are able to get 77.3% classification accuracy without any network fine-tuning. With fine-tuning on the training set, the accuracy can be further improved to 84.0%, significantly surpassing the Fisher vector approach. These numbers are achieved by using the penultimate layer (fc_7) in the network as image descriptors and linear SVMs.

Although it is impractical to get multiple views from 2D images, we can use jittering to mimic the effect of views. For each hand-drawn sketch, we do in-plane rotation with three angles: -45°, 0°, 45°, and also horizontal reflections (hence 6 samples per image). We apply the two CNN variants (regular CNN and MVCNN) discussed earlier for aggregating multiple views of 3D shapes, and get 85.5% (CNN w/o view-pooling) and 86.3% (MVCNN w/ view-pooling on fc_7) classification accuracy respectively. The latter also has the advantage of a single, more compact descriptor for each hand-drawn sketch.

With a deeper network architecture (VGG-VD, a network with 16 weight layers from [113]), we achieve 87.2% accuracy, further advancing the classification performance, and closely approaching human performance.

[4]The dataset does not come with a standard training/val/test split.

Examples of correctly and wrongly classified hand-drawn sketches are shown in Figure 3.5. Most misclassified sketches contain visually similar components with the target class, *e.g.* spider and crab have a similar layout of legs, and some very abstract sketches are difficult to recognize even for humans.

3.2.3 Sketch-based 3D shape retrieval

Due to the growing number of online 3D repositories, many approaches have been investigated to perform efficient 3D shape retrieval. Most online repositories (*e.g.* 3D Warehouse [123], TurboSquid [126], Shapeways [124]) provide only text-based search engines or hierarchical catalogs for 3D shape retrieval. However, it is hard to convey stylistic and geometric variations using only textual descriptions, thus sketch-based shape retrieval [142, 108, 32] has been proposed as an alternative for users to retrieve shapes with an approximate sketch of the desired 3D shape in mind. Sketch-based retrieval is challenging since it involves two heterogeneous data domains (hand-drawn sketches and 3D shapes), and sketches can be highly abstract and visually different from target 3D shapes. Here we demonstrate the potential strength of MVCNNs in sketch-based shape retrieval.

For this experiment, we construct a dataset containing 193 sketches and 790 CAD models from 10 categories existing in both SketchClean and ModelNet40. Following [32], we produce renderings of 3D shapes with a style similar to hand-drawn sketches (see Figure 3.6).

This is achieved by detecting Canny edges on the depth buffer (also known as z-buffer) from 12 viewpoints. These edge maps are then passed through CNNs to obtain image descriptors. Descriptors are also extracted from 6 perturbed samples of each query sketch in the manner described in Section 3.2.2. Finally we rank 3D shapes w.r.t. "average minimum distance" (Equation 3.1) to the sketch descriptors. Representative retrieval results are shown in Figure 3.7.

We are able to retrieve 3D objects from the same class with the query sketch, as well as being visually similar, especially in the top few matches. Our performance is 36.1% mAP on this dataset. Here we use the VGG-M network trained on ImageNet without any fine-tuning on either sketches or 3D shapes. With a fine-tuning procedure that optimizes a distance measure between hand-drawn sketches and 3D shapes, *e.g.*, by using a Siamese Network [22], retrieval performance can be further improved.

3.3 Conclusions

While the world is full of 3D shapes, as humans at least, we understand that world is mostly through 2D images. We have shown that using images of shapes as inputs to modern learning architectures, we can achieve performance better than any previously published results, including those that operate on direct 3D representations of shapes.

While even a näive usage of these multiple 2D projections yields impressive discrimination performance, by building descriptors that are aggregations of information from multiple views, we can achieve compactness, efficiency, and better accuracy. In addition, by relating the content of 3D shapes to 2D representations like sketches, we can retrieve these 3D shapes at high accuracy using sketches, and leverage the implicit knowledge of 3D shapes contained in their 2D views.

There are a number of directions to explore in future work. One is to experiment with different combinations of 2D views. Which views are the most informative? How many views are necessary for a given level of accuracy? Can informative views be selected on the fly?

Another obvious question is whether our view aggregating techniques can be used for building compact and discriminative descriptors for real-world 3D objects from multiple views, or automatically from video, rather than merely for 3D polygon mesh models. Such investigations could be immediately applicable to widely studied problems such as object recognition and face recognition.

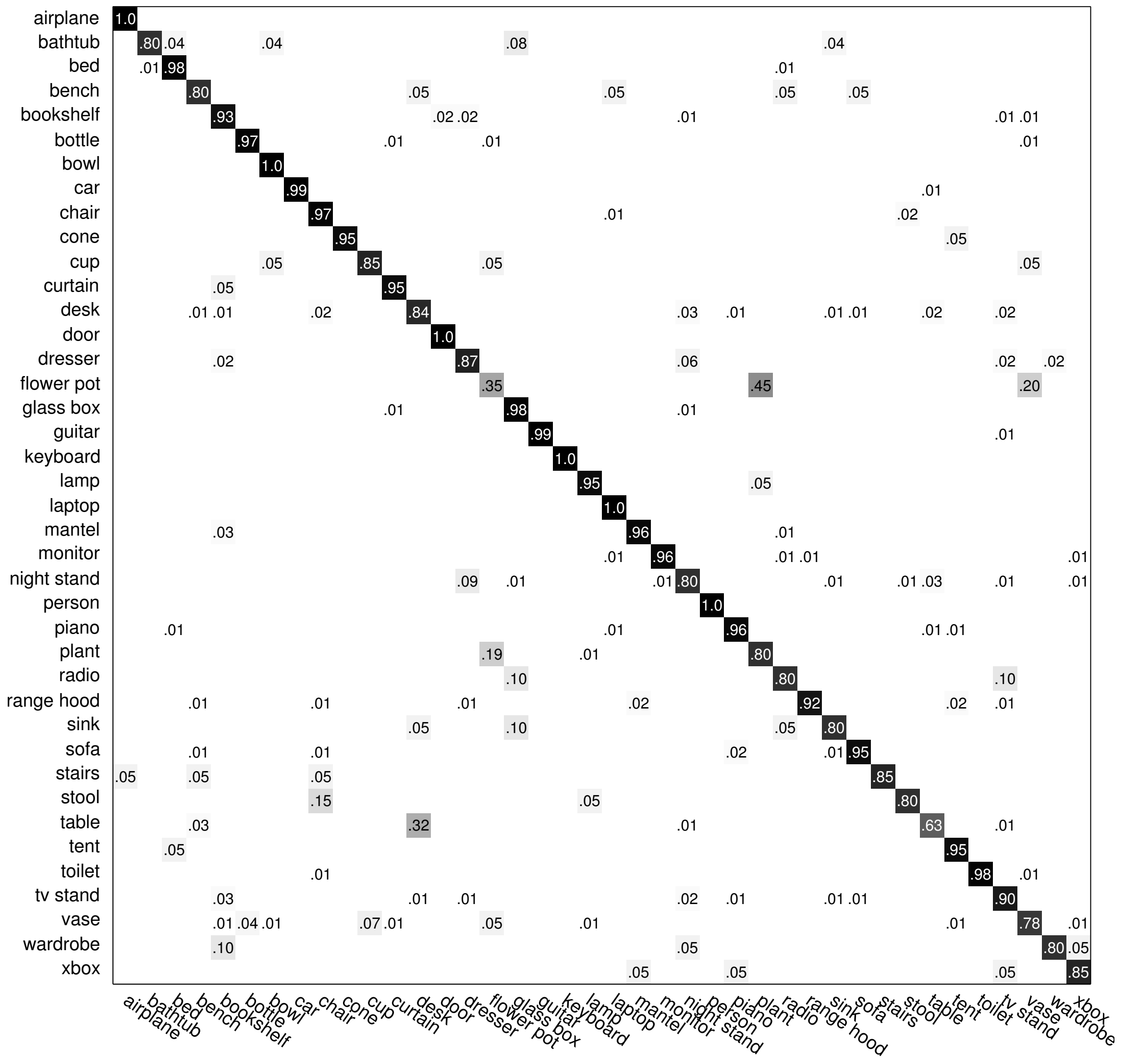


Figure 3.3: Confusion matrix of ModelNet40 classification.

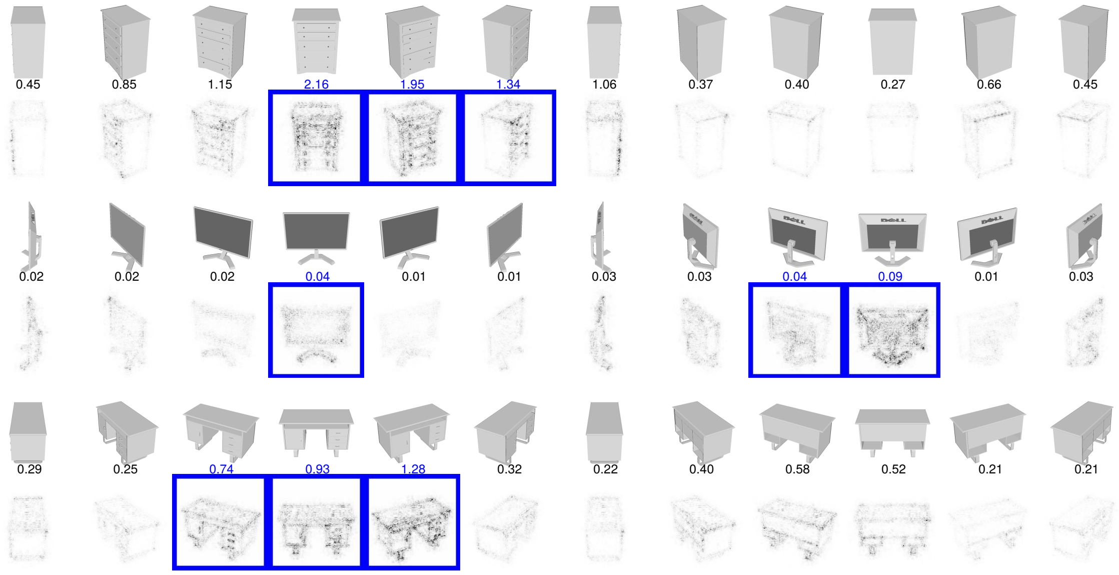


Figure 3.4: Saliency maps on samples from ModelNet. Top three views with the highest saliency are highlighted in blue and the relative magnitude of gradient energy for each view is shown on top. The saliency maps are computed by back-propagating the gradients of the class score onto the images via the view-pooling layer. Notice that the handles of the dresser are the most discriminative features. (Figures are contrast enhanced for visibility.)

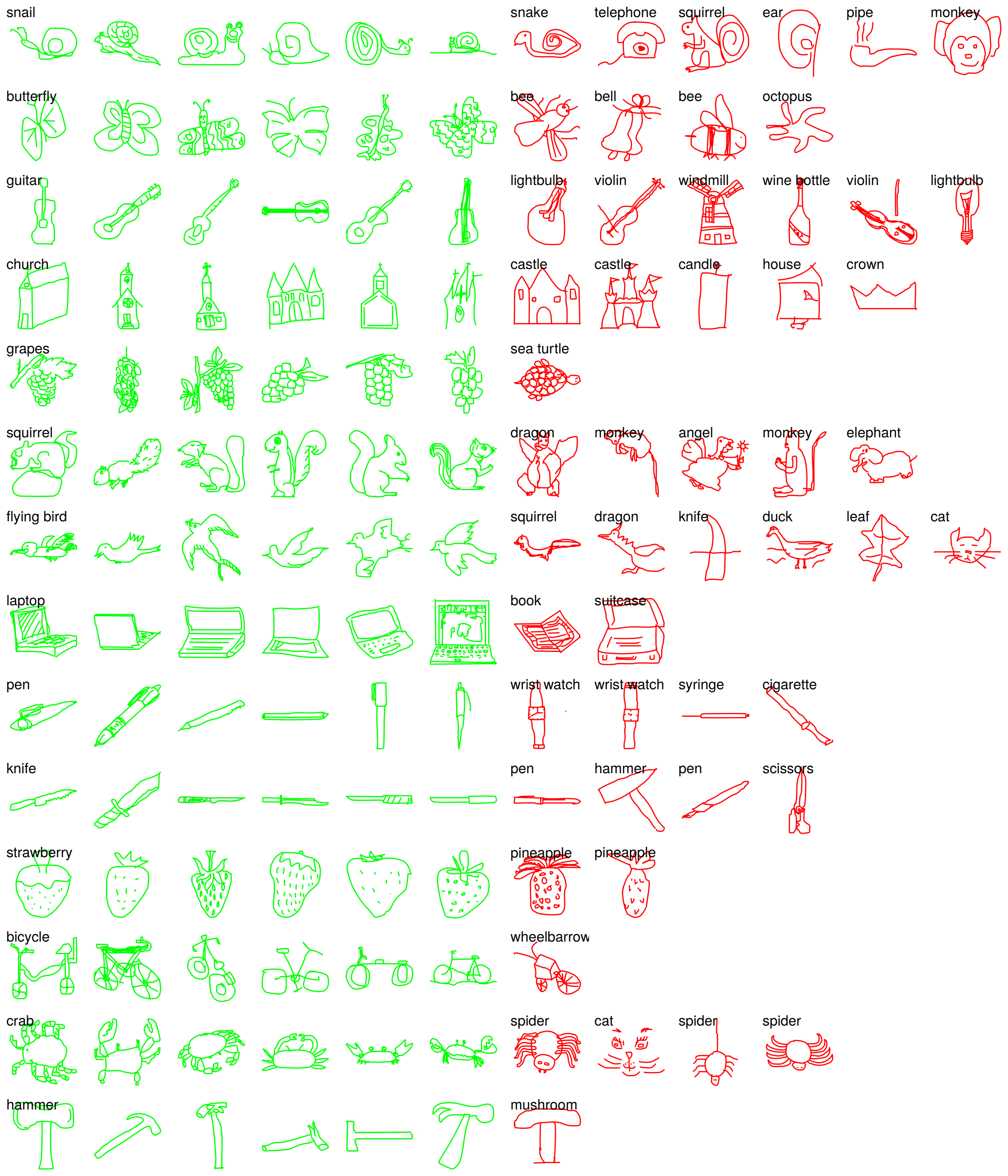


Figure 3.5: Examples of correctly and wrongly classified hand-drawn sketches. All sketches in each row are classified into the class labeled on top left. False positives are in red, with their ground truth classes labeled on top.

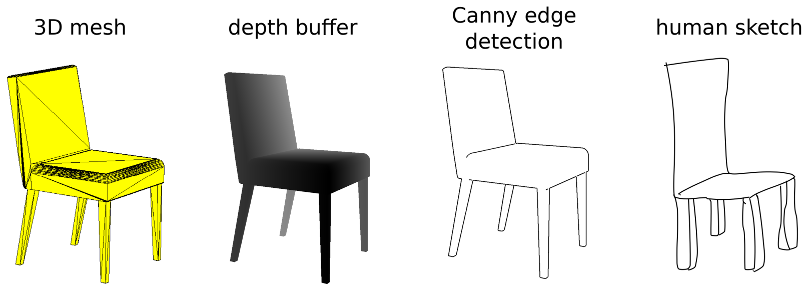


Figure 3.6: Line-drawing style rendering from 3D shapes.

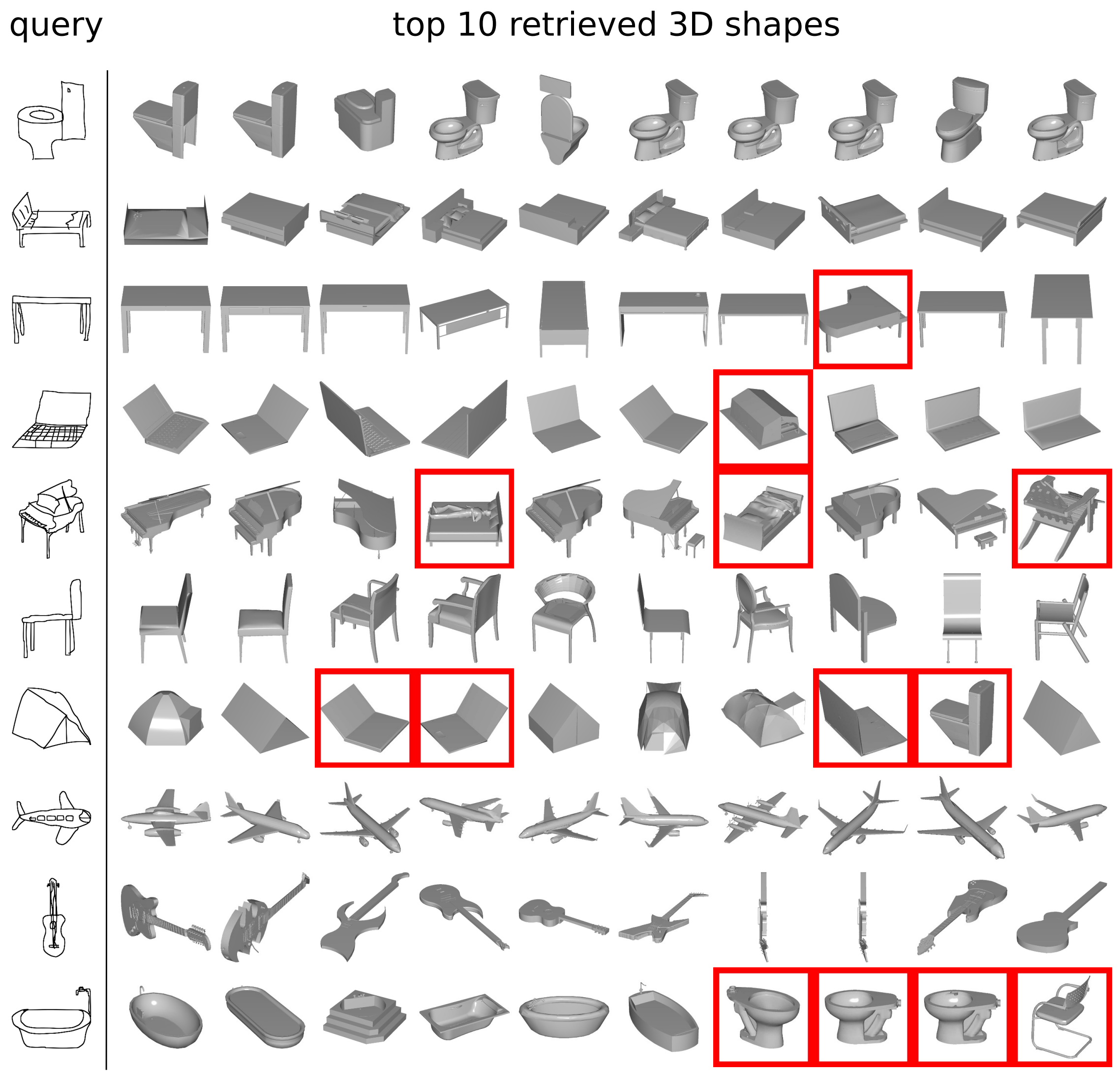


Figure 3.7: Sketch-based 3D shape retrieval examples. Top matches are shown for each query, with mistakes highlighted in red.

CHAPTER 4

3D POINT CLOUD PROCESSING WITH SPARSE LATTICE NETWORKS

In this chapter, we introduce a generic and flexible neural network architecture for processing point clouds that alleviates some of the aforementioned issues with existing deep architectures. Our key observation is that the bilateral convolution layers (BCLs) proposed in [57, 61] have several favorable properties for point cloud processing. BCL provides a systematic way of filtering unordered points while enabling flexible specifications of the underlying lattice structure on which the convolution operates. BCL smoothly maps input points onto a sparse lattice, performs convolutions on the sparse lattice and then smoothly interpolates the filtered signal back onto the original input points.

4.1 Bilateral Convolution Layer

In this section, we briefly review the Bilateral Convolution Layer (BCL) that forms the basic building block of our SPLATNet architecture for point clouds. BCL provides a way to incorporate sparse high-dimensional filtering inside neural networks. In [57, 61], BCL was proposed as a learnable generalization of bilateral filtering [121, 2], hence the name 'Bilateral Convolution Layer'. Bilateral filtering involves a projection of a given 2D image into a higher-dimensional space (*e.g.*, space defined by position and color) and is traditionally limited to hand-designed filter kernels. BCL provides a way to learn filter kernels in high-dimensional spaces for bilateral filtering. BCL is also shown to be useful for information propagation across video frames [56]. We observe

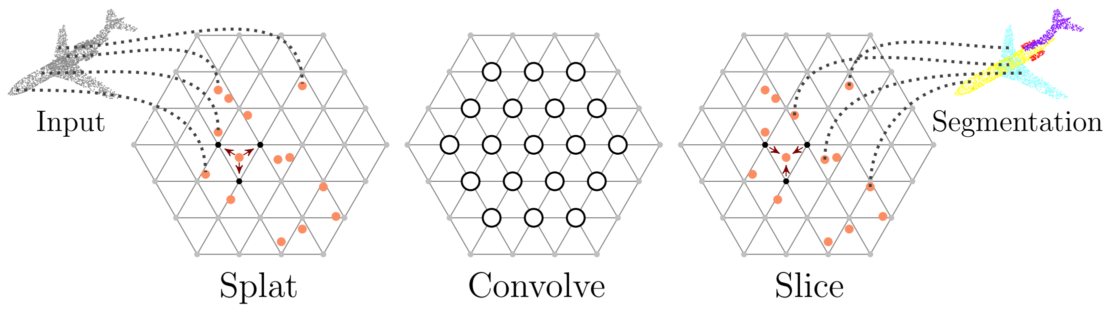


Figure 4.1: Bilateral Convolution Layer. *Splat*: BCL first interpolates input features F onto a d_l-dimensional permutohedral lattice defined by the lattice features L at input points. *Convolve*: BCL then does d_l-dimensional convolution over this sparsely populated lattice. *Slice*: The filtered signal is then interpolated back onto the input signal. For illustration, input and output are shown as point cloud and the corresponding segmentation labels.

that BCL has several favorable properties to filter data that is inherently sparse and high-dimensional, like point clouds. Here, we briefly describe how a BCL works and then discuss its properties.

4.1.1 Inputs to BCL

Let $F \in \mathbb{R}^{n \times d_f}$ be the given *input features* to a BCL, where n denotes the number of input points and d_f denotes the dimensionality of input features at each point. For 3D point clouds, input features can be low-level features such as color, position, *etc.*, and can also be high-level features such as features generated by a neural network.

One of the interesting characteristics of BCL is that it allows a flexible specification of the lattice space in which the convolution operates. This is specified as *lattice features* at each input point. Let $L \in \mathbb{R}^{n \times d_l}$ denote lattice features at input points with d_l denoting the dimensionality of the feature space in which convolution operates. For instance, the lattice features can be point position and color ($XYZRGB$) that define a 6-dimensional filtering space for BCL. For standard 3D spatial filtering of point clouds, L is given as the position (XYZ) of each point. Thus BCL takes input features F and lattice features L of input points and performs d_l-dimensional filtering of the points.

4.1.2 Processing steps in BCL

As illustrated in Figure 4.1, BCL has three processing steps, *splat*, *convolve* and *slice*, that work as follows.

1. **Splat.** BCL first projects the input features F onto the d_l-dimensional lattice defined by the lattice features L, via barycentric interpolation. Following [1], BCL uses a permutohedral lattice instead of a standard Euclidean grid for efficiency purposes. The size of lattice simplices or space between the grid points is controlled by scaling the lattice features ΛL, where Λ is a diagonal $d_l \times d_l$ scaling matrix.

2. **Convolve.** Once the input points are projected onto the d_l-dimensional lattice, BCL performs d_l-dimensional convolution on the splatted signal with learnable filter kernels. Just like in standard spatial CNNs, BCL allows an easy specification of filter neighborhood in the d_l-dimensional space.

3. **Slice.** The filtered signal is then mapped back to the input points via barycentric interpolation. The resulting signal can be passed on to other BCLs for further processing. This step is called "slicing". BCL allows slicing the filtered signal onto a different set of points other than the input points. This is achieved by specifying a different set of lattice features $L^{out} \in \mathbb{R}^{m \times d_l}$ at m output points of interest.

All the above three processing steps in BCL can be written as matrix multiplications:

$$\hat{F}_c = S_{slice} B_{conv} S_{splat} F_c, \tag{4.1}$$

where F_c denotes the c^{th} column/channel of the input feature F and $\hat{F}_c$ denotes the corresponding filtered signal.

4.1.3 Point cloud density normalization

BCL has a normalization scheme to deal with uneven point density, or more specifically, the fact that some lattice vertices are supported by more data points than others. Input signals are filtered directly with the learnable filter kernels, and are also filtered in a separate second round with their values replaced by 1s with a Gaussian kernel. The filter responses in the second round are then used for normalizing responses from the first round. This is similar to using homogeneous coordinates, which are widely adopted in bilateral filtering implementations such as [1].

4.1.4 Properties of BCL

There are several properties of BCL that makes it particularly convenient for point cloud processing. Here, we mention some of those properties:

- The input points to BCL need not be ordered or lie on a grid as they are projected onto a d_l-dimensional grid defined by lattice features L^{in}.
- The input and output points can be different for BCL with the specification of different input and output lattice features L^{in} and L^{out}.
- Since BCL allows separate specifications of input and lattice features, input signals can be projected into a different dimensional space for filtering. For instance, a 2D image can be projected into 3D space for filtering.
- Just like in standard spatial convolutions, BCL allows an easy specification of filter neighborhood.
- Since a signal is usually sparse in high-dimension, BCL uses hash tables to index the populated vertices and does convolutions only at those locations. This helps in efficient processing of sparse inputs.

Refer to [1] for more information about sparse high-dimensional Gaussian filtering on a permutohedral lattice and refer to [57] for more details on BCL.

4.2 Sparse Lattice Networks

With BCLs as building blocks, we propose a new neural network architecture, which we refer to as SPLATNet (SParse LATtice Networks), that does hierarchical and spatially-aware feature learning for unordered points.

SPLATNet has several advantages for point cloud processing:

- SPLATNet takes the point cloud as input and does not require any pre-processing to voxels or images.
- SPLATNet allows an easy specification of filter neighborhood as in standard CNN architectures.
- With the use of hash table, our network can efficiently deal with sparsity in the input point cloud by convolving only at locations where data is present.
- SPLATNet computes hierarchical and spatially-aware features of an input point cloud with sparse and efficient lattice filters.
- In addition, our network architecture allows an easy mapping of 2D points into 3D space and vice-versa. Following this, we propose a joint 2D-3D deep architecture that processes both the multi-view 2D images and the corresponding 3D point cloud in a single forward pass while being end-to-end learnable.

The inputs and outputs of two versions of the proposed network, SPLATNet$_{\text{3D}}$ and SPLATNet$_{\text{2D-3D}}$, are depicted in Figure 4.2. We demonstrate the above advantages with experiments on point cloud segmentation. Experiments on both RueMonge2014 facade segmentation [102] and ShapeNet part segmentation [140] demonstrate the superior performance of our technique compared to prior state-of-the-art techniques, while being computationally efficient.

4.2.1 3D point cloud processing with SPLATNet$_{\text{3D}}$

We first introduce SPLATNet$_{\text{3D}}$, an instantiation of our proposed network architecture which operates directly on 3D point clouds and is readily applicable to many important 3D tasks. The input to SPLATNet$_{\text{3D}}$ is a 3D point cloud $P \in \mathbb{R}^{n \times d}$, where n denotes the number of points and $d \geq 3$ denotes the number of feature dimensions including point locations XYZ. Additional features are often available either directly from 3D sensors or through pre-processing. These can be RGB color, surface normal, curvature, *etc.* at the input points. Note that input features F of the first BCL and lattice features L in the network each comprises a subset of the d feature dimensions: $d_f \leq d, d_l \leq d$.

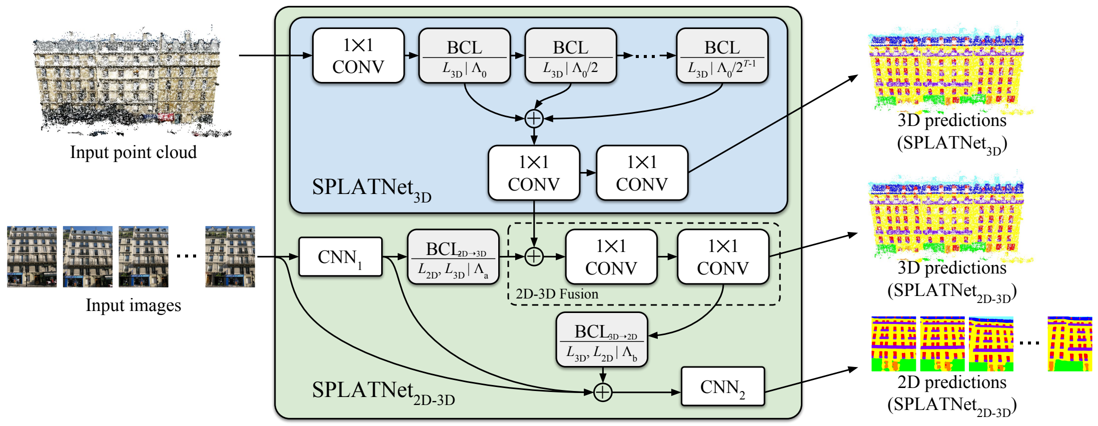

Input point cloud
Input images
1×1 CONV
BCL $L_{3D} | \Lambda_0$
BCL $L_{3D} | \Lambda_0/2$
BCL $L_{3D} | \Lambda_0/2^{T-1}$
SPLATNet$_{3D}$
CNN$_1$
BCL$_{2D \to 3D}$ $L_{2D}, L_{3D} | \Lambda_a$
2D-3D Fusion
BCL$_{3D \to 2D}$ $L_{3D}, L_{2D} | \Lambda_b$
CNN$_2$
SPLATNet$_{2D\text{-}3D}$
3D predictions (SPLATNet$_{3D}$)
3D predictions (SPLATNet$_{2D\text{-}3D}$)
2D predictions (SPLATNet$_{2D\text{-}3D}$)

Figure 4.2: Illustration of inputs, outputs and network architectures for SPLATNet$_{\text{3D}}$ and SPLATNet$_{\text{2D-3D}}$.

As output, SPLATNet$_{\text{3D}}$ produces per-point predictions. Tasks like 3D semantic segmentation and 3D object part labeling fit naturally under this framework. With simple techniques such as global pooling [95], SPLATNet$_{\text{3D}}$ can be modified to produce a single output vector and thus can be extended to other tasks such as classification.

4.2.1.1 Network architecture.

The architecture of SPLATNet$_{3\text{D}}$ is depicted in Figure 4.2. The network starts with a single 1×1 CONV layer followed by a series of BCLs. The 1×1 CONV layer processes each input point separately without any data aggregation. The functionality of BCLs is already explained in Section 4.1. For SPLATNet$_{3\text{D}}$, we use T BCLs each operating on a 3D lattice ($d_l = 3$) constructed using 3D point locations XYZ as lattice features, $L^{in} = L^{out} \in \mathbb{R}^{n \times 3}$. We note that different BCLs can use different lattice scales Λ. Recall from Section 4.1 that Λ is a diagonal matrix that controls the spacing between the grid points in the lattice. For BCLs in SPLATNet$_{3\text{D}}$, we use the same lattice scales along each of the X, Y and Z directions, *i.e.*, $\Lambda = \lambda I_3$, where λ is a scalar and I_3 denotes a 3×3 identity matrix. We start with an initial lattice scale λ_0 for the first BCL and subsequently divide the lattice scale by a factor of 2 ($\lambda_t = \lambda_{t-1}/2$) for the next $T-1$ BCLs. In other words, SPLATNet$_{3\text{D}}$ with T BCLs use the following lattice scales: $(\Lambda_0, \Lambda_0/2, \ldots, \Lambda_0/2^{T-1})$. Lower lattice scales imply coarser lattices and larger receptive fields for the filters. Thus, in SPLATNet$_{3\text{D}}$, deeper BCLs have longer-range connectivity between input points compared to earlier layers. We will discuss more about the effects of different lattice spaces and their scales later. Like in standard CNNs, SPLATNet allows an easy specification of filter neighborhoods. For all the BCLs, we use filters operating on one-ring neighborhoods and refer to the supp. material for details on the number of filters per layer.

The responses of the T BCLs are concatenated and then passed through two additional 1×1 CONV layers. Finally, a softmax layer produces point-wise class label probabilities. The concatenation operation aggregates information from BCLs operating at different lattice scales. Similar techniques of concatenating outputs from network layers at different depths have been useful in 2D CNNs [48]. All parameterized layers, except for the last CONV layer, are followed by ReLU and BatchNorm. More details about the network architecture are given in the supp. material.

4.2.1.2 Lattice spaces and their scales.

The use of BCLs in SPLATNet allows easy specifications of lattice spaces via lattice features and also lattice scales via a scaling matrix.

Changing the lattice scales Λ directly affects the resolution of the signal on which the convolution operates. This gives us direct control over the receptive fields of network layers. Figure 4.3 shows lattice cell visualizations for different lattice spaces and scales. Using coarser lattice can increase the effective receptive field of a filter. Another way to increase the receptive field of a filter is by increasing its neighborhood size. But, in high-dimensions, this will significantly increase the number of filter parameters. For instance, 3D filters of size $3, 5, 7$ on a regular Euclidean grid have $3^3 = 27, 5^3 = 125, 7^3 = 343$ parameters respectively. On the other hand, making the lattice coarser would not increase the number of filter parameters leading to more computationally efficient network architectures.

We observe that it is beneficial to use finer lattices (larger lattice scales) earlier in the network, and then coarser lattices (smaller lattice scales) going deeper. This is consistent with the common knowledge in 2D CNNs: increasing receptive field gradually through the network can help build hierarchical representations with varying spatial extents and abstraction levels.

Although we mainly experiment with XYZ lattices in this work, BCL allows for other lattice spaces such as position and color space ($XYZRGB$) or normal space. Using different lattice spaces enforces different connectivity across input points that may be beneficial to the task. In one of the experiments, we experimented with a variant of SPLATNet$_{\text{3D}}$, where we add an extra BCL with position and normal lattice features ($XYZn_xn_yn_z$) and observed minor performance improvements.

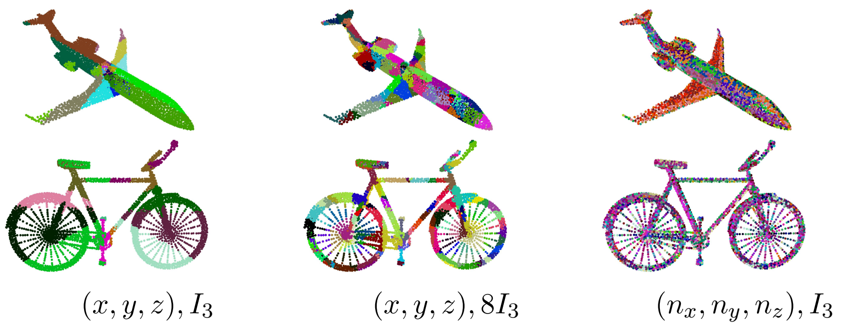


Figure 4.3: Effect of different lattice spaces and scales. Visualizations for different lattice feature spaces $L = (x, y, z), (x, y, z), (n_x, n_y, n_z)$ along with lattice scales $\Lambda = I_3, 8I_3, I_3$. (n_x, n_y, n_z) refers to point normals. All points falling in the same lattice cell are colored the same.

4.2.2 Joint 2D-3D processing with SPLATNet$_{\text{2D-3D}}$

Oftentimes, 3D point clouds are accompanied by 2D images of the same target. For instance, many modern 3D sensors capture RGBD streams and perform 3D reconstruction to obtain 3D point clouds, resulting in both 2D images and point clouds of a scene together with point correspondences between 2D and 3D. One could also easily sample point clouds along with 2D renderings from a given 3D mesh. When such aligned 2D-3D data is present, SPLATNet provides an extremely flexible framework for joint processing. We propose SPLATNet$_{\text{2D-3D}}$, another SPLATNet instantiation designed for such joint processing.

The network architecture of the SPLATNet$_{\text{2D-3D}}$ is depicted in the green box of Figure 4.2. SPLATNet$_{\text{2D-3D}}$ encompasses SPLATNet$_{\text{3D}}$ as one of its components and adds extra computational modules for joint 2D-3D processing. Next, we explain each of these extra components of SPLATNet$_{\text{2D-3D}}$, in the order of their computations.

CNN$_1$. First, we process the given multi-view 2D images using a 2D segmentation CNN, which we refer to as CNN$_1$. In our experiments, we use the DeepLab [19] architecture for CNN_1 and initialize the network weights with those pre-trained on PASCAL VOC segmentation [33].

$\mathbf{BCL_{2D\to 3D}}$. CNN_1 outputs features of the image pixels, whose 3D locations often do not exactly correspond to points in the 3D point cloud. We project information from the pixels onto the point cloud using a BCL with only *splat* and *slice* operations. As mentioned in Section 4.1, one of the interesting properties of BCL is that it allows for different input and output points by separate specifications of input and output lattice features, L^{in} and L^{out}. Using this property, we use BCL to *splat* 2D features onto the 3D lattice space and then *slice* the 3D splatted signal on the point cloud. We refer to this BCL, without a convolution operation, as $\mathrm{BCL_{2D\to 3D}}$ as illustrated in Figure 4.4. Specifically, we use 3D locations of the image pixels as input lattice features, $L^{in} = L_{2D} \in \mathbb{R}^{m\times 3}$, where m denotes the number of input image pixels. In addition, we use 3D locations of points in the point cloud as output lattice features, $L^{out} = L_{3D} \in \mathbb{R}^{n\times 3}$, which are the same lattice features used in $\mathrm{SPLATNet_{3D}}$. The lattice scale, Λ_a, controls the smoothness of the projection and can be adjusted according to the sparsity of the point cloud.

2D-3D Fusion. At this point, we have the result of CNN_1 projected onto 3D points and also the intermediate features from $\mathrm{SPLATNet_{3D}}$ that exclusively operates on the input point cloud. Since both of these signals are embedded in the same 3D space, we concatenate these two signals and then use a series of 1×1 CONV layers for further processing. The output of the '2D-3D Fusion' module is passed on to a softmax layer to compute class probabilities at each input point of the point cloud.

$\mathbf{BCL_{3D\to 2D}}$. Sometimes, we are also interested in segmenting 2D images and want to leverage relevant 3D information for better 2D segmentation. For this purpose, we

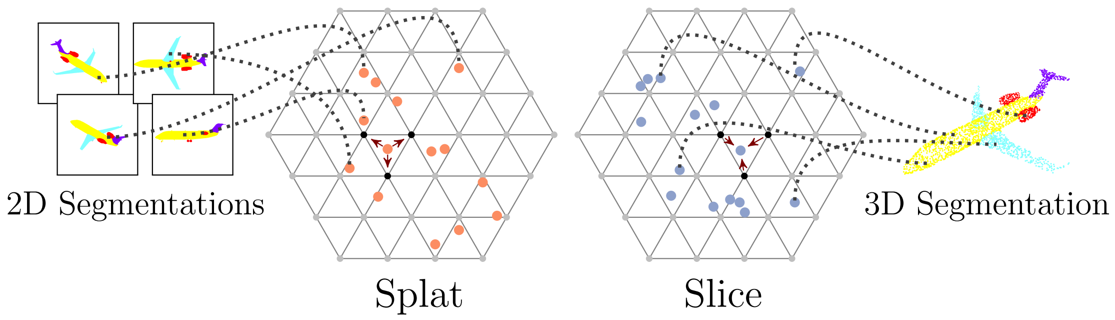


Figure 4.4: 2D to 3D projection. Illustration of 2D to 3D projection using *splat* and *slice* operations. Given input features of 2D images, pixels are projected onto a 3D permutohedral lattice defined by 3D positional lattice features. The splatted signal is then sliced onto the points of interest in a 3D point cloud.

back-project the 3D features computed by the '2D-3D Fusion' module onto the 2D images by a $\text{BCL}_{\text{2D}\rightarrow\text{3D}}$ module. This is the reverse operation of $\text{BCL}_{\text{2D}\rightarrow\text{3D}}$, where the input and output lattice features are swapped. Similarly, a hyper-parameter Λ_b controls the smoothness of the projection.

CNN_2. We then concatenate the output from CNN_1, input images and the output of $\text{BCL}_{\text{3D}\rightarrow\text{2D}}$, and pass them through another 2D CNN, CNN_2, to obtain refined 2D semantic predictions. In our experiments, we find that a simple 2-layered network is good enough for this purpose.

All components in this 2D-3D joint processing framework are differentiable, and can be trained end-to-end. Depending on the availability of 2D or 3D ground-truth labels, loss functions can be defined on either one of the two domains, or on both domains in a multi-task learning setting. More details of the network architecture are provided in the supp. material. We believe that this joint processing capability offered by $\text{SPLATNet}_{\text{2D-3D}}$ can result in better predictions for both 2D images and 3D point clouds. For 2D images, leveraging 3D features helps in view-consistent predictions

Table 4.1: Results on facade segmentation. Average IoU scores and approximate runtimes for point cloud labeling and 2D image labeling using different techniques. Runtimes indicate the time taken to segment the entire test data (202 images sequentially for 2D and a point cloud for 3D).

(a) Point cloud labeling

Method	Average IoU	Runtime (min)
With only 3D data		
OctNet [101]	59.2	-
Autocontext$_{\text{3D}}$ [37]	54.4	16
SPLATNet$_{\text{3D}}$ (Ours)	**65.4**	0.06
With both 2D and 3D data		
Autocontext$_{\text{2D-3D}}$ [37]	62.9	87
SPLATNet$_{\text{2D-3D}}$ (Ours)	**69.8**	1.20

(b) Multi-view image labeling

Method	Average IoU	Runtime (min)
Autocontext$_{\text{2D}}$ [37]	60.5	117
Autocontext$_{\text{2D-3D}}$ [37]	62.7	146
DeepLab$_{\text{2D}}$ [19]	69.3	0.84
SPLATNet$_{\text{2D-3D}}$ (Ours)	**70.6**	4.34

across multiple viewpoints. For point clouds, incorporating 2D CNNs help leverage powerful 2D deep CNN features computed on high-resolution images.

4.3 Experiments

We evaluate SPLATNet on tasks on two different benchmark datasets of RueMonge2014 [102] and ShapeNet [140]. On RueMonge2014, we conducted experiments on the tasks of 3D point cloud labeling and multi-view image labeling. On ShapeNet, we evaluated SPLATNet on 3D part segmentation. We use Caffe [59] neural network framework for all the experiments.

4.3.1 RueMonge2014 facade segmentation

Here, the task is to assign semantic label to every point in a point cloud and/or corresponding multi-view 2D images.

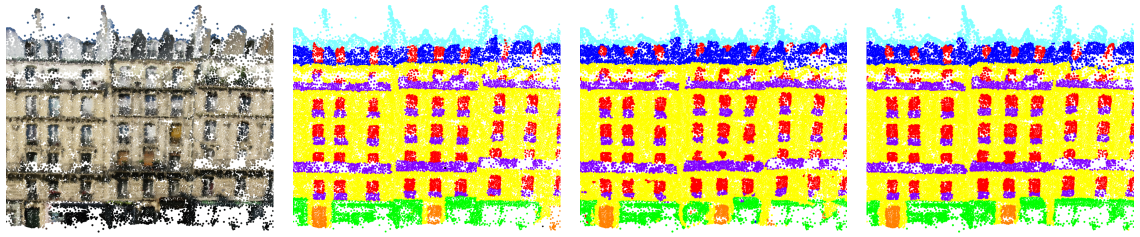


Figure 4.5: Sample visual results of SPLATNet$_{3D}$ and SPLATNet$_{2D\text{-}3D}$ for Facade point cloud labeling.

4.3.1.1 Dataset

RueMonge2014 [102] provides a standard benchmark for 2D and 3D facade segmentation and also inverse procedural modeling. The dataset consists of 428 high-resolution and multi-view images obtained from a street in Paris. A point cloud with approximately 1M points is reconstructed using the multi-view images. A ground-truth labeling with seven semantic classes of door, shop, balcony, window, wall, sky and roof are provided for both 2D images and the point cloud. Sample point cloud sections and 2D images with their corresponding ground truths are shown in Figure 4.5 and 4.6 respectively. For evaluation, Intersection over Union (IoU) score is computed for each of the seven classes and then averaged to get a single overall IoU.

4.3.1.2 Implementation details

We experiment with both SPLATNet variants for the facade segmentation task. Here we provide detailed description of the network architectures and the training setup.

Network architecture of SPLATNet$_{3D}$. We use 5 BCLs ($T = 5$) followed by 2 1×1 CONV layers in SPLATNet$_{3D}$ for the facade segmentation task. We omit the initial 1×1 CONV layer since we find it has no effect on the overall performance.

The number of output channels in each layer are: `B64-B128-B128-B128-B64-C64-C7`. Note that although written as a linear structure, the network has skip connections from all BCLs (layers start with "`B`") to the penultimate 1×1 CONV layer. We use an initial scale $\Lambda_0 = 32I_3$ for scaling lattice features XYZ, and divide the scale in half after each BCL: $32I_3, 16I_3, 8I_3, 4I_3, 2I_3$. Input features to the network comprise of a 7-dimensional vector at each point representing RGB color, normal and height above the ground. The unit of raw XYZ inputs is meter, with Y (aligned with gravity axis) having a range of 7.1 meters. For all the BCLs, we use filters operating on one-ring neighborhoods on the lattice.

Network architecture of SPLATNet$_{\text{2D-3D}}$. We use SPLATNet$_{\text{3D}}$ as described above as the 3D component of the 2D-3D joint model. The "2D-3D Fusion" component has 2 1×1 CONV layers: `C64-C7`. DeepLab [19] segmentation architecture is used as CNN$_1$. CNN$_2$ is a small network with 2 CONV layers: `C32-C7`, where the first layer has 3×3 filters and 32 output channels, and the second one has 1×1 filters and 7 output channels. We use $\Lambda_a = 64$ and $\Lambda_b = 1000$ for 2D$\leftrightarrow$3D projections with BCLs. Note that the dataset provides one-to-many mappings from 3D points to pixels. By using a very large scale (*i.e.*, $\Lambda_b = 1000$), 3D unaries are directly mapped to the corresponding 2D pixel locations without any interpolation.

Training. We randomly sample facade segments of 60k points and use a batch size of 4 when training SPLATNet$_{\text{3D}}$. CNN$_1$ is initialized with Pascal VOC [33] pre-trained weights and fine-tuned for 2D facade segmentation. Adam optimizer [62] with an initial learning rate of 0.0001 is used for training both SPLATNet$_{\text{3D}}$ and SPLATNet$_{\text{2D-3D}}$. Since the training data is small, we augment point cloud training data with random rotations, translations, and small color perturbations. We also augment 2D image data with small color perturbations during training.

4.3.1.3 Results on point cloud labeling

Experimental results with average IoU and runtime are shown in Table 4.1a. Results show that, with only 3D data, our method achieves an IoU of 65.4 which is a considerable improvement (6.2 IoU $\uparrow$) over the state-of-the-art deep network, OctNet [101].

Since this dataset comes with multi-view 2D images, one could leverage the information present in 2D data for better point cloud labeling. We use SPLATNet$_{\text{2D-3D}}$ to leverage 2D information and obtain better 3D segmentations. Table 4.1a shows the experimental results when using both the 2D and 3D data as input. SPLATNet$_{\text{2D-3D}}$ obtains an average IoU of 69.8 outperforming the previous state-of-the-art by a large margin (6.9 IoU $\uparrow$), thereby setting up a new state-of-the-art on this dataset. This is also a significant improvement from the IoU obtained with SPLATNet$_{\text{3D}}$ demonstrating the benefit of leveraging 2D and 3D information in a joint framework. Runtimes in Table 4.1a also indicate that our SPLATNet approach is much faster compared to traditional Autocontext techniques. Sample visual results for 3D facade labeling are shown in Figure 4.5.

4.3.1.4 Results on multi-view image labeling

Table 4.1b shows the results of multi-view image labeling on this dataset using different techniques. Using DeepLab (CNN$_1$) already outperforms existing state-of-the-art by a large margin. Leveraging 3D information via SPLATNet$_{\text{2D-3D}}$ boosts the performance to 70.6 IoU. An increase of 1.3 IoU from only using CNN$_1$ demonstrates the potential of our joint 2D-3D framework in leveraging 3D information for better 2D segmentation.

4.3.2 ShapeNet part segmentation

The task of part segmentation is to assign a part category label to each point in a point cloud representing a 3D object.

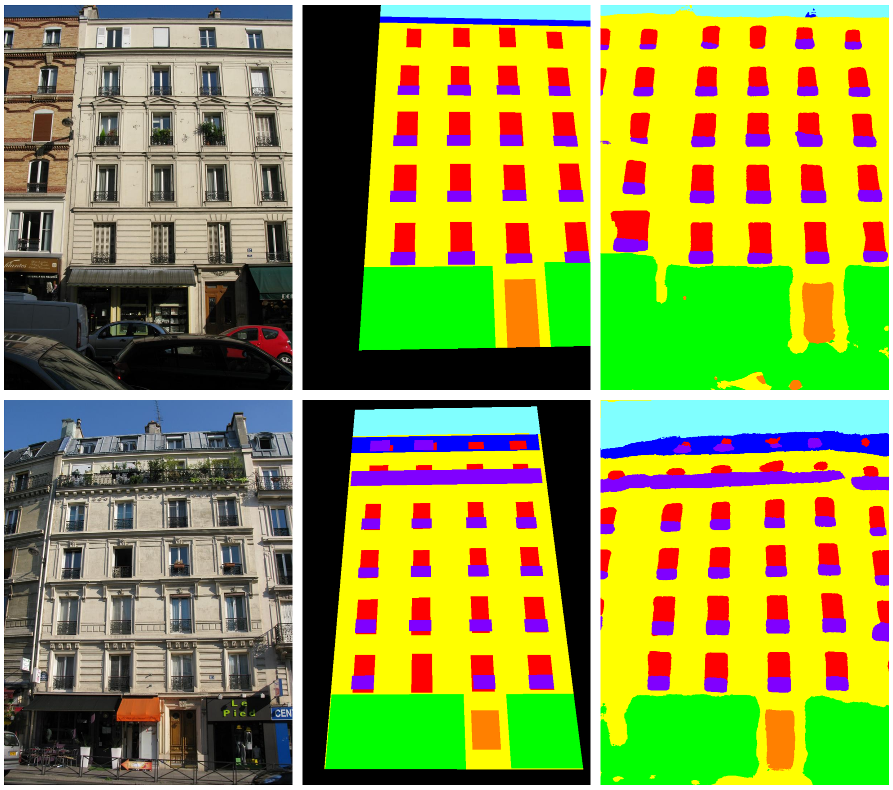


Figure 4.6: Sample visual results of $\text{SPLATNet}_{\text{2D-3D}}$ for 2D facade segmentation.

4.3.2.1 Dataset

The ShapeNet Part dataset [140] is a subset of ShapeNet, which contains 16681 objects from 16 categories, each with 2-6 part labels. The objects are consistently aligned and scaled to fit into a unit cube, and the ground-truth annotations are provided on sampled points on the shape surfaces. It is common to assume that the category of the input 3D object is known, narrowing the possible part labels to the ones specific to the given object category. We report standard IoU scores for evaluation of part segmentation. An IoU score is computed for each object and then averaged within the objects in a category to compute mean IoU (mIoU) for each object category. In addition to reporting mIoU score for each object category, we also

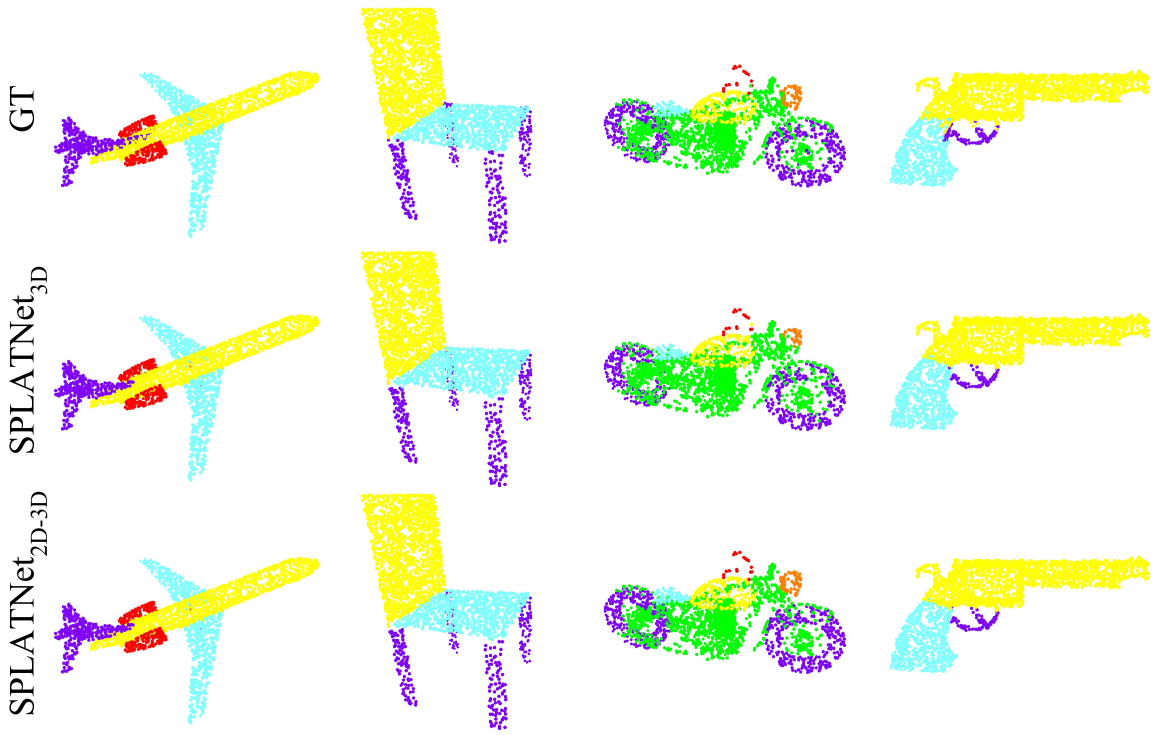


Figure 4.7: Sample visual results of SPLATNet$_{\text{3D}}$ and SPLATNet$_{\text{2D-3D}}$ for ShapeNet part segmentation.

report “class average mIoU” which is the average mIoU across all object categories, and also “instance average mIoU”, which is the average mIoU across all objects.

4.3.2.2 Implementation details

We experiment with both SPLATNet variants for the object part segmentation task. Here we provide detailed description of the network architectures and the training setup.

Network architecture of SPLATNet$_{\text{3D}}$. We use a 1×1 CONV layer in the beginning, followed by 5 BCLs ($T = 5$), and then 2 $1{\times}1$ CONV layers in SPLATNet$_{\text{3D}}$ for the ShapeNet part segmentation task. The number of output channels in each layer are: `C32-B64-B128-B256-B256-B256-C128-Cx`. “x” in the last CONV layer denotes the number of part categories, and ranges from 2-6 for different object categories. Point locations XYZ are used as input features as well as lattice features L for all the BCLs. We use an initial scale $\Lambda_0 = 64I_3$ for scaling lattice features XYZ, and divide the scale in half after each BCL: $64I_3, 32I_3, 16I_3, 8I_3, 4I_3$.

Network architecture of SPLATNet$_{\mathbf{2D\text{-}3D}}$. We use SPLATNet$_{\text{3D}}$ as described above as the 3D component of the joint model. The "2D-3D Fusion" component has 2 1×1 CONV layers: `C128-Cx`. The same DeepLab architecture is used for CNN$_1$. We use $\Lambda_a = 32$ in BCL$_{\text{2D}\rightarrow\text{3D}}$. Since 2D prediction is not needed, CNN$_2$ and BCL$_{\text{3D}\rightarrow\text{2D}}$ are omitted.

Training. Since the category of the input object is assumed to be known, we train separate networks for each object category. CNN$_1$ is initialized the same way as in the facade experiment. Adam optimizer with an initial learning rate of 0.0001 is used. We augment point cloud data with random rotations, translations, and scalings during training. We train our networks until validation loss plateaus. Training SPLATNet$_{\text{3D}}$ and SPLATNet$_{\text{2D-3D}}$ take about 2.5 and 3 days respectively on a single graphics card (Nvidia GeForce GTX 1080 Ti). With default settings, training PointNet++ takes 3.5 days on the same hardware.

4.3.2.3 Results on 3D object part segmentation

We evaluate both SPLATNet$_{\text{3D}}$ and SPLATNet$_{\text{2D-3D}}$ for this task.

SPLATNet$_{\text{3D}}$ uses only 3D point clouds as input. Experimental results are shown in Table 4.2. SPLATNet$_{\text{3D}}$ obtains a class average mIoU of 82.0 and an instance average mIoU of 84.6, which is on-par with the best networks that only take point clouds as input (PointNet++ [96] uses surface normals as additional inputs).

We also adopt the SPLATNet$_{\text{2D-3D}}$ network, which operates on both 2D and 3D data, for this task. For the joint framework to work, we need rendered 2D views and corresponding 3D locations for each pixel in the renderings. We first render 3-channel images: Phong shading [93], depth, and height from ground. Cameras are placed on the 20 vertices of a dodecahedron from a fixed distance, pointing towards the object's center. The 2D-3D correspondences can be generated by carrying the XYZ coordinates of 3D points into the rendering rasterization pipeline so that each pixel also acquires coordinate values from the surface point projected onto it. Results in Table 4.2 show that incorporating 2D information allows SPLATNet$_{\text{2D-3D}}$ to improve noticeably from SPLATNet$_{\text{3D}}$ with 1.7 and 0.8 increase in class and instance average mIoU respectively. SPLATNet$_{\text{2D-3D}}$ obtains a class average IoU of 83.7 and an instance average IoU of 85.4, outperforming existing state-of-the-art approaches.

Table 4.2: Results on ShapeNet part segmentation showing class average mIoU, instance average mIoU and mIoU scores for all the categories on the task of point cloud labeling using different techniques.

#instances			2690	76	55	898	3758	69	787	392	1547	451	202	184	283	66	152	5271
	class avg.	instance avg.	air-plane	bag	cap	car	chair	ear-phone	guitar	knife	lamp	laptop	motor-bike	mug	pistol	rocket	skate-board	table
Yi *et al.* [140]	79.0	81.4	81.0	78.4	77.7	75.7	87.6	61.9	92.0	85.4	82.5	95.7	70.6	91.9	**85.9**	53.1	69.8	75.3
3DCNN [95]	74.9	79.4	75.1	72.8	73.3	70.0	87.2	63.5	88.4	79.6	74.4	93.9	58.7	91.8	76.4	51.2	65.3	77.1
Kd-network [63]	77.4	82.3	80.1	74.6	74.3	70.3	88.6	73.5	90.2	**87.2**	81.0	94.9	57.4	86.7	78.1	51.8	69.9	80.3
PointNet [95]	80.4	83.7	**83.4**	78.7	82.5	74.9	89.6	73.0	91.5	85.9	80.8	95.3	65.2	93.0	81.2	57.9	72.8	80.6
PointNet++ [96]	81.9	85.1	82.4	79.0	87.7	77.3	**90.8**	71.8	91.0	85.9	83.7	95.3	71.6	94.1	81.3	58.7	76.4	**82.6**
SyncSpecCNN [141]	82.0	84.7	81.6	81.7	81.9	75.2	90.2	74.9	**93.0**	86.1	**84.7**	95.6	66.7	92.7	81.6	60.6	**82.9**	82.1
SPLATNet$_{3D}$	82.0	84.6	81.9	83.9	88.6	79.5	90.1	73.5	91.3	84.7	84.5	**96.3**	69.7	95.0	81.7	59.2	70.4	81.3
SPLATNet$_{2D\text{-}3D}$	**83.7**	**85.4**	83.2	**84.3**	**89.1**	**80.3**	90.7	**75.5**	92.1	87.1	83.9	**96.3**	**75.6**	**95.8**	83.8	**64.0**	75.5	81.8

On one Nvidia GeForce GTX 1080 Ti, SPLATNet$_{\text{3D}}$ runs at 9.4 shapes/sec, while SPLATNet$_{\text{2D-3D}}$ is slower at 0.4 shapes/sec due to a relatively large 2D network operating on 20 high-resolution (512×512) views, which takes up more than 95% of the computation time. In comparison, PointNet++ runs at 2.7 shapes/sec on the same hardware[1].

4.3.2.4 Filtering in higher-dimensional spaces

We experiment with a variant of SPLATNet$_{\text{3D}}$ where an additional BCL with 6-dimensional position and normal lattice features ($XYZn_xn_yn_z$) is added between the last two 1×1 CONV layers. This modification gave only a marginal improvement of 0.2 IoU over standard SPLATNet$_{\text{3D}}$ in terms of both class and instance average mIoU scores.

4.3.2.5 Limitations with the dataset

We observed a few types of labeling issues in the ShapeNet Part dataset:

- Some object part categories are frequently labeled incorrectly. *E.g.*, skateboard axles are often mistakenly labeled as “deck” or “wheel” (Figure 4.8a).
- Some object parts, *e.g.* “fin” of some rockets, have incomplete range or coverage (Figure 4.8b).
- Some object part categories are labeled inconsistently between shapes. *E.g.*, airplane landing gears are seen labeled as “body”, “engine”, or “wings” (Figure 4.8c).
- Some categories have parts that are labeled as “other”, which can be confusing for the classifier as these parts do not have clear semantic meanings or

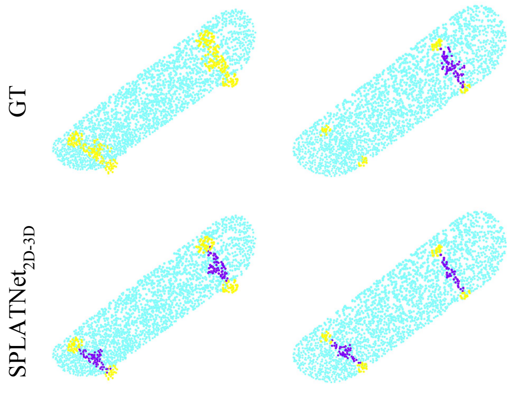
[1]We use the public implementation released by the authors (`https://github.com/charlesq34/pointnet2`) with settings: model = ‘pointnet2_part_seg_msg_one_hot’, VOTE_NUM = 12, num_point = 3000 (in consistence with our experiments).

structures. *E.g.*, in the case of earphones, anything that is not "headband" or "earphone" are given the same label ("other") (Figure 4.8d).

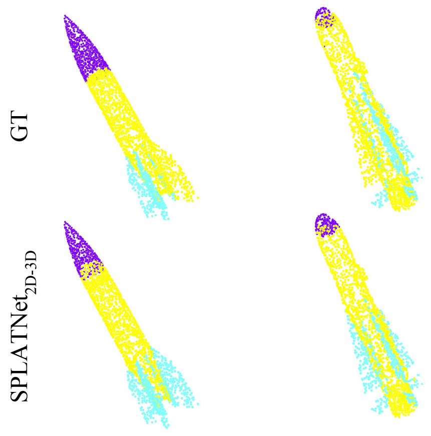
The first two issues make evaluations and comparisons on the benchmark less reliable, while the other two make learning ill-posed or unnecessarily hard for the networks. Thus we suspect pushing the performance further would pose a potential risk of overfitting the dataset.

4.4 Conclusions

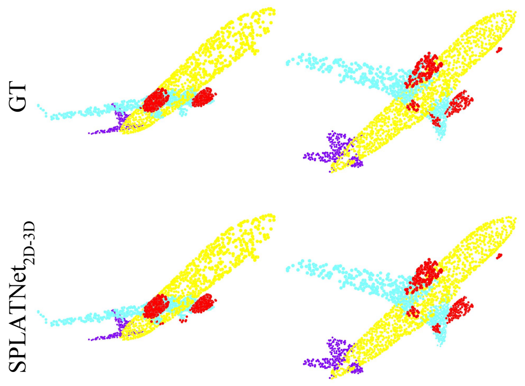
In this chapter, we propose the SPLATNet architecture for point cloud processing. SPLATNet directly takes point clouds as input and computes hierarchical and spatially-aware features with sparse and efficient lattice filters. In addition, SPLATNet allows an easy mapping of 2D information into 3D and vice-versa, resulting in a novel network architecture for joint processing of point clouds and multi-view images. Experiments on two different benchmark datasets show that the proposed networks compare favorably against state-of-the-art approaches for segmentation tasks. In the future, we would like to explore the use of additional input features (*e.g.*, texture) and also the use of other high-dimensional lattice spaces in our networks.



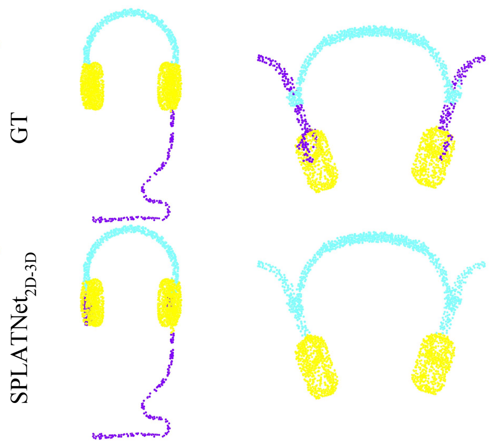
(a) Incorrect labels



(b) Incomplete labels



(c) Inconsistent labels



(d) Confusing labels

Figure 4.8: Four types of labeling issues in the ShapeNet Part dataset are shown here. Two examples from the test set are given for each type, where the first row shows the ground-truth labels and the second row shows our predictions with SPLATNet$_{2D\text{-}3D}$. Our predictions appear to be more accurate than the ground truth in some cases (see the skateboard axles in Figure 4.8a and the rocket fins in Figure 4.8b).

CHAPTER 5

PIXEL-ADAPTIVE CONVOLUTIONAL NEURAL NETWORKS

Convolution is a basic operation in many image processing and computer vision applications and the major building block of CNN architectures. It forms one of the most prominent ways of propagating and integrating features across image pixels due to its simplicity and highly optimized CPU/GPU implementations. In this chapter, we concentrate on two important characteristics of standard spatial convolution and aim to alleviate some of its drawbacks: *Spatial Sharing* and its *Content-Agnostic* nature.

Spatial Sharing: A typical CNN shares filters' parameters across the whole input. In addition to affording translation invariance to the CNN, spatially invariant convolutions significantly reduce the number of parameters compared with fully connected layers. However, spatial sharing is not without drawbacks. For dense pixel prediction tasks, such as semantic segmentation, the loss is spatially varying because of varying scene elements on a pixel grid. Thus the optimal gradient direction for parameters differs at each pixel. However, due to the spatial sharing nature of convolution, the loss gradients from all image locations are globally pooled to train each filter. This forces the CNN to learn filters that minimize the error across all pixel locations at once, but may be sub-optimal at any specific location.

Content-Agnostic: Once a CNN is trained, the same convolutional filter banks are applied to all the images and all the pixels irrespective of their content. The image content varies substantially across images and pixels. Thus a single trained CNN may not be optimal for all image types (*e.g.* images taken in daylight and at night) as well as different pixels in an image (*e.g.* sky vs. pedestrian pixels). Ideally, we would like CNN filters to be adaptive to the type of image content, which is not the case with standard CNNs. These drawbacks can be tackled by learning a large number of filters in an attempt to capture both image and pixel variations. This, however, increases the number of parameters, requiring a larger memory footprint and an extensive amount of labeled data. A different approach is to use *content-adaptive* filters inside the networks.

Existing content-adaptive convolutional networks can be broadly categorized into two types. One class of techniques make traditional image-adaptive filters, such as bilateral filters [2, 121] and guided image filters [50] differentiable, and use them as layers inside a CNN [69, 76, 147, 21, 18, 57, 77, 14, 36, 79, 131, 134]. These content-adaptive layers are usually designed for enhancing CNN results but not as a replacement for standard convolutions. Another class of content-adaptive networks involve learning position-specific kernels using a separate sub-network that predicts convolutional filter weights at each pixel. These are called "Dynamic Filter Networks" (DFN) [139, 58, 26, 135] (also referred to as cross-convolution [139] or kernel prediction networks [4]) and have been shown to be useful in several computer vision tasks. Although DFNs are generic and can be used as a replacement to standard convolution layers, such a kernel prediction strategy is difficult to scale to an entire network with a large number of filter banks.

In this chapter, we propose a new content-adaptive convolution layer that addresses some of the limitations of the existing content-adaptive layers while retaining several favorable properties of spatially invariant convolution. Figure 5.1 illustrates our content-adaptive convolution operation, which we call "Pixel-Adaptive Convolution" (PAC). Unlike a typical DFN, where different kernels are predicted at different pixel locations, we *adapt* a standard spatially invariant convolution filter $\mathbf{W}$ at each

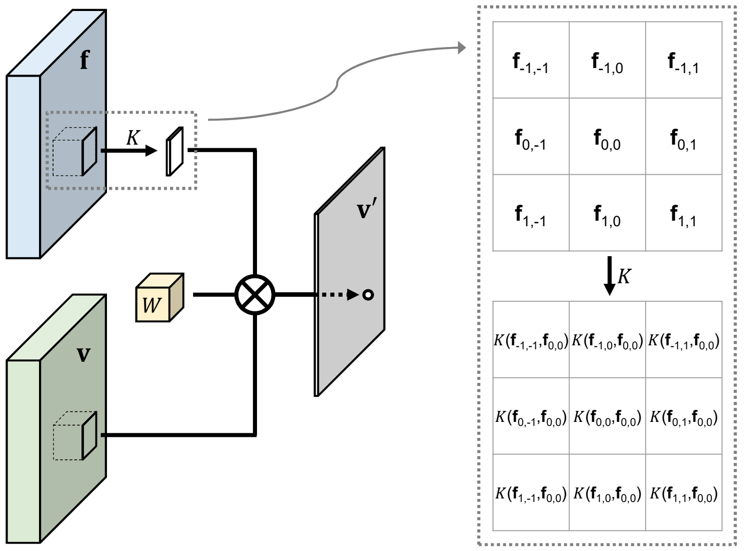

f
K
v′
W
v
$\mathbf{f}_{-1,-1}$
$\mathbf{f}_{-1,0}$
$\mathbf{f}_{-1,1}$
$\mathbf{f}_{0,-1}$
$\mathbf{f}_{0,0}$
$\mathbf{f}_{0,1}$
$\mathbf{f}_{1,-1}$
$\mathbf{f}_{1,0}$
$\mathbf{f}_{1,1}$
K
$K(\mathbf{f}_{-1,-1},\mathbf{f}_{0,0})$
$K(\mathbf{f}_{-1,0},\mathbf{f}_{0,0})$
$K(\mathbf{f}_{-1,1},\mathbf{f}_{0,0})$
$K(\mathbf{f}_{0,-1},\mathbf{f}_{0,0})$
$K(\mathbf{f}_{0,0},\mathbf{f}_{0,0})$
$K(\mathbf{f}_{0,1},\mathbf{f}_{0,0})$
$K(\mathbf{f}_{1,-1},\mathbf{f}_{0,0})$
$K(\mathbf{f}_{1,0},\mathbf{f}_{0,0})$
$K(\mathbf{f}_{1,1},\mathbf{f}_{0,0})$

Figure 5.1: Pixel-Adaptive Convolution (PAC) modifies a standard convolution on an input $\mathbf{v}$ by modifying the spatially invariant filter $\mathbf{W}$ with an adapting kernel K. The adapting kernel is constructed using either pre-defined or learned features $\mathbf{f}$. $\otimes$ denotes element-wise multiplication of matrices followed by a summation. Only one output channel is shown for the illustration.

pixel by multiplying it with a spatially varying filter K, which we refer to as an "adapting kernel". This adapting kernel has a pre-defined form (e.g., Gaussian or Laplacian) and depends on the pixel features. For instance, the adapting kernel that we mainly use in this work is Gaussian: $e^{-\frac{1}{2}||\mathbf{f}_i-\mathbf{f}_j||^2}$, where $\mathbf{f}_i \in \mathbb{R}^d$ is a d-dimensional feature at the i^{th} pixel. We refer to these pixel features $\mathbf{f}$ as "adapting features", and they can be either pre-defined, such as pixel position and color features, or can be learned using a CNN.

We observe that PAC, despite being a simple modification to standard convolution, is highly flexible and can be seen as a generalization of several widely-used filters. Specifically, we show that PAC is a generalization of spatial convolution, bilateral filtering [2, 121], and pooling operations such as average pooling and detail-preserving pooling [103]. We also implement a variant of PAC that does pixel-adaptive transposed convolution (also called deconvolution) which can be used for learnable guided upsampling of intermediate CNN representations. We discuss more about these generalizations and variants in Section 5.2.

As a result of its simplicity and being a generalization of several widely used filtering techniques, PAC can be useful in a wide range of computer vision problems. In this work, we demonstrate its applicability in three different vision problems. In Section 5.3, we use PAC in joint image upsampling networks and obtain state-of-the-art results on both depth and optical flow upsampling tasks. In Section 5.4, we use PAC in a learnable conditional random field (CRF) framework and observe consistent improvements with respect to the widely used fully-connected CRF [69]. In Section 5.5, we demonstrate how to use PAC as a drop-in replacement of trained convolution layers in a CNN and obtain performance improvements after fine-tuning. In summary, we observe that PAC is highly versatile and has wide applicability in a range of computer vision tasks.

5.1 Related Work

5.1.1 Content-adaptive filtering

Some important content-adaptive filtering techniques include bilateral filtering [2, 121], guided image filtering [50], non-local means [12, 3], and propagated image filtering [99], to name a few. A common line of research is to make these filters differentiable and use them as content-adaptive CNN layers. Early work [147, 21] in this direction back-propagates through bilateral filtering and can thus leverage fully-connected CRF inference [69] on the output of CNNs. The work of [57] and [36] proposes to use bilateral filtering layers inside CNN architectures. Chandra *et al.* [14] propose a layer that performs closed-form Gaussian CRF inference in a CNN. Chen *et al.* [18] and Liu *et al.* [79] propose differentiable local propagation modules that have roots in domain transform filtering [39]. Wu *et al.* [134] and Wang *et al.* [131] propose neural network layers to perform guided filtering [50] and non-local means [131] respectively inside CNNs. Since these techniques are tailored towards a particular CRF or adaptive filtering technique, they are used for specific tasks and cannot be directly used as a replacement of general convolution. Closest to our work are the sparse, high-dimensional neural networks [57] which generalize standard 2D convolutions to high-dimensional convolutions, enabling them to be content-adaptive. Although conceptually more generic than PAC, such high-dimensional networks can not learn the adapting features and have a larger computational overhead due to the use of specialized lattices and hash tables.

5.1.2 Dynamic filter networks

Introduced by Jia *et al.* [58], dynamic filter networks (DFN) are an example of another class of content-adaptive filtering techniques. Filter weights are themselves directly predicted by a separate network branch, and provide custom filters specific to different input data. The work is later extended by Wu *et al.* [135] with an additional attention mechanism and a dynamic sampling strategy to allow the position-specific kernels to also learn from multiple neighboring regions. Similar ideas have been applied to several task-specific use cases, *e.g.* motion prediction [139], semantic segmentation [49], and Monte Carlo rendering denoising [4]. Explicitly predicting all position-specific filter weights requires a large number of parameters, so DFNs typically require a sensible architecture design and are difficult to scale to multiple dynamic-filter layers. Our approach differs in that PAC reuses spatial filters just as standard convolution, and only modifies the filters in a position-specific fashion. Dai *et al.* propose deformable convolution [26], which can also produce position-specific modifications to the filters. Different from PAC, the modifications there are represented as *offsets* with an emphasis on learning geometric-invariant features.

5.1.3 Self-attention mechanism

PAC is also related to the self-attention mechanism originally proposed by Vaswani *et al.* for machine translation [127]. Self-attention modules compute the responses at each position while attending to the global context. Thanks to the use of global information, self-attention has been successfully used in several applications, including image generation [146, 91] and video activity recognition [131]. Attending to the whole image can be computationally expensive, and, as a result, can only be afforded on low-dimensional feature maps, *e.g.* as in [131]. A PAC layer produces responses that are sensitive to a more local context (which can be alleviated through *dilation*), and is therefore much more efficient.

5.2 Pixel-Adaptive Convolution

In this section, we start with a formal definition of standard spatial convolution and then explain our generalization of it to arrive at our pixel-adaptive convolution (PAC). Later, we will discuss several variants of PAC and how they are connected to different image filtering techniques.

5.2.1 Extending spatial convolution to higher-dimensions

Formally, a spatial convolution of image features $\mathbf{v} = (\mathbf{v}_1, \ldots, \mathbf{v}_n), \mathbf{v}_i \in \mathbb{R}^c$ over n pixels and c channels with filter weights $\mathbf{W} \in \mathbb{R}^{c' \times c \times s \times s}$ can be written as

$$\mathbf{v}'_i = \sum_{j \in \Omega(i)} \mathbf{W}\left[\mathbf{p}_i - \mathbf{p}_j\right] \mathbf{v}_j + \mathbf{b} \tag{5.1}$$

where $\mathbf{p}_i = (x_i, y_i)^\intercal$ are pixel coordinates, $\Omega(\cdot)$ defines an $s \times s$ convolution window, and $\mathbf{b} \in \mathbb{R}^{c'}$ denotes biases. With a slight abuse of notation, we use $[\mathbf{p}_i - \mathbf{p}_j]$ to denote indexing of the spatial dimensions of an array with 2D spatial offsets. This convolution operation results in a c'-channel output, $\mathbf{v}'_i \in \mathbb{R}^{c'}$, at each pixel i.

Equation 5.1 highlights how the weights only depend on pixel position and thus are agnostic to image content. In other words, the weights are *spatially shared* and, therefore, *image-agnostic*. As outlined earlier, these properties of spatial convolutions are limiting: we would like the filter weights $\mathbf{W}$ to be content-adaptive.

One approach to make the convolution operation content-adaptive, rather than only based on pixel locations, is to generalize $\mathbf{W}$ to depend on the pixel features, $\mathbf{f} \in \mathbb{R}^d$:

$$\mathbf{v}'_i = \sum_{j \in \Omega(i)} \mathbf{W}\left(\mathbf{f}_i - \mathbf{f}_j\right)\mathbf{v}_j + \mathbf{b} \tag{5.2}$$

where $\mathbf{W}$ can be seen as a high-dimensional filter operating in a d-dimensional feature space. In other words, we can apply Equation 5.2 by first projecting the input signal $\mathbf{v}$ into a d-dimensional space, and then performing d-dimensional convolution with $\mathbf{W}$. Traditionally, such high-dimensional filtering is limited to hand-specified filters such as Gaussian filters [1]. Recent work [57] lifts this restriction and proposes a technique to freely parameterize and learn $\mathbf{W}$ in high-dimensional space. Although generic and used successfully in several computer vision applications [57, 56, 115], high-dimensional convolutions have several shortcomings. First, since data projected on a higher-dimensional space is sparse, special lattice structures and hash tables are needed to perform the convolution [1] resulting in considerable computational overhead. Second, it is difficult to learn features $\mathbf{f}$ resulting in the use of hand-specified feature spaces such as position and color features, $\mathbf{f} = (x, y, r, g, b)$. Third, we have to restrict the dimensionality d of features (say, < 10) as the projected input image can become too sparse in high-dimensional spaces. In addition, the advantages that come with spatial sharing of standard convolution are lost with high-dimensional filtering.

5.2.2 Pixel-Adaptive Convolution

Instead of bringing convolution to higher dimensions, which has the above-mentioned drawbacks, we choose to modify the spatially invariant convolution in Equation 5.1 with a spatially varying kernel $K \in \mathbb{R}^{c' \times c \times s \times s}$ that depends on pixel features $\mathbf{f}$:

$$\mathbf{v}'_i = \sum_{j \in \Omega(i)} K\left(\mathbf{f}_i, \mathbf{f}_j\right) \mathbf{W}\left[\mathbf{p}_i - \mathbf{p}_j\right] \mathbf{v}_j + \mathbf{b} \tag{5.3}$$

where K is a kernel function that has a fixed parametric form such as Gaussian: $K(\mathbf{f}_i, \mathbf{f}_j) = \exp(-\frac{1}{2}(\mathbf{f}_i - \mathbf{f}_j)^\intercal(\mathbf{f}_i - \mathbf{f}_j))$. Since K has a pre-defined form and is not parameterized as a high-dimensional filter, we can perform this filtering on the 2D grid itself without moving onto higher dimensions. We call the above filtering operation (Equation 5.3) as "Pixel-Adaptive Convolution" (PAC) because the standard spatial convolution $\mathbf{W}$ is adapted at each pixel using pixel features $\mathbf{f}$ via kernel K. We call these pixel features $\mathbf{f}$ as "adapting features" and the kernel K as "adapting kernel". The adapting features $\mathbf{f}$ can be either hand-specified such as position and color features $\mathbf{f} = (x, y, r, g, b)$ or can be deep features that are learned end-to-end.

5.2.3 PAC as a generalization of other techniques

PAC, despite being a simple modification to standard convolution, generalizes several widely used filtering operations, including

- *Spatial Convolution* can be seen as a special case of PAC with adapting kernel being constant $K(\mathbf{f}_i, \mathbf{f}_j) = 1$. This can be achieved by using constant adapting features, $\mathbf{f}_i = \mathbf{f}_j, \forall i, j$. In brief, standard convolution (Equation 5.1) uses fixed, spatially shared filters, while PAC allows the filters to be modified by the adapting kernel K differently across pixel locations.

- *Bilateral Filtering* [121] is a basic image processing operation that has found wide-ranging uses [90] in image processing, computer vision and also computer graphics. Standard bilateral filtering operation can be seen as a special case of PAC, where $\mathbf{W}$ also has a fixed parametric form, such as a 2D Gaussian filter, $\mathbf{W}[\mathbf{p}_i - \mathbf{p}_j] = \exp(-\frac{1}{2}(\mathbf{p}_i - \mathbf{p}_j)^\intercal \Sigma^{-1}(\mathbf{p}_i - \mathbf{p}_j))$.

- *Pooling* operations can also be modeled by PAC. Standard average pooling corresponds to the special case of PAC where $K(\mathbf{f}_i, \mathbf{f}_j) = 1,\ \mathbf{W} = \frac{1}{s^2} \cdot \mathbf{1}$. *Detail Preserving Pooling* [103, 133] is a recently proposed pooling layer that is useful to preserve high-frequency details when performing pooling in CNNs. PAC can model the detail-preserving pooling operations by incorporating an adapting kernel that emphasizes more distinct pixels in the neighborhood, *e.g.* $K(\mathbf{f}_i, \mathbf{f}_j) = \alpha + (|\mathbf{f}_i - \mathbf{f}_j|^2 + \epsilon^2)^\lambda$.

The above generalizations show the generality and the wide applicability of PAC in different settings and applications. We experiment using PAC in three different problem scenarios, which will be discussed in later sections.

Some filtering operations are even more general than the proposed PAC. Examples include high-dimensional filtering shown in Equation 5.2 and others such as dynamic filter networks (DFN) [58] discussed in Section 5.1. Unlike most of those general filters, PAC allows efficient learning and reuse of spatially invariant filters because it is a direct modification of standard convolution filters. PAC offers a good trade-off between standard convolution and DFNs. In DFNs, filters are solely generated by an auxiliary network and different auxiliary networks or layers are required to predict kernels for different dynamic-filter layers. PAC, on the other hand, uses learned pixel embeddings $\mathbf{f}$ as adapting features, which can be reused across several different PAC layers in a network. When related to sparse high-dimensional filtering in Equation 5.2, PAC can be seen as factoring the high-dimensional filter into a product of standard spatial filter $\mathbf{W}$ and the adapting kernel K. This allows efficient implementation of PAC in 2D space alleviating the need for using hash tables and special lattice structures in high dimensions. PAC can also use learned pixel embeddings $\mathbf{f}$ instead of hand-specified ones in existing learnable high-dimensional filtering techniques such as [57].

5.2.4 Variants of PAC

In addition to the standard PAC layer described in Section 5.2.2, some variants of the layer are also useful depending on the usage scenarios.

5.2.4.1 Transposed PAC

Transposed convolution (sometimes called "deconvolution" in deep learning literature) is a commonly used operation in neural networks that involve upsampling. We refer to the transposed variant of PAC as PAC$^\intercal$. Similar to standard transposed convolution, PAC$^\intercal$ uses fractional striding and results in an upsampled output. As we show in Section 5.3, PAC$^\intercal$ plays an important role in the proposed joint upsampling network.

5.2.4.2 PAC on sparse inputs

The standard PAC layer is defined on dense 2D inputs. Removing this constraint would open up opportunities for applications where sparse data are given as input.

In order to process sparse inputs, we can modify the layer to take a third input (together with features $\mathbf{v}$ and $\mathbf{f}$), observation mask $M \in \{0,1\}^{H\times W}$. Each binary bit in M indicates whether the corresponding value in $\mathbf{v}$ is observed or not. Note that $\mathbf{f}$ still needs to be dense — this turns out not a very restrictive requirements and many applications can still fit under the formulation. The PAC layer can then be easily modified to incorporate the observation mask:

$$\mathbf{v}'_i = \frac{1}{Z_i} \sum_{j\in\Omega(i)} M_j K\left(\mathbf{f}_i, \mathbf{f}_j\right) \mathbf{W}\left[\mathbf{p}_i - \mathbf{p}_j\right] \mathbf{v}_j + \mathbf{b}, \tag{5.4}$$

where Z_i is a normalization factor to compensate the differences in signal density across pixel positions:

$$Z_i = \sum_{j \in \Omega(i)} M_j \tag{5.5}$$

In case of very sparse input, it is likely that there are output positions which receive no observed inputs, *i.e.* $M_j = 0, \forall j \in \Omega(i)$, and the output value at those positions are undefined based on Equation 5.4. To properly handle this, we also need to define an output observation mask:

$$M'_i = \mathbb{1}[Z_i > 0] \tag{5.6}$$

The output masks can then be used as input masks to the subsequent layers, till the signal is dense everywhere, *i.e.* $M_i = 1, \forall i$, from where regular PAC layers (or just standard spatial convolutional layers) can be directly applied.

While we have not experimented with tasks that make use of the sparse variant of PAC, here we discuss a few potential use cases:

- **Sparse depth map completion.** Depth maps acquired from LiDAR sensors are often very sparse. Using the accompanying RGB images as adapting feature $\mathbf{f}$ can help provide guidance to fill in the gaps to produce dense depth maps. The depth completion benchmark in the KITTI suite [122] provides an ideal evaluation environment for this task.

- **Monte Carlo rendering upsampling.** Since it is expensive to obtain a large number of samples per pixel (SPP), computer graphics systems often render at a low sampling rate and then upsample as a post-processing step. There is already some existing work ([4], [129], *etc.*) on applying neural networks for this purpose, however our layer can be used to leverage additional 3D clues, *e.g.* depth buffer, normal directions, to facilitate the upsampling procedure.

5.2.5 Implementation

We implemented PAC as a network layer in PyTorch with GPU acceleration[1]. Our implementation enables back-propagation through the features $\mathbf{f}$, permitting the use of learnable deep features as adapting features.

The implementation of PAC and its variants allows easy and flexible specification of different options that are commonly used in standard convolution: filter size, number of input and output channels, striding, padding and dilation factor.

5.3 Joint Upsampling

Joint upsampling is the task of upsampling a low-resolution signal with the help of a corresponding high-resolution guidance image. An example is upsampling a low-resolution depth map given a corresponding high-resolution RGB image as guidance. Joint upsampling is useful when some sensors output at a lower resolution than cameras, or can be used to speed up computer vision applications where full-resolution results are expensive to produce. PAC allows filtering operations to be guided by the adapting features, which can be obtained from a separate guidance image, making it an ideal choice for joint image processing. We investigate the use of PAC for joint upsampling applications. In this section, we introduce a network architecture that relies on PAC for deep joint upsampling, and show experimental results on two applications: joint depth upsampling and joint optical flow upsampling.

5.3.1 Deep joint upsampling with PAC

A deep joint upsampling network takes two inputs, a low-resolution signal $\mathbf{x} \in \mathbb{R}^{c \times h/m \times w/m}$ and a high-resolution guidance $\mathbf{g} \in \mathbb{R}^{c_g \times h \times w}$, and outputs upsampled signal $\mathbf{x}_{\uparrow} \in \mathbb{R}^{c \times h \times w}$. Here m is the required upsampling factor. Similar to [74], our upsampling network has three components (as illustrated in Figure 5.2):

[1]Code is available at `https://suhangpro.github.io/pac/`

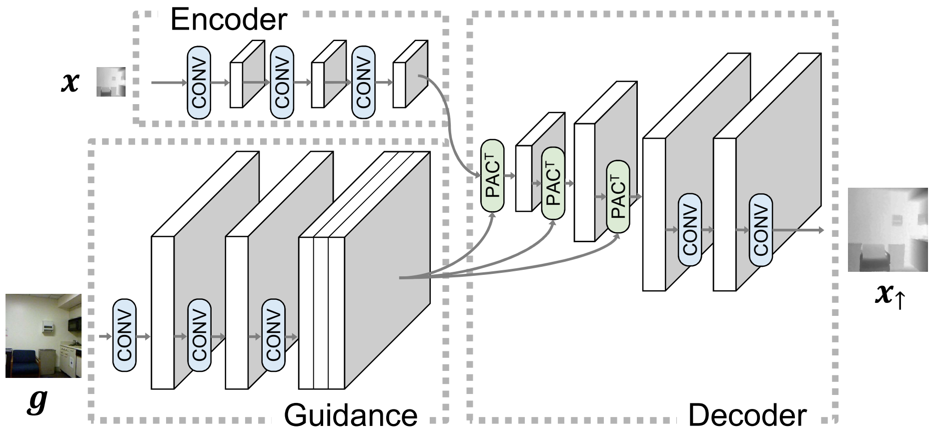


Figure 5.2: Joint upsampling with PAC. Network architecture showing encoder, guidance and decoder components. Features from the guidance branch are used to adapt PAC$^\top$ kernels that are applied on the encoder output resulting in upsampled signal.

- *Encoder* branch operates directly on the low-resolution signal with convolution (CONV) layers.
- *Guidance* branch operates solely on the guidance image, and generates adapting features that will be used in all PAC$^\top$ layers later in the network.
- *Decoder* branch starts with a sequence of PAC$^\top$, which perform transposed pixel-adaptive convolution, each of which upsamples the feature maps by a factor of 2. PAC$^\top$ layers are followed by two CONV layers to generate the final upsampled output.

The layers in each branch of the joint depth upsampling networks are listed in Table 5.1. Since we use each PAC$^\top$ for $2\times$ upsampling, $4\times$, $8\times$, $16\times$ networks requires 2, 3, 4 PAC$^\top$ layers respectively. The final output from the guidance branch is equally divided in the channel dimension for use as adapting features for the PAC$^\top$ layers in the decoder. All CONV and PAC$^\top$ layers use 5×5 filters, and are followed by ReLU except for the last CONV. We use Gaussian kernels for K in all PAC$^\top$ layers.

	standard			lite		
	4×	8×	16×	4×	8×	16×
Encoder	C32	C32	C32	C12	C12	C8
	C32	C32	C32	C16	C16	C16
	C32	C32	C32	C22	C16	C16
Guidance	C32	C32	C32	C12	C12	C8
	C32	C32	C32	C22	C16	C16
	C32	C48	C64	C24	C36	C40
Decoder	P32	P32	P32	P12	P12	P8
	P32	P32	P32	P16	P16	P16
	C32	P32	P32	C22	P16	P16
	C1	C32	P32	C1	C20	P16
		C1	C32		C1	C16
			C1			C1
#Params	183K	222K	260K	56K	56K	56K

Table 5.1: Network architectures for joint depth upsampling. "C" stands for regular CONV, "P" stands for PAC$^{\top}$ (the transposed convolution variant of PAC), and the number after them represents the number of output channels.

We design two variants of our model, *standard* and *lite*. The *standard* variant has a simpler design, but has varying number of parameters for different upsampling factors, and overall consume more memory than DJF [74], a previous state-of-the-art approach on joint depth upsampling. For the *lite* variant, we reduce the number of filters and make sure the networks roughly match the number of parameters compared to DJF.

Similar network architectures are also used for optical flow upsampling. First layer of encoder and last layer in decoder are modified to fit the two (u, v) channels in optical flow instead of one channel in depth maps, *i.e.* using "C2" instead of "C1" in Table 5.1.

5.3.2 Joint depth upsampling

Here, the task is to upsample a low-resolution depth by using a high-resolution RGB image as guidance. We experiment with the NYU Depth V2 dataset [110], which has 1449 RGB-depth pairs. Following [74], we use the first 1000 samples for training and the rest for testing. The low-resolution depth maps are obtained from the ground-truth depth maps using nearest-neighbor downsampling. Table 5.2 shows root mean square error (RMSE) of different techniques and for different upsampling factors m (4×, 8×, 16×). Results indicate that our network outperforms others in comparison and obtains state-of-the-art performance. Sample visual results are shown in Figure 5.3.

We train our network with the Adam optimizer using a learning rate schedule of $[10^{-4}\times$ 3.5k, $10^{-5}\times$ 1.5k, $10^{-6}\times$ 0.5k] and with mini-batches of 256×256 crops. We found this training setup to be superior to the one recommended in DJF [74], and also compare with our own implementation of it under such a setting ("DJF (Our impl.)" in Table 5.2). We keep the network architecture similar to that of previous state-of-the-art technique, DJF [74]. In DJF, features from the guidance branch are simply concatenated with encoder outputs for upsampling, whereas we use guidance features to adapt PAC$^{\intercal}$ kernels. Although with similar number of layers, our network has more parameters compared with DJF. We also trained a lighter version of our network ("Ours-lite") that matches the number of parameters of DJF, and still observe better performance showing the importance of PAC$^{\intercal}$ for upsampling.

5.3.3 Joint optical flow upsampling

We also evaluate our joint upsampling network for upsampling low-resolution optical flow using the original RGB image as guidance. Estimating optical flow is a challenging task, and even recent state-of-the-art approaches [116] resort to simple bilinear upsampling to predict optical flow at the full resolution. Optical flow is smoothly varying within motion boundaries, where accompanying RGB images can offer strong clues, making joint upsampling an appealing solution. We use the same network architecture as in the depth upsampling experiments, with the only differ-

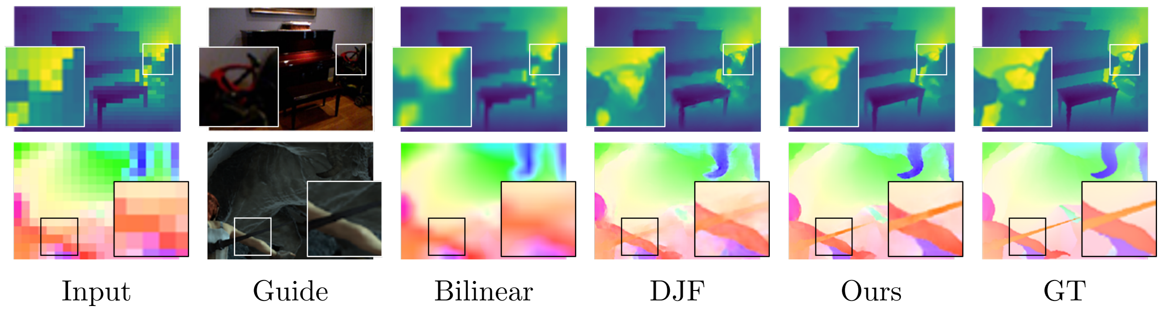


Figure 5.3: Deep joint upsampling. Results of different methods for 16× joint depth upsampling (top row) and 16× joint optical flow upsampling (bottom row). Our method produces results that have more details and are more faithful to the edges in the guidance image.

ence being that instead of single-channel depth, input and output are two-channel flow with u, v components. We experiment with the Sintel dataset [13] (clean pass). The same training protocol in Section 5.3.2 is used, and the low-resolution optical flow is obtained from bilinear downsampling of the ground-truth. We compare with baselines of bilinear interpolation and DJF [74], and observe consistent advantage (Table 5.3). Figure 5.3 shows a sample visual result indicating that our network is

Table 5.2: Joint depth upsampling. Results (in RMSE) show that our upsampling network consistently outperforms other techniques for different upsampling factors.

Method	4×	8×	16×
Bicubic	8.16	14.22	22.32
MRF	7.84	13.98	22.20
GF [50]	7.32	13.62	22.03
JBU [68]	4.07	8.29	13.35
Ham *et al.* [46]	5.27	12.31	19.24
DMSG [55]	3.78	6.37	11.16
FBS [5]	4.29	8.94	14.59
DJF [74]	3.54	6.20	10.21
DJF+ [75]	3.38	5.86	10.11
DJF (Our impl.)	2.64	5.15	9.39
Ours-lite	2.55	4.82	8.52
Ours	**2.39**	**4.59**	**8.09**

capable of restoring fine-structured details and also produces smoother predictions in areas with uniform motion.

Table 5.3: Joint optical flow upsampling. End-Point-Error (EPE) showing the improved performance compared with DJF [74].

	4×	8×	16×
Bilinear	0.465	0.901	1.628
DJF [74]	0.176	0.438	1.043
Ours	**0.105**	**0.256**	**0.592**

5.4 Conditional Random Fields

In this section, we draw the connection between conditional random fields (CRFs) and PAC, and present a new kind of CRF model that allows efficient inference within deep neural networks with the help of PAC layers.

CRFs are a class of statistical modeling method introduced by Lafferty *et al.* [72]. A CRF models the probabilistic distribution over two sets of variables: observations $\mathbf{X} = \{\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_m\}$ and outputs $\mathbf{Y} = \{\mathbf{Y}_1, \mathbf{Y}_2, \ldots, \mathbf{Y}_n\}$. Outputs and observations often come in corresponding pairs (in such cases $m = n$), however this is not always a requirement. Such distributions can have wide ranges of applications. A simple example in natural language processing (NLP) is part-of-speech tagging, where $\mathbf{X}$ represents the individual words in a sentence and $\mathbf{Y}$ represents their part-of-speech labels. Another common usage is image segmentation tasks in computer vision where $\mathbf{X}$ represents the pixel intensities and $\mathbf{Y}$ represents the label of each pixel (*e.g.* indicating whether a pixel belongs to foreground or background). Depending on the application, both $\mathbf{X}$ and $\mathbf{Y}$ can be either continuous or discrete variables.

Instead of directly modeling a joint distribution over $\mathbf{X}$ and $\mathbf{Y}$, a CRF describes the conditional distribution, $p(\mathbf{Y}|\mathbf{X})$. A significant advantage of adopting such a discriminative framework is that CRFs do not need to consider dependencies that involve only variables in $\mathbf{X}$. Roughly speaking, the model over $(\mathbf{X}, \mathbf{Y})$ is a CRF if variables in $\mathbf{Y}$ obey the Markov property when conditioned on $\mathbf{X}$ globally. More concretely, Lafferty *et al.* define a CRF on $\mathbf{X}$ and $\mathbf{Y}$ as follows [72]:

Definition. Let $G = (V, E)$ be an undirected graph such that $\mathbf{Y} = (\mathbf{Y}_v)_{v \in V}$, so that $\mathbf{Y}$ is indexed by the vertices of G. Then $(\mathbf{X}, \mathbf{Y})$ is a CRF when the variables $\mathbf{Y}_v$, conditioned on $\mathbf{X}$, obey the Markov property with respect to the graph, *i.e.* $p(\mathbf{Y}_v|\mathbf{X}, \mathbf{Y}_w, w \neq v) = p(\mathbf{Y}_v|\mathbf{X}, \mathbf{Y}_w, w \sim v)$, where $w \sim v$ means that w and v are directly connected as an edge in E.

For NLP tasks such as part-of-speech tagging, the graph G takes a simple chain structure; while for image segmentation tasks G is usually a grid where each pixel is connected with its four neighboring pixels. In either case, the only cliques in the graph are edges and vertices, and according to the Hammersley–Clifford theorem the distribution over random field $\mathbf{Y}$ given $\mathbf{X}$ has the form:

$$p_\theta(\mathbf{y}|\mathbf{x}) \propto \exp(-\sum_{v \in V} \psi_1(\mathbf{y}_v, \mathbf{x}, \theta) - \sum_{(u,v) \in E} \psi_2(\mathbf{y}_u, \mathbf{y}_v, \mathbf{x}, \theta)) \tag{5.7}$$

where ψ_1 is *unary potential* function over vertices, ψ_2 is *pairwise potential* function over edges, and θ are parameters of the CRF that usually need to be learned from training data.

5.4.1 Fully-connected CRFs

Early adoptions of CRFs in computer vision tasks were limited to models with pairwise potentials on neighboring pixels or patches [43, 35, 109] for efficiency reasons. The lack of longer-range connections often results in overly smooth boundaries in the predictions. Fully-Connected CRF (Full-CRF) [69] was proposed to offer the benefits of dense pairwise connections among pixels, which resorts to approximate high-dimensional filtering [1] for efficient inference. Consider a semantic labeling problem, where each pixel i in an image I can take one of the semantic labels $l_i \in \{1, ..., \mathcal{L}\}$. Full-CRF has unary potentials usually defined by a classifier such as CNN: $\psi_u(l_i) \in \mathbb{R}^{\mathcal{L}}$. And, the pairwise potentials are defined for every pair of pixel locations (i, j): $\psi_p(l_i, l_j|I) = \mu(l_i, l_j)K(\mathbf{f}_i, \mathbf{f}_j)$, where K is a kernel function and μ is a compatibility function. A common choice for μ is the Potts model: $\mu(l_i, l_j) = [l_i \neq l_j]$. [69] utilizes two Gaussian kernels with hand-crafted features as the kernel function:

$$K(\mathbf{f}_i, \mathbf{f}_j) = \underbrace{w_1 \exp\left\{-\frac{\|\mathbf{p}_i - \mathbf{p}_j\|^2}{2\theta_\alpha^2} - \frac{\|I_i - I_j\|^2}{2\theta_\beta^2}\right\}}_{\text{appearance kernel}} + \underbrace{w_2 \exp\left\{-\frac{\|\mathbf{p}_i - \mathbf{p}_j\|^2}{2\theta_\gamma^2}\right\}}_{\text{smoothness kernel}} \tag{5.8}$$

where $w_1, w_2, \theta_\alpha, \theta_\beta, \theta_\gamma$ are model parameters, and are typically found by a grid-search. The *appearance kernel* encourages nearby similar pixels to be assigned the same class. The *smoothness kernel* introduces additional spatial smoothness in the predictions and helps remove small isolated regions.

Inference in Full-CRF amounts to maximizing the following Gibbs distribution: $p(\mathbf{l}|I) \propto \exp(-\sum_i \psi_u(l_i) - \sum_{i<j} \psi_p(l_i, l_j))$, $\mathbf{l} = (l_1, l_2, ..., l_n)$. Exact inference of Full-CRF is hard, and [69] relies on mean-field approximation which is optimizing for an approximate distribution $Q(\mathbf{l}) = \prod_i Q_i(l_i)$ by minimizing the KL-divergence between $P(\mathbf{l}|I)$ and the mean-field approximation $Q(\mathbf{l})$. This leads to the following mean-field (MF) inference step that updates marginal distributions Q_i iteratively for $t = 0, 1, ...$:

$$Q_i^{(t+1)}(l) \leftarrow \frac{1}{Z_i} \exp\left\{ -\psi_u(l) - \sum_{l' \in \mathcal{L}} \mu(l, l') \sum_{j \neq i} K(\mathbf{f}_i, \mathbf{f}_j) Q_j^{(t)}(l') \right\} \tag{5.9}$$

The main computation in each MF iteration, $\sum_{j \neq i} K(\mathbf{f}_i, \mathbf{f}_j) Q_j^{(t)}$, can be viewed as high-dimensional Gaussian filtering. Previous work [69, 70] relies on permutohedral lattice convolution [1] to achieve efficient implementation.

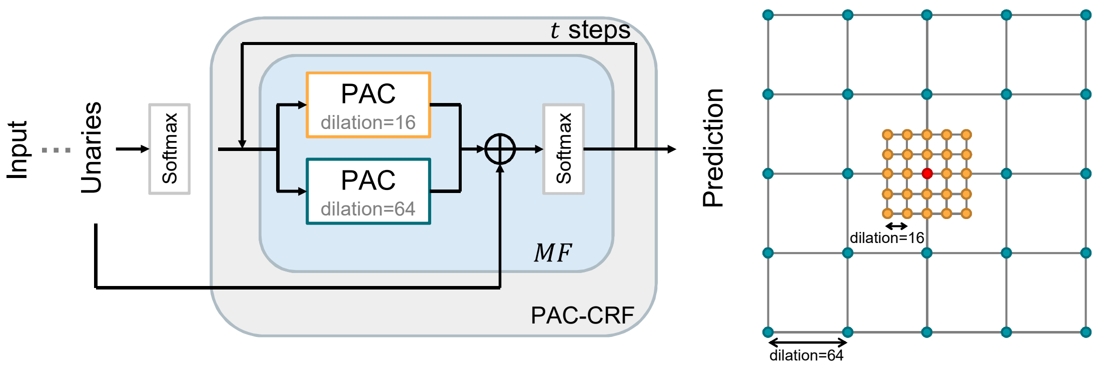


Figure 5.4: PAC-CRF. Illustration of inputs, outputs and the operations in each mean-field (MF) step of PAC-CRF inference. Also shown is the coverage of two 5×5 PAC filters, with dilation factors 16 and 64 respectively.

5.4.2 Efficient, learnable CRFs with PAC

Existing work [147, 57] back-propagates through the above MF steps to combine CRF inference with CNNs resulting in end-to-end training of CNN-CRF models. While there exists optimized CPU implementations, permutohedral lattice convolution cannot easily utilize GPUs because it "does not follow the SIMD paradigm of efficient GPU computation" [119]. Another drawback of relying on permutohedral lattice convolution is the approximation error incurred during both inference and gradient computation.

We propose PAC-CRF, which alleviates these computation issues by relying on PAC for efficient inference, and is easy to integrate with existing CNN backbones. PAC-CRF also has additional learning capacity, which leads to better performance compared with Full-CRF in our experiments.

5.4.2.1 PAC-CRF

In PAC-CRF, we define pairwise connections over fixed windows Ω^k around each pixel instead of dense connections: $\sum_k \sum_i \sum_{j \in \Omega^k(i)} \psi_p^k(l_i, l_j|I)$, where the k-th pairwise potential is defined as:

$$\psi_p^k(l_i, l_j|I) = K^k(\mathbf{f}_i, \mathbf{f}_j)\mathbf{W}^k_{l_j l_i}[\mathbf{p}_j - \mathbf{p}_i] \tag{5.10}$$

Here $\Omega^k(\cdot)$ specifies the pairwise connection pattern of the k-th pairwise potential originated from each pixel, and K^k is a fixed Gaussian kernel. Intuitively, this formulation allows the label compatibility transform μ in Full-CRF to be modeled by $\mathbf{W}$, and to vary across different spatial offsets.

5.4.2.2 Inference

As with Full-CRFs, we can perform mean-field inference for PAC-CRFs. We will start from the mean-field update equation for general pairwise CRFs, Equation 5.11. Detailed derivation for it can be found in Koller and Friedman [67, Chapter 11.5].

$$Q_i(l) = \frac{1}{Z_i}\exp\Big\{ -\psi_u(l) - \sum_{j\in\Omega(i)} \mathbf{E}_{l_j\sim Q_j}\psi_p(l, l_j)\Big\} \tag{5.11}$$

Considering that we use multiple neighborhoods (with different dilation factors) in parallel, the update equation becomes

$$Q_i(l) = \frac{1}{Z_i}\exp\Big\{ -\psi_u(l) - \sum_k \sum_{j\in\Omega^k(i)} \mathbf{E}_{l_j\sim Q_j}\psi_p^k(l, l_j)\Big\} \tag{5.12}$$

Substituting the pairwise potential with

$$\psi_p^k(l_i, l_j) = K^k(\mathbf{f}_i, \mathbf{f}_j)\mathbf{W}^k_{l_j l_i}[\mathbf{p}_j - \mathbf{p}_i] \tag{5.13}$$

the update rule becomes

$$
\begin{aligned}
Q_i(l) =& \frac{1}{Z_i} \exp\Big\{ -\psi_u(l) - \sum_k \sum_{j \in \Omega^k(i)} \mathbf{E}_{l_j \sim Q_j} \Big\{ K^k(\mathbf{f}_i, \mathbf{f}_j) \mathbf{W}^k_{l_j l}[\mathbf{p}_j - \mathbf{p}_i] \Big\} \Big\} \\
=& \frac{1}{Z_i} \exp\Big\{ -\psi_u(l) - \sum_k \sum_{l' \in \mathcal{L}} \sum_{j \in \Omega^k(i)} K^k(\mathbf{f}_i, \mathbf{f}_j) \mathbf{W}^k_{l' l}[\mathbf{p}_j - \mathbf{p}_i] Q_j(l') \Big\} \qquad (5.14)
\end{aligned}
$$

Using Equation 5.14 in an iterative fashion leads to the final update rule of mean-field inference:

$$
Q_i^{(t+1)}(l) \leftarrow \frac{1}{Z_i} \exp\Big\{ -\psi_u(l) - \sum_k \underbrace{\sum_{l' \in \mathcal{L}} \sum_{j \in \Omega^k(i)} K^k(\mathbf{f}_i, \mathbf{f}_j) \mathbf{W}^k_{l' l}[\mathbf{p}_j - \mathbf{p}_i] Q_j^{(t)}(l') \Big\}}_{\text{PAC}} \qquad (5.15)
$$

MF update now consists of PACs instead of sparse high-dimensional filtering as in Full-CRF (Equation 5.9). As outlined in Section 5.1, there are several advantages of PAC over high-dimensional filtering. With PAC-CRF, we can freely parameterize and learn the pairwise potentials in Equation 5.10 that also use a richer form of compatibility transform $\mathbf{W}$. PAC-CRF can also make use of learnable features $\mathbf{f}$ for pairwise potentials instead of pre-defined ones in Full-CRF. Figure 5.4 (left) illustrates the computation steps in each MF update with two pairwise PAC kernels.

5.4.2.3 Interpretation of the PAC-CRF formulation

Recall that the pairwise potentials in Full-CRF is defined as the product of a compatibility function and a kernel function, $\psi_p(l_i, l_j | I) = \mu(l_i, l_j) K(\mathbf{f}_i, \mathbf{f}_j)$, where the kernel function K has two terms, *appearance kernel* and *smoothness kernel* (Equation 5.8).

For the moment, let's consider a special case of PAC-CRF where the pairwise potential includes a single Gaussian kernel and uses a two-dimensional filter $\mathbf{W}$:

$$
\begin{aligned}
K(\mathbf{f}_i, \mathbf{f}_j) &= \mathbf{W}[\mathbf{p}_j - \mathbf{p}_i] G(\mathbf{f}_i, \mathbf{f}_j) \\
&= \mathbf{W}[\mathbf{p}_j - \mathbf{p}_i] \exp\left\{ -\frac{1}{2} \|\mathbf{f}_i - \mathbf{f}_j\|^2 \right\}
\end{aligned} \tag{5.16}
$$

There are two major differences between this kernel function and the one used in Full-CRF:

1. The *smoothness kernel* in Full-CRF can now be considered as represented by filter $\mathbf{W}$. It can still be initialized as a Gaussian, but arbitrary filter is allowed to be learned.

2. The *appearance kernel* now operates on $\mathbf{f}$ directly without the need of decomposing it into multiple parts, and without the individual scaling factors ($\theta_\alpha, \dots$).

Both changes give the pairwise potential more learning capacity. Note that $\mathbf{f}$ can be the output of some other network layers. A simple linear layer can learn appropriate scaling factors, while in other cases a more complex network may be preferred. For input with more than RGB channels (*e.g.* 3D data with color, depth, normal, curvature, *etc.*), hand-engineering and finding parameters for kernel functions like Equation 5.8 can be time-consuming and suboptimal, so allowing the function to be learned from data in an end-to-end fashion is particularly desirable.

Note that in Equation 5.16, $\mathbf{W}$ is a 2D matrix, and the corresponding pairwise potential is defined as

$$
\psi_p(l_i, l_j) = \mu(l_i, l_j) \mathbf{W}[\mathbf{p}_j - \mathbf{p}_i] K(\mathbf{f}_i, \mathbf{f}_j) \tag{5.17}
$$

where $\mu(l_i, l_j)$ is the compatibility matrix. The full pairwise potential of PAC-CRF, $\psi_p(l_i, l_j) = K(\mathbf{f}_i, \mathbf{f}_j) \mathbf{W}_{l_j l_i}[\mathbf{p}_j - \mathbf{p}_i]$, can be seen as a further step of generalization, where $\mathbf{W}$ is now a 4D tensor. Intuitively, this formulation allows the label compatibility pattern to be spatially varying across different pixel locations. Equation 5.17 can be seen as a special case factorizing the 4D tensor as the product of two 2D matrices.

5.4.2.4 Long-range connections with dilated PAC

The major source of heavy computation in Full-CRF is the dense pairwise pixel connections. In PAC-CRF, the pairwise connections are defined by the local convolution windows Ω^k. To have long-range pairwise connections while keeping the number of PAC parameters managable, we make use of dilated filters [20, 143]. Even with a relatively small kernel size (5×5), with a large dilation, *e.g.* 64, the CRF can effectively reach a neighborhood of 257×257. A concurrent work [119] also propose a convolutional version of CRF (Conv-CRF) to reduce the number of connections in Full-CRF. However, [119] uses connections only within small local windows. We argue that long-range connections can provide valuable information, and our CRF formulation uses a wider range of connections while still being efficient. Our formulation allows using multiple PAC filters in parallel, each with different dilation factors. In Figure 5.4 (right), we show an illustration of the coverage of two 5×5 PAC filters, with dilation factors 16 and 64 respectively. This allows PAC-CRF to achieve a good trade-off between computational efficiency and long-range pairwise connectivity.

5.4.3 Semantic segmentation with PAC-CRF

The task of semantic segmentation is to assign a semantic label to each pixel in an image. Full-CRF is proven to be a valuable post-processing tool that can considerably improve CNN segmentation performance [20, 147, 57]. Here, we experiment with PAC-CRF on top of the FCN semantic segmentation network [80]. We choose FCN for simplicity and ease of comparisons, as FCN only uses standard convolution layers and does not have many bells and whistles.

In the experiments, we use scaled RGB color, $[\frac{R}{\sigma_R}, \frac{G}{\sigma_G}, \frac{B}{\sigma_B}]^\intercal$, as the guiding features for the PAC layers in PAC-CRF . The scaling vector $[\sigma_R, \sigma_G, \sigma_B]^\intercal$ is learned jointly with the PAC weights $\mathbf{W}$. We try two internal configurations of PAC-CRF: a single 5×5 PAC kernel with dilation of 32, and two parallel 5×5 PAC kernels with dilation factors of 16 and 64. 5 MF steps are used for a good balance between speed and accuracy. We first freeze the backbone FCN network and train only the PAC-CRF part for 40 epochs, and then train the whole network for another 40 epochs with reduced learning rates.

5.4.3.1 Dataset

We follow the training and validation settings of FCN [80] which is trained on PascalVOC images and validated on a reduced validation set of 736 images. We also submit our final trained models to the official evaluation server to get test scores on 1456 test images.

5.4.3.2 Baselines

We compare PAC-CRF with three baselines: Full-CRF [69], BCL-CRF [57], and Conv-CRF [119]. For Full-CRF, we use the publicly available C++ code, and find the optimal CRF parameters through grid search. For BCL-CRF, we use 1-neighborhood filters to keep the runtime manageable and use other settings as suggested by the authors. For Conv-CRF, the same training procedure is used as in PAC-CRF. We use the more powerful variant of Conv-CRF with learnable compatibility transform (referred to as "Conv+C" in [119]), and we learn the RGB scales for Conv-CRF in the same way as for PAC-CRF. We follow the suggested default settings for Conv-CRF and use a filter size of 11×11 and a blurring factor of 4. Note that like Full-CRF (Equation 5.8), the other baselines also use two pairwise kernels.

Table 5.4: Semantic segmentation with PAC-CRF. Validation and test mIoU scores along with the runtimes of different techniques. PAC-CRF results in better improvements than Full-CRF [69] while being faster. PAC-CRF also outperforms Conv-CRF [119] and BCL [57]. Runtimes are averaged over all validation images.

Method	mIoU (val / test)	CRF Runtime
Unaries only (FCN)	65.51 / 67.20	-
Full-CRF [69]	+2.11 / +2.45	629 ms
BCL-CRF [57]	+2.28 / +2.33	2.6 s
Conv-CRF [119]	+2.13 / +1.57	38 ms
PAC-CRF, 32	+3.01 / +2.21	39 ms
PAC-CRF, 16-64	**+3.39** / **+2.62**	78 ms

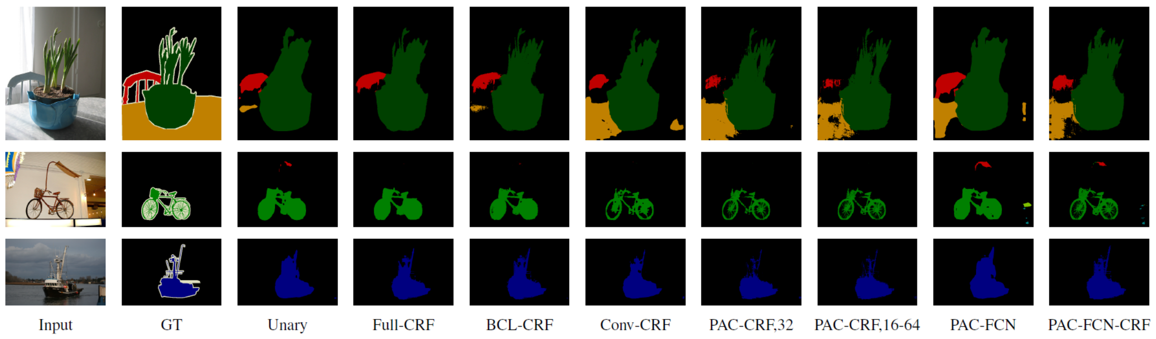


Figure 5.5: Semantic segmentation with PAC-CRF and PAC-FCN. We show three examples from the validation set. Compared to Full-CRF [69], BCL-CRF [57], and Conv-CRF [119], PAC-CRF can recover finer details faithful to the boundaries in the RGB inputs.

5.4.3.3 Results

Table 5.4 reports validation and test mean Intersection over Union (mIoU) scores along with average runtimes of different techniques. Our two-filter variant ("PAC-CRF, 16-64") achieves better mIoU compared with all baselines, and also compares favorably in terms of runtime. The one-filter variant ("PAC-CRF, 32") performs slightly worse than Full-CRF and BCL-CRF, but has even larger speed advantage, offering a strong option where efficiency is needed. Sample visual results are shown in Figure 5.5.

Mean-field inference steps. Table 5.5 shows how mIoU changes with different mean-field update steps. We find no additional improvements beyond 5 steps, and use 5 steps for all other experiments.

Table 5.5: Validation mIoU when using different number of mean-field update steps in PAC-CRF.

Mean-field steps	1	3	5	7
mIoU	68.38	68.72	**68.90**	**68.90**
time	19 ms	49 ms	78 ms	109 ms

On the contribution of dilation. Just like standard convolution, PAC supports dilation to increase the receptive field without increasing the number of parameters. This capability is leveraged by PAC-CRF to allow long-range connections. For a similar purpose, Conv-CRF applies Gaussian blur to pairwise potentials to increase the receptive field. To quantify the improvements due to dilation, we try another baseline where we add dilation to Conv-CRF. The improved performance (+2.13/+1.57 $\rightarrow$ +2.50/+1.91) validates that dilation is indeed an important ingredient, while the remaining gap shows that the PAC formulation is essential to the full gain.

5.4.3.4 Limitations

While being quantitatively better and retaining more visual details overall, PAC-CRF produces some amount of noise around boundaries. Figure 5.6 shows a few such examples. The artifacts are likely due to a known "gridding" effect of dilated filters, and a similar behavior has been reported in earlier work [144].

In [144], the proposed solution involves several modifications to the networks architectures: using decreasing dilation factors towards later layers in the networks,

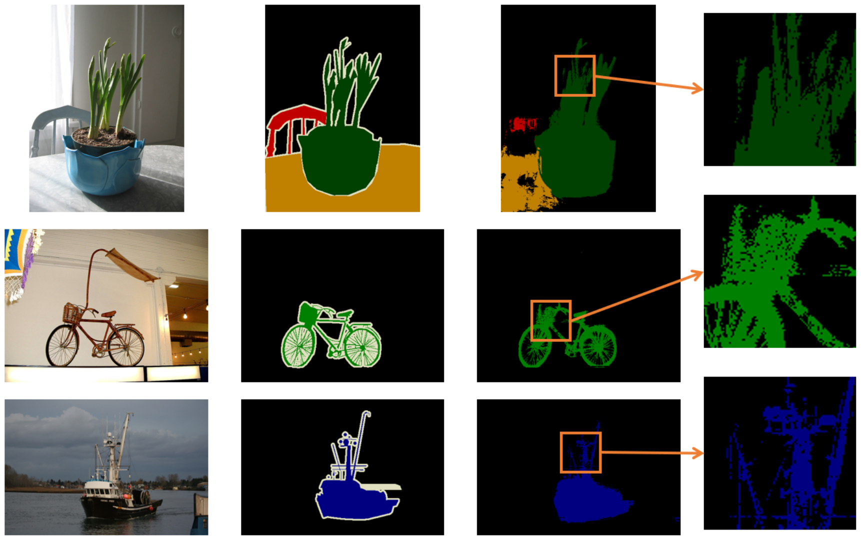


Figure 5.6: Artifacts in the PAC-CRF predictions. From left to right: RGB image, ground-truth labeling, PAC-CRF output with zoomed-in details.

appending a few (non-dilated) convolutional layers at the end of the networks, *etc*. These measures, however, do not fit naturally within PAC-CRF.

5.5 Layer Hot-Swapping with PAC

So far, we design specific architectures around PAC for different use cases. In this section, we offer a strategy to use PAC for simply upgrading existing CNNs with minimal modifications through what we call layer *hot-swapping*.

5.5.1 Layer hot-swapping

Network fine-tuning has become a common practice when training networks on new data or with additional layers. Typically, in fine-tuning, newly added layers are initialized randomly. Since PAC generalizes standard convolution layers, it can directly replace convolution layers in existing networks while retaining the pre-trained weights. We refer to this modification of existing pre-trained networks as layer *hot-swapping*.

5.5.2 Semantic segmentation with layer hot-swapping

We continue to use semantic segmentation as an example, and demonstrate how layer hot-swapping can be a simple yet effective modification to existing CNNs. Figure 5.7 illustrates a FCN [80] before and after the hot-swapping modifications. We swap out the last CONV layer of the last three convolution groups, CONV3_3, CONV4_3, CONV5_3, with PAC layers with the same configuration (filter size, input and output channels, *etc.*), and use the output of CONV2_2 as the guiding feature for the PAC layers. By this example, we also demonstrate that one could use earlier layer features (CONV2_2 here) as adapting features for PAC. Using this strategy, the network parameters do not increase when replacing CONV layers with PAC layers. All the layer weights are initialized with trained FCN parameters. To ensure a better starting condition for further training, we scale the guiding features by a small constant (0.0001) so that the PAC layers initially behave very closely to their original CONV counterparts. We use 8825 images for training, including the Pascal VOC

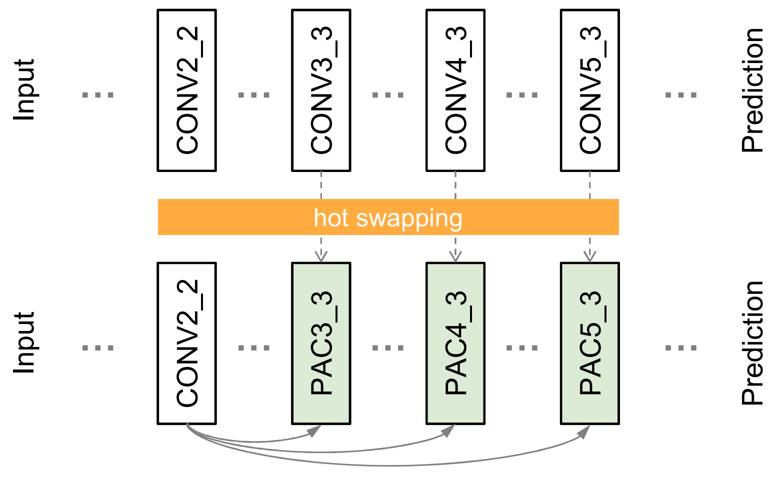


Figure 5.7: Layer hot-swapping with PAC. A few layers of a network before (top) and after (bottom) hot-swapping. Three CONV layers are replaced with PAC layers, with adapting features coming from an earlier convolution layer. All the original network weights are retained after the modification.

2011 training images and the additional training samples from [47]. Validation and testing are performed in the same fashion as in Section 5.4.

Results are reported in Table 5.6. We show that our simple modification (PAC-FCN) provides about 2 mIoU improvement on test (67.20 $\rightarrow$ 69.18) for the semantic segmentation task, while incurring virtually no runtime penalty at inference time. Note that PAC-FCN has the same number of parameters as the original FCN model. The improvement brought by PAC-FCN is also complementary to any additional CRF post-processing that can still be applied. After combined with a PAC-CRF (the 16-64 variant) and trained jointly, we observe another 2 mIoU improvement. Sample visual results are shown in Figure 5.5.

Table 5.6: FCN hot-swapping CONV with PAC. Validation and test mIoU scores along with runtimes of different techniques. Our simple hot-swapping strategy provides 2 IoU gain on test. Combining with PAC-CRF offers additional improvements.

Method	PAC-CRF	mIoU (val / test)	Runtime
FCN-8s	-	65.51 / 67.20	39 ms
FCN-8s	16-64	68.90 / 69.82	117 ms
PAC-FCN	-	67.44 / 69.18	41 ms
PAC-FCN	16-64	**69.87** / **71.34**	118 ms

5.6 Conclusions

In this chapter, we propose PAC, a new type of filtering operation that can effectively learn to leverage guidance information. We show that PAC generalizes several popular filtering operations and demonstrate its applicability on different uses ranging from joint upsampling, semantic segmentation networks, to efficient CRF inference. PAC generalizes standard spatial convolution, and can be used to directly replace standard convolution layers in pre-trained networks for performance gain with minimal computation overhead.

CHAPTER 6

CONCLUSIONS

In this dissertation, we present our effort on a few fronts where directly applying existing deep neural network designs would pose challenges. 3D data can come in many different types of representations, most of which not amenable to the powerful CNN operations and architectures known for their success in image recognition tasks.

3D shapes represented as polygon meshes do not have a regular spatial layout. To apply CNNs, many choose to voxelize the shapes so 3D data lives on a regular grid and then one can easily extend 2D CNNs into 3D versions. We propose Multi-View CNNs in Chapter 3 as an alternative to the voxel-based representations. Relying on 2D features from multiple views and a simple mechanism to pool information from all views, Multi-View CNNs are remarkably efficient compared to 3D networks on high-resolution voxels. Moreover, the performance advantages we showcase in the experiments prove that 2D projections of 3D objects can carry a large amount of useful information for recognition tasks.

Another very common type of 3D data representation is point cloud. Its importance has risen rapidly recently due to the wide adoption of 3D sensors in applications such as autonomous driving. Like polygon meshes, point clouds also do not have a regular grid structure. Since point clouds can be very sparse, voxelization is often a poor choice for its computational overheads. In Chapter 4, we present the SPLAT-Net architectures that implicitly impose some lattice structure over the point clouds. Besides its strength we demonstrate in point cloud semantic segmentation tasks, a unique advantage of SPLATNet is that it can incorporate multi-view 2D inputs when available for end-to-end training and inference.

In Chapter 5, we present a new type of operation named Pixel-Adaptive Convolution (PAC). PAC is inspired by sparse high-dimensional filtering, the underlying operation utilized in SPLATNet. Although PAC's subjects are images, which live on dense regular grids, PAC essentially *projects* the pixels onto a high-dimensional space and then performs sparse high-dimensional filtering in that space. We demonstrate several usage examples of PAC, including joint upsampling, semantic segmentation and efficient CRF inference.

6.1 Future Directions

Going forward, we identify a few potential directions for future work based on the observations and conclusions in this dissertation:

- **Developing a principled approach to better combine 2D and 3D inputs into a unified representation that is optimized for the end task**. In Chapter 3, we investigate the trade-off between 2D and 3D representations. In Chapter 4, we present a strategy to integrate both 2D and 3D data in a single network. However, in these cases, the 2D and 3D representations are still largely separate and retain their original structures. Consider the task of understanding the environment for an autonomous vehicle, while a detailed 3D reconstruction of the environment can definitely be useful, it is not likely that an intelligent agent under normal conditions would require a full 3D reconstructed world to operate the vehicle. We can envision a hybrid representation that strives the best balance between performance and efficiency.

- **Unsupervised learning for 3D representations**. Even with the recent developments in large-scale 3D datasets, the amount of 3D labeled data available for supervised training is still very limited compared to 2D images. The superior performance of Multi-View CNN benefits substantially from the pre-training enabled by the large image datasets. Unsupervised learning, and self-supervised learning in particular, has gain encouraging success recently [44] in representation learning for images, offering pre-training performances closely matching supervised pre-training for some tasks. We believe many of the proven ideas and techniques there can be borrowed for learning 3D representations as well and can help mitigate the issue of insufficient labeled data.

BIBLIOGRAPHY

[1] Adams, Andrew, Baek, Jongmin, and Davis, Myers Abraham. Fast high-dimensional filtering using the permutohedral lattice. *Computer Graphics Forum 29*, 2 (2010), 753–762.

[2] Aurich, Volker, and Weule, Jörg. Non-linear Gaussian filters performing edge preserving diffusion. In *DAGM*. Springer, 1995, pp. 538–545.

[3] Awate, Suyash P, and Whitaker, Ross T. Higher-order image statistics for unsupervised, information-theoretic, adaptive, image filtering. In *Proc. CVPR* (2005), vol. 2, IEEE, pp. 44–51.

[4] Bako, Steve, Vogels, Thijs, McWilliams, Brian, Meyer, Mark, Novák, Jan, Harvill, Alex, Sen, Pradeep, Derose, Tony, and Rousselle, Fabrice. Kernel-predicting convolutional networks for denoising monte carlo renderings. *ACM Trans. Graph. 36*, 4 (2017), 97.

[5] Barron, Jonathan T, and Poole, Ben. The fast bilateral solver. In *Proc. ECCV* (2016), Springer, pp. 617–632.

[6] Boscaini, D., Masci, J., Melzi, S., Bronstein, M. M., Castellani, U., and Vandergheynst, P. Learning class-specific descriptors for deformable shapes using localized spectral convolutional networks. In *Proc. SGP* (2015).

[7] Boscaini, Davide, Masci, Jonathan, Rodolà, Emanuele, and Bronstein, Michael M. Learning shape correspondence with anisotropic convolutional neural networks. In *Proc. NeurIPS* (2016).

[8] Brock, André, Lim, Theodore, Ritchie, James M., and Weston, Nick. Generative and discriminative voxel modeling with convolutional neural networks. *arXiv:1608.04236* (2016).

[9] Bronstein, A., Bronstein, M., Ovsjanikov, M., and Guibas, L. Shape Google: Geometric words and expressions for invariant shape retrieval. *ACM Trans. Graph. 30* (2011).

[10] Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., and Vandergheynst, P. Geometric deep learning: Going beyond euclidean data. *IEEE Signal Processing Magazine 34*, 4 (2017), 18–42.

[11] Bruna, Joan, Zaremba, Wojciech, Szlam, Arthur, and LeCun, Yann. Spectral networks and locally connected networks on graphs. In *Proc. ICLR* (2014).

[12] Buades, Antoni, Coll, Bartomeu, and Morel, J-M. A non-local algorithm for image denoising. In *Proc. CVPR* (2005), vol. 2, IEEE, pp. 60–65.

[13] Butler, D. J., Wulff, J., Stanley, G. B., and Black, M. J. A naturalistic open source movie for optical flow evaluation. In *Proc. ECCV* (Oct. 2012), A. Fitzgibbon et al. (Eds.), Ed., Part IV, LNCS 7577, Springer-Verlag, pp. 611–625.

[14] Chandra, Siddhartha, and Kokkinos, Iasonas. Fast, exact and multi-scale inference for semantic image segmentation with deep Gaussian CRFs. In *Proc. ECCV* (2016), Springer, pp. 402–418.

[15] Chatfield, K., Simonyan, K., Vedaldi, A., and Zisserman, A. Return of the devil in the details: Delving deep into convolutional nets. In *Proc. BMVC* (2014).

[16] Chaudhuri, Siddhartha, and Koltun, Vladlen. Data-driven suggestions for creativity support in 3D modeling. *ACM Trans. Graph. 29*, 6 (2010).

[17] Chen, Dingyun, Tian, Xiaopei, Shen, Yute, and Ouhyoung, Ming. On visual similarity based 3D model retrieval. *Proc. Eurographics 22*, 3 (2003), 223–232.

[18] Chen, Liang-Chieh, Barron, Jonathan T, Papandreou, George, Murphy, Kevin, and Yuille, Alan L. Semantic image segmentation with task-specific edge detection using CNNs and a discriminatively trained domain transform. In *Proc. CVPR* (2016), pp. 4545–4554.

[19] Chen, Liang-Chieh, Papandreou, George, Kokkinos, Iasonas, Murphy, Kevin, and Yuille, Alan L. Semantic image segmentation with deep convolutional nets and fully connected CRFs. In *Proc. ICLR* (2015).

[20] Chen, Liang-Chieh, Papandreou, George, Kokkinos, Iasonas, Murphy, Kevin, and Yuille, Alan L. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs. *PAMI 40*, 4 (2018), 834–848.

[21] Chen, Liang-Chieh, Schwing, Alexander, Yuille, Alan, and Urtasun, Raquel. Learning deep structured models. In *Proc. ICML* (2015), pp. 1785–1794.

[22] Chopra, Sumit, Hadsell, Raia, and LeCun, Yann. Learning a similarity metric discriminatively, with application to face verification. In *Proc. CVPR* (2005).

[23] Cimpoi, M., Maji, S., Kokkinos, I., Mohamed, S., , and Vedaldi, A. Describing textures in the wild. In *Proc. CVPR* (2014).

[24] Cyr, Christopher M., and Kimia, Benjamin B. A similarity-based aspect-graph approach to 3D object recognition. *IJCV 57*, 1 (2004).

[25] Dai, Angela, Chang, Angel X., Savva, Manolis, Halber, Maciej, Funkhouser, Thomas, and Nießner, Matthias. Scannet: Richly-annotated 3D reconstructions of indoor scenes. In *Proc. CVPR* (2017).

[26] Dai, Jifeng, Qi, Haozhi, Xiong, Yuwen, Li, Yi, Zhang, Guodong, Hu, Han, and Wei, Yichen. Deformable convolutional networks. *arXiv:1703.06211 1*, 2 (2017), 3.

[27] Dalal, Navneet, and Triggs, Bill. Histograms of oriented gradients for human detection. In *Proc. CVPR* (2005).

[28] Defferrard, Michaël, Bresson, Xavier, and Vandergheynst, Pierre. Convolutional neural networks on graphs with fast localized spectral filtering. *arXiv:1606.09375* (2016).

[29] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *Proc. CVPR* (2009).

[30] Donahue, Jeff, Jia, Yangqing, Vinyals, Oriol, Hoffman, Judy, Zhang, Ning, Tzeng, Eric, and Darrell, Trevor. DeCAF: A deep convolutional activation feature for generic visual recognition. *CoRR abs/1310.1531* (2013).

[31] Eitz, Mathias, Hays, James, and Alexa, Marc. How do humans sketch objects? *ACM Trans. Graph. 31*, 4 (2012), 44:1–44:10.

[32] Eitz, Mathias, Richter, Ronald, Boubekeur, Tamy, Hildebrand, Kristian, and Alexa, Marc. Sketch-based shape retrieval. *ACM Trans. Graph. 31*, 4 (2012).

[33] Everingham, M., Eslami, S. M. A., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A. The Pascal Visual Object Classes Challenge: A retrospective. *IJCV 111*, 1 (Jan. 2015), 98–136.

[34] Ezuz, Danielle, Solomon, Justin, Kim, Vladimir G., and Ben-Chen, Mirela. GWCNN: A metric alignment layer for deep shape analysis. *Computer Graphics Forum 36*, 5 (2017).

[35] Fulkerson, Brian, Vedaldi, Andrea, and Soatto, Stefano. Class segmentation and object localization with superpixel neighborhoods. In *Proc. ICCV* (2009), IEEE, pp. 670–677.

[36] Gadde, Raghudeep, Jampani, Varun, Kiefel, Martin, Kappler, Daniel, and Gehler, Peter V. Superpixel convolutional networks using bilateral inceptions. In *Proc. ECCV* (2016), Springer, pp. 597–613.

[37] Gadde, Raghudeep, Jampani, Varun, Marlet, Renaud, and Gehler, Peter. Efficient 2D and 3D facade segmentation using auto-context. *PAMI* (2017).

[38] Garcia-Garcia, Alberto, Gomez-Donoso, Francisco, Rodríguez, José García, Orts, Sergio, Cazorla, Miguel, and López, Jorge Azorín. PointNet: A 3D convolutional neural network for real-time object class recognition. In *Proc. IJCNN* (2016).

[39] Gastal, Eduardo SL, and Oliveira, Manuel M. Domain transform for edge-aware image and video processing. *ACM Trans. Graph. 30*, 4 (2011), 69.

[40] Girshick, R. B., Donahue, J., Darrell, T., and Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proc. CVPR* (2014).

[41] Girshick, Ross. Fast r-cnn. In *Proc. ICCV* (2015), pp. 1440–1448.

[42] Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A., and Bengio, Y. Maxout networks. *ArXiv e-prints* (Feb. 2013).

[43] Gould, Stephen, Rodgers, Jim, Cohen, David, Elidan, Gal, and Koller, Daphne. Multi-class segmentation with relative location prior. *IJCV 80*, 3 (2008), 300–316.

[44] Goyal, Priya, Mahajan, Dhruv, Gupta, Abhinav, and Misra, Ishan. Scaling and Benchmarking Self-Supervised Visual Representation Learning. In *Proc. ICCV* (2019).

[45] Graham, Benjamin, and van der Maaten, Laurens. Submanifold sparse convolutional networks. *arXiv:1706.01307* (2017).

[46] Ham, Bumsub, Cho, Minsu, and Ponce, Jean. Robust image filtering using joint static and dynamic guidance. In *Proc. CVPR* (2015), pp. 4823–4831.

[47] Hariharan, Bharath, Arbelaez, Pablo, Bourdev, Lubomir, Maji, Subhransu, and Malik, Jitendra. Semantic contours from inverse detectors. In *Proc. ICCV* (2011).

[48] Hariharan, Bharath, Arbeláez, Pablo, Girshick, Ross, and Malik, Jitendra. Hypercolumns for object segmentation and fine-grained localization. In *Proc. CVPR* (2015), pp. 447–456.

[49] Harley, Adam W, Derpanis, Konstantinos G, and Kokkinos, Iasonas. Segmentation-aware convolutional networks using local attention masks. In *Proc. ICCV* (2017), vol. 2, p. 7.

[50] He, Kaiming, Sun, Jian, and Tang, Xiaoou. Guided image filtering. *PAMI 35*, 6 (2013), 1397–1409.

[51] He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition. In *Proc. CVPR* (2016), pp. 770–778.

[52] Henaff, Mikael, Bruna, Joan, and LeCun, Yann. Deep convolutional networks on graph-structured data. *arXiv:1506.05163* (2015).

[53] Horn, B. K. P. Extended gaussian images. *Proc. of the IEEE 72*, 12 (1984), 1671–1686.

[54] Hua, Binh-Son, Tran, Minh-Khoi, and Yeung, Sai-Kit. Pointwise convolutional neural networks. In *Proc. CVPR* (2018), pp. 984–993.

[55] Hui, Tak-Wai, Loy, Chen Change, and Tang, Xiaoou. Depth map super-resolution by deep multi-scale guidance. In *Proc. ECCV* (2016), Springer, pp. 353–369.

[56] Jampani, Varun, Gadde, Raghudeep, and Gehler, Peter V. Video propagation networks. In *Proc. CVPR* (2017).

[57] Jampani, Varun, Kiefel, Martin, and Gehler, Peter V. Learning sparse high dimensional filters: Image filtering, dense CRFs and bilateral neural networks. In *Proc. CVPR* (2016), pp. 4452–4461.

[58] Jia, Xu, De Brabandere, Bert, Tuytelaars, Tinne, and Gool, Luc V. Dynamic filter networks. In *Proc. NeurIPS* (2016), pp. 667–675.

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

[60] Kazhdan, M., Funkhouser, T., and Rusinkiewicz, S. Rotation invariant spherical harmonic representation of 3D shape descriptors. *Proc. Symposium of Geometry Processing* (2003).

[61] Kiefel, Martin, Jampani, Varun, and Gehler, Peter V. Permutohedral lattice CNNs. In *ICLR workshops* (May 2015).

[62] Kingma, Diederik, and Ba, Jimmy. Adam: A method for stochastic optimization. *arXiv:1412.6980* (2014).

[63] Klokov, Roman, and Lempitsky, Victor. Escape from cells: Deep Kd-Networks for the recognition of 3D point cloud models. In *Proc. ICCV* (2017).

[64] Knopp, Jan, Prasad, Mukta, Willems, Geert, Timofte, Radu, and Van Gool, Luc. Hough transform and 3D SURF for robust three dimensional classification. In *Proc. ECCV* (2010).

[65] Koenderink, Jan J, and Van Doorn, Andrea J. The singularities of the visual mapping. *Biological cybernetics 24*, 1 (1976), 51–59.

[66] Kokkinos, I., Bronstein, M., Litman, R., and Bronstein, A. Intrinsic shape context descriptors for deformable shapes. In *Proc. CVPR* (2012).

[67] Koller, Daphne, and Friedman, Nir. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

[68] Kopf, Johannes, Cohen, Michael F, Lischinski, Dani, and Uyttendaele, Matt. Joint bilateral upsampling. *ACM Trans. Graph. 26*, 3 (2007), 96.

[69] Krähenbühl, Philipp, and Koltun, Vladlen. Efficient inference in fully connected CRFs with Gaussian edge potentials. In *Proc. NeurIPS* (2011), pp. 109–117.

[70] Krähenbühl, Philipp, and Koltun, Vladlen. Parameter learning and convergent inference for dense random fields. In *Proc. ICML* (2013), pp. 513–521.

[71] Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In *Proc. NeurIPS* (2012).

[72] Lafferty, John, McCallum, Andrew, and Pereira, Fernando CN. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. ICML* (2001).

[73] Li, Yangyan, Bu, Rui, Sun, Mingchao, Wu, Wei, Di, Xinhan, and Chen, Baoquan. PointCNN: Convolution on x-transformed points. In *Proc. NeurIPS* (2018), pp. 820–830.

[74] Li, Yijun, Huang, Jia-Bin, Ahuja, Narendra, and Yang, Ming-Hsuan. Deep joint image filtering. In *Proc. ECCV* (2016), Springer, pp. 154–169.

[75] Li, Yijun, Huang, Jia-Bin, Ahuja, Narendra, and Yang, Ming-Hsuan. Joint image filtering with deep convolutional networks. *PAMI* (2018).

[76] Li, Yujia, and Zemel, Richard. Mean field networks. *arXiv:1410.5884* (2014).

[77] Lin, Guosheng, Shen, Chunhua, Van Den Hengel, Anton, and Reid, Ian. Efficient piecewise training of deep structured models for semantic segmentation. In *Proc. CVPR* (2016), pp. 3194–3203.

[78] Lin, Tsung-Yu, RoyChowdhury, Aruni, and Maji, Subhransu. Bilinear CNN models for fine-grained visual recognition. In *Proc. ICCV* (2015), pp. 1449–1457.

[79] Liu, Sifei, Mello, Shalini De, Gu, Jinwei, Zhong, Guangyu, Yang, Ming-Hsuan, and Kautz, Jan. Learning affinity via spatial propagation networks. In *Proc. NeurIPS* (2017).

[80] Long, Jonathan, Shelhamer, Evan, and Darrell, Trevor. Fully convolutional networks for semantic segmentation. In *Proc. CVPR* (2015), pp. 3431–3440.

[81] Lowe, D. G. Object recognition from local scale-invariant features. In *Proc. ICCV* (1999).

[82] Macrini, D., Shokoufandeh, A., Dickinson, S., Siddiqi, K., and Zucker, S. View-based 3-D object recognition using shock graphs. In *Proc. ICPR* (2002), vol. 3.

[83] Mao, Jiageng, Wang, Xiaogang, and Li, Hongsheng. Interpolated convolutional networks for 3d point cloud understanding. In *Proc. ICCV* (2019), pp. 1578–1587.

[84] Maron, Haggai, Galun, Meirav, Aigerman, Noam, Trope, Miri, Dym, Nadav, Yumer, Ersin, Kim, Vladimir G., and Lipman, Yaron. Convolutional neural networks on surfaces via seamless toric covers. *ACM Trans. Graph. 36*, 4 (2017).

[85] Masci, Jonathan, Boscaini, Davide, Bronstein, Michael, and Vandergheynst, Pierre. Geodesic convolutional neural networks on Riemannian manifolds. In *ICCV workshops* (2015).

[86] Maturana, Daniel, and Scherer, Sebastian. 3D convolutional neural networks for landing zone detection from LiDAR. In *Proc. ICRA* (2015).

[87] Monti, Federico, Boscaini, Davide, Masci, Jonathan, Rodola, Emanuele, Svoboda, Jan, and Bronstein., Michael M. Geometric deep learning on graphs and manifolds using mixture model CNNs. In *Proc. CVPR* (2017).

[88] Murase, Hiroshi, and Nayar, Shree K. Visual learning and recognition of 3-D objects from appearance. *IJCV 14*, 1 (1995).

[89] Osada, Robert, Funkhouser, Thomas, Chazelle, Bernard, and Dobkin, David. Shape distributions. *ACM Trans. Graph. 21*, 4 (2002).

[90] Paris, Sylvain, Kornprobst, Pierre, Tumblin, Jack, Durand, Frédo, et al. Bilateral filtering: Theory and applications. *Foundations and Trends® in Computer Graphics and Vision 4*, 1 (2009), 1–73.

[91] Parmar, Niki, Vaswani, Ashish, Uszkoreit, Jakob, Kaiser, Łukasz, Shazeer, Noam, and Ku, Alexander. Image transformer. *arXiv:1802.05751* (2018).

[92] Perronnin, F., Sánchez, J., and Mensink, T. Improving the Fisher kernel for large-scale image classification. In *Proc. ECCV* (2010).

[93] Phong, Bui Tuong. Illumination for computer generated pictures. *Commun. ACM 18*, 6 (1975).

[94] Qi, C. R., Su, H., Nießner, M., Dai, A., Yan, M., and Guibas, L. J. Volumetric and multi-view CNNs for object classification on 3D data. In *Proc. CVPR* (2016).

[95] Qi, Charles R, Su, Hao, Mo, Kaichun, and Guibas, Leonidas J. PointNet: Deep learning on point sets for 3D classification and segmentation. In *Proc. CVPR* (2017).

[96] Qi, Charles R., Yi, Li, Su, Hao, and Guibas, Leonidas. PointNet++: Deep hierarchical feature learning on point sets in a metric space. In *Proc. NeurIPS* (2017).

[97] Razavin, A. Sharif, Azizpour, H., Sullivan, J., and Carlsson, S. CNN features off-the-shelf: An astounding baseline for recognition. In *DeepVision workshop* (2014).

[98] Ren, Shaoqing, He, Kaiming, Girshick, Ross, and Sun, Jian. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Proc. NeurIPS* (2015), pp. 91–99.

[99] Rick Chang, Jen-Hao, and Frank Wang, Yu-Chiang. Propagated image filtering. In *Proc. CVPR* (2015), pp. 10–18.

[100] Riegler, Gernot, Ulusoy, Ali Osman, Bischof, Horst, and Geiger, Andreas. Oct-NetFusion: Learning depth fusion from data. In *Proc. 3DV* (2017).

[101] Riegler, Gernot, Ulusoys, Ali Osman, and Geiger, Andreas. Octnet: Learning deep 3D representations at high resolutions. In *Proc. CVPR* (2017).

[102] Riemenschneider, Hayko, Bódis-Szomorú, András, Weissenberg, Julien, and Van Gool, Luc. Learning where to classify in multi-view semantic segmentation. In *Proc. ECCV* (2014).

[103] Saeedan, Faraz, Weber, Nicolas, Goesele, Michael, and Roth, Stefan. Detail-preserving pooling in deep networks. In *cvpr* (2018), pp. 9108–9116.

[104] Sanchez, Jorge, Perronnin, Florent, Mensink, Thomas, and Verbeek, Jakob. Image classification with the Fisher vector: Theory and practice. *IJCV* (2013).

[105] Savva, Manolis, Yu, Fisher, Su, Hao, Aono, M, Chen, B, Cohen-Or, D, Deng, W, Su, Hang, Bai, Song, Bai, Xiang, et al. SHREC16 track: Largescale 3D shape retrieval from shapenet core55. In *Proc. Eurographics workshop on 3D object retrieval* (2016).

[106] Schneider, Rosália G., and Tuytelaars, Tinne. Sketch classification and classification-driven analysis using Fisher vectors. *ACM Trans. Graph. 33*, 6 (Nov. 2014), 174:1–174:9.

[107] Sedaghat, N., Zolfaghari, M., Amiri, E., and Brox, T. Orientation-boosted voxel nets for 3D object recognition. In *Proc. BMVC* (2017).

[108] Shao, Tianjia, Xu, Weiwei, Yin, Kangkang, Wang, Jingdong, Zhou, Kun, and Guo, Baining. Discriminative sketch-based 3D model retrieval via robust shape matching. In *Computer Graphics Forum* (2011), Wiley Online Library.

[109] Shotton, Jamie, Winn, John, Rother, Carsten, and Criminisi, Antonio. Textonboost: Joint appearance, shape and context modeling for multi-class object recognition and segmentation. In *Proc. ECCV* (2006).

[110] Silberman, Nathan, Hoiem, Derek, Kohli, Pushmeet, and Fergus, Rob. Indoor segmentation and support inference from rgbd images. In *Proc. ECCV* (2012), Springer, pp. 746–760.

[111] Simonyan, K., Parkhi, O. M., Vedaldi, A., and Zisserman, A. Fisher vector faces in the wild. In *Proc. BMVC* (2013).

[112] Simonyan, Karen, Vedaldi, Andrea, and Zisserman, Andrew. Deep inside convolutional networks: Visualising image classification models and saliency maps. *CoRR abs/1312.6034* (2013).

[113] Simonyan, Karen, and Zisserman, Andrew. Very deep convolutional networks for large-scale image recognition. *CoRR abs/1409.1556* (2014).

[114] Sinha, Ayan, Bai, Jing, and Ramani, Karthik. Deep learning 3D shape surfaces using geometry images. In *Proc. ECCV* (2016).

[115] Su, Hang, Jampani, Varun, Sun, Deqing, Maji, Subhransu, Kalogerakis, Evangelos, Yang, Ming-Hsuan, and Kautz, Jan. SPLATNet: Sparse lattice networks for point cloud processing. In *Proc. CVPR* (2018).

[116] Sun, Deqing, Yang, Xiaodong, Liu, Ming-Yu, and Kautz, Jan. PWC-Net: CNNs for optical flow using pyramid, warping, and cost volume. In *Proc. CVPR* (2018), pp. 8934–8943.

[117] Szegedy, Christian, Liu, Wei, Jia, Yangqing, Sermanet, Pierre, Reed, Scott, Anguelov, Dragomir, Erhan, Dumitru, Vanhoucke, Vincent, and Rabinovich, Andrew. Going deeper with convolutions. In *Proc. CVPR* (2015), pp. 1–9.

[118] Tatarchenko, M., Dosovitskiy, A., and Brox, T. Octree generating networks: Efficient convolutional architectures for high-resolution 3D outputs. In *Proc. ICCV* (2017).

[119] Teichmann, Marvin T. T., and Cipolla, Roberto. Convolutional CRFs for semantic segmentation. *arXiv:1805.04777* (2018).

[120] Thomas, Hugues, Qi, Charles R, Deschaud, Jean-Emmanuel, Marcotegui, Beatriz, Goulette, François, and Guibas, Leonidas J. Kpconv: Flexible and deformable convolution for point clouds. In *Proc. ICCV* (2019), pp. 6411–6420.

[121] Tomasi, Carlo, and Manduchi, Roberto. Bilateral filtering for gray and color images. In *Proc. ICCV* (1998).

[122] Uhrig, Jonas, Schneider, Nick, Schneider, Lukas, Franke, Uwe, Brox, Thomas, and Geiger, Andreas. Sparsity invariant cnns. In *Proc. 3DV* (2017), IEEE, pp. 11–20.

[123] 3D Warehouse. `https://3dwarehouse.sketchup.com/`.

[124] Shapeways. `http://www.shapeways.com/`.

[125] The Princeton ModelNet. `http://modelnet.cs.princeton.edu/`. [Online; accessed March 2015].

[126] TurboSquid. `http://www.turbosquid.com/`.

[127] Vaswani, Ashish, Shazeer, Noam, Parmar, Niki, Uszkoreit, Jakob, Jones, Llion, Gomez, Aidan N, Kaiser, Łukasz, and Polosukhin, Illia. Attention is all you need. In *Proc. NeurIPS* (2017), pp. 5998–6008.

[128] Vedaldi, A., and Fulkerson, B. VLFeat: An open and portable library of computer vision algorithms. `http://www.vlfeat.org/`, 2008.

[129] Vogels, Thijs, Rousselle, Fabrice, McWilliams, Brian, Röthlin, Gerhard, Harvill, Alex, Adler, David, Meyer, Mark, and Novák, Jan. Denoising with kernel prediction and asymmetric loss functions. *ACM Trans. Graph. 37*, 4 (2018), 124.

[130] Wang, Peng-Shuai, Liu, Yang, Guo, Yu-Xiao, Sun, Chun-Yu, and Tong, Xin. O-CNN: Octree-based convolutional neural networks for 3D shape analysis. *ACM Trans. Graph. 36*, 4 (2017).

[131] Wang, Xiaolong, Girshick, Ross, Gupta, Abhinav, and He, Kaiming. Non-local neural networks. In *Proc. CVPR* (2018).

[132] Wang, Yue, Sun, Yongbin, Liu, Ziwei, Sarma, Sanjay E, Bronstein, Michael M, and Solomon, Justin M. Dynamic graph cnn for learning on point clouds. *ACM Trans. Graph. 38*, 5 (2019), 1–12.

[133] Weber, Nicolas, Waechter, Michael, Amend, Sandra C, Guthe, Stefan, and Goesele, Michael. Rapid, detail-preserving image downscaling. *ACM Trans. Graph. 35*, 6 (2016), 205.

[134] Wu, Huikai, Zheng, Shuai, Zhang, Junge, and Huang, Kaiqi. Fast end-to-end trainable guided filter. In *Proc. CVPR* (2018), pp. 1838–1847.

[135] Wu, Jialin, Li, Dai, Yang, Yu, Bajaj, Chandrajit, and Ji, Xiangyang. Dynamic sampling convolutional neural networks. *arXiv:1803.07624* (2018).

[136] Wu, Wenxuan, Qi, Zhongang, and Fuxin, Li. PointConv: Deep convolutional networks on 3D point clouds. In *Proc. CVPR* (2019), pp. 9621–9630.

[137] Wu, Zhirong, Song, Shuran, Khosla, Aditya, Yu, Fisher, Zhang, Linguang, Tang, Xiaoou, and Xiao, Jianxiong. 3D shapenets: A deep representation for volumetric shapes. In *Proc. CVPR* (2015).

[138] Xu, Yifan, Fan, Tianqi, Xu, Mingye, Zeng, Long, and Qiao, Yu. SpiderCNN: Deep learning on point sets with parameterized convolutional filters. In *Proc. ECCV* (2018), pp. 87–102.

[139] Xue, Tianfan, Wu, Jiajun, Bouman, Katherine, and Freeman, Bill. Visual dynamics: Probabilistic future frame synthesis via cross convolutional networks. In *Proc. NeurIPS* (2016), pp. 91–99.

[140] Yi, Li, Kim, Vladimir G, Ceylan, Duygu, Shen, I, Yan, Mengyan, Su, Hao, Lu, ARCewu, Huang, Qixing, Sheffer, Alla, Guibas, Leonidas, et al. A scalable active framework for region annotation in 3D shape collections. *ACM Trans. Graph. 35*, 6 (2016), 210.

[141] Yi, Li, Su, Hao, Guo, Xingwen, and Guibas, Leonidas. SyncSpecCNN: Synchronized spectral CNN for 3D shape segmentation. In *Proc. CVPR* (2017).

[142] Yoon, Sang Min, Scherer, Maximilian, Schreck, Tobias, and Kuijper, Arjan. Sketch-based 3D model retrieval using diffusion tensor fields of suggestive contours. In *Proc. International Conference on Multimedia* (2010).

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

[144] Yu, Fisher, Koltun, Vladlen, and Funkhouser, Thomas. Dilated residual networks. In *Proc. CVPR* (2017), pp. 472–480.

[145] Zaheer, Manzil, Kottur, Satwik, Ravanbakhsh, Siamak, Poczos, Barnabas, Salakhutdinov, Ruslan R, and Smola, Alexander J. Deep sets. In *Proc. NeurIPS* (2017), pp. 3394–3404.

[146] Zhang, Han, Goodfellow, Ian, Metaxas, Dimitris, and Odena, Augustus. Self-attention generative adversarial networks. *arXiv:1805.08318* (2018).

[147] Zheng, Shuai, Jayasumana, Sadeep, Romera-Paredes, Bernardino, Vineet, Vibhav, Su, Zhizhong, Du, Dalong, Huang, Chang, and Torr, Philip HS. Conditional random fields as recurrent neural networks. In *Proc. ICCV* (2015), pp. 1529–1537.