A Tiny Diagnostic Dataset and Diverse Modules for Learning-Based Optical Flow Estimation

by

Shuang Xie

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements
For the M.C.S. degree in
Computer Science

uOttawa

School of Electrical Engineering and Computer Science, Faculty of Engineering,
University of Ottawa
Ottawa, Canada

Abstract

Recent work has shown that flow estimation from a pair of images can be formulated as a supervised learning task to be resolved with convolutional neural networks (CNN). However, the basic straightforward CNN methods estimate optical flow with motion and occlusion boundary blur. To tackle this problem, we propose a tiny diagnostic dataset called FlowClevr to quickly evaluate various modules that can use to enhance standard CNN architectures. Based on the experiments of the FlowClevr dataset, we find that a deformable module can improve model prediction accuracy by around 30% to 100% in most tasks and more significantly reduce boundary blur. Based on these results, we are able to design modifications to various existing network architectures improving their performance. Compared with the original model, the model with the deformable module clearly reduces boundary blur and achieves a large improvement on the MPI sintel dataset, an omni-directional stereo (ODS) and a novel omni-directional optical flow dataset.

Acknowledgements

I would like to thank my supervisor, Professor Jochen Lang for his guidance. His trust in my abilities and the academic latitudes he provided were extremely invaluable during my M.C.S study. Professor Lang gave me far more than just the required supervisory feedback and instead provided me with excellent guidance throughout my research journey and the production of this work.

Likewise, I would like to thank all the members of the VIVA Lab for their thoughtful comments on my thesis. My sincere thanks go to Pokong Lai for his insightful suggestions and guidance throughout my research.

Finally, I express my profound gratitude to my parents and my boyfriend, Yang Liu, for their unfailing support and continuous encouragement throughout my years of study and throughout the process of researching and writing this thesis. This accomplishment would not have been possible without them.

Table of Contents

Abstract ii

Acknowledgements iii

Table of Contents iv

List of Figures vii

1 Introduction **1**

1.1 Motivation of the problem 1

1.2 Application of the problem 3

1.3 Thesis statement 4

1.4 Contributions 4

1.5 Thesis outline 5

2 Related Work **7**

2.1 Optical flow estimation methods 8

2.1.1 Variational approaches 8

2.1.2 Learning based methods 9

2.2 Disparity estimation methods 10

2.3 Boundary refinement in flow and disparity estimation 12

2.4 Convolutional neural network 13

2.4.1 Convolution Operation 14

2.4.2 Correlation Convolution Operation 16

2.4.3 Deformable Convolution Operation 17

2.4.4 Coordinate Convolution Operation 19

2.4.5 Depthwise Separable convolution 20

2.5 Datasets 21

2.5.1 Optical flow dataset 23

2.5.2 Diagnostic dataset 24

2.5.3 Omni-directional stereo dataset 26

3 Datasets **27**

3.1 FlowClevr Dataset 27

3.2 Basic Information on the FlowClevr Dataset 28

3.2.1 Spatial Transformation of the Square 29

3.2.1.1 Stretching 30

3.2.1.2 Shearing 30

3.2.1.3 Rotation 31

3.2.2 Motion Vector Calculation 32

3.2.3 Training and Testing Set 33

3.3 Omni-directional Flow Dataset 34

3.3.1 Basic Information on the Omni-directional Flow Dataset 34

3.3.2 Generating the Image Pair and its Corresponding Optical Flow Groundtruth in the Omni-directional Flow Dataset 35

4 Flow Estimation Networks **37**

4.1 SimpleNet 38

4.1.1 Basic SimpleNet 38

4.1.2 SimpleNet with different modules 39

4.1.2.1 SimpleNet with correlation convolution 39

4.1.2.2 SimpleNet with coordinate convolution 40

4.1.2.3 SimpleNet with deformable convolution 41

4.2 FlownetC 42

4.2.1 FlownetC 42

4.2.2 FlownetC-D 44

4.2.3 FlownetC-DD 44

4.3 Spatial Pyramid Network 45

4.3.1 SPyNet 46

4.3.2 SPyNet-D 46

4.3.3 SPyNet-DD 47

4.4 PWC-Net . . . 47
4.4.1 PWC-Net . . . 49
4.4.2 PWC-Net-D . . . 50
4.4.3 PWC-Net-DD . . . 51

5 Panoramic Optical Flow and Depth 52
5.1 SepUNet . . . 52
5.2 SepUNet with deformable modules . . . 54

6 Experiments 56
6.1 SimpleNet for FlowClevr dataset . . . 57
6.1.1 Experiment setup . . . 57
6.1.2 Experimental result . . . 58
6.1.3 Analysis . . . 61
6.2 Experiments for MPI Sintel dataset . . . 64
6.2.1 Experiment setup . . . 64
6.2.2 Experimental result . . . 65
6.2.3 Analysis . . . 67
6.3 SepUNet for Omni-directional Flow datasets and Omni-directional Stereo datasets . . . 71
6.3.1 Experiment setup . . . 71
6.3.2 Experimental result and analysis for Omni-directional Flow datasets 73
6.3.3 Experimental result and analysis for Omni-directional Stereo datasets 76

7 Conclusions 79
7.1 Conclusions . . . 79
7.2 Limitations and future work . . . 81
References . . . 82

List of Figures

2.1 2D convolution operation. The green image represents a simple input feature for 2D convolution, the yellow image is the kernel of size 3×3 and the result convolved feature is the pink image. 15

2.2 2D convolution operation with padding. The green image represents a simple input feature with zeros padding around the bounder; the orange image is the kernel of size 3 x 3 and the convolved feature (pink image) is the same size with input feature. 16

2.3 Illustration of the sampling locations in $3*3$ standard and deformable convolutions. (a) regular sampling grid (green points) of standard 2D convolution operation. (b) the blue arrows indicate the offsets applied to regular sampling grids, the deformed sampling locations are shown as dark blue points.(c)(d) are special cases of (b), showing that the deformable convolution generalizes various transformations for scale, (anisotropic) aspect ratio and rotation. 18

2.4 Illustration of 3×3 deformable convolution. 19

2.5 The input of the CoordConv layer is concatenating the input image with hard-coded coordinates, the most basic version of the hard-coded coordinates are i coordinate and j coordinate. 20

2.6 Illustration of i,j coordinate channels. (a) The channel for i coordinate, the integer value in each row is equal to the row number. (b) The channel for j channel, the integer value in each column is equal to the column number. 21

2.7 Depthwise separable convolution replace standard convolution(a) with depthwise convolution(b) and pointwise convolution(c). 22

2.8 The visualization example Not-so-Clevr dataset. The white areas in the figures equal to 1 and the black areas are filled with 0. (a) The example images. (b) The example one-hot label. (c) The pixelwise sum of all one-hot label in the whole train/test set. 25

3.1 The corresponding relationship between the normalized flow vector map and color map. 29
3.2 Example of transformed objects and their flow including the respective original images. 30
3.3 The visualization examples of the flow estimation of rendered object pairs. 32
3.4 The visualization of train/test set split. (a) Quadrant split. (b) Uniform split. 33
3.5 The visualization examples of the image pairs and its relevant flow samples in Omnidirectional Flow Dataset. 36

4.1 Illustration of SimpleNet architecture. 39
4.2 Illustration of SimpleCorrNet architecture. 40
4.3 Illustration of SimpleCoordNet architecture. 41
4.4 Illustration of SimpleDeformNet architecture. 42
4.5 The FlowNetC architecture. 43
4.6 Refinement of the coarse feature maps to the high resolution prediction. [18] 43
4.7 Illustration of FlownetC-D. The green arrow represents the deformable convolution module. 44
4.8 Illustration of FlownetC-DD. The pink arrow represents two stacked deformable convolution modules. 45

4.9 Inference in a 3-Level Pyramid Network: The network G_0 computes the residual flow v_0 at the highest level of the pyramid (smallest image) using the low resolution images I_0^1 and I_0^2. At each pyramid level, the network G_k computes a residual flow v_k which propagates to each of the next lower levels of the pyramid in turn, to finally obtain the flow V_2 at the highest resolution. 47
4.10 The architecture of G_k in SPyNet and its variants. (a) represents G_k architecture in SPyNet. (b) represents G_k architecture in SpyNet-D. (c)represents G_k architecture in SpyNet-DD. 48
4.11 Network architecture of PWC-Net at the top two levels. For the rest of the pyramidal levels, the flow estimation modules have the same structure as the second to the top level. u represents upsampling operation and w represents warping operation. 49
4.12 The optical flow estimator network at pyramid level 2. Each convolutional layer is followed by a leaky ReLU unit except the last (light green) one that outputs the optical flow. c_l^t denotes extracted features of image t at level l, w_l denotes the predicted flow at level l. 50
4.13 The optical flow estimator network in PWC-Net-D at pyramid level 2. . . 50
4.14 The optical flow estimator network in PWC-Net-DD at pyramid level 2. . 51

5.1 General structure of SepUNet. The coloured arrows represents the corresponding modules shown in the bottom sub figure. 53
5.2 General structure of SepUNet-D. The coloured arrows represents the corresponding modules shown in the bottom sub figure. 54
5.3 General structure of SepUNet-DD. The coloured arrows represents the corresponding modules shown in the bottom sub figure. 55

6.1 The learning rate of optimizer in the whole training process. 58

6.2 The average EPE in FlowCLEVR for 7 networks ("SN" denotes "SimpleNet"), where the x-axis is the EPE value. The red, green, blue and gray bar represent the dataset with rotation, shearing, non-uniform scaling (stretching) and no transformation, respectively, while the forward and backward slash present the average EPE on training and testing set, respectively. The standard error and the value of average EPE are noted on the top of the bars. Results shown are for the uniform dataset, the results for the quadrant set are very similar (see the supplemental material). . . 59

6.3 The average EPE in FlowCLEVR quadrant set for 7 networks ("SN" denotes "SimpleNet"), where the x-axis is the EPE value. The red, green, blue and gray bar represent the dataset with rotation, shearing, non-uniform scaling (stretching) and no transformation, respectively, while the left and right slash present the average EPE on training and test set, respectively. The standard error and the value of average EPE are noted on the top of the bars. 60

6.4 The training loss trend of SimpleNet, SimpleNet-Coord, SimpleNet-Deform, SimpleNet-Corr. In each subfigure, the x axis is the training epoch, while the y axis is the L1 loss. 62

6.5 The training loss trend of SimpleNet-DD, SimpleNet-CC, SimpleNet-D, SimpleNet-DD. In each subfigure, the x axis is the training epoch, while the y axis is the L1 loss. 63

6.6 The visualization flow predicted by different networks on Sintel training clean set. 65

6.7 The visualization flow predicted by different networks on Sintel training final set. 67

6.8 The visualization flow predicted by different networks on Sintel testing clean set. 68

6.9 The visualization flow predicted by different networks on Sintel testing final set. 69

6.10 Visualization of SepUNet, SepUNet-D and SepUNet-DD results on the ODF training dataset. 74
6.11 Visualization results of SepUNet, SepUNet-D and SepUNet-DD on ODF testing dataset. 74
6.12 Visualization results of SepUNet, SepUNet-D and SepUNet-DD on ODS training dataset. 76
6.13 Visualization results of SepUNet, SepUNetCD and SepUNet-DD on ODS testing dataset. 77

Chapter 1

Introduction

1.1 Motivation of the problem

Optical flow estimation is a well-known problem in computer vision introduced by Gibson [25] to describe the human visual perception of a stimulus object. It is a low-level computer vision problem but is the core for many high-level computer vision applications such as object tracking [61, 56, 31] and autonomous driving [60, 24, 47]. However, due to the camera aperture, optical flow is not directly measurable [27, 28]. Hence, the estimation is typically solved by complex energy minimization introduced by Horn and Schunck [27], which is time-consuming and computationally expensive.

With the significant progress of deep learning in recent years, one appealing approach

is to adopt fast, end-to-end convolutional neural networks (CNN) into optical flow estimation. However, unlike energy minimization methods, basic straightforward CNN methods estimate the optical flow with vague boundaries. To tackle this problem, Dosovitskiy *et al.* [18] utilizes the variational method [9] and boundary limitation [45] to make the boundaries slightly clearer but the method requires expensive computation. Solutions based on pure end-to-end learning are also pursued by researchers. Deqing *et al.* [79] adopts the dilated convolution operation [89] to refine details of final optical flow estimation. Tak-Wai *et al.* [29] proposes a flow regularization model which folds and packs the filter into a 3d tensor and makes the filter flow- and image-aware. Many of these CNN refinement methods perform on par with typical variational approaches and run much faster.

Vague boundaries are not only a problem in flow estimation, but also in object segmentation and in disparity estimation. Chen *et al.* [13] propose atrous spatial pyramid pooling to help the network gain context awareness and reduce boundary blur between the foreground object and background images in the segmentation task. Dai *et al.* [17] bring the deformable convolutional module to object segmentation to learning the shape and context information which decreases the error caused by boundary blur. Chang *et al.* [11] adopt the stacked hourglass module to refine the stereo estimation further to eliminate the boundary error as much as possible. Lai *et al.* [41] also finds that camera distortion can increase boundary blur and reduce the accuracy in depth estimation. They use a panoramic loss to reduce the error.

On the other hand, due to the camera aperture, it is difficult to obtain an optical flow dataset with ground truth directly from the real-word. It is difficult to obtain an optical flow dataset with ground truth directly from the real-word. The large scale datasets of optical flow [30] are mainly from synthetic 3D virtual environments. The scale of datasets recorded from the real-world [22] is limited, making it difficult to train a CNN from scratch. In addition, datasets for diagnostic and panoramic images are also missing in optical flow estimation.

1.2 Application of the problem

Optical flow has widely been used in motion estimation and video compression. The applications of optical flow include the tasks of inferring the motion of the observer and the objects in the scene, but also tasks related to the estimation of the structure of objects and the environment. Optical flow is recognized as useful information in many areas such as object detection and tracking, image dominant plane extraction, movement detection, robot navigation, and visual odometry.

An omni-directional image is an image which has a field of view that covers approximately the entire sphere or at least a full circle in the horizontal plane. Omni-directional images or videos are important in areas where coverage of a large visual field are needed, such as in panoramic photography, robotics, and for AR/VR headset. Optical flow in omni-directional images is able to provide useful information for vision tasks in these kinds of applications, such as object detection and tracking, robot navigation and 3d modeling and rendering for free viewpoint video with VR headsets.

The omni-directional stereo image is able to extract the depth information for the panoramic view. Based on this, there are several systems which employ a limited number of cameras with fisheye lenses for capturing omni-directional video. The depth map can be combined with a color panorama to produce a 3D mesh thus providing full 6 DoF viewing. In this case, the extract video is able to live-streaming the VR content with 6 DoF. The optical flow for the omni-directional image is also able to calculate the depth and provide the 6 DoF viewing by single camera when the position of the camera is given.

Depth information for the panoramic view can be calculated from a set of omni-directional stereo images. Because of this, there are several systems [41] which employ a limited number of cameras with fisheye lenses for capturing omni-directional video. The depth map can be combined with a color panorama in a 2.5D mesh serving as a basis for full 6 DoF viewing, i.e., the video can be correctly rendered for a viewer during head rotation and translation. In this case, real-time stereo estimation is necessary for live-streaming of VR content with 6 DoF. Optical flow for omni-directional images enables interpolation between omni-directional video frames or in still images, e.g., Matzen et

al. [58] use optical flow to find dense correspondence between equirectangular images in their 360 video capture system. Optical flow also gives relative depth cues which is relevant to AR/VR.

1.3 Thesis statement

Learning-based optical flow methods estimate flow with large errors at boundaries causing blurred flow boundaries. We aim to show in this thesis that these methods can be improved by developing and evaluating layers on a dedicated dataset with a simple network, and that the flow estimation methods developed with such a dataset will scale to state-of-the-art methods applied to large standard datasets.

1.4 Contributions

A well-known issue of CNN-based flow estimation is the vague boundaries. A hypothesis of the reason for this situation is that the basic convolution operation can not perceive the shape of the object in the whole image field since the convolution operation learns the filter locally but shares the weights and bias in the whole feature map. Therefore, we propose a diagnostic flow estimation dataset, evaluate different convolution modules on it and study which module can improve the results. We apply our insights gained with diagnostic dataset in experiments on the MPI Sintel dataset, an omni-directional stereo dataset and a novel omni-directional flow dataset. In this thesis, our main contributions are:

- We propose a simple tiny diagnostic FlowClevr dataset for quickly training and testing the effectiveness of various neural network layers for optical flow estimation;
- We propose a synthetic omni-directional flow dataset for training and testing the flow estimation on the panoramic image.
- We evaluate different modules on the FlowClevr datasets and find the effective modules which can improve the original networks in the range from 30% to 100%.

- We modify the state-of-the-art flow estimation networks with the effective modules and achieved better performance in most networks on MPI Sintel clean datasets and MPI Sintel final datasets, respectively.

- We also applied the effective modules to the panoramic flow and disparity estimation task and achieved 10.04% improvement in end-point-error on omni-directional flow datasets and 9.84% improvement in depth error on omni-directional stereo datasets.

- The full implementation of the models with Pytorch and the generated synthetic diagnostic datasets are available on Github: https://github.com/ShuangXieIrene/disp-flow-pytorch

This thesis also includes collaborative work on panoramic flow and disparity estimation made by me and Dr. Po Kong Lai, under the supervision of Dr. Jochen Lang and Dr. Robert Laganiere. In the collaboration, I have mainly focused on the training and testing different architectures, while Pokong Lai proposed the SepUNet and generated the omni-directional flow dataset and the omni-directional depth dataset.

1.5 Thesis outline

The thesis is organized as follows:

- Chapter 2 introduces the background and related work on optical flow and disparity estimation. We group the methods for optical flow estimation into two categories, Variational-based methods, and learning based methods. We also introduce the computation procedures of the basic convolution operation and its variants: correlation operation, deformable convolution operation, coordinate convolution operation, depthwise convolution operation. After that, we introduce the methods for disparity estimation. At the end of this chapter, we present widely used optical flow and disparity datasets and the diagnostic Clevr dataset.

- Chapter 3 presents a simple tiny diagnostic dataset for optical flow estimation called FlowCLEVR and a synthetic omni-directional flow dataset called ODF. The details of the generation and train/test split are discussed in these two chapters.

- Chapter 4 presents the model architectures we used for optical flow estimation. It includes two sections. We first present a SimpleNet architecture and its derivatives with different modules used with the FlowClevr dataset. After that, we present FlowNetC, SPyNet and PWC-Net architectures and their derivatives with effective modules used in MPI Sintel datasets. Chapter 5 describes SepUNet architecture and its variants used in panoramic flow and disparity estimation tasks.

- Chapter 6 presents the experimental results and analysis of the models in Flow-Clevr, MPI Sintel, and ODF/ODS datasets.

- Chapter 7 summarize the thesis and presents its limitations and possible future research directions.

Chapter 2

Related Work

In this chapter, we first introduce the variational approaches and learning based approaches to estimate optical flow. We then describe the approaches to estimate disparity from aligned images. The motion boundary problem and its solutions in disparity and optical flow estimation is introduced in Section 2.3. In the convolution operation section, we introduce the calculation and limitations of convolution, correlation, deformable convolution operation, coordinate convolution operation, and depthwise separable convolution. Finally, we present the widely used optical flow datasets, the diagnose Clevr datasets and the ODS dataset.

2.1 Optical flow estimation methods

Optical flow estimation is a well-known problem in computer vision introduced by Gibson [25] to describe the visual stimulus provided to animals moving through the world. To be more specific, optical flow is the motion between two frames at time t and $t + \Delta t$, the motion of the two frames are normally caused by the motion of the camera or the object in the scene. The current optical flow estimation can be divided into two approaches: variational approaches which estimate optical flow by optimizing the energy function, learning based approaches which use learned parameters to tackle the task.

2.1.1 Variational approaches

Optical flow is a highly challenging low-level vision problem due to the complicated phenomena such as motion discontinuity, untextured regions, and sensor noise. To overcome those difficulties, Lucas and Kanade [54] propose an image registration method that is using spatial intensity information to calculate the best match region between the two frames. However, this method minimizes the energy function between the compared windows and it can generalize the aperture problem. To be more specific, the windows' size is either too small to provide sufficient information or too big and hence spanning over the motion boundaries.

To alleviate this problem, Jepson and Black [34] use parametric and mixtures models to represent multiple motions within a window. They use the extension of the EM-algorithm to compute a maximum likelihood estimate for the motion parameters. Although they provide robust optical flow estimations results of the motion boundaries, the window-based model fails to estimate accurate optical flow result of the motion of deformable non-rigid bodies and poorly-textured regions.

Horn and Schunck [27] first propose a global model for optical flow estimation. They pioneer the variational approach to optical flow by coupling the brightness constancy and spatial smoothness assumptions using a global energy function. The energy function is usually composed of a data term that encourages agreement between frames and a spatial term (i.e., regularization) that enforces consistency of the flow field. Markov

random fields (MRF) are closely related to energy-based models [43] in that the node and clique potentials of an MRF are often defined in terms of energy or cost functions in general exponential families. Thus equivalence can be drawn between the negative log-posterior of MRF models and global energy functions. However, this method still has limitations on the motion boundaries due to the brightness constancy and spatial smoothness assumptions violated near motion boundaries.

2.1.2 Learning based methods

Early work on learning optical flow was first introduced by Simoncelli and Adelson [75]. They study the data matching errors for optical flow. Freeman et al. [21] learn parameters of an MRF model for image motion using synthetic blob world examples. Roth and Black [72] proposed a supervised learning technique for general optical flow estimation. They learn the spatial statistics of optical flow using the proposed Field-of-Experts(FoE) model [71]. Sun et al. [78] learn a full model for optical flow, but the learning has been limited to a few training sequences [4]. Li and Huttenlocker [46] use stochastic optimization to tune the parameters for the Black and Anandan method [7], but the number of parameters learned is limited. Wulff and Black [86] learn a PCA motion basis of optical flow estimated by GPU-Flow [85] on real movies. Their method is fast but produces over-smoothed flow.

Recent work has shown that optical flow estimation can be cast as a supervised learning task to be solved with deep learning methods.

Inspired by the success of applying convolutional neural networks(CNNs) on other computer vision tasks [39], Dosovitskiy et al.[18] construct FlownetS and FlownetC to learn optical flow. FlownetS is based on the U-Net[70] encoder-decoder architecture with skip connections between the contracting and the expending parts of the network. To further help the network learn the correspondence between two frames, they introduce correlation layer in FlowNet, named FlowNetC. However, FlowNetC archives similar result as FlowNetS due to the limitation of correlation operation on large motions.

The successor of FlowNet, called Flownet2 [30], stacks several FlowNetC and FlowNetS networks into a large model which achieve similar performance with state-of-the-art meth-

ods but the model, as a result, requires 640MB memory for one input pair.

Ilg et al. [30] develop Flownet 2.0 architecture that stacks several basic Flownet models. They also introduce a warping operation that compensate for some already estimated preliminary motion in the second image. Using the warping operation in the stacked architecture can improve the results significantly. Flownet 2.0 decrease the estimation error by more than 50% than original Flownet and performs on par with state-of-the-art methods, while running at interactive frame rates.

Chang et al.[11] propose a pyramid stereo matching network (PSMNet) consisting of spatial pyramid pooling and 3D CNN modules. The spatial pyramid helps to learn the context information by aggregating different scales and locations to form a cost volume. They develop two kinds of 3D CNN modules to regularize the cost volume: a basic architecture and a stacked hourglass architecture. In their experiments, PSMNet significantly decreases errors in ill-posed regions such as occlusion areas, repeated patterns, textureless regions, and reflective surfaces.

PWCNet [79] introduces the pyramidal process and feature warping to address the huge model size problem of the stack networks. PWCNet performs on par with the state-of-the-art methods and Flownet2, but the model size is 17 times smaller, and the speed is two times faster than FlowNet2.

Compared with the traditional methods to estimate optical flow, recent deep learning methods achieve more accurate results with faster speed. In the next section, we focus on describing CNN and its basic operation: convolution. Also, based on the limitation of convolution such as fixed-size constraint, we also explore correlation convolution, deformable convolution, and coordinate convolution.

2.2 Disparity estimation methods

Disparity refers to the distance along the epipolar line for the same object seen. If the two images are taken by cameras at different horizontal positions, and one pixel at position (x, y) in the left image, appears at position $(x-d, y)$ in the right image. The disparity for that pixel in the left image is d. If we know the disparity of an object, we can compute

its depth z using the equation 2.1 under the condition of using pinhole cameras with parallel optical axes that are scanline aligned, where f is the focal length of the camera and B is the distance between the camera centers.

$$z = \frac{fB}{d} \tag{2.1}$$

The described problem of stereo matching is important in many fields such as autonomous driving, robotics, and 3D scene reconstruction. A typical stereo matching algorithm [73] consists of four steps: matching cost computation, cost aggregation, optimization, and disparity refinement. The current state of the art studies focuses on how to estimate disparity using CNNs accurately. The CNNs-based approaches can be roughly divided into three categories: matching cost learning, regularity learning, and end-to-end disparity learning.

Matching cost learning is using CNNs to measure the similarity between image patches. Han et al. [26] proposed MatchNet, which consists of a deep convolutional network that extracts features from patches and a network of three fully connected layers that computes similarity between the extracted features. Concurrently, Zbontar et al.[90] investigated a series of CNN architectures for binary classification of pairwise matching and applied in disparity estimation. In contrast to an independent binary classification scheme between image patches, Luo et al. [55] proposed to learn a probability distribution over all disparity values. This strategy employs a diverse set of training samples without considering the imbalance in the training samples. However, these architectures that rely on patch-based Siamese networks are lacking the means to exploit context information for finding correspondence in ill-posed regions.

Regularity learning is based on the observation that disparity images are generally piecewise smooth; some existing works add smoothness constraints in the learning process. Menze et al. [60] applied adaptive smoothness constraints using texture and edge information for a dense stereo estimation. Besides, disparity can also be regularized by incorporating high-level vision tasks. For example, disparity was estimated concurrently by solving the problem of semantic segmentation, e.g., [8, 40].

End-by-end disparity learning is using CNNs to predict whole disparity maps without

post-processing. Mayer et al. [59] present end-to-end networks for the estimation of disparity (DispNet) and optical flow (FlowNet). Pang et al. [64] extend DispNet [59] and introduce a two-stage network called cascade residual learning (CRL). The first and second stages calculate the disparity map and its multi-scale residuals, respectively. Then the outputs of both stages are summed to form the final disparity map. Both DispNet [59] and CRL [64] reuse hierarchical information, concatenating features from lower layers with those from higher layers. However, these models do not do well in inherently ill-posed regions, such as object occlusions, repeated patterns, or textureless regions. To tackle this problem, Chang et al. [11] propose a pyramid stereo matching network (PSMNet), which consists of two main modules: spatial pyramid pooling (SPP) and 3D CNN. The SPP module incorporates different levels of feature maps to form a cost volume. The 3D CNN further learns to regularize the cost volume via repeated top-down/bottom-up processes.

2.3 Boundary refinement in flow and disparity estimation

Boundary blur always exists in the variational approaches and learning based approaches of flow and disparity estimation. The addition of descriptor matching is the most popular way to clear the estimation boundary. Brox and Malik use penalization of the difference between flow and HOG matches to the energy function to improve the boundary blur by HOG features [9]. Weinzaepfel *et al.* [84] replaces the HOG matches by an approach based on similarities of non-rigid patches to get a finer refinement. Xu *et al.* [87] merged the estimated flow with matching candidates at each level of the coarse-to-fine scheme to refine the boundary based on motion detail. Lu *et al.* [53] propose a variant of PatchMatch [5], which uses SLIC superpixels [1] as basic blocks in order to better respect image boundaries. The purpose is to produce a nearest-neighbor-field (NNF) which is later translated into a flow. However, SLIC superpixels are only locally aware of image edges, whereas our edge-aware distance can capture regions at the image scale. Similarly, Chen *et al.* [14] propose to compute an approximate NNF, and then estimate

the dominant motion patterns using RANSAC. Then, they use a multi-label graph-cut to solve the assignment of each pixel to a motion pattern candidate. Their multi-label optimization can be interpreted as a motion segmentation problem or as a layered model [77]. Another approach is directly using the estimated boundary to refine the flow and disparity estimation. Ren [14] proposes to use edge-based affinities to group pixels and estimate a piece-wise affine flow. Jerome *et al.* [68] also use edge preserving interpolation of correspondences technology to refine the estimation result based on the edge information of the objects. FlowNet [18] also computes image boundaries of the objects and uses the coarse to fine scheme with 20 iterations to refine the boundary of estimation. These descriptor matching methods can get a reliable result when the boundary is clear in the original image. However, if there is motion blur in the original image, these matching methods often fail to eliminate boundary blur in the estimation result. Therefore, descriptor matching could help the estimation performing well in the clean pass of MPI Sintel dataset, while it fails to improve the final performance in the final pass of MPI Sintel dataset. To tackle this problem, Xu *et al.* introduces multi-stage dilated convolution to use the context information to refine the boundary rather than estimate the boundary itself. Compared with the descriptor matching refinement, it also achieves outstanding performance in the final pass of the MPI Sintel dataset. However, limited to dilated convolutional operation, the final estimation for PWC-Net disappear in regions where the object is thin. Therefore, a context- and detail-aware method is needed to refine the boundary in flow and disparity estimation.

2.4 Convolutional neural network

Neural networks have been studied since the middle of the last century [82] and have experienced a surge in development at the end of the last century. A neural network is a hierarchical model inspired by the structure of the brain. A network consists of a series of nodes connected according to specific rules [44]. The two most obvious characteristics of neural networks are the introduction of non-linear activation functions and the use of a nested design between net layers. This allows it to represent a highly non-linear

(relative to the input) function. Thus neural networks have a strong modeling capability for complex data patterns [44].

Neural networks have developed into many types. A commonly used neural network is a CNN. CNNs were introduced by Y. LeCun [44] and gained prominence in the field of computer vision after the success of AlexNet in the general image classification task in 2012 [16]. They have achieved success in recognition tasks, image classification [39], heart rate estimation [66], optical flow estimation [18], disparity estimation [18], and depth estimation [19]. However, the convolution operation in CNNs has limitations. In the next subsections, we introduce the calculation and limitations of the convolution operation and its variants.

2.4.1 Convolution Operation

The convolution operation is the core building block of a Convolutional Network that does most of the computation.

The 2D convolution operation parameters consist of a set of learnable filters. Every filter is small spatially (along width and height), but extends through the full depth of the input volume. During the forward pass, each filter slides across the width and height of the input volume and compute dot products between the entries of the filter and the input at any position. Each filter that slides over the width and height of the input volume will produce a 2-dimensional activation map that gives the responses of that filter at every spatial position. Stacking all the activation maps that are produced by the entire set of filters along the depth dimension will produce the output volume.

In order to introduce the 2D convolution operation, one example is shown in Figure 2.1. The green image represents a 5×5 input volume; the yellow image represents a 3×3 filter, and the output volume is the pink image. The 3×3 filter slides over the 5×5 input volume by 1 pixel with stride 1. For every position, we compute element-wise multiplication (between input volume and filter) and sum up the multiplication outputs to compute the final value that forms a single element of the output volume.

Another hyperparameter that controls the size of the output volume is zero-padding, which pad the input volume with zeros around the border. Commonly, the size of zero

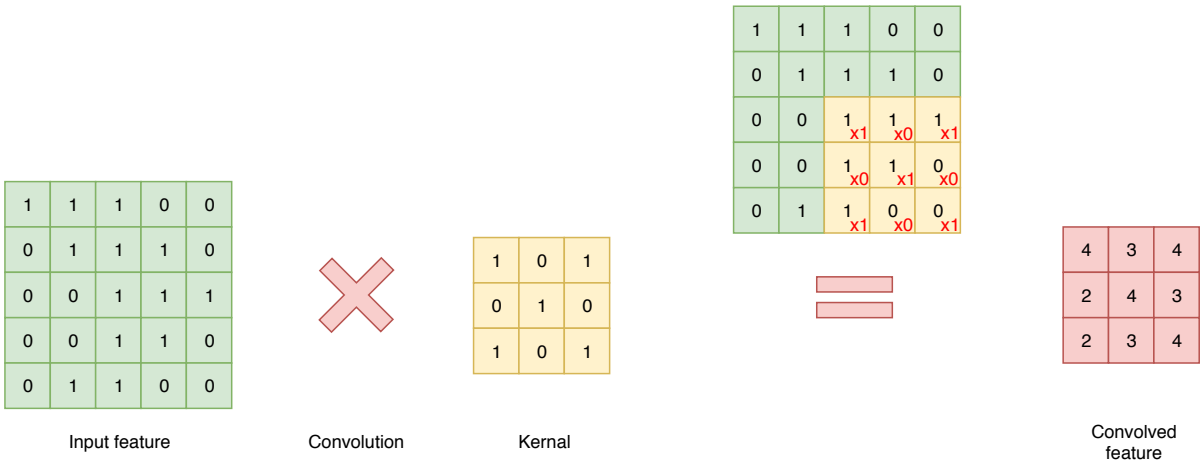

Figure 2.1: 2D convolution operation. The green image represents a simple input feature for 2D convolution, the yellow image is the kernel of size 3×3 and the result convolved feature is the pink image.

padding is chosen to preserve the spatial size of the input volume precisely, so the input and output width and height are the same. The use of zero padding is shown in Figure2.2.

The two major features of the convolution operation are local connectivity and parameter sharing. Normally, the input image is high dimensional, so instead of connecting the neurons to all the previous neurons, we will connect the neurons to a local region in the previous volume. The spatial extent of the local connectivity is a hyperparameter called the receptive field of the neuron which is equal to the kernel size. The local connectivity is asymmetric; the connections are local in the width and height dimensions, however, in the depth dimension, the extent of the connectivity is always equal to the input volume depth. The convolution operation uses shared weights to compute the output feature on each local region in the whole image, which is called parameter sharing. Parameter sharing is helpful to control the number of parameters. Also, based on the parameter sharing, convolution operation is spatially invariant. The learned parameters of the convolution operation apply to the whole image and do not depend on the pixel position of the input image.

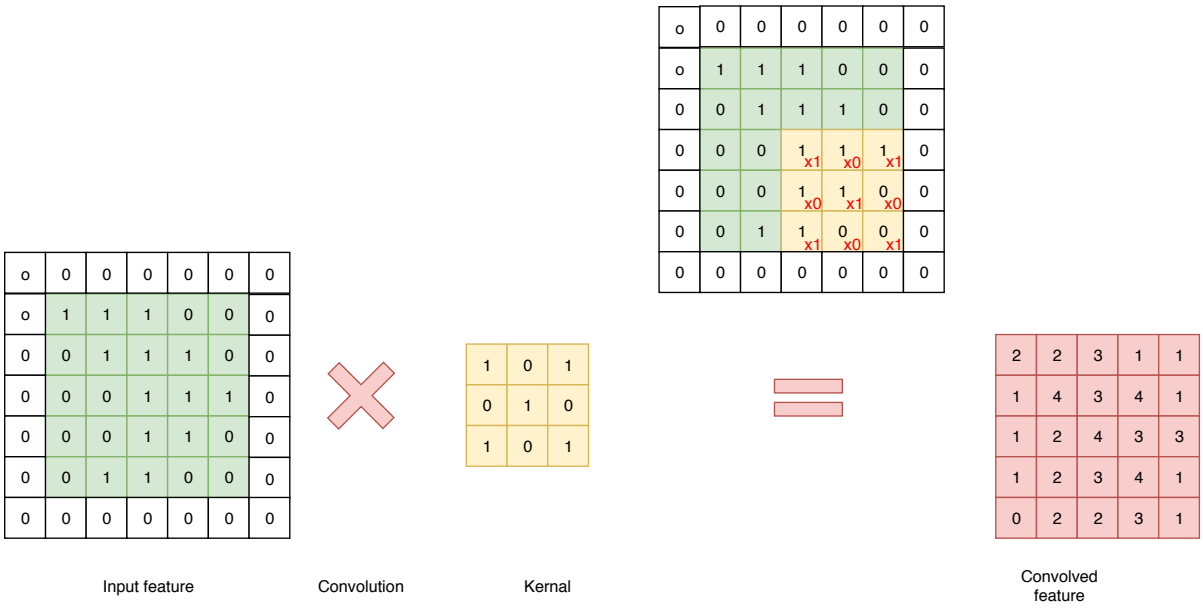

Figure 2.2: 2D convolution operation with padding. The green image represents a simple input feature with zeros padding around the bounder; the orange image is the kernel of size 3 x 3 and the convolved feature (pink image) is the same size with input feature.

2.4.2 Correlation Convolution Operation

Fischer et al. [18] proposed a correlation layer that computes the correlation of feature maps through calculating multiple patch convolutions between two feature maps. Adding a correlation layer in the network helps the network to learn the correspondence between images.

Correlation convolution operation performs multiplicative patch comparisons between two feature maps. Given two multi-channel feature maps f_1, f_2, with w, h, and c being their width, height and number of channels, the correlation layer lets the network compare each patch from f_1 with each path from f_2. Eq. 2.2 shows the computation between a square patch of size $K := 2k + 1$ centered at x_1 and the same size patch centered at x_2. The operation is shown in Eq. 2.2 is the same than the convolution operation in neural networks. Correlation convolution operation has no trainable weights. This is because

that it convolves data with other data instead of convolving data with a trainable kernel.

$$c(x_1, x_2) = \sum_{o\epsilon[-k,k]\times[-k,k]} < f_1(x_1 + o), f_2(x_2 + o) > \tag{2.2}$$

Computing $c(x1, x2)$ involves $c \times K^2$ multiplications. Comparing all patch combinations involves $w^2 \times h^2$ such computations yields a large result and makes efficient forward and backward passes intractable. Hence, to further reduce the computation cost, a maximum displacement d is introduced, for each location x1, we compute correlations c(x1, x2) only in a neighborhood of size $D := 2d + 1$, by limiting the range of x2. The relative displacements are organized in channels, so the output volume size is $w \times h \times D^2$.

2.4.3 Deformable Convolution Operation

Dai et al. [17] proposed a deformable convolutional layer to solve the problem that CNNs are inherently limited in learning geometric transformations due to the spatial invariance of the convolution operation. Compared with 2D convolution operation, deformable convolution adds a 2D offset to the regular grid sampling locations in the standard convolution, which enables free form deformation of the sampling grid. Some examples of the offset added to the regular grid sampling locations are illustrated in Figure 2.3. Figure 2.3 (a) shows the regular 2D convolution grid (green points) that defines the receptive field size and dilation. Figure 2.3 (b) shows the deformed sampling grid (dark blue points) that applies the learned offsets (light blue arrows) on a regular sampling grid. Figure 2.3 (c)(d) are special cases of (b).

The deformable convolution consists of three parts. First, a convolutional layer is applied over the input feature to learn the offsets. The output of the convolutional layer are the offset fields. The spatial dimension of the offsets fields is the same with the input feature map, and the depth dimension is 2N corresponing to N 2D offsers. Then, the regular grid is augmented using the learned offsets fields. Last, another convolutional layer is applied over the same input feature map but using the deformed sampling locations. The process of the deformable convolution is illustrated in Figure 2.4, the offsets are obtained by applying a convolutional layer over the same input feature map. The

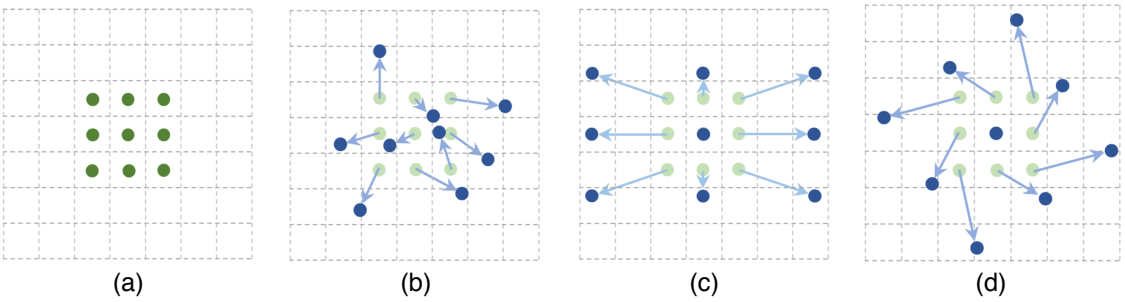

Figure 2.3: Illustration of the sampling locations in 3∗3 standard and deformable convolutions. (a) regular sampling grid (green points) of standard 2D convolution operation. (b) the blue arrows indicate the offsets applied to regular sampling grids, the deformed sampling locations are shown as dark blue points.(c)(d) are special cases of (b), showing that the deformable convolution generalizes various transformations for scale, (anisotropic) aspect ratio and rotation.

channel dimension 2N corresponds to N 2D offsets (e.g., $N = 3 \times 3$ in Figure 2.4). The calculation of the deformable convolution is expressed in Equation 2.3,

$$y(p) = \sum_{k=1}^{K} w_k \cdot x(p + p_k + \Delta p_k) \cdot \Delta m_k \tag{2.3}$$

where $x(p)$ and $y(p)$ denote the features at location p from the input feature maps x and output feature maps y, respectively; K expresses the sample location of a give kernel; w_k and p_k denote the weight and pre-specified offset for the k-th location, respectively; Δp_k and Δm_k are the learnable offset and modulation scalar for the k-th location, respectively. The modulation scalar Δm_k lies in the range [0, 1], while Δp_k is a real number with unconstrained range.

Compared with the original convolution operation, deformable convolution operation has additional learnable offsets and modulation scalar for all kernel. These features allow the deformable convolution kernel to learn flexible geometric information and to be sensitive to the context information. The deformable feature has been applied successfully the success in detection and segmentation tasks. However, the sampling may be on irregular offset locations $p_n + \Delta p_n$. This is because Δp may be fractional. In this case, we will use bilinear interpolation to solve this problem. This is shown in Eq. 2.4, where

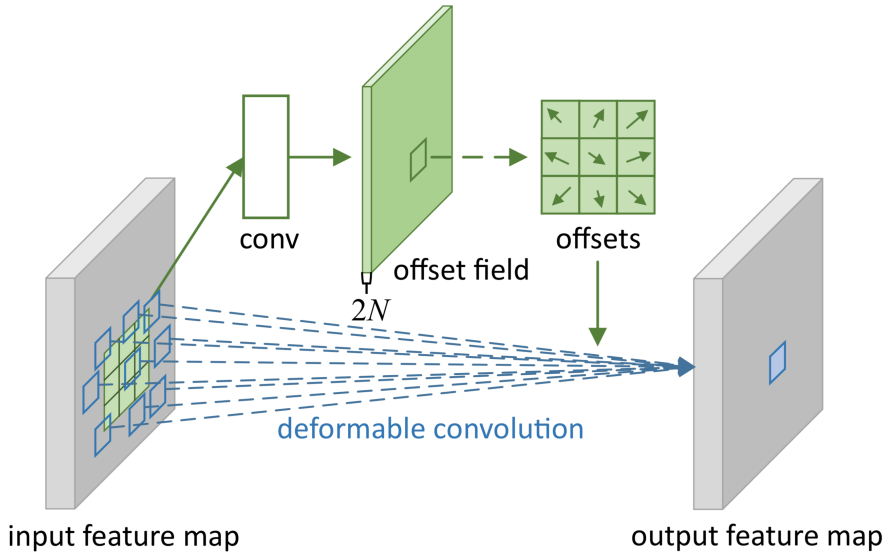

Figure 2.4: Illustration of 3×3 deformable convolution.

$p = p_0 + p_n + \Delta p_n$, q enumerates all integral spatial locations in the feature map x, and $g(a, b) = max(0, 1 - |a - b|)$.

$$x(p) = \sum_q g(q_x, p_x) \cdot g(q_y, p_y) \cdot x(q) \tag{2.4}$$

2.4.4 Coordinate Convolution Operation

The coordinate convolution operation is a simple extension of 2D convolution operation, which can be implemented by concatenating extra channels to the input representation. The extra channels are initialized and filled with constant and untrained coordinate information. The example of the coordinate convolution operation is shown in Figure 2.5, the input of the coordinate convolution layer adds the coordinate information by concatenating i and j coordinate channels channel-wise. Specifically, the i coordinate channel is a $h \times w$ matrix with the first row filled with 0's, the second rows filled with 1's, etc. The j coordinate channel is a $h \times w$ matrix with the first column filled with 0's, the second column filled with 1's. One example of the i and j coordinate channels concatenated to 5×5 input representation is shown in Figure 2.6, the green and pink images represent i coordinate channel and j coordinate channel, respectively. For i coordinate channel,

the integer value in each row is equal to the row number. For j coordinate channel, the integer value in each column is equal to the column number.

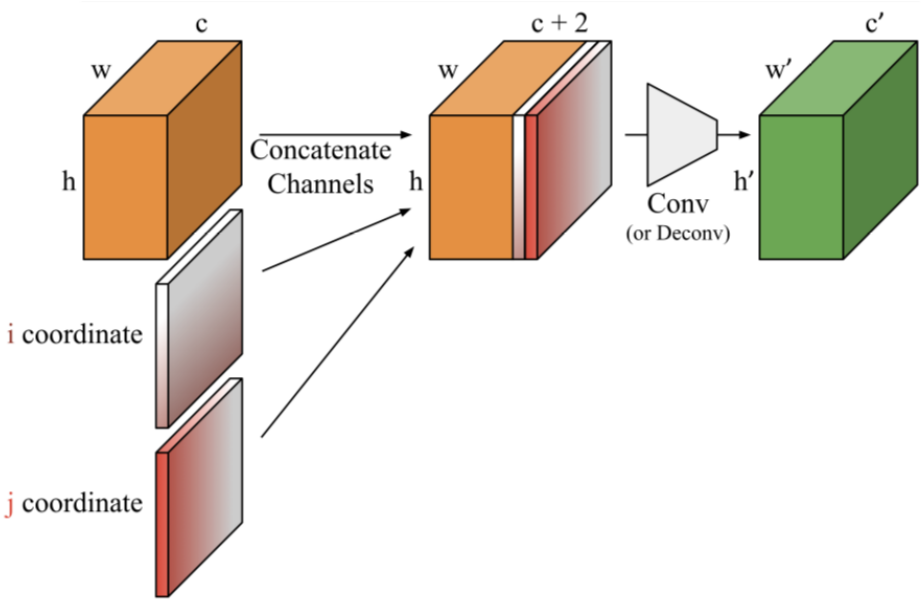

Figure 2.5: The input of the CoordConv layer is concatenating the input image with hard-coded coordinates, the most basic version of the hard-coded coordinates are i coordinate and j coordinate.

2.4.5 Depthwise Separable convolution

Depthwise separable convolution separates a standard convolution into a depthwise convolution and a 1×1 convolution called a pointwise convolution. As shown in Figure 2.7(a), a standard convolution is parameterized by convolution kernel K of size $D_k \times D_k \times M \times N$ where $D_k \times D_k$ is the feature map size, M is the number of input channels and N is the number of output channels. The depthwise separable convolution first applies a depthwise convolution(Fig. 2.7(b)) then applies a pointwise convolution(Fig. 2.7(c)). Depthwise convolution is parameterized by the convolution kernel K_1 of size $D_k \times D_k \times 1 \times M$. The kernel size of depthwise convolution is the same with standard convolution kernel size but a single kernel is applied to each input channel. The number of output channels is M since the number of input channel to depthwise convolution is M. Pointwise convolution is a 1×1 convolution. In order to make the number of depthwise separable convolution

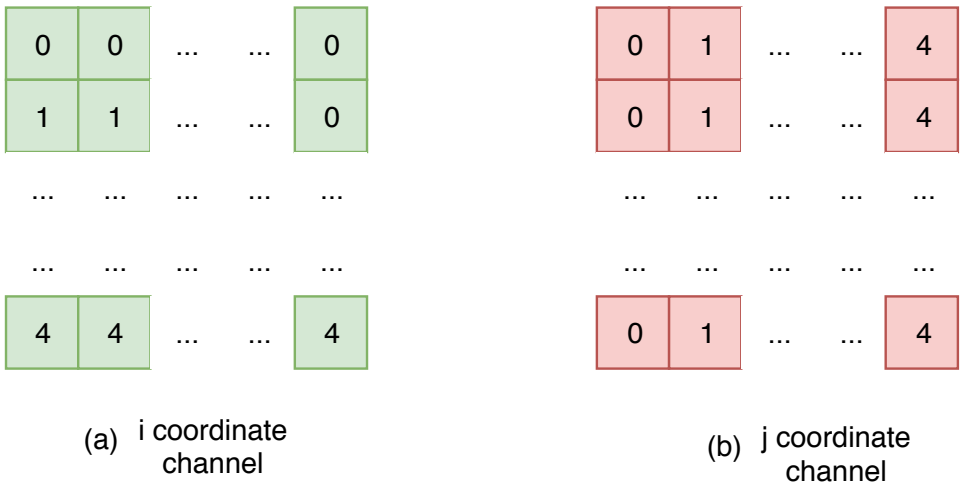

Figure 2.6: Illustration of i,j coordinate channels. (a) The channel for i coordinate, the integer value in each row is equal to the row number. (b) The channel for j channel, the integer value in each column is equal to the column number.

output channel equal to the number of standard convolution output channel, pointwise convolution is parameterized by convolution kernel K_2 of size $1 \times 1 \times M \times N$.

Depthwise Separable convolution works almost as well as standard convolution, but the computation cost is smaller. Assume the input feature map to standard convolution and depthwise separable convolution has a size of $D_F \times D_F \times M$:

- the cost of standard convolution: $D_k \times D_k \times M \times N \times D_F \times D_F$
- the cost of depthwise separable convolution: $D_k \times D_k \times M \times D_F \times D_F + M \times N \times D_F \times D_F$ where $D_k \times D_k \times M \times D_F \times D_F$ is the cost of depthwise convolution and $M \times N \times D_F \times D_F$ is the cost of 1×1 pointwise convolution.

By expressing convolution as a two-step process of filtering and combining, the depthwise separable convolution operation can get a reduction in the computation of: $\frac{D_k \times D_k \times M \times D_F \times D_F + M \times N \times D_F \times D_F}{D_k \times D_k \times M \times N \times D_F \times D_F} = \frac{1}{N} + \frac{1}{D_k^2}$.

2.5 Datasets

Due to the camera aperture, ground truth for optical flow datasets is difficult to obtain from the real-word. Large scale datasets of optical flow [59, 10] are often from synthetic

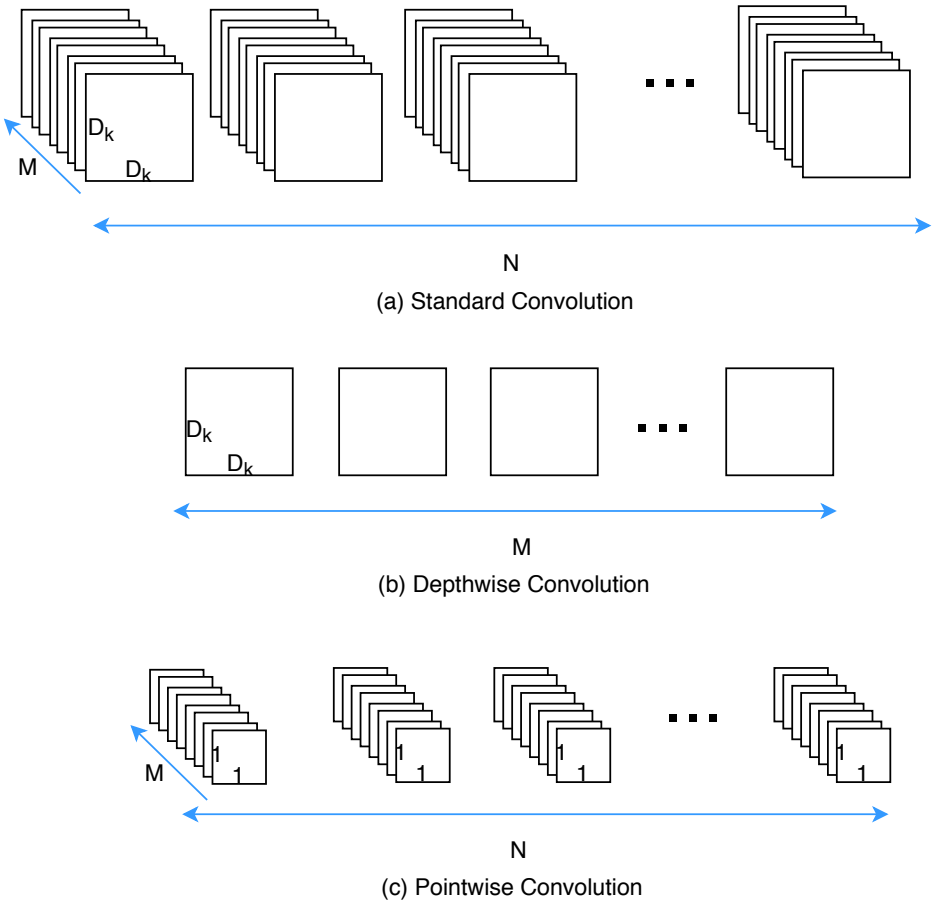

Figure 2.7: Depthwise separable convolution replace standard convolution(a) with depthwise convolution(b) and pointwise convolution(c).

3D virtual environments. The scale of datasets recorded from the real-world [22] is limited and may be too small to train a CNN from scratch in supervised learning. A diagnostic dataset like CLEVR [35], which is a diagnostic dataset for compositional language and elementary visual reasoning tasks, is missing in optical flow estimation.

In this section, we first introduce optical flow datasets and diagnostic datasets. Then, we introduce Omni-directional stereo (ODS) images, which are popular media formats for displaying content captured from visual sensors for virtual reality (VR) headsets [2, 58, 63].

2.5.1 Optical flow dataset

Synthetic, rendered data has been used for benchmarking purposes for a long time because obtaining ground truth optical flow in the real world is an extremely hard task. Heeger and David [62] used virtual scenes (including Lynn Quam's famous Yosemite 3D sequence) to evaluate optical flow estimation methods quantitatively. The major drawback of using synthetic datasets is that methods may still not perform well on real data. To overcome this problem, Baker et al. [4] proposed the Middlebury dataset to capture indoor sequences under both ambient and UV lights in a controlled lab environment and obtain optical flow for the video sequences. However, this approach does not work for the outdoor scenes as illumination can only be controlled in indoor scenes. To obtain optical flow in outdoor scenes, Liu et al. [48] use human annotations as ground truth in 10 indoor and outdoor real-world videos. However, this methodology is very time-consuming.

Currently, KITTI [23], MPI-Sintel [10] and Scene Flow datasets [59] are most widely used and challenging datasets for optical flow. KITTI is mainly used for autonomous driving applications. They use LIDAR [20] to capture semi-dense ground truth. The 2012 set [22] is the original optical flow evaluation benchmark which consists of 194 training and 195 test scenes of a static environment captured by a moving camera. The 2015 set [60] is the new optical flow evaluation dataset which consists of 200 training and 200 test scenes with moving cameras and moving objects. Because it is extended to dynamic scenes which include large motion, severe illumination changes, and occlusions, the 2015 set is more challenging. Also, commercial games have been used to acquire high-fidelity

data for training and benchmarking learning-based optical flow methods [69].

The MPI-Sintel benchmark is derived from the open source 3D animated short graphics movie "Sintel". This dataset contains a clean pass and final pass. The final pass has important features that are not present in the Middlebury flow evaluation, such as strong atmospheric effects, motion blur, camera noise, and atmospheric effects. Those features cause severe problems in existing methods. Also, because the 3D movie is open source, we can render scenes under conditions of different complexity to evaluate where the flow algorithm fails.

Scene Flow datasets are similar in spirit to the MPI-Sintel benchmark. In contrast to MPI-Sintel, Scene Flow datasets are large enough to facilitate training of convolutional networks and include ground truth for scene flow. Mayer et.al[59] created a collection of three such datasets: Flyingthings3D, Monkaa, and Driving. The central part of Flyingthings3D consists of everyday objects flying along randomized 3D trajectories. The motivation for creating this dataset is to facilitate training of large convolutional networks, which should benefit from the large variety. Monkaa is made from the open source Blender assets of the animated short film Monkaa10. The Driving scene is a mostly naturalistic, dynamic street scene from the viewpoint of a driving car, made to resemble the KITTI datasets. It uses car models from the same pool as the FlyingThings3D dataset and additionally employs highly detailed tree models from 3D Warehouse11 and simple street lights.

2.5.2 Diagnostic dataset

A diagnostic dataset is used to analyze the progress and discover short-comings of the modules in artificial intelligence systems. Johnson et al. [35] proposed a diagnostic dataset CLEVR for studying the ability of Visual Question Answering (VQA) systems to perform visual reasoning. Not-so-Clevr dataset [49] is a simple version of the Clevr dataset [35] to analysis the performance of the model in classification, regression & object rendering tasks.

The CLEVR dataset contains 100k rendered images and about one million automatically generated questions, of which 853k are unique. It has challenging images and

questions that test visual reasoning abilities such as counting, comparing, logical reasoning, and storing information in memory.

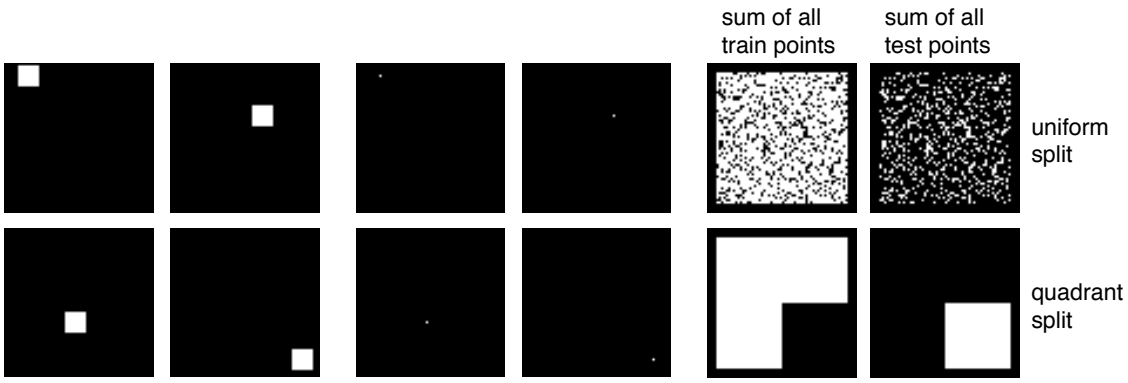

Figure 2.8: The visualization example Not-so-Clevr dataset. The white areas in the figures equal to 1 and the black areas are filled with 0. (a) The example images. (b) The example one-hot label. (c) The pixelwise sum of all one-hot label in the whole train/test set.

The Not-so-Clevr dataset contains a list of images randomly painted with a single greyscale 9×9 square. The size of the image is 64×64, and the squares are restricted to lie on the images entirely. Therefore, the square centers are limited to the smaller central area of 56×56 in the image. The number of examples in this dataset is 3136 which is the enumeration of all square centers. The square centers are also textured in the background images as label image. The examples of the images are shown in Figure 2.8a and corresponding label are shown in 2.8b. These 3136 examples are split to train/test set in two ways: *uniform*, which randomly split the whole dataset into train/test set by 80/20, and *quadrant*, which split three of four quadrants into a training set and the fourth quadrants into the test set. The split train/test dataset are visualized in Figure 2.8c by summing of one-hot label of train/test set.

2.5.3 Omni-directional stereo dataset

A large portion of prior works in ODS imaging cover different capture systems [2, 12, 33, 38] with recent works focusing on content formats and pipelines for 6 DoF virtual reality (VR) videos [80, 81, 50].

Early works utilized rotating cameras to generate ODS images [33, 65] which made recording dynamic scenes difficult. More recent works were developed with VR video capture in mind replacing the rotating camera with a ring of cameras all pointing outwards capturing video simultaneously [2, 74]. Both Anderson et al. [2] and Schroers et al. [74] use 16 overlapping camera streams and optical flow to stitch together an ODS image. Stitching is a computationally heavy task with Anderson et al. [2] reporting a single machine runtime of $\approx$ 24 minutes per frame and Schroers et al. [74] with $\approx$ 20 minutes per frame. As a result, camera ring capture rigs are unsuitable for regular consumers and live-streaming applications. However, there does not exist a wealth of real-world datasets for the general problem of panoramic depth map estimation from ODS images. Po et al. [41] generated training samples by leveraging existing textured 3D models of indoor environments. Specifically, they made use of the textured meshes from the Stanford 2D-3D-Semantics Dataset [3] and the panoramic ray-tracing renderer from Blender.

Chapter 3

Datasets

In Section 3.1, we introduce a small diagnostic dataset, FlowClevr, to effectively evaluate the performance of different neural network layers for the task of optical flow estimation. We then introduce a synthetic omni-directional flow dataset for training and testing a large deformed panoramic image in Section 3.3

3.1 FlowClevr Dataset

In this section, we propose a FlowClevr dataset, a tiny dataset for fast-testing different model performance on optical flow prediction. We introduce the necessary information of the FlowClevr dataset and all the spatial transformations in the first two sections. Then,

we describe how to calculate the 2D motion vector as the optical flow ground truth in the third section. The split of training set and test set is introduced in the last section.

3.2 Basic Information on the FlowClevr Dataset

The FlowClver Dataset is inspired by Not-so-Clevr dataset [49]. This dataset is composed of a list of two 64×64 images as original images and the corresponding flow images between the two images as labels. The original images contain a single object transformed from 9×9 square. Each i square contains four information:

- $C_i \epsilon R^2$, the center location in (x, y) Cartesian coordinates,
- $P_i \epsilon R^{64\times64}$, a one-hot representation of the center pixel,
- $I_i \epsilon R^{64\times64}$, the resulting 64×64 images of the square and
- T_i, the transformation of the square.

To ensure every pixel of the square lies within the images, the center of each 9×9 square needs to be restricted into the center 48×48 sub image. Enumerating all the possible central positions of the squares results in a 2304 example data for each transformation. Based on the transformation method, the dataset can be subdivided into five sub-datasets: original square, stretching square, rotating square, shearing square, and mixed transformation square. We split those datasets into training and test set with a ratio of 4 to 1 in two ways:

- Uniform split, randomly split the center points into train and test set.
- Quadrant split, take three of four quadrants as the training dataset into train set, the fourth quadrant as the test set.

Since optical flow is defined as the pattern of apparent motion of objects, only the objects with contrast or texture are calculated in flow estimation [83]. Therefore, we add

brick and grass textures to our foreground object and background image, respectively. Please note that the FlowClevr dataset is only a small diagnostic dataset, it can only be used for evaluating the modules. The weights trained on this dataset are not suitable for transfer learning to the other optical flow dataset. The optical flow for the FlowClevr dataset can be visualized with a color map as it is common in optical flow, e.g., when reporting results for the MPI Sintel dataset. The optical flow of the image pair is first normalized to $[1, 1]$ by the maximum optical flow value. The relationship between the normalized flow vector map and color map is shown in Figure 3.1. The normalization is only for visualization purposes, the motion itself is estimated over the complete image as a fraction of image size.

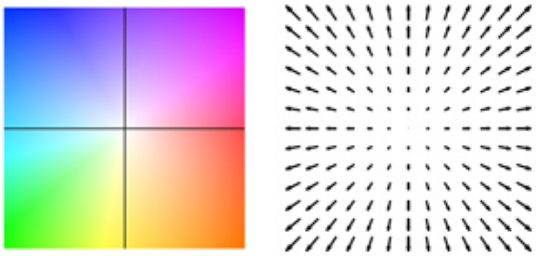

Figure 3.1: The corresponding relationship between the normalized flow vector map and color map.

3.2.1 Spatial Transformation of the Square

There are three transformation methods applied to the original square: *stretching*, *shearing* and *rotating*. All the three transformations keep the index of the one hot label (the 9×9 square center x_0) fixed. In the next subsections, we introduce the transformations for stretching, shearing, and rotation, respectively.

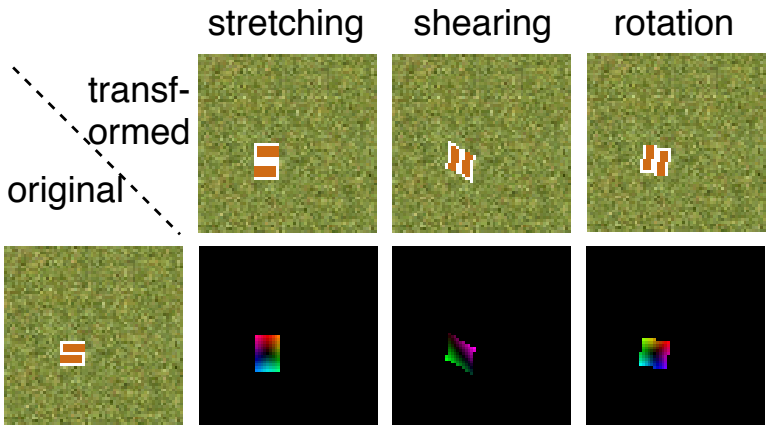

Figure 3.2: Example of transformed objects and their flow including the respective original images.

3.2.1.1 Stretching

The stretching transformation enlarges all the distances along the x-axis and the y-axis by constant factors k_x and k_y. The equation for stretching is:

$$\begin{cases} x_{stretching} = k_x(x - x_o) + x_o, \quad k_x \in [1/2, 2], and \\ y_{stretching} = k_y(y - y_o) + y_o, \quad k_y \in [1/2, 2]. \end{cases} \tag{3.1}$$

In Equation 3.1, the x and y are the pixel indices of the original image; x_o and y_o are the indices of the one hot label in the original image; The $x_{stretching}$ and $y_{stretching}$ are the indices of transformed images. The k_x and k_y are the constant stretching factors along the x-axis and y-axis. Based on the centers of the squares, the one-hot labels are limited to the smaller central area of 48×48 in the image, and the length of the transformed object has to be smaller than 17. In order for the objects to lie on the image completely, k_x and k_y need to be in the range of $[1/2, 2]$. The flow vector between the original object and the stretched object is shown in Figure 3.2.

3.2.1.2 Shearing

The shearing transformation displaces each point in a fixed direction by a constant factor k to its signed distance from a line that is parallel to that direction. A parallel shearing transformation to the x-axis has the y value unchanged and x value equalled to

$x+k_x(y-y_0)$. A parallel shearing transformation to the y-axis has the x value unchanged and y value equalled to $y + k_y(x - x_0)$. Shearing is defined as:

$$\begin{cases} x_{shearing} = x + k_x(y - y_o), & k_x \in [-1, 1], and \\ y_{shearing} = y + k_y(x - x_o), & k_y \in [-1, 1]. \end{cases} \tag{3.2}$$

In Equation 3.1, the x and y are the pixel indices of the original image; x_o and y_o are the indices of the one hot label in the original image; The $x_{shearing}$ and $y_{shearing}$ are the indices of transformed images. The k_x and k_y are the constant shearing factors along the x and y-axis. Based on the centers of the squares, the one-hot labels are limited to the smaller central area of 48×48 in the image, and the length of the transformed object has to be smaller than 17. In order for the objects to lie on the image completely, k_x and k_y need to be in the range of $[-1, 1]$. The flow vector between the original object and the sheared object is shown in Figure 3.2.

3.2.1.3 Rotation

The rotation transformation rotates each point around a fixed point by an angle θ. The direction of rotation can be clockwise or anticlockwise. The equation of rotation is:

$$\begin{cases} x_{rotation} = cos\theta(x - x_o) + sin\theta(y - y_o) + x_o, \\ \qquad\qquad\qquad\qquad \theta \in (-\pi/2, \pi/2), and \\ y_{rotation} = -sin\theta(y - y_o) + cos\theta(x - x_o) + y_o, \\ \qquad\qquad\qquad\qquad \theta \in (-\pi/2, \pi/2). \end{cases} \tag{3.3}$$

In Equation 3.3, the x and y are the pixel indices of the original image; x_o and y_o are the indices of the one hot label in the original image; The $x_{rotation}$ and $y_{rotation}$ are the indices of transformed images. Based on the centers of the squares, the one-hot labels are limited to the smaller central area of 48×48 in the image, and the length of the transformed object is smaller than 17. In order for the objects to lie on the image completely, θ value need to be in the range of $[-\pi/2, \pi/2]$. The flow vector between the original object and the rotated object is shown in Figure 3.2.

3.2.2 Motion Vector Calculation

The flow estimation in FlowCLEVR dataset is the 2D motion of the rendered object pair $(t, t+1)$. The t objects are a sequential enumeration of all the rendered objects in the dataset, while the $t+1$ objects are randomly selected from the nearby area of each t objects. The distance of the center point between t and $t+1$ object is in the range of $[-8, 8]$ for x and y-axis. Due to the enumeration of the rendered objects for time t, there also are 2304 object pairs in the dataset.

The final flow F of $(t, t+1)$ object is calculated in Equation 3.4, where I_t and I_{t+1} are the 2D motion between the original square and transformed object for time t and $t+1$; $I_{t \to t+1}$ is the 2D motion of the original squares for time t and $t+1$. Since the original squares for different times are completely the same, $I_{t \to t+1}$ could be simplified to the center distance of the rendered distance. The example of the flow estimation of rendered object pairs for different is shown in Figure 3.3.

$$F = -I_t + I_{t \to t+1} + I_{t+1} \tag{3.4}$$

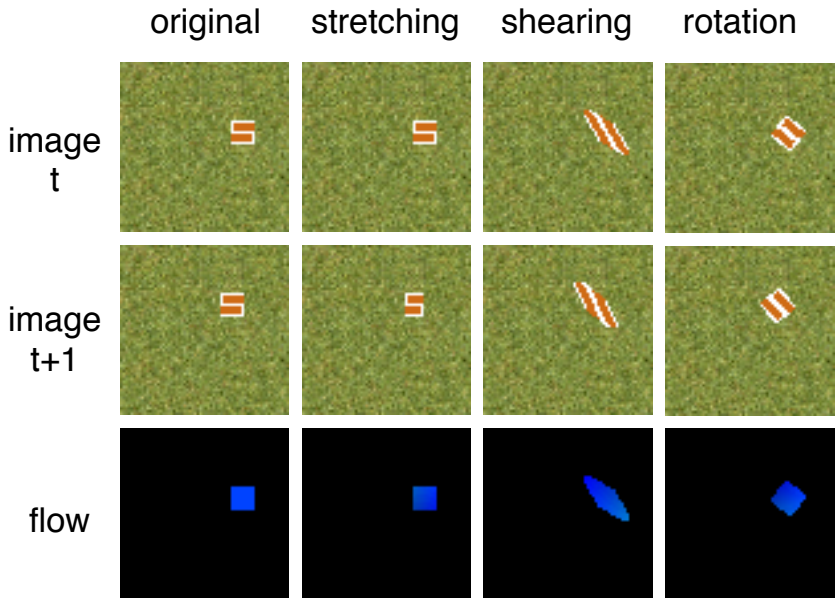

Figure 3.3: The visualization examples of the flow estimation of rendered object pairs.

The original image is a digital image of limited resolution. Applying a forward ge-

ometric transformation to each pixel will result in the location of pixels no longer as integer. In the FlowClevr dataset, we simply round the location of the pixel in the transformed image to an integer value while for the groundtruth the flow is kept as a float value. More accurate transformed images could be obtained by looping over the transformed image and looking up colour values in the source image possibly using bilinear or higher order interpolation.

3.2.3 Training and Testing Set

Similar with the Not-so-CLEVR dataset [49], we also split the 2304 object pairs in two ways: *uniform*, which randomly split the whole dataset 80/20 into training and test set, and *quadrant*, which takes three quadrants as training set and the fourth quadrant as test set. The uniform splitting method is adopted by learning based estimation tasks [37], while the quadrant is designed to test the generalization of the model when the location of trainset and testset is different. The splitted training and testing sets are visualized in Figure 3.4 by summing of center point of the rendered objects in the train/test set. Besides, we create the dataset for each transformation. Therefore, we have 8 datasets which are *uniform-original*, *uniform-stretching*, *uniform-shearing*, *uniform-rotation*, *quadrant-original*, *quadrant-stretching*, *quadrant-shearing*, and *quadrant-rotation*.

(a) (b)

Figure 3.4: The visualization of train/test set split. (a) Quadrant split. (b) Uniform split.

3.3 Omni-directional Flow Dataset

Since currently there does not exist a wealth of real-world datasets for the general problem of panoramic flow estimation, the synthetic omni-directional images and its relevant groundtruth is generated by the existing textured 3D models of indoor environments. Also, omni-directional images contain information of the full (or nearly full) optical flow field, estimating the optical flow from omni-directional images applies to navigation, localization, and VR headset. In this chapter, we introduce the basic information of the proposed Omni-directional Flow Dataset (ODF). Then, we describe how we get the pair of omni-directional images and the way we calculate the optical flow groundturh of the pair of omni-directional images. It should be noticed that Po provides great help in ODF dataset by providing the code for generating the panoramic images and calculating the optical flow groundtruth.

3.3.1 Basic Information on the Omni-directional Flow Dataset

Since the ground truth of optical flow is hard to obtain in the real-world image especially for omni-directional images, we propose a synthetic omni-directional stereo dataset with optical flow groundtruth. Specifically, we made use of the textured meshes from the Stanford 2D-3D-Semantics Dataset [3] and the panoramic ray-tracing renderer from Blender. The image sets are rendered by ray-tracing at random positions and orientations. In our experiments, we set the max movement of the virtual camera to be 200mm to get the visible movement between the omni-directional image pair. We generate 60000 image pairs from the area_01 of the Stanford 2D-3D-Semantics Dataset and sample 50000 image pairs as the training set and 10000 image pairs as the test set. For each sample pair, the dataset includes:

- Two RGB-D images which the distance of the camera center is 200mm,
- the camera poses of the rendered images,
- the image-based optical flow groundtruth of the image pair and

- the angle-based optical flow groundtruth of the image pair.

The original size RGB-D image pairs and its corresponding flow groundtruth is 1024×512. Our intentions are to provide a dataset of pre-rendered images with a higher resolution for future work. For our experiments, we downsample the initial dataset from 1024×512 to 256×128 to train and evaluate our model. The values in the image-based optical flow groundtruth are divided by 4 to get the correct image-based optical flow and the values angle-based optical flow keep the same. We visualize the optical flow for ODF dataset by color map and vector map. The value of optical flow for the color map is first normalized by its log_{10} value and its corresponding color of the visualized optical flow is also based on 3.1. The vector map is the steamline figure shows the sparse optical flow. The direction of the streamline is the direction of the optical flow, while the color of the steamline is based on the absolute value of the optical flow. When the absolute value of optical flow is high, the color of the optical flow is red; when the absolute value of optical flow is low, the color of the optical flow is blue.

3.3.2 Generating the Image Pair and its Corresponding Optical Flow Groundtruth in the Omni-directional Flow Dataset

All the images in Omnidirectional Flow Dataset are generated from the textured meshes from the Stanford 2D-3D-Semantics Dataset [3] and the panoramic ray-tracing renderer from Blender.

In order to ensure the positional component of the camera poses selected for rendering were contained within the mesh model, we first compute a voxelization of the mesh M where a voxel v is considered occupied if it intersects with a triangle of M and unoccupied otherwise. The centroids of each unoccupied voxel were output as a point cloud which was manually pruned to remove points which exist outside of the mesh boundaries. This manual cleaning step is required since the meshes were not water tight which makes it non-trivial to determine whether or not a voxel is in the interior or exterior of M. The rotational component of the camera poses was represented as Euler angles and their values randomly sampled from the range of $\pm 20°$. The minor rotations are helpful to

mimic the rotational data augmentation in the dataset.

After polishing, there are 64403 potential camera poses available in the area_01 of the Stanford 2D-3D-Semantics Dataset. The initial positions are random choose from the potential positions and the image pairs are recorded by two panoramic cameras nearby the initial position as t and t+1 image. The height axis of the cameras is the same and the distance between the two cameras is 200mm.

When the image pairs are generated, the image-based flow groundtruth and the angle-based flow groundtruth is calculated. For image-based flow groundtruth, we project the t panoramic image back to mesh and find the 3D point cloud on the mesh. Then the 3D point cloud is projected to the t+1 camera based on its camera pose. Therefore, the 2D motion vector of the corresponding points in the t and t+1 image could be calculated, which is the image-based flow groundtruth. When we project the motion vector to the unit sphere of t camera, the angle difference between the two points is the angle-based flow estimation. The examples of image pair and their corresponding image-based flow and angle-based flow are shown in Figure 3.5.

Figure 3.5: The visualization examples of the image pairs and its relevant flow samples in Omnidirectional Flow Dataset.

Chapter 4

Flow Estimation Networks

In this chapter, we first propose SimpleNet, a network that stacks several convolutional layers and makes it work on simple optical flow estimation task by testing on FlowClevr dataset. We introduce several modules to add to SimpleNet, namely, the correlation module, coordinate convolution module, deformable convolution module, and depthwise separable convolution module. Then, we propose the FlownetC architecture with those modules and make the networks work on a more complex optical flow task by testing on the MPI-Sintel dataset.

4.1 SimpleNet

SimpleNet and its variants are designed for our FlowClevr dataset. Given a dataset consisting of image pairs and ground truth flows, we train a network to predict the x–y flow fields directly from the images. A simple choice is to stack both input images together and feed them through a rather generic network, allowing the network to decide itself how to process the image pair to extract motion information. However, we can never be sure that a local gradient optimization like stochastic gradient descent can get the network to this point. Therefore, it could be beneficial to hand-design an architecture which is less generic but may perform better with the given data and optimization techniques.

A straightforward step is to create two separate, yet identical processing streams for the two images and to combine them at a later stage. SimpleNet is a simple architecture that designed based on the idea to evaluate the SimpleNet with different modules quickly. In the next subsections, we introduce the architecture of SimpleNet and how we modify SimpleNet with different modules.

4.1.1 Basic SimpleNet

Basic SimpleNet is designed by stacking five 2D convolutional layers. The input of the network is two 64×64 greyscale images, and the output of the network is the predicted optical flow between the two input images.

The architecture of SimpleNet is illustrated in Figure 4.1. There are three stages: in Stage 1, two convolutional layers are applied to the left and right images separately to extract features, the weights are shared between the convolutional layers applied to the left images and the right images. In Stage 2, the feature maps from left and right images are concatenated channel-wise. There are two more convolutional layers applied to the concatenated feature map to extract shared information. The flow is estimated in Stage 3. The feature map size and channels are shown in Figure 4.1, and the kernel size of all the convolutional layers is 3×3.

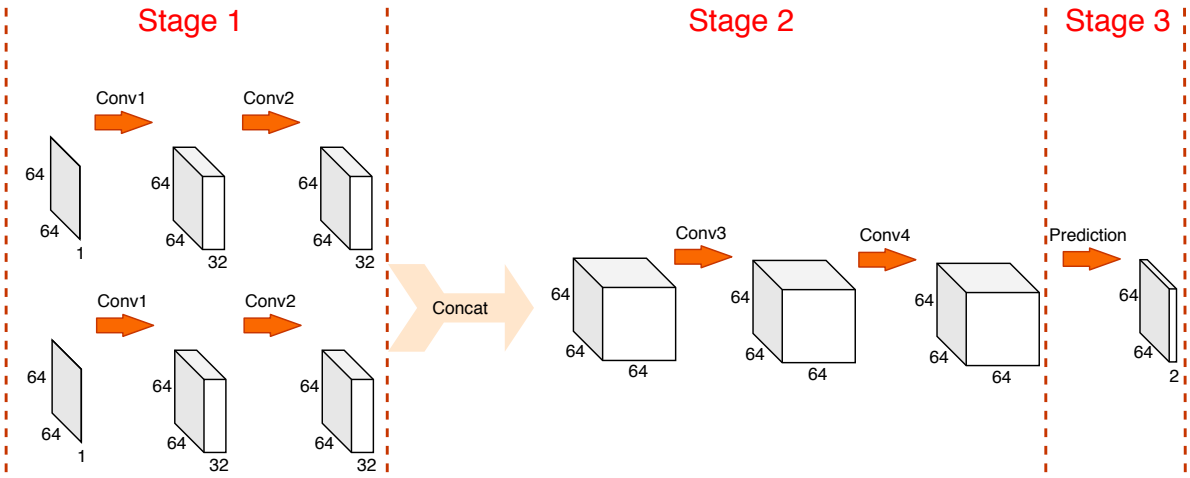

Figure 4.1: Illustration of SimpleNet architecture.

4.1.2 SimpleNet with different modules

To further test the performance of different modules on the FlowCLEVR dataset, we modify the SimpleNet architecture with those modules. In the following subsection, we discuss how to apply the modules to the SimpleNet architecture.

4.1.2.1 SimpleNet with correlation convolution

The correlation operation is used to find the actual correspondences based on the features. It has been found to be effective in disparity and flow estimation task [18, 79]. In the original SimpleNet, it first produces meaningful feature map representations of the two images separately in stage 1 and then concatenates them on a higher level at the beginning of stage 2. This resembles a common approach of first extracting features from patches of both images and then combining the feature maps channel-wise. With the correlation module, it can find the correspondence between given feature representations of two images. The correlation module that performs multiplicative patch comparisons between two feature maps is applied in Stage 2 as shown in Figure 4.2. The resulting CNNs are called SimpleNet-Corr.

As shown in Figure 4.2, in stage 2, we compute the correlation between the features maps from stage 1. The maximum displacement parameter is 10. For each pixel in the first map, we compute the correlation in the region of 21×21 in the second feature map,

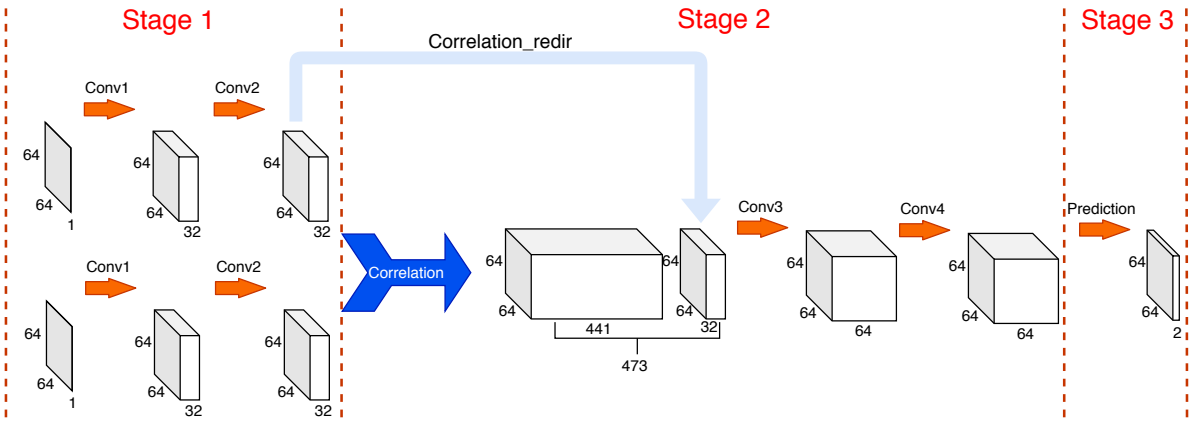

Figure 4.2: Illustration of SimpleCorrNet architecture.

so the number of output feature map channel is 21^2. Then, this $64 \times 64 \times 441$ feature map is concatenated with a $64 \times 64 \times 32$ feature map. That $64 \times 64 \times 32$ feature map is the first feature map after a convolutional layer which called convolution redirector in Figure 4.2.

4.1.2.2 SimpleNet with coordinate convolution

Coordinate convolution [49] operation helps the network learn the coordinate information by adding two i and j positions information to the input. It has been applied in object detection and generative modeling [49] but not to optical flow estimation. Because the flow estimation task is to predict the relative position of each pixel between two frames, adding the position information to the network may help to get more accurate flow prediction. With the coordinate convolution module, SimpleNet may perform better by adding the pixel coordinate information. The Coordinate convolution module is applied in stage 1. The resulting CNN is called SimpleNet-Coord.

As shown in Figure 4.3, in Stage 1, we extend the first standard convolution to Coordinate convolution by adding two extra channels to the input images. Concretely, the i coordinate channel is a 64×64 rank-1 matrix with its first row filled with 0's, its second row with 1's, its third with 2's, etc. The j coordinate channel is similar, but with columns filled in with constant values instead of rows. In all experiments, we apply a final linear scaling of both i and j coordinate values to make them fall in the range

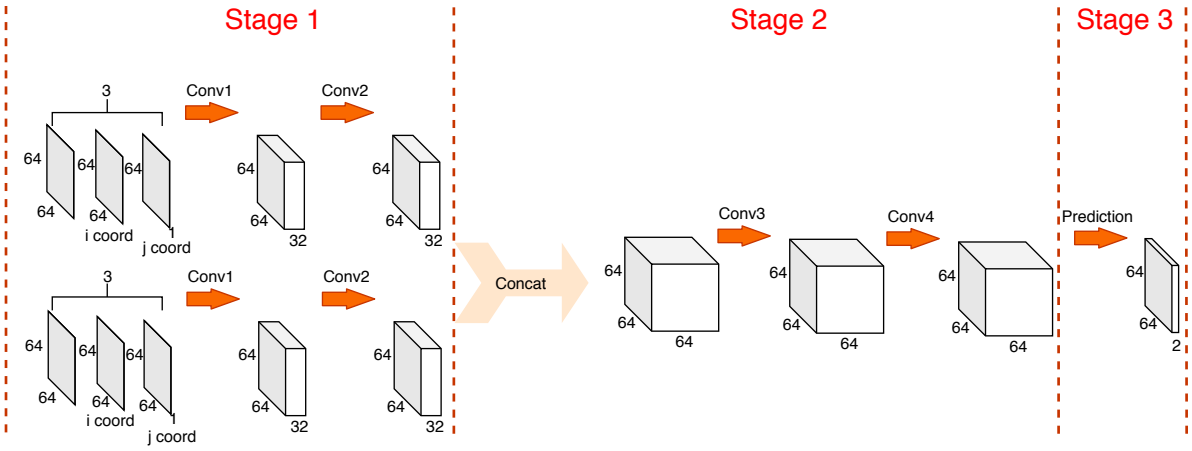

Figure 4.3: Illustration of SimpleCoordNet architecture.

$[-1, 1]$.

4.1.2.3 SimpleNet with deformable convolution

Deformable convolution [17] can enhance a CNNs' capability of modeling geometric transformations by adding 2D offsets to the regular grid sampling locations in the standard convolution. Deformable convolution has been shown to be useful for dense spatial transformation in CNNs for sophisticated vision tasks, such as object detection [17, 88] and semantic segmentation [17, 13] but not in optical flow estimation. The flow estimation task also involves geometric transformations between two frames. Hence, we introduce the deformable module to SimpleNet in stage 3. The resulting CNN is called SimpleNet-Deform.

Figure 4.4 illustrates the deformable convolution adding to the last convolutional layer. It consists of two stages: first, use a convolutional layer ($Conv\_offset$) to learn the offsets of the receptive field size. Because the kernel size is 3×3 and the offset is the displacement that applies to the receptive field, the number of the offset feature map channel is 18 ($3 \times 3 \times 2$). Then, apply another convolutional layer with the receptive field added as offsets.

Dai et al. also experiment with different numbers of deformable layers and found that applying two such layers to the last convolutional layers has a good trade-off for different tasks. Hence, we also modify the last two convolutional layers in stage 3 to deformable

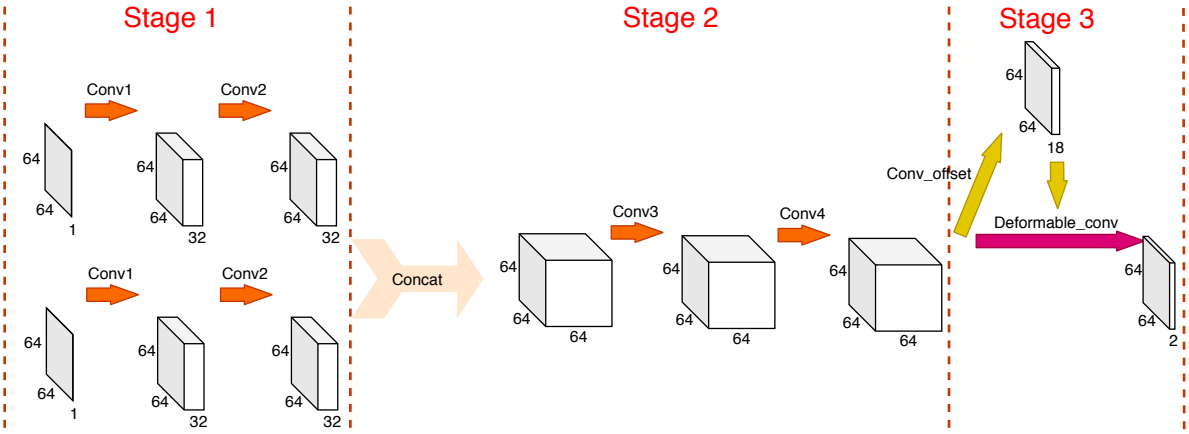

Figure 4.4: Illustration of SimpleDeformNet architecture.

convolutional layers to test the performance of SimpleNet with the different number of deformable convolution.

4.2 FlownetC

FlowNetC [18] learns to predict the optical flow field from a pair of images. It is a commonly used network for optical flow estimation. Based on the testing result that correlation and deformable convolution modules appear to boost the performance on the FlowCLEVR dataset, we further apply deformable convolution operation to FlowNetC. To further test if applying two such layers instead of one to the last convolution layers has a better trade-off, we also modify FlowNetC to FlowNetC-D & FlowNetC-DD by applying the deformable convolution modules to the flow estimation layers. The architectures for FlowNetC, FlowNetC-D, and FlowNetC-DD are introduced in the next subsections.

4.2.1 FlownetC

The FlowNetC architecture is UNet-based with correlation modules as shown in Figure 4.5. It is composed of two part: contractive and expansive parts. The contractive part is to produce meaningful representations of the two images separately and then combine them on a higher level. In this part, two convolutional layers are first applied to the two images separately to extract lower layer feature maps. Then, a correlation layer is

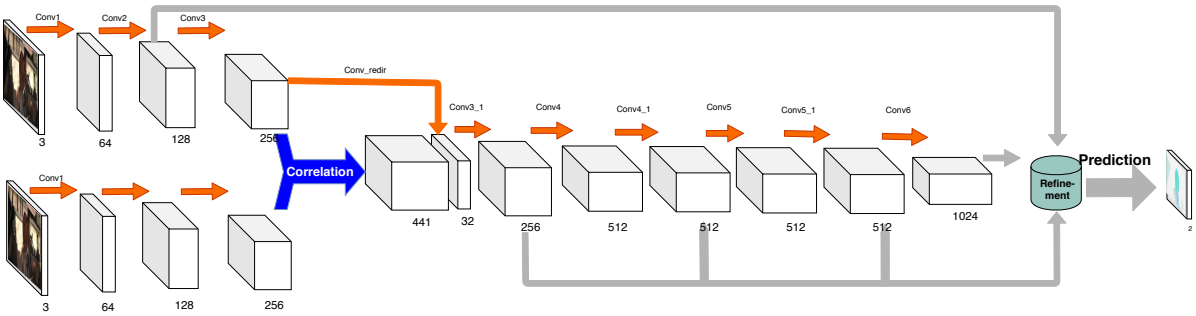

Figure 4.5: The FlowNetC architecture.

applied to perform multiplicative patch comparisons between two feature maps to find the correspondences between two images. Following that, several convolutional layers are further applied to extract deeper feature maps. The details of the 'contractive part' are shown in Figure 4.5. The channel of each feature map is shown in Figure 4.5. The kernel size for the first 3 convolutional layers are 7×7, 5×5 and 5×5 respectively, while the kernel size for the remaining convolutional layers is 3×3.

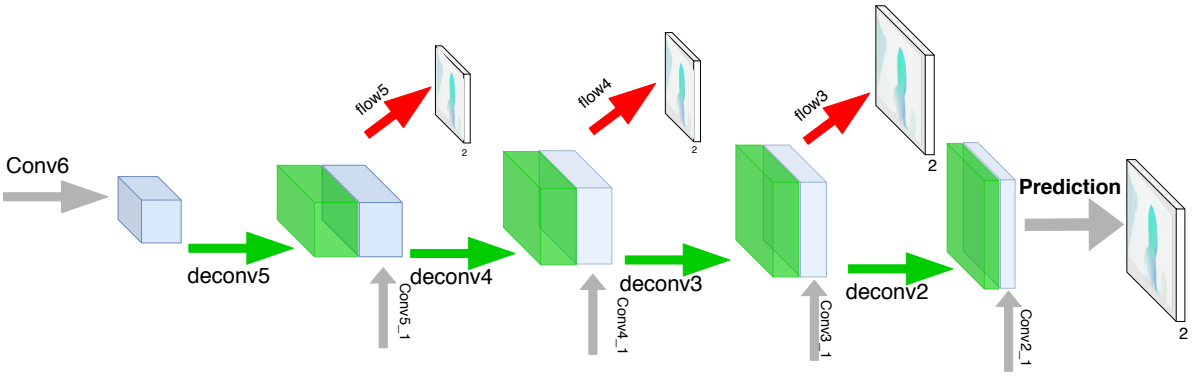

Figure 4.6: Refinement of the coarse feature maps to the high resolution prediction. [18]

The contractive part reduces the resolution, the feature map is then refined to higher resolution in the expansive part. The central part of the refinement approach is to use deconvolutional layers to extend the feature maps. To perform the refinement, the deconvolutional layers are applied to feature maps, which is shown in Figure 4.6. Then, concatenate the extended feature maps with corresponding feature maps from the 'contractive' part of the network and an upsampled flow prediction. This way both the high-level information passed from coarser feature maps and fine local information pro-

vided in lower layer feature maps are preserved. Each step increases the resolution twice. Also, the step is repeated four times.

4.2.2 FlownetC-D

Based on the evaluation result shown in Section 6.1.2, SimpleNet-Deform has better performance than SimpleNet on the flow estimation task in FlowCLEVR. We propose a modified FlowNetC architecture which uses a deformable convolution as the flow prediction layer and evaluates its performance with the complex optical flow estimation task in the Sintel datasets.

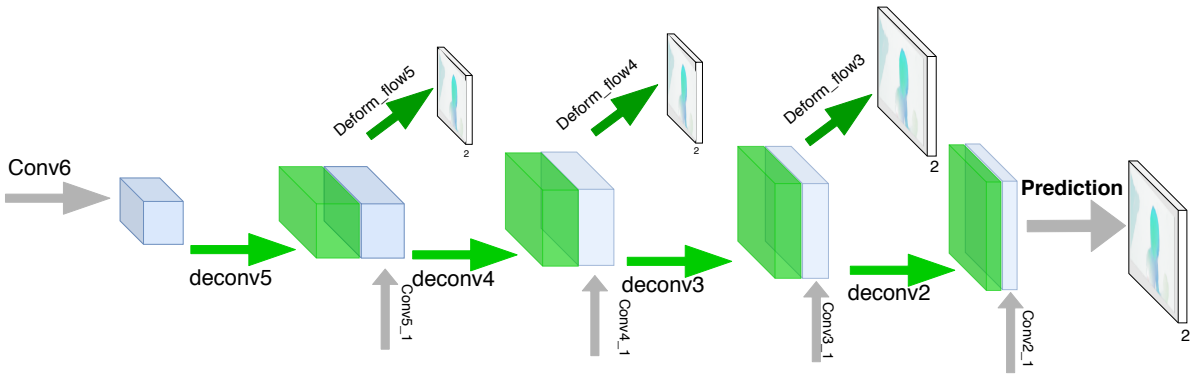

Figure 4.7: Illustration of FlownetC-D. The green arrow represents the deformable convolution module.

As FlowNetC predicts flow at multiple levels of scale, we apply the deformable convolution module to each convolutional layer that predicts the flow in the refinement part. To be more specific, we first use a convolutional layer to learn the offset field. Then we apply the learned offset field to the reception field in each prediction layer. As depicted in Figure 4.7, the pink arrow represents the deformable convolution module.

4.2.3 FlownetC-DD

The testing results of SimpleNet and its derivatives on FlowCLEVR dataset also show that SimpleNetCorr-DD performs better than SimpleNetCorr-D in most cases. Hence, we

also modify FlowNetC by replacing each prediction layer with stacking two deformable convolution modules.

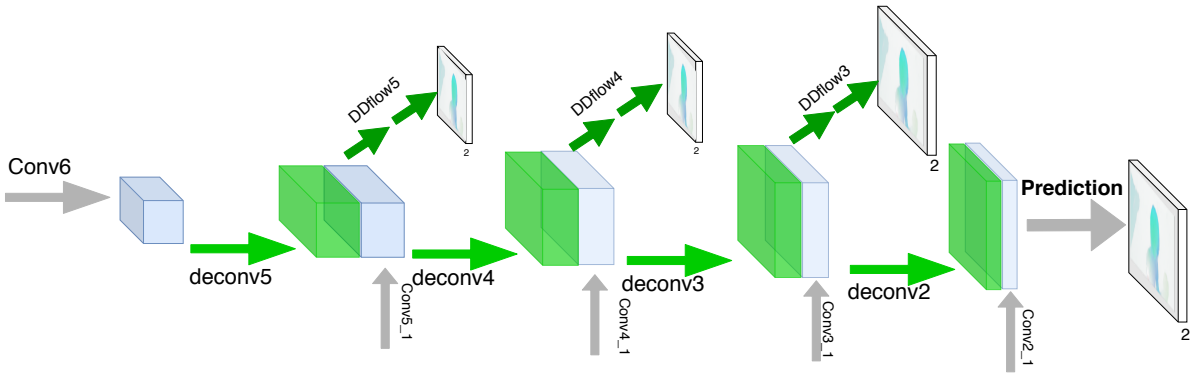

Figure 4.8: Illustration of FlownetC-DD. The pink arrow represents two stacked deformable convolution modules.

In the refinement part, before the upsampled coarser flow prediction, the feature maps after up convolution are concatenated with the corresponding feature maps from the 'contractive' part of the network. Because the prediction layer is after the concatenated layer, we add another deformable convolution operation before the deformable prediction layer in FlowNetC-D. The architecture of the FlowNetC-DD is illustrated in Figure 4.8, where the pink arrow represents two stacked deformable convolution modules.

4.3 Spatial Pyramid Network

Spatial Pyramid Network (SPyNet) [67] is another effective network proposed to solve the optical flow task by combining a traditional coarse-to-fine pyramid matching method with deep learning. It achieves comparable or lower error than FlowNet on standard benchmarks but is 96% smaller and faster than FlowNet. To test the performance of the deformable module applied to SPyNet, we modify SPyNet to SPyNet-D & SPyNet-DD by applying deformable convolution layers in the last convolution layers in networks. The architectures of SPyNet, SPyNet-D, and SPyNet-DD are introduced in the following subsections.

4.3.1 SPyNet

Unlike FlowNetC, SPyNet used the cascade structure to predict and optimize the optical flow from coarse-to-fine. The sub-network first predicts the optical flow from the down-sampled images. The predicted optical flow is then upsampling to the higher resolution and is concatenated with the input image with the same spatial dimension. This concatenated image is refined the optical flow in the current spatial dimension. An example of a 3-level pyramid SPyNet is shown in Figure 4.9. To be more specific, the network warps one image of a pair at each pyramid level by the current flow estimate and computes an update to the flow. SPyNet starts with downsampled images I_0^1 and I_0^2 and an initial flow estimate that is zero everywhere to compute the residual flow $v_0 = V_0$ at the top of the pyramid. The resulting flow, $u(V_0)$ is upsampled and passed to the network G_1 along with I_0^1 and $w(I_1^2, u(V_0))$ to compute the residual flow v_1. The flow V_k is similarly propagated to higher resolution layers of the pyramid until the flow V_k is at full resolution. The architecture for each G_k is shown in Figure 4.10 (a). In our experiments, we use a 6-level SPyNet. All the kernel sizes of the convolutional layers in the SPyNet and its variants are 7×7.

4.3.2 SPyNet-D

Similar with FlowNetC-D, SPyNet-D modifies the last convolution layer to deformable convolution layer in G_k at each pyramid level. The architecture of the SPyNet-D is illustrated in Figure 4.10 (b), each convolutional layer is followed by a Rectified Linear Unit (ReLU), except for the last one. We use a 7x7 convolutional kernel for each of the layers. The image I_k^1 and the warped image $w(I_k^2, u(V_{k-1}))$ have 3 channels each (RGB). The upsampled flow $u(V_{k-1})$ is 2 channel (horizontal and vertical). The image frames are stacked together with the up-sampled flow to form an 8 channel input to each G_k. The output is 2 channel flow corresponding to velocity in x and y directions.

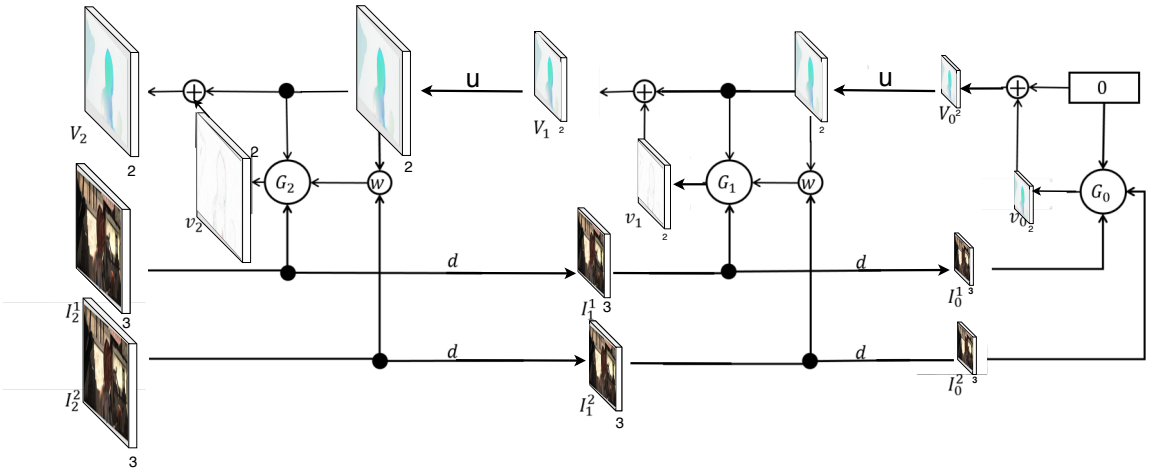

Figure 4.9: Inference in a 3-Level Pyramid Network: The network G_0 computes the residual flow v_0 at the highest level of the pyramid (smallest image) using the low resolution images I_0^1 and I_0^2. At each pyramid level, the network G_k computes a residual flow v_k which propagates to each of the next lower levels of the pyramid in turn, to finally obtain the flow V_2 at the highest resolution.

4.3.3 SPyNet-DD

Similar with FlowNetC-DD, SPyNet-DD modifies the last two convolution layers to deformable convolution layers in G_k at each pyramid level. The illustration of SPyNet-DD architecture is shown in Figure 4.10 (c), each convolutional layer is followed by a ReLU, except for the last two deformable convolution layers.

4.4 PWC-Net

PWC-Net [79] has been designed based on three straightforward principles: pyramidal processing, warping, and the use of a cost volume. It is the most state-of-the-art network that outperforms all published optical flow methods on the MPI Sintel final pass and KITTI 2015 benchmarks. To test the performance of the deformable module applied to PWC-Net, we modify PWC-Net to PWC-Net-D & PWC-Net-DD by applying deformable convolution layers in the last convolution layers. The architectures of PWC-Net, PWC-Net-D, and PWC-Net-DD are introduced in the following subsections.

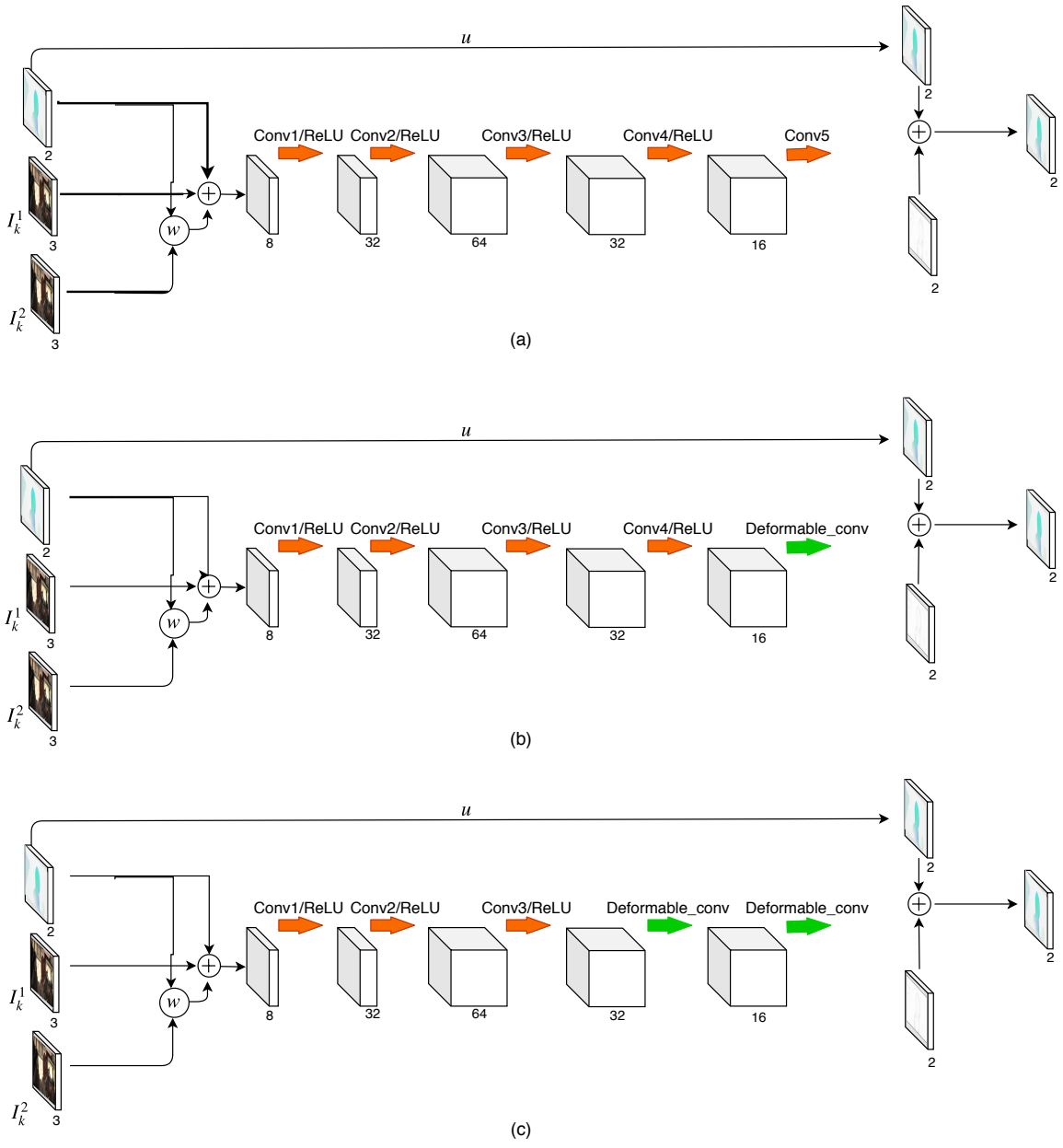

Figure 4.10: The architecture of G_k in SPyNet and its variants. (a) represents G_k architecture in SPyNet. (b) represents G_k architecture in SpyNet-D. (c)represents G_k architecture in SpyNet-DD.

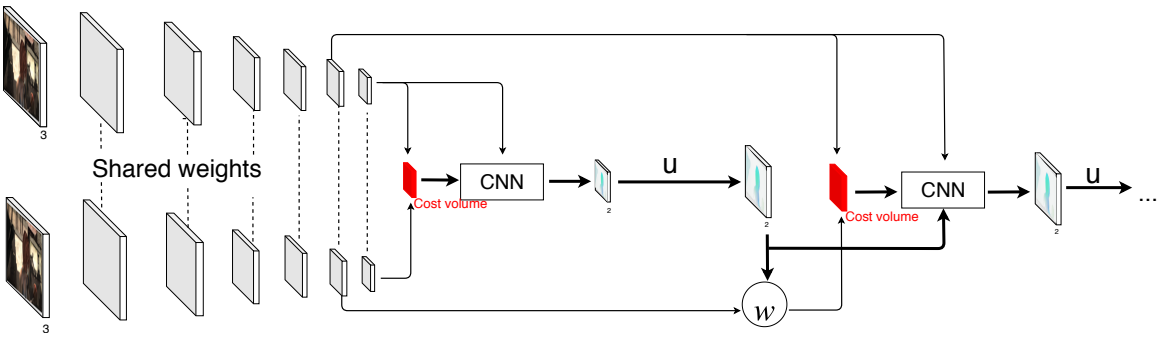

Figure 4.11: Network architecture of PWC-Net at the top two levels. For the rest of the pyramidal levels, the flow estimation modules have the same structure as the second to the top level. u represents upsampling operation and w represents warping operation.

4.4.1 PWC-Net

The network architecture of PWC-Net is shown in Figure 4.11. PWC-Net first builds a feature pyramid from the two input images. At the top level of the pyramid, PWC-Net constructs a cost volume by comparing the features of a pixel in the first image with corresponding features in the second image. As the top level is of small spatial resolution, the cost volume can be constructed by using a small search range. The cost volume and features of the first image are then fed to a CNN to estimate the flow at the top level. PWC-Net then upsamples and rescales the estimated flow to the next level. At the second level, PWC-Net warps features of the second image toward the first using the upsampled flow and then constructs a cost volume using features of the first image and the warped features. As warping compensates the large motion, the cost volume can still be constructed by using a small search range. The cost volume, features of the first image, and the upsampled flow are then fed to a CNN to estimate flow at the current level, which is then upsampled to the next (third) level. In our experiments, the process repeats until the sixth level.

The architecture of CNN in Figure 4.11 is shown in Figure 4.12, its inputs are the cost volume, features of the first image, and upsampled optical flow and its output is the flow w^l at the l-th level. The numbers of feature channels at each convolutional layers are respectively 128, 128, 96, 64, and 32, which are kept fixed at all pyramid

levels, while the kernel size of the convolutional layers in PWC-Net and its variance is 3×3. The estimators at different levels have their parameters instead of sharing the same parameters. This estimation process is repeated until the desired level, l0.

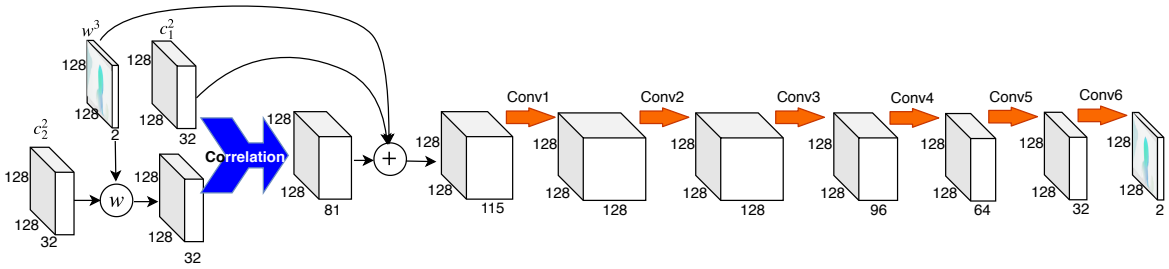

Figure 4.12: The optical flow estimator network at pyramid level 2. Each convolutional layer is followed by a leaky ReLU unit except the last (light green) one that outputs the optical flow. c_l^t denotes extracted features of image t at level l, w_l denotes the predicted flow at level l.

4.4.2 PWC-Net-D

Similar with FlowNetC-D and SPyNet-D, PWC-Net-D modifies the last convolutional layer to deformable convolutional layer in optical flow estimator network. The network architecture of PWC-Net-D is illustrated in Figure 4.13, each convolutional layer is followed by a leaky ReLU unit except the last deformable convolutional layer that outputs the optical flow.

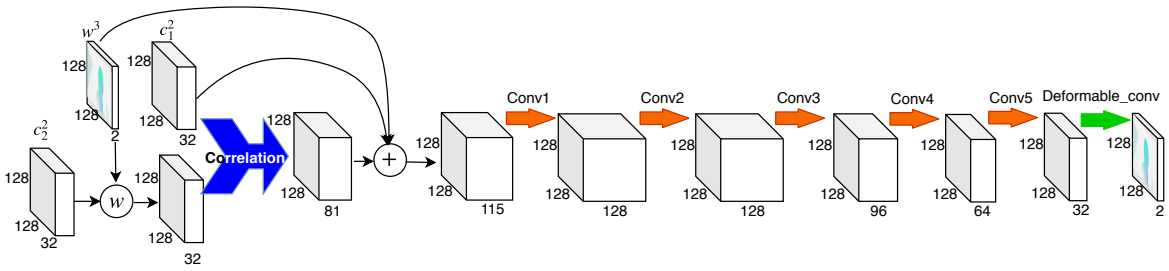

Figure 4.13: The optical flow estimator network in PWC-Net-D at pyramid level 2.

4.4.3 PWC-Net-DD

Similar with FlowNetC-DD and SPyNet-DD, PWC-Net-DD modifies the last two convolutional layer to architecture convolutional layers in optical flow estimator network. The network architecture of PWC-Net-DD is illustrated in Figure 4.14; each convolutional layer is followed by a leaky ReLU unit except the last two deformable convolutional layers.

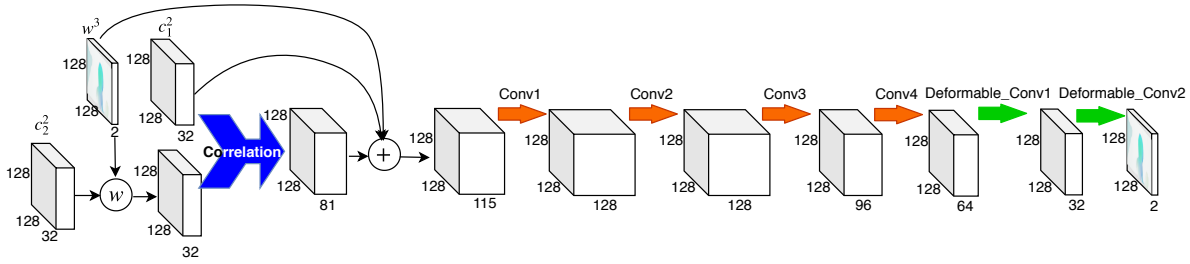

Figure 4.14: The optical flow estimator network in PWC-Net-DD at pyramid level 2.

Chapter 5

Panoramic Optical Flow and Depth

In this chapter, we first describe our baseline network architecture named SepUnet which is first proposed by Lai et al. [41]. Then, we apply the deformable modules to SepUNet and propose two modified networks to improve the model performance on the ODS and ODF datasets, namely SepUNet-D and SepUNet-DD.

5.1 SepUNet

The encoder-decoder architecture with skip connections has been successful in a variety of applications such as semantic segmentation [51, 70], object detection [6] and image restoration [57]. In general, skip connections allow for earlier learned features to have

greater influence in the final output. Intuitively, this is helpful in depth estimation since it allows a more global context to be considered in the output depth map.

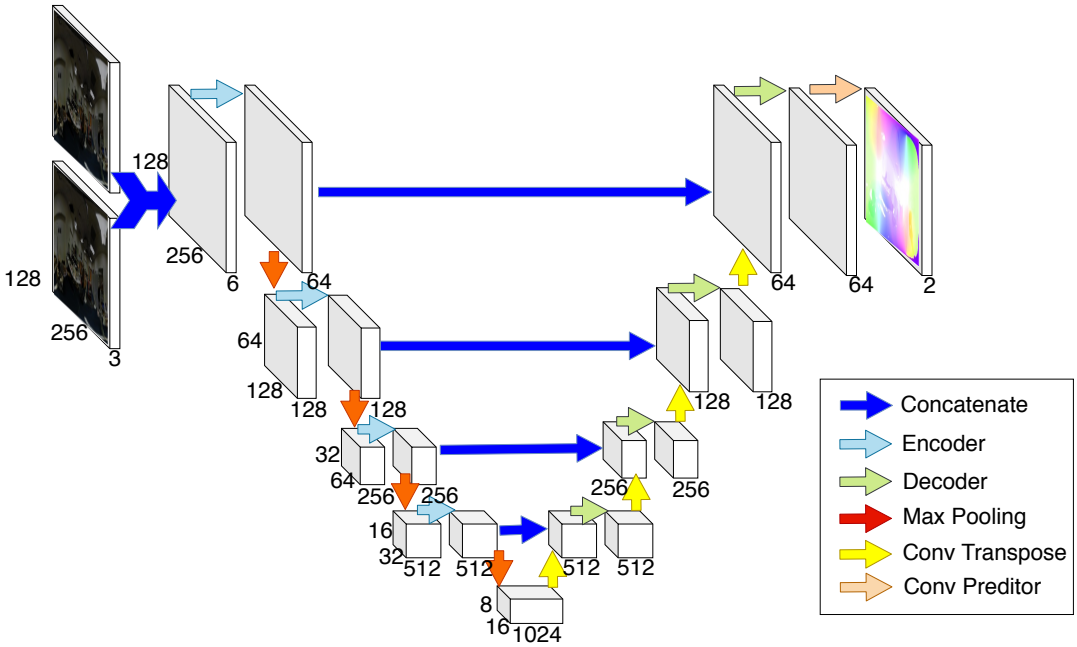

Figure 5.1: General structure of SepUNet. The coloured arrows represents the corresponding modules shown in the bottom sub figure.

SepUNet is adopted the encoder-decoder architecture proposed by U-Net [70]. The encoder and decoder portions use separable depth-wise convolutions [15] with a kernel size of (3×3) as the weight layers, ReLU [63] as the activation followed up with batch normalization [32]. It uses separable depth-wise convolutions [15] instead of the standard convolutions to reduce the parameter and improve the speed. Each subsequent block then doubles the number of feature maps while halving the image dimensions by max-pooling with a kernel size of (2×2). The decoder blocks are constructed similarly but increase the image dimensions while halving the number of feature maps by transposed convolutions with a kernel size of (4×4). The final portion of the network is a series of fully connected layers with dropout [76] before splitting into the desired outputs. Figure 5.1 provides an illustration of SepUNet and a detailed view of the encoder and decoder blocks.

5.2 SepUNet with deformable modules

Based on the experiment results result shown in Section 6.1 and 6.2, to further improve SepUNet performance on ODS and ODF datasets by refining the details of predicted disparity and optical flow boundaries, we apply the deformable modules to SepUNet and propose two modified networks: SepUNet-D and SepUNet-DD. The architecture of the proposed networks are shown in Figure 5.2 and 5.3. The deformable convolutional predictor replaces original convolutional predictor in SepUNet-D to predict the flow/disparity map. The SepUNet-DD architecture further replaces convolutional layer in the last decoder with deformable layers to achieve better performance.

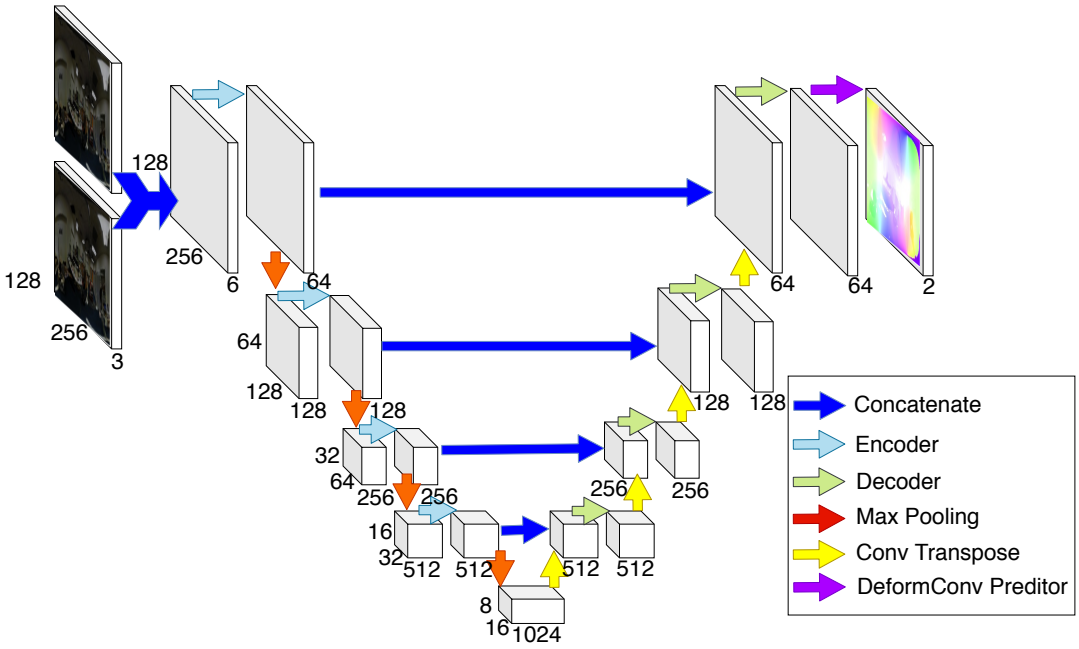

Figure 5.2: General structure of SepUNet-D. The coloured arrows represents the corresponding modules shown in the bottom sub figure.

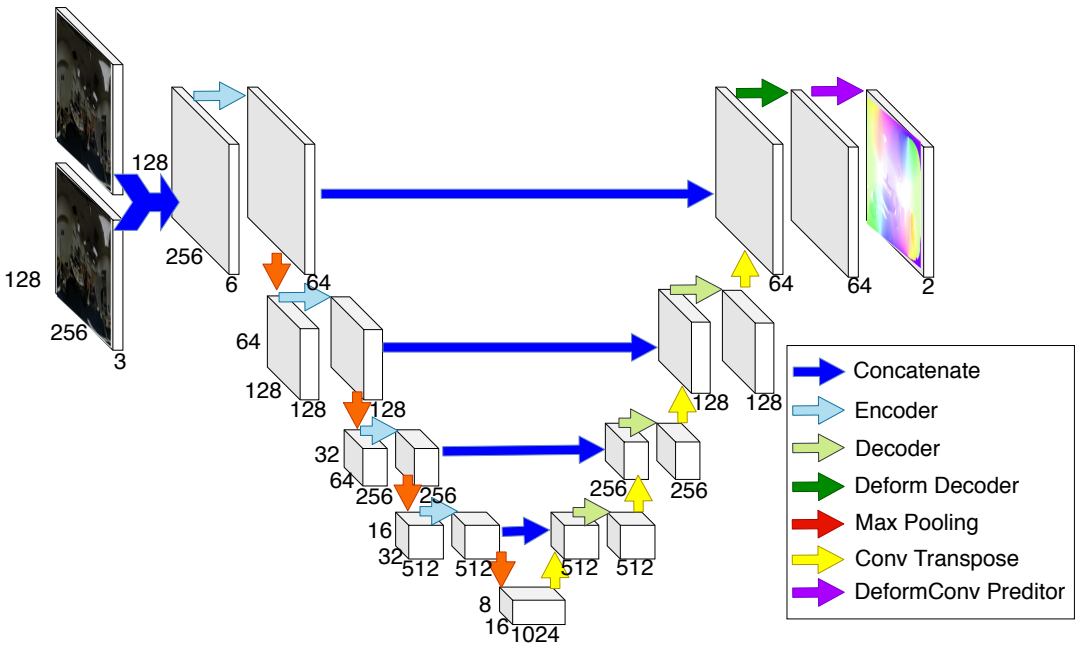

Figure 5.3: General structure of SepUNet-DD. The coloured arrows represents the corresponding modules shown in the bottom sub figure.

Chapter 6

Experiments

In this chapter, we present the results using the proposed methods on the FlowCLEVR dataset, Sintel dataset and ODS and ODF datasets. In Section 6.1, we report the experimental setup and the results for the different modules with SimpleNet on the FlowCLEVR dataset. In Section 6.2, we consider the verification experiments for the efficient modules with FlowNet, SPyNet, and PWC-Net on the Sintel dataset and report corresponding results. Finally, to further verify the efficient modules on panoramic images, the experimental setup and results for SepUNet and its variants on the ODS and ODF datasets are shown in Section 6.3. All computations are done on an Intel i7-4790K 4.0GHZ PC with 16 GB RAM and an NVIDIA 1080Ti graphic card with 11 GB memory. The program is implemented in Python with Pytorch.

6.1 SimpleNet for FlowClevr dataset

In this section, we report the results of SimpleNet with different modules on the FlowClevr dataset. We experiment with two different splits into training and test sets: *uniform* and *quadrant*. The experiment setup, result and the analysis based on the results are described in the following subsections.

6.1.1 Experiment setup

For the FlowCLEVR dataset, we use 8 models to analyse the performance of the correlation operation, deformable convolution operation, and the coordinate convolution operation, which are SimpleNet, SimpleNet-Corr, SimpleNet-Coord, SimpleNet-Deform, SimpleNet-CC, SimpleNet-DD, SimpleNet-CoorD, SimpleNet-DD. Among them,
SimpleNet-Corr, SimpleNet-Coord, SimpleNet-Deform are the derivatives of SimpleNet with correlation module, coordinate module, and deformable module respectively; SimpleN
et-CC, SimpleNet-CoorD, SimpleNet-DD are the derivatives of SimpleNet-Corr with the coordinate module and deformable module respectively; SimpleNet-DD is the derivative of SimpleNet-Deform with another deformable module.

The L1 loss is adopted as the training loss, while the end-point-error (EPE) [4] is chosen as the evaluation metric. L1 loss is used to minimize the error which is the sum of absolute differences between the true values and the predicted values. EPE is a standard evaluation metric for optical flow estimation, which is the Euclidean distance between the predicted flow and the ground truth 2D vector, averaged over all pixels. The equation of the L1 loss and EPE is expressed in Equation 6.1 and 6.2, respectively. In Equation 6.1, y_i and $h(x_i)$ denote each target value and the corresponding estimated value, respectively. In Equation 6.2, I' and I^{GT} denote the flow field predicted by the network and corresponding supervised ground truth, respectively.

$$L_1 = \sum_{i=0}^{n} |y_i - h(x_i)| \tag{6.1}$$

$$EPE = \left| I' - I^{GT} \right|_2 \tag{6.2}$$

We use the original 64×64 image pair to train the models from scratch. Adam is used to optimize the networks for fast convergence. We fix the parameters of Adam as recommended in [36]: $\beta_1 = 0.9$ and $\beta_2 = 0.999$. The learning rate for the optimizer is started with $\lambda = 4e - 3$ and Stochastic Gradient Descent with Warm Restarts (SGDR) [52] is used to cyclically descend the learning rate with $1e - 4$ weight decay rate. The learning rate of the optimizer in the whole training process is shown in Figure 6.1. The models are trained for 100 epochs with a batch size of 64 during training.

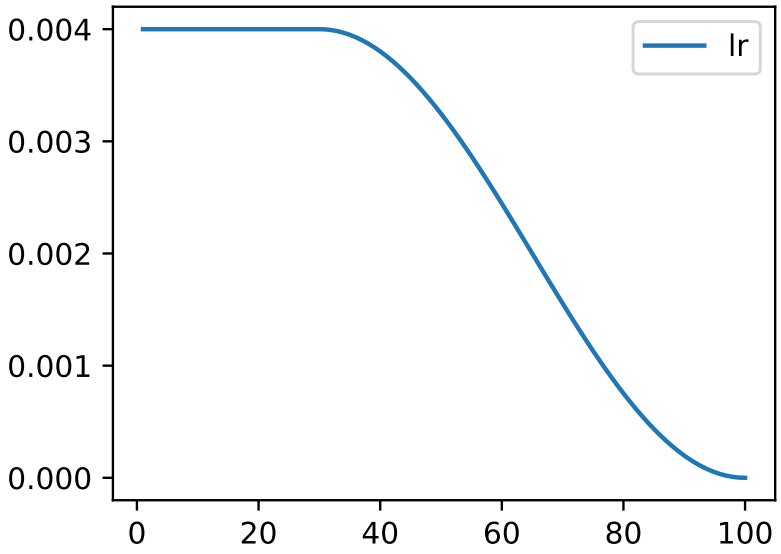

Figure 6.1: The learning rate of optimizer in the whole training process.

6.1.2 Experimental result

In Figure 6.2, the deformable operation reduces the EPE for all transform types. When compared to SimpleNet, the EPE of SimpleNetC is approx. 5 times lower when the images have just translation and approx. 2 times lower when linear transformations are applied. For all networks with deformable layers (ie: "-D" and "-DD" variants), we observed a large reduction in EPE when compared to SimpleNet (20 times) and SimpleNet-C (3 times) for images which have just translation. For images which have linear transformation,

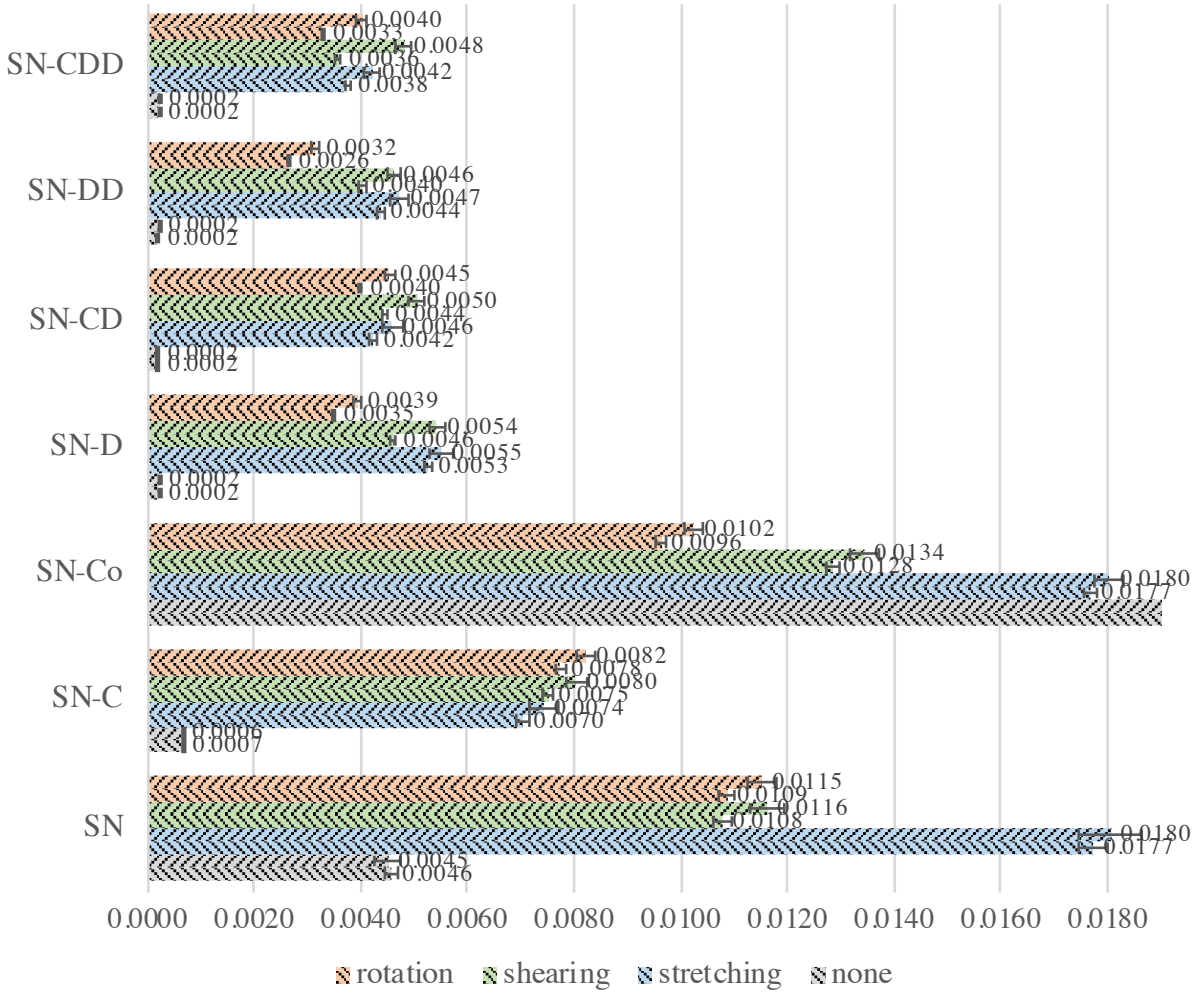

Figure 6.2: The average EPE in FlowCLEVR for 7 networks ("SN" denotes "SimpleNet"), where the x-axis is the EPE value. The red, green, blue and gray bar represent the dataset with rotation, shearing, non-uniform scaling (stretching) and no transformation, respectively, while the forward and backward slash present the average EPE on training and testing set, respectively. The standard error and the value of average EPE are noted on the top of the bars. Results shown are for the uniform dataset, the results for the quadrant set are very similar (see the supplemental material).

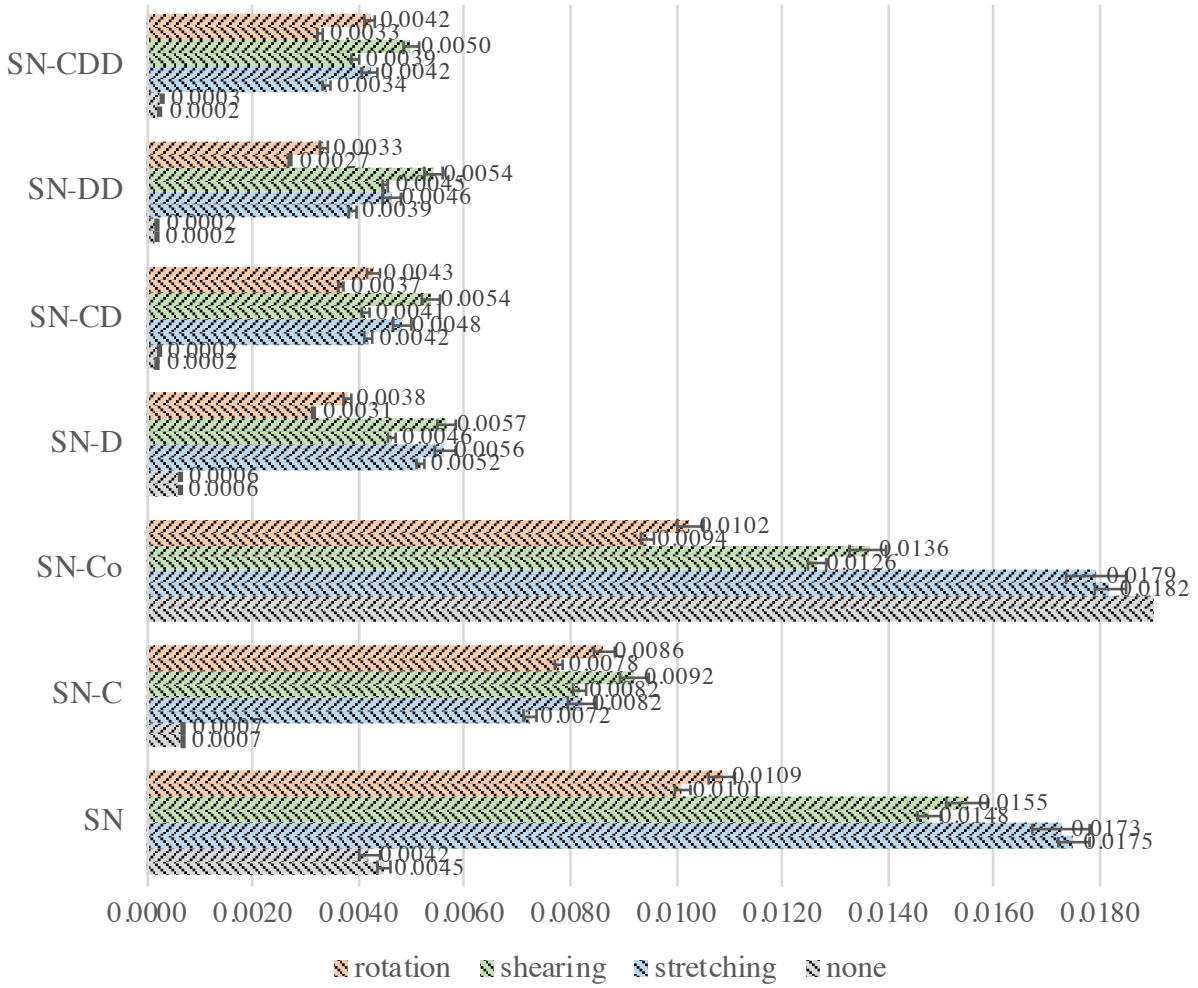

Figure 6.3: The average EPE in FlowCLEVR quadrant set for 7 networks ("SN" denotes "SimpleNet"), where the x-axis is the EPE value. The red, green, blue and gray bar represent the dataset with rotation, shearing, non-uniform scaling (stretching) and no transformation, respectively, while the left and right slash present the average EPE on training and test set, respectively. The standard error and the value of average EPE are noted on the top of the bars.

we observed a smaller reduction in EPE when compared to SimpleNet (4 times) and SimpleNetC (2.5 times). When we compared the two deformable variants (ie: "-D" vs "-DD"), we found that they have both similar performance. We have not found improved performance on flow estimation tasks with coordinate convolution with SimpleNet-Co. The result is similar split by quadrant, which is shown in Figure 6.3.

All the training losses of the eight networks are shown in Figures 6.4 and 6.5. In Figure 6.4, the SimpleNet-Deform converges much faster than SimpleNet, SimpleNet-Coord and SimpleNet-Corr. From Figure 6.5, the SimpleNet-CC converges slower than the other networks on all the datasets.

6.1.3 Analysis

In Figures 6.4 and 6.5. In Figure 6.4, the deformable convolution operation reduces the EPE 7.2 to 20.8 times in the all original datasets for SimpleNet. The correlation operation reduces the EPE by 3.9-5.0 times in the original datasets for SimpleNet. The EPEs value with SimpleNet-Coord is much worse than SimpleNet on the original dataset. In all shearing and stretching datasets, SimpleNet-Deform reduces the EPE by 2.3-3.3 times and SimpleNet-Corr reduces the EPE by 1.38-2.34 times. SimpleNet-Coord performs slightly worse than SimpleNet on all stretching and uniform shearing datasets but performs slightly better than SimpleNet on all the rotation and quadrant shearing datasets. What's more, in Figure 6.4 SimpleNet-Deform converges much faster than SimpleNet, SimpleNet-Coord, SimpleNet-Deform and SimpleNet-Corr.

In Figures 6.4 and 6.5. In Figure 6.4,, we can see the performance of SimpleNet with mixed modules. SimpleNet-CC performs the worst compared the other three networks. Similarly, in Figure 6.5, SimpleNet-CC converges slower than the other networks on all the datasets. SimpleNet-D, SimpleNet-DD and SimpleNet-DD have similar performance on all datasets except the original datasets. In Figure 6.5, the inflection points of SimpleNet-D, SimpleNet-DD and SimpleNetDD are close for all datasets.

Therefore, based on the performance shown in the diagnostic FlowCLEVR dataset, deformable convolution and correlation can improve the flow estimation a lot in all object transformation. However, the coordinate convolution operation does not have significant

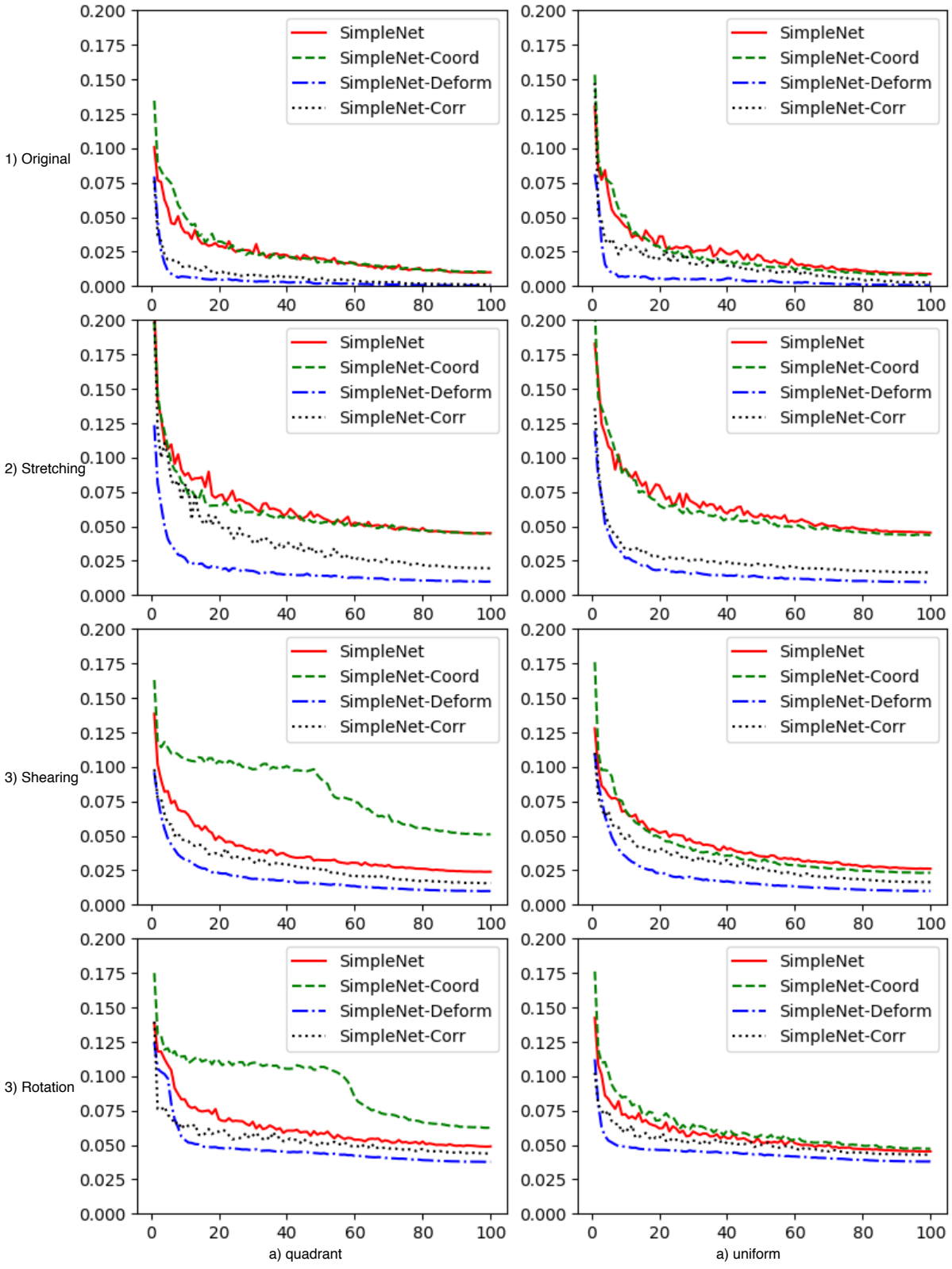

Figure 6.4: The training loss trend of SimpleNet, SimpleNet-Coord, SimpleNet-Deform, SimpleNet-Corr. In each subfigure, the x axis is the training epoch, while the y axis is the L1 loss.

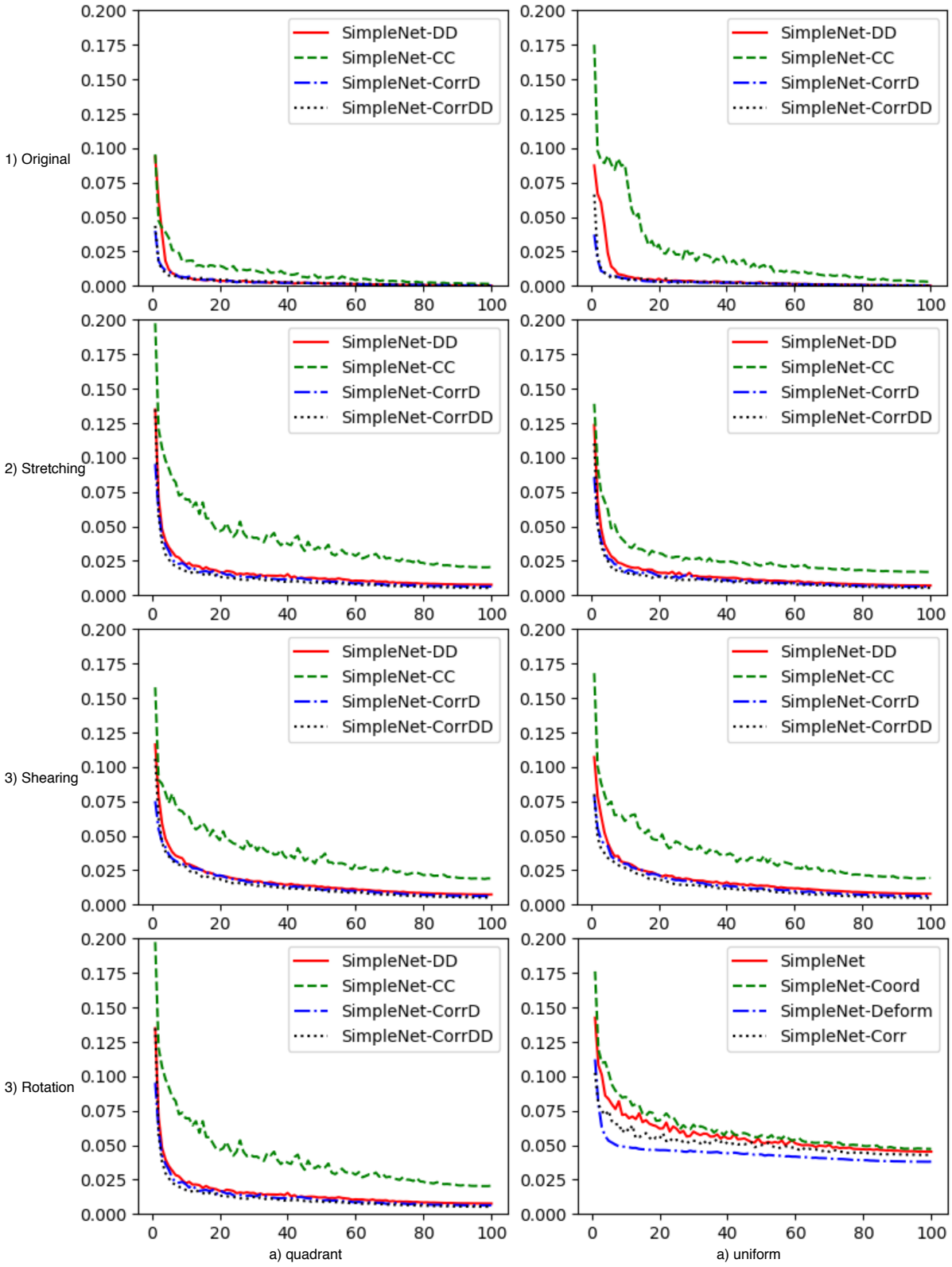

Figure 6.5: The training loss trend of SimpleNet-DD, SimpleNet-CC, SimpleNet-D, SimpleNet-DD. In each subfigure, the x axis is the training epoch, while the y axis is the L1 loss.

help to predict the flow.

6.2 Experiments for MPI Sintel dataset

Based on the performance of different modules on our FlowClevr dataset, we further test the deformable convolution operation operation performance using public datasets. We train and evaluate FlowNetC, SPyNet, and PWC-Net with/without these modules using the MPI Sintel datasets. The experiment setup, results and analysis are introduced in the following subsections.

6.2.1 Experiment setup

In the experiment, we use the variants of FlowNetC, SPyNet and PWC-Net to train and test flow estimation in the Sintel datasets. We use random cropped 768×384 image batches to train the model starting with the pre-trained weights provided by the authors [18, 67, 79]. The flow prediction layers in the models are first trained for 30 epochs with 4 batches and the initial learning rate is 0.004. And then the whole model is trained for 70 epochs with 4 batches and the initial learning rate is 0.001. For all training we adopt Adam [36] as optimizer and SGDR [52] as learning weight decay method.

Multi-scale loss [18] is adopted as training loss, while the EPE [4] is chosen as the evaluation metric. The equation of the multi-scale loss L is

$$L = \sum_{l=l_0}^{L} \alpha_l \left| I'_l - I_l^{GT} \right|_2 + \gamma \left| \Theta \right|_2 . \tag{6.3}$$

In Equation 6.3, I' and I^{GT} denote the flow field predicted by the network and corresponding supervised ground truth, respectively; l represent the lth pyramid level of the flow field; $|\cdot|_2$ computes the L2 norm of a vector; $|\Theta|_2$ is the regularization loss for all learnable parameters Θ in the models; α and γ are scalars and determine the relative weight of the multi-scale loss and regularization loss. The lower the value of multi-scale loss L and EPE, the better the model performs. The value of α and γ are the same as in the original paper [30, 79].

6.2.2 Experimental result

Table 6.1 shows the EPEs value of FlowNetC, SPyNet, PWC-Net and their variants with deformable convolution operations on the MPI Sintel datasets. The smallest EPEs value (the smaller the better) for each network group are marked in bold. In Table 6.1, FlowNetC-D and FlowNetC-DD perform better than FlowNetC on all Sintel datasets. The best performance is achieved by FlowNetC-DD, which has 6.095 EPE on the Sintel clean testing dataset and 7.306 on the Sintel final testing dataset. For SPyNet and its variants, SPyNet-DD performs best compared with other models. Its EPE values on the Sintel clean testing dataset and Sintel final test dataset are 6.03 and 7.43. Compared with the original SPyNet, it has 9.2% improvement on the Sintel clean test dataset and 11.1% in Sintel final test dataset. For PWC-Net and its derivatives, the PWC-Net mentioned in the paper [79] has achieved the best performance. However, the PWC-Net trained by ourselves performs slightly worse than the PWC-Net-D. Besides, the sample flows predicted by different networks are visualized in Figure 6.6, 6.7, 6.8 and 6.9.

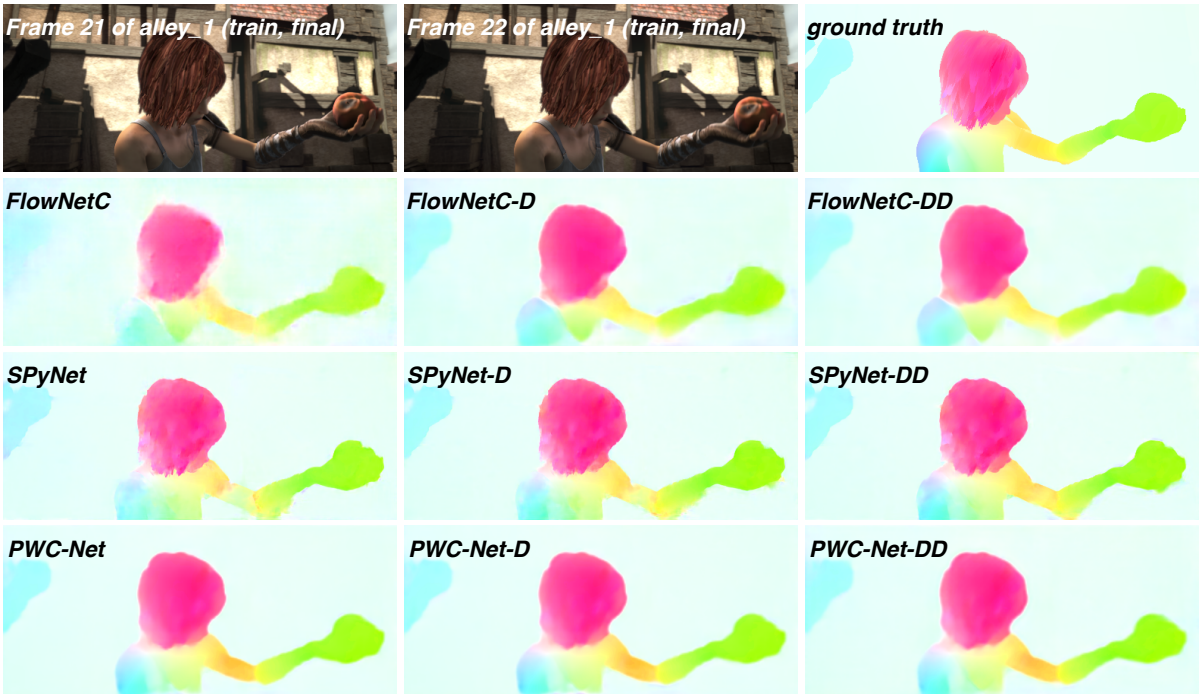

Figure 6.6: The visualization flow predicted by different networks on Sintel training clean set.

Table 6.1: Average EPEs (in pixels) of FlowNetC, FlowNetC-D and FlowNetC-DD on Sintel datasets.

	Sintel Clean		Sintel Final	
Method	Train	Test	Train	Test
FlowNetC	3.78	6.85	5.28	8.51
FlowNetC-D	**2.65**	6.19	3.31	7.34
FlowNetC-DD	2.69	**6.10**	**3.29**	**7.31**
SPyNet	3.17	6.64	4.32	8.36
SPyNet-D	2.61	6.20	3.65	7.67
SPyNet-DD	**2.47**	**6.03**	**3.32**	**7.43**
PWC-Net-small	2.27	5.05	2.45	5.32
PWC-Net	2.02	**4.39**	**2.08**	**5.04**
PWC-Net(our train)	1.91	4.69	2.20	5.60
PWC-Net-D	**1.86**	4.65	2.20	5.56
PWC-Net-DD	1.94	4.71	2.27	5.59

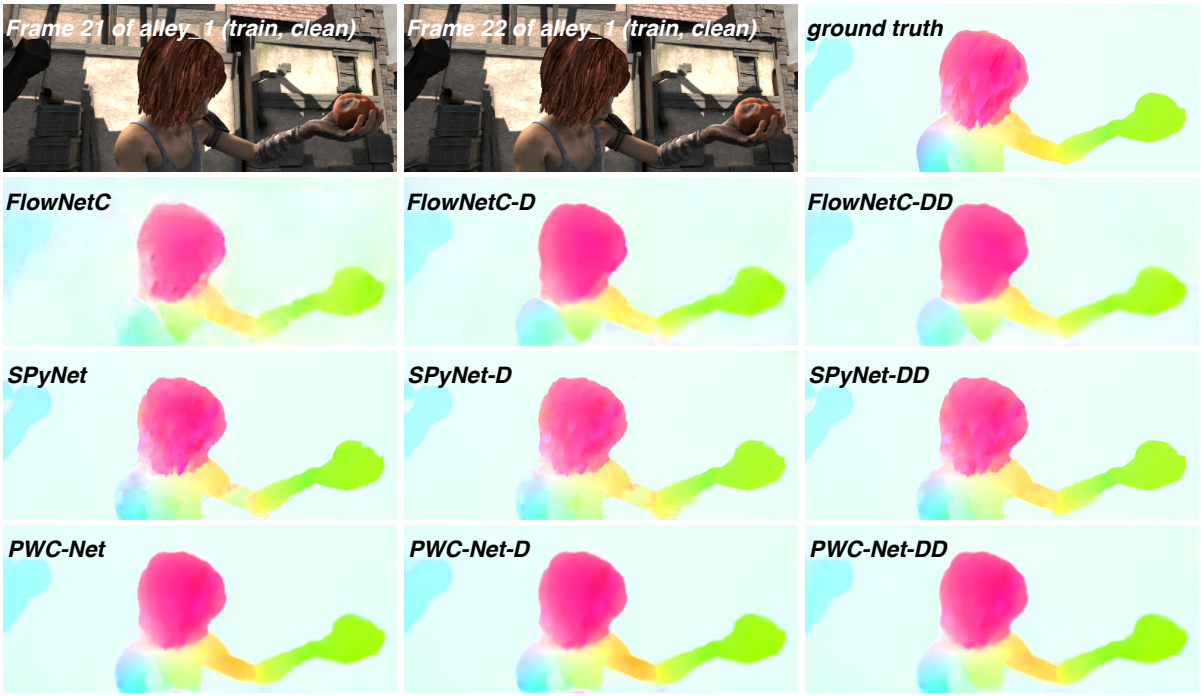

Figure 6.7: The visualization flow predicted by different networks on Sintel training final set.

6.2.3 Analysis

In Table 6.1, the DD module brings clear improvements in EPE in FlownetC and SPyNet. Compared with the original FlowNetC, FlowNetC-DD performs better by 11.02% and 14.15% on the MPI Sintel Clean and Final test sets. Similarly, SPyNet-DD also brings the EPE error down by 9.19% and 11.12% in comparison to SPyNet. However, for PWC-Net and its derivatives, the PWC-Net mentioned in the paper [79] has achieved best performance. However the PWC-Net trained by ourselves performs slightly worse than the PWC-Net-D. The DD module in SPyNet improves the results more than the D module. By comparing with the FlowNetC-D, the FlowNetC-DD performs 1.45% and 0.41% better in test EPE. The EPE from SPyNet-D to the SPyNet-DD also decreases 2.74% and 3.13%, respectively, in the two test sets.

Figure 6.6 and Figure 6.7 show the predicted flow from frame 21 to frame 22 of sequence 'alley_1' in the MPI Sintel training clean/final set. The foreground and back-

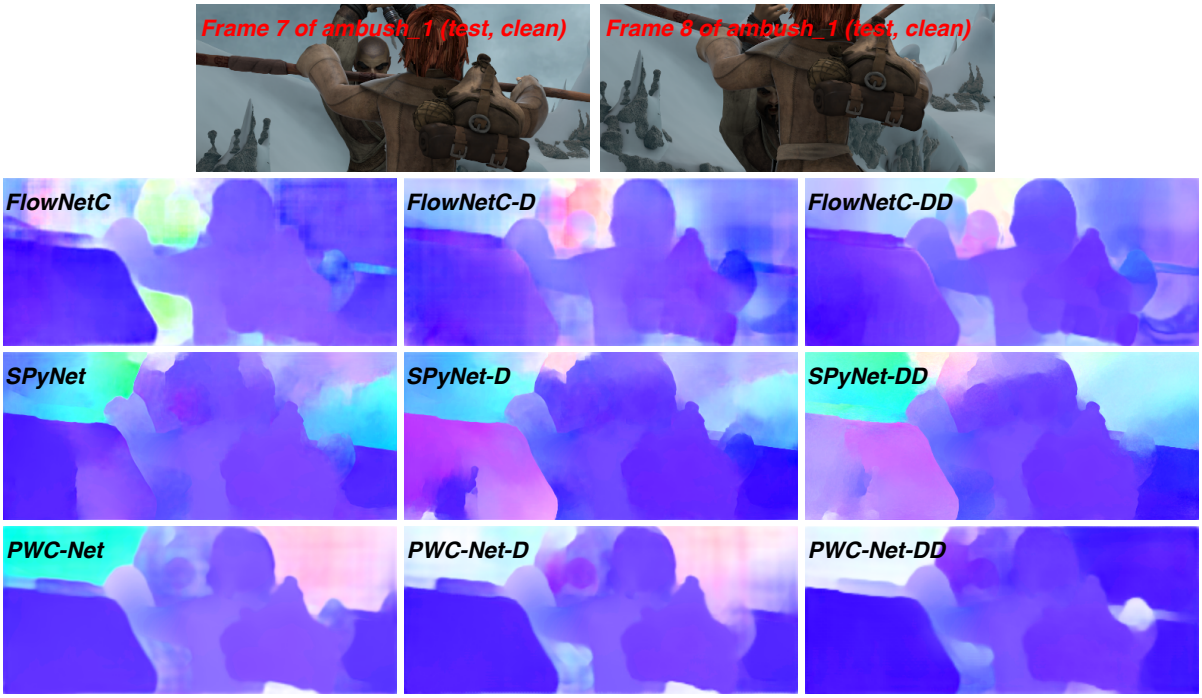

Figure 6.8: The visualization flow predicted by different networks on Sintel testing clean set.

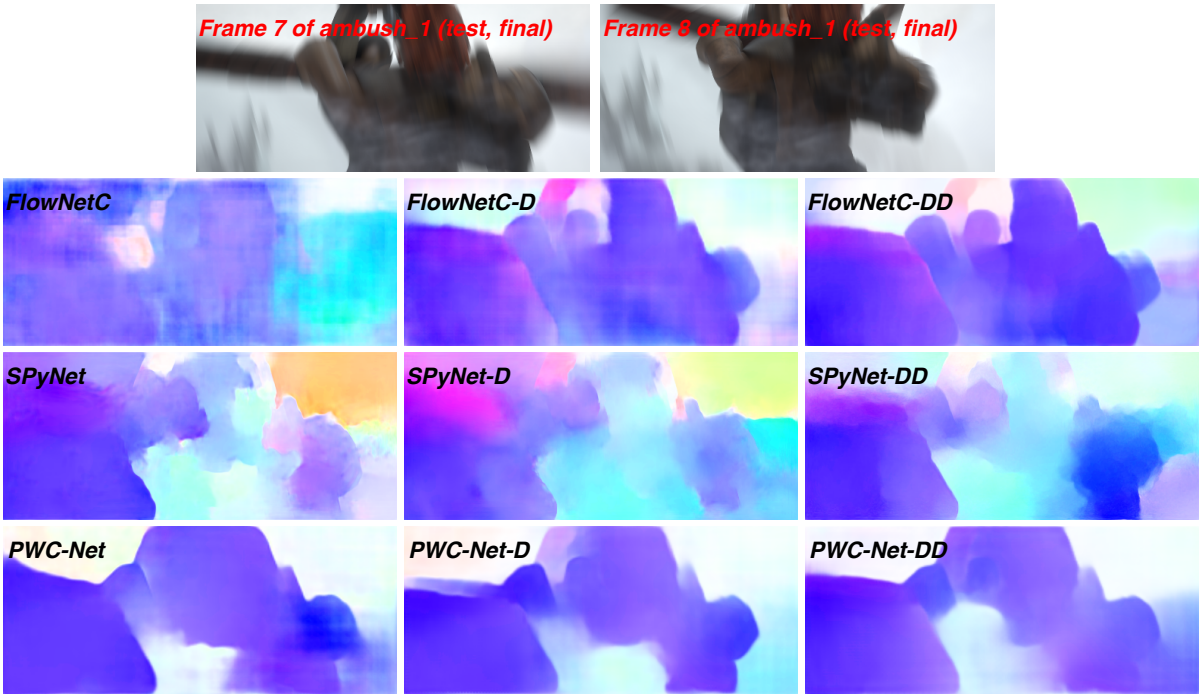

Figure 6.9: The visualization flow predicted by different networks on Sintel testing final set.

ground motion between these two frames is relatively slight. In general, SPyNet and its variants have more details in flow estimation compared with other models. This is caused by the output prediction of flow by SPyNet has the same size than the input image, while the size of output predicted by FlowNetC and PWC-Net is 0.25 smaller than the size of the original images. Based on the color of the visualized prediction results, SPyNet-DD is closer to the ground truth. It can also be found that part of the arm of the avatar is missing in the predicted flow of the original SPyNet, while this problem is solved by the DD module based on its predicted flow. In Figure 6.6, FlowNetC-DD predicts slightly sharper boundaries than FlowNetC-D, but much better boundaries than FLowNetC. When comparing the results from FlowNetC in Figure 6.6 and Figure 6.7, the results on Sintel Final are much worse than on Sintel Clean especially around the boundary regions. However, the visualization results from FlowNetC-D and FlowNetC-DD in Figure 6.6 are only slightly better than in Figure 6.7. Based on this observation, we can draw the conclusion that adding the deformable operation to FlowNetC makes the network more robust to noise. Since PWC-Net already produces decent prediction results in these two images, the D and DD modules do not result in obvious improvements or changes in this case.

Figures 6.8 and 6.9 show the predicted flow of networks from frame 7, frame 8 of sequence 'Ambush_1' in the MPI Sintel training clean/final sets. This test sample shows a large motion of characters and background. In Figure 6.8, we can see that the optical flow estimation of FlowNetC has a boundary blur problem with large displacements. However, FlowNetC-D and FlowNetC-DD are much better with the large displacements as the shapes of the characters are relatively clear. In Figure 6.9, the result of frame 7 from Sintel Final is completely blurry in the FlowNetC result. One explanation is that the maximum displacement of the correlation does not allow to predict very large motions. FlowNetC-DD performs best among the three networks in the Sintel Final dataset with large displacement as the deformable module helps the network learn the shape information. Limited to the design of the SPyNet, it fails to perform well when large displacement happens in the flow estimation. But the D and DD module still help to refine the shape of the avatars in Figure 6.8. Meanwhile, even though the EPE error

of SPyNet-D and SPyNet-DD is lower than original SPyNet in Figure 6.8, SPyNet, and its variants fail to provide clean predictions of the flow based in the visualization result. Even for PWC-Net, the boundary blur problem can be observed in the flow estimation in Figure 6.8 and Figure 6.9. PWC-Net-D and PWC-Net-DD slightly improve the difference between the two avatars and the stick in these two figures.

6.3 SepUNet for Omni-directional Flow datasets and Omni-directional Stereo datasets

Based on the experimental results with the flow estimation datasets, we also applied the deformable modules to the SepUNet architecture to further evaluate the modules performance in ODF and ODS datasets. The modified SepUNet are called SepUNet-D and SepUNet-DD which have been discussed in Section 5.2. We train and test SepUNet and its variants on the omni-directional datasets. The experiment setup, results, and analysis for each dataset are introduced in the following subsections.

6.3.1 Experiment setup

In this experiment, we train and evaluate SepUNet, SepUNet-D, SepUNet-DD on the ODF and ODS datasets. The 24 grouped image batches with size 128×256 are used to train the whole model from scratch for 50 epochs. The initial learning rate is 0.004, while the Adam [36] is adopted as optimizer and SGDR [52] is chosen as learning weight decay method.

For the ODF dataset, L1 loss is chosen to train the network, while the EPE and logarithm of the EPE (LEPE) are used to evaluate the model. The LEPE is adopted as an evaluation metric since the boundary of the panoramic images has much larger flow than the other areas. Therefore, LEPE could provide a better insight of the accuracy of flow estimation in the non-margin areas. The equation of EPE and LEPE are shown in Equation 6.2 and 6.4, respectively. In Equation 6.2 and 6.4, I' and I^{GT} denote the flow

field predicted by the network and corresponding supervised ground truth, respectively.

$$LEPE = \left| L(I') - L(I^{GT}) \right|_2, where\ L(I) = \begin{cases} log(I), & I \geq 1 \\ 0, & -1 \leq I < 1 \\ -log(-I), & I < -1 \end{cases} \tag{6.4}$$

For the ODS dataset, the L1 loss is chosen to train the network, while the prediction result is converted to depth in order to evaluate the methods with other state-of-the-art methods. We adopt relative error (REL), root mean square error (RMSE) and log depth error (Log10) to compare the basic prediction differences and use accuracy with different thresholds to calculate similarity. The equation for the REL, RMSE, Log10 and accuracy are shown in Equation 6.5, 6.6, 6.7 and 6.8, respectively.

$$REL = \frac{1}{N} \sum_{i=0}^{N} |\frac{y_i - \hat{y_i}}{y_i}| \tag{6.5}$$

$$RMSE = \sqrt{\frac{1}{N} \sum_{i=0}^{N} (y_i - \hat{y_i})^2} \tag{6.6}$$

$$Log10 = \frac{1}{N} \sum_{i=0}^{N} |log(y_i) - log(\hat{y_i})| \tag{6.7}$$

$$Accuracy(\delta) = \frac{1}{N} \sum_{i=0}^{N} \Delta(max(\frac{y_i}{\hat{y_i}}, \frac{\hat{y_i}}{y_i}) < \delta) \tag{6.8}$$

In Equation 6.5, 6.6, 6.7 and 6.8, N is the total number of the pixels in the prediction result; y_i and $\hat{y_i}$ represent the ground truth and prediction result in the pixel i, respectively; Δ is the impulse function of the condition. When the condition is true, Δ is 1; else Δ equals to 0. In our experiments, the threshold δ of the accuracy is set to 1.25, 1.25^2 and 1.25^3. We also adopt the absolute border error (ABE) and the relative border error (RBE) proposed by Lai et al. [41]. Intuitively, ABE measures how close the predictions are to the groundtruth along the boundaries of the omnidirectional images, while RBE measures how well a given image is able to connect along the boundaries.

	EPE		LEPE	
	Train	Test	Train	Test
SepUNet	1.994	3.372	0.284	0.459
SepUNet-D	**1.848**	3.085	**0.269**	0.413
SepUNet-DD	1.939	**3.068**	0.271	**0.402**

Table 6.2: Evaluation results for panoramic flow estimation in ODF dataset. Bold numbers indicates the best performance.

6.3.2 Experimental result and analysis for Omni-directional Flow datasets

The results of the ODF experiments with SepUNet and its variants are summarized in Table 6.2 based on the above mentioned metrics. From Table 6.2 The SepUNet-D and SepUNet-DD has clearly improved the EPE and LEPE. The visualization of the results with the corresponding inputs are also shown in Figure 6.10 and 6.11. Figure 6.10 shows the visualization of the previous frame, next frame with their corresponding groundtruth flow and the predicted flow of networks in 50000 training ODF dataset. The testing samples with the ground truth and predicted flow in 10000 testing ODF dataset are illustrated in Figure 6.11. The optical flow shown in the 6.10 and 6.11 is normalized by $L(I)$ in Equation 6.4 and visualized in the same way than the MPI Sintel dataset [10].

In Table 6.2, by comparing to the SepUNet with different modules, we found that the SepUNet-DD had the lowest EPE and LEPE. The deformable modules have improved for the SepUNet. Compared with the original SepUNet, SepUNet-D and SepUNet-DD achieved 7.32% and 2.76% improvement in training EPE and 8.51% and 9.02% improvement in test EPE. For LEPE, these two models also improve 5.28% and 4.58% in training and 10.02% and 12.42% improvement in testing. The training EPE improvement indicates that the deformable modules help a model to converge faster during training, while the reduced testing EPE shows that the modified models are more robust for unseen images. Since the training and testing LEPE improvement in the modified models is

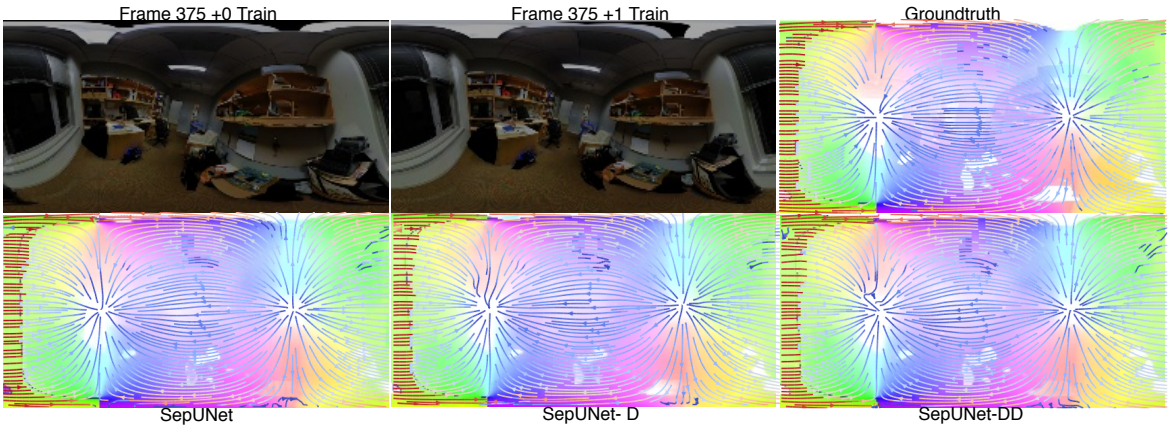

Figure 6.10: Visualization of SepUNet, SepUNet-D and SepUNet-DD results on the ODF training dataset.

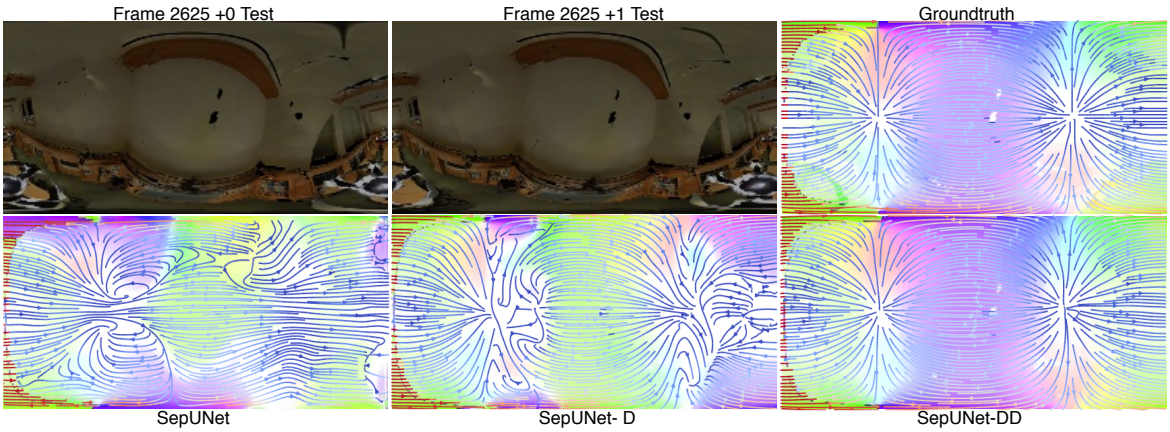

Figure 6.11: Visualization results of SepUNet, SepUNet-D and SepUNet-DD on ODF testing dataset.

larger than the corresponding EPE improvement, the models with deformable modules work better in non-boundary areas.

In Figure 6.10, compared with the groundtruth, all of the results are missing the detail of objects such as the table. However, compared with SepUNet-D and SepUNet-DD, the visualized prediction result of SepUNet are also missing the red and yellow color. A similar situation can also be seen with the test set in Figure 6.13: the color of the SepUNet result is wrong. The color space represents the direction of the optical flow. It means that the margin region has a great influence on the SepUNet results and the SepUNet fails to learn the optical flow direction when close to the image boundary. However,

the deformable module can help the model to learn the spatial transformation and to predict accuracy optical flow with right direction. However, the deformable module fails to predict the correct flow for thin objects. Compared with SepUNet-D in Figure 6.13, the prediction result of SepUNet-DD has better accuracy in optical flow visualized by better matching colors to the ground truth.

	Error (lower is better)					
Method	REL	RMSE	Log10	$ABE_{k=1}$	$ABE_{k=16}$	RBE
SepUNet	2.146	0.703	0.066	0.234	0.209	0.176
SepUNet-D	**2.065**	0.400	0.038	0.143	0.111	0.181
SepUNet-DD	2.120	**0.392**	**0.036**	0.139	0.108	0.180
PSMNet [11]	6.203	0.396	0.059	**0.114**	**0.095**	**0.134**
FCRN [42]	8.223	0.713	0.107	0.300	0.288	0.202
UResNet [91]	16.906	2.037	0.326	1.187	1.441	0.329
RectNet [91]	16.132	1.738	0.291	1.133	1.196	0.523

Table 6.3: Evaluation error results for panoramic depth map estimation in ODS dataset. Bold numbers indicates the best performance.

	Accuracy (higher is better)		
Method	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
SepUNet	0.874	0.957	0.976
SepUNet-D	0.957	0.985	0.990
SepUNet-DD	**0.960**	**0.987**	**0.992**
PSMNet [11]	0.954	0.975	0.980
FCRN [42]	0.785	0.922	0.962
UResNet [91]	0.213	0.399	0.560
RectNet [91]	0.240	0.453	0.634

Table 6.4: Evaluation accuracy results for panoramic depth map estimation in ODS dataset. Bold numbers indicates the best performance.

6.3.3 Experimental result and analysis for Omni-directional Stereo datasets

The results of the ODS experiments with SepUNet and its variants are summarized in Table 6.3 and 6.4 based on the above mentioned metrics, while the result of the current state-of-the-art approaches in mono and stereo image networks first reported by Po *et al.* [41] are also listed in the same table. From Table 6.3 and 6.4, we can see that SepUNet-D and SepUNet-DD boost the performance of SepUNet in nearly every evaluation metrics. The performance of SepUNet-DD compares on par with all the state-of-art CNN architectures on the ODS dataset. The networks visualization results of the corresponding inputs are also shown in Figure 6.12 and 6.13. Figure 6.12 shows the visualization of the left frame, right frame with their corresponding groundtruth depth and the predicted depth of networks in 40000 training ODS dataset. The testing samples with the ground truth and predicted depth in 10000 testing ODS dataset are illustrated in Figure 6.13.

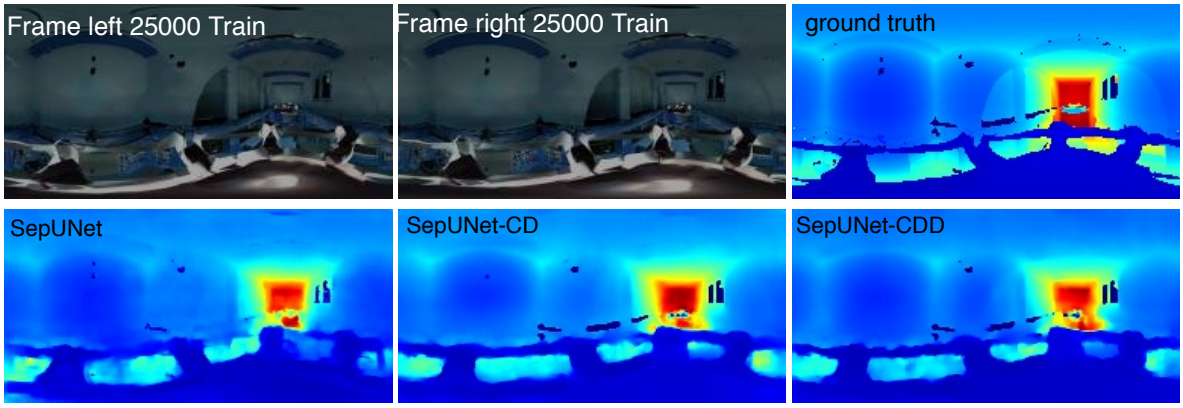

Figure 6.12: Visualization results of SepUNet, SepUNet-D and SepUNet-DD on ODS training dataset.

In Table 6.3 and 6.4, by comparing to the current state-of-the-art approaches in mono and stereo image networks we found that the SepUNet-DD had the lowest error across almost all metrics except across ABE and RBE evaluation metrics, where PSMNet [11] has 12.04%-25.56% better performance boost than SepUNet-DD. When comparing SepUNet and its derivatives, we can see that SepUNet-DD outperforms SepUNet by

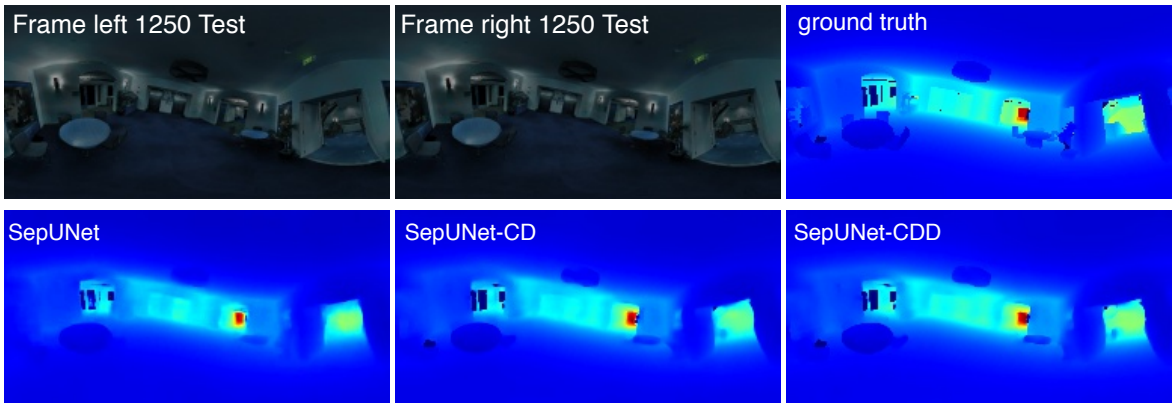

Figure 6.13: Visualization results of SepUNet, SepUNetCD and SepUNet-DD on ODS testing dataset.

44.24% on log10 evaluation metrics and has a 1.43%-9.84% higher accuracy at different thresholds. Compared with the original SepUNet, the deformable layer does not bring too much improvement on REL value. However, based on the increased performance in RMSE, Log10, ABE and Accuracy, the deformable module can greatly improve the performance in the whole image and at the border. However, this module does not improve the consistency of the predicted image as indicated by the evaluation result in RBE.

On the other hand, the mono image input networks (FCRN [42], UResNet [91] and RectNet [91]) do not perform on par with the stereo input networks (SepUNet and its derivatives and PSMNet [11]).Since there are only 40000 images, this limits the possible fine-tuning for state-of-the-art networks. For the result mentioned in the state-of-the-art network, the original networks have been trained by their respective authors for the best performance without limitation on the number of training epochs. It also suggests that stereo input networks are more robust with unseen images.

In Figure 6.12, the predicted result of SepUNet misses the edges of the armrests and the boundary of the items are blurred. A similar situation can also be found in the SepUNet results of Figure 6.13: small items are missed and the predicted results are not smooth. Based on the shown output of SepUNet-D and SepUNet-DD, most of the boundary of large objects and small items can be clearly seen. The edge of items and the wall becomes much sharper than in the output of SepUNet. The consistency

of the output image also looks much better. Even though there are still small flaws in the output images such as the items on the ceiling of the room, the deformable module clearly eliminates boundary blur and greatly improves overall output results.

Chapter 7

Conclusions

7.1 Conclusions

In this thesis, we investigate effective neural network modules for optical flow estimation. Optical flow has many applications in computer graphics, augmented reality and in 3D modeling. However, CNN-based learning flow estimation methods have the boundary blur problem especially when the image has large distortion. In order to estimate the performance of different modules, the CNN-based learning flow estimation methods are time- and computation-consuming to get the results for the specific dataset.

Therefore, in this thesis, we first propose a diagnostic flow estimation called FlowClevr dataset to quickly evaluate the performance of different modules in neural networks

for flow estimation. Based on the evaluation results, we find that the models with a deformable module and the models with a correlation module can reduce the EPE by 30% to 100% in most cases and reduces boundary blur. These modules are further verified in the state-of-the-art CNN-based flow estimation networks on the MPI sintel dataset. The flow estimation networks are modified by replacing some of the original modules with correlation layers and deformable convolutional layers. Compared with the original models, the modified models achieved 9.19% to 14.15% improvement of EPE during testing in most cases. Modified models also predict the flow with clearer boundaries in training and testing images. Therefore, the modules performance show a similar trend between public datasets and the diagnostic dataset and it can reduce the boundary blur efficiently.

We then apply the efficient modules to the ODF dataset. The equirectangular representation of the panoramic images in the dataset have large distortion and their optical flow groundtruth has large values. The SepUNet proposed by Lai et al. is adopted to estimate the optical flow in the ODF dataset. The original SepUNet is modified with the correlation and deformable convolutional modules and we named the modified models as SepUNet-D and SepUNet-DD. With efficient modules, the modified network SepUNet-DD can reduce the testing EPE by 9.02% and LEPE by 10.04% comparing with the original SepUNet. To verify the efficient modules performance in other tasks, we also compare the performances of SepUNet, SepUNet-D, and SepUNet-DD on the ODS dataset. Compared with the SepUNet, SepUNet-DD can reduce the Log10 Error by 44.24% and improve the accuracy by 9.84%. What's more, the SepUNet-DD has much clearer boundary in the visualized prediction result. Based on that, we conclude that the correlation and deformable convolutional modules are able to reduce the EPE and boundary blur in the equirectangular representation of the panoramic images. Also, the results in the ODS dataset shows that the modules have similar effects on the disparity/depth estimation task.

7.2 Limitations and future work

The current proposed diagnostic FLowClevr dataset can be a great indicator on what module to choose for flow estimation tasks. However, since the FLowClevr dataset only contains linear 2D spatial transformations in the image plane, it does not include 3D spatial transformations involving depth, synthetic and real world datasets typically include such more realistic changes. Therefore, one potential future work for FlowClevr dataset is to widen the set of transformations to better simulate the spatial transformation in the public datasets and in the real world cases.

In addition, even though we run SepUNet and its variants on real-world omni-directional videos [41], we cannot evaluate the performance since we do not have the ground truth. It would be therefore worthwhile to test and evaluate the networks on additional synthetic and real-world scenarios.

References

[1] Radhakrishna Achanta et al. "SLIC Superpixels Compared to State-of-the-art Superpixel Methods". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34.11 (2012), pp. 2274–2282.

[2] Robert Anderson et al. "Jump: Virtual Reality Video". In: *ACM Transactions on Graphics* 35.6 (2016), p. 198.

[3] Iro Armeni et al. "Joint 2D-3D-semantic Data for Indoor Scene Understanding". In: *arXiv:1702.01105* (2017).

[4] Simon Baker et al. "A database and evaluation methodology for optical flow". In: *International Journal of Computer Vision* 92.1 (2011), pp. 1–31.

[5] Connelly Barnes et al. "The Generalized Patchmatch Correspondence Algorithm". In: *Proceedings of European Conference on Computer Vision*. Springer. 2010, pp. 29–43.

[6] Sean Bell et al. "Inside-outside Net: Detecting Objects in Context with Skip Pooling and Recurrent Neural Networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 2874–2883.

[7] Michael J Black and Paul Anandan. "The robust estimation of multiple motions: Parametric and piecewise-smooth flow fields". In: *Computer vision and image understanding* 63.1 (1996), pp. 75–104.

[8] Michael Bleyer et al. "Object Stereo—joint Stereo Matching and Object Segmentation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2011, pp. 3081–3088.

[9] Thomas Brox and Jitendra Malik. "Large Displacement Optical Flow: Descriptor Matching in Variational Motion Estimation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33.3 (2011), pp. 500–513.

[10] Daniel J Butler et al. “A Naturalistic Open Source Movie for Optical Flow Evaluation”. In: *Proceedings of European Conference on Computer Vision*. Springer. 2012, pp. 611–625.

[11] Jia-Ren Chang and Yong-Sheng Chen. “Pyramid Stereo Matching Network”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 5410–5418.

[12] Vincent Chapdelaine-Couture and Sébastien Roy. “The Omnipolar Camera: A New Approach to Stereo Immersive Capture”. In: *Proceedings of the IEEE International Conference on Computational Photography*. IEEE. 2013, pp. 1–9.

[13] Liang-Chieh Chen et al. “Deeplab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40.4 (2018), pp. 834–848.

[14] Zhuoyuan Chen et al. “Large Displacement Optical Flow from Nearest Neighbor Fields”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2013, pp. 2443–2450.

[15] François Chollet. “Xception: Deep Learning with Depthwise Separable Convolutions”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2017, pp. 1800–1807.

[16] Dan Cireşan, Ueli Meier, and Jürgen Schmidhuber. “Multi-column Deep Neural Networks for Image Classification”. In: *arXiv:1202.2745* (2012).

[17] Jifeng Dai et al. “Deformable Convolutional Networks”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 764–773.

[18] Alexey Dosovitskiy et al. “Flownet: Learning Optical Flow with Convolutional Networks”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 2758–2766.

[19] David Eigen, Christian Puhrsch, and Rob Fergus. “Depth Map Prediction from a Single Image Using a Multi-scale Deep Network”. In: *Proceedings of the Annual*

Conference on Advances in Neural Information Processing Systems. 2014, pp. 2366–2374.

[20] Frederick G Fernald. "Analysis of Atmospheric Lidar Observations: Some Comments". In: *Applied Optics* 23.5 (1984), pp. 652–653.

[21] William T Freeman, Egon C Pasztor, and Owen T Carmichael. "Learning low-level vision". In: *International journal of computer vision* 40.1 (2000), pp. 25–47.

[22] Andreas Geiger, Philip Lenz, and Raquel Urtasun. "Are We Ready for Autonomous Driving? The KITTI Vision Benchmark Suite". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2012, pp. 3354–3361.

[23] Andreas Geiger et al. "Vision Meets Robotics: The KITTI Dataset". In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1231–1237.

[24] Andrea Giachetti, Marco Campani, and Vincent Torre. "The Use of Optical Flow for Road Navigation". In: *IEEE Transactions on Robotics and Automation* 14.1 (1998), pp. 34–48.

[25] James J Gibson. "The Perception of the Visual World." In: (1950).

[26] Xufeng Han et al. "Matchnet: Unifying Feature and Metric Learning for Patch-based Matching". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 3279–3286.

[27] Berthold KP Horn and Brian G Schunck. "Determining Optical Flow". In: *Artificial Intelligence* 17.1-3 (1981), pp. 185–203.

[28] Tak-Wai Hui and Ronald Chung. "Determining Motion Directly from Normal Flows upon the use of a Spherical Eye Platform". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2013, pp. 2267–2274.

[29] Tak-Wai Hui, Xiaoou Tang, and Chen Change Loy. "LiteFlowNet: A Lightweight Convolutional Neural Network for Optical Flow Estimation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 8981–8989.

[30] Eddy Ilg et al. “Flownet 2.0: Evolution of Optical Flow Estimation with Deep Networks”. In: *IEEE Conference on Computer Vision and Pattern Recognition*. Vol. 2. 2017, p. 6.

[31] Hirochika Inoue, Tetsuya Tachikawa, and Masayuki Inaba. “Robot Vision System with a Correlation Chip for Real-time Tracking, Optical Flow and Depth Map Generation”. In: *Proceedings of the IEEE International Conference on Robotics and Automation*. IEEE. 1992, pp. 1621–1626.

[32] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *Proceedings of the International Conference on Machine Learning*. 2015, pp. 448–456.

[33] Hiroshi Ishiguro, Masashi Yamamoto, and Saburo Tsuji. “Omni-directional Stereo for Making Global Map”. In: *Proceedings of the International Conference on Computer Vision*. IEEE. 1990, pp. 540–547.

[34] Allan D Jepson and Michael J Black. *Mixture Models for Optical Flow Computation*. Department of Computer Science, University of Toronto, 1993.

[35] Justin Johnson et al. “Clevr: A Diagnostic Dataset for Compositional Language and Elementary Visual Reasoning”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 2901–2910.

[36] Diederik P Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *arXiv:1412.6980* (2014).

[37] Ron Kohavi et al. “A Study of Cross-validation and Bootstrap for Accuracy Estimation and Model Selection”. In: *Proceedings of the International Joint Conferences on Artificial Intelligence Organization*. Vol. 14. 2. Montreal, Canada. 1995, pp. 1137–1145.

[38] Robert Konrad et al. “Spinvr: Towards Live-streaming 3D Virtual Reality Video”. In: *ACM Transactions on Graphics* 36.6 (2017), p. 209.

[39] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet Classification with Deep Convolutional Neural Networks". In: *Proceedings of the Annual Conference on Advances in neural information processing systems*. 2012, pp. 1097–1105.

[40] Lubor Ladickẁ et al. "Joint Optimization for Object Class Segmentation and Dense Stereo Reconstruction". In: *International Journal of Computer Vision* 100.2 (2012), pp. 122–133.

[41] P Lai et al. "Real-time Panoramic Depth Maps from Omni-directional Stereo Images for 6 DoF Videos in Virtual Reality". In: *Proceeding of IEEE Conference on Virtual Reality and 3D User Interfaces*. IEEE. 2019, pp. 1–4.

[42] Iro Laina et al. "Deeper Depth Prediction with Fully Convolutional Residual Networks". In: *Proceedings of the International Conference on 3D Vision*. IEEE. 2016, pp. 239–248.

[43] Yann LeCun and Fu Jie Huang. "Loss Functions for Discriminative Training of Energy-Based Models". In: *Proceedings of the International Conference on Artificial Intelligence and Statistics*. Vol. 6. 2005, p. 34.

[44] Yann LeCun, Yoshua Bengio, et al. "Convolutional networks for images, speech, and time series". In: *The handbook of brain theory and neural networks* 3361.10 (1995), p. 1995.

[45] Marius Leordeanu, Rahul Sukthankar, and Cristian Sminchisescu. "Efficient closed-form solution to generalized boundary detection". In: *European Conference on Computer Vision*. Springer. 2012, pp. 516–529.

[46] Yunpeng Li and Daniel P Huttenlocher. "Learning for optical flow using stochastic optimization". In: *European Conference on Computer Vision*. Springer. 2008, pp. 379–391.

[47] David Lieb, Andrew Lookingbill, and Sebastian Thrun. "Adaptive Road Following using Self-Supervised Learning and Reverse Optical Flow". In: *Proceedings of the Conference on Robotics: Science and systems*. 2005, pp. 273–280.

[48] Ce Liu et al. “Human-assisted Motion Annotation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2008, pp. 1–8.

[49] Rosanne Liu et al. “An Intriguing Failing of Convolutional Neural Networks and the Coordconv Solution”. In: *Proceedings of the Annual Conference on Advances in Neural Information Processing Systems*. 2018, pp. 9628–9639.

[50] Yang Liu et al. “Technical evaluation of HoloLens for multimedia: a first look”. In: *IEEE MultiMedia* 25.4 (2018), pp. 8–18.

[51] Jonathan Long, Evan Shelhamer, and Trevor Darrell. “Fully Convolutional Networks for Semantic Segmentation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 3431–3440.

[52] Ilya Loshchilov and Frank Hutter. “SGDR: Stochastic Gradient Descent With Warm Restarts”. In: *arXiv:1608.03983* (2016).

[53] Jiangbo Lu et al. “Patch Match Filter: Efficient Edge-aware Filtering Meets Randomized Search for Fast Correspondence Field Estimation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2013, pp. 1854–1861.

[54] Bruce D Lucas, Takeo Kanade, et al. “An iterative image registration technique with an application to stereo vision”. In: (1981).

[55] Wenjie Luo, Alexander G Schwing, and Raquel Urtasun. “Efficient Deep Learning for Stereo Matching”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 5695–5703.

[56] Yasushi Mae et al. “Object Tracking in Cluttered Background Based on Optical Flows and Edges”. In: *Proceedings of the International Conference for Pattern Recognition*. IEEE. 1996, p. 196.

[57] Xiaojiao Mao, Chunhua Shen, and Yu-Bin Yang. “Image Restoration Using Very Deep Convolutional Encoder-decoder Networks with Symmetric Skip Connections”. In: *Proceedings of the Annual Conference on Advances in Neural Information Processing Systems*. 2016, pp. 2802–2810.

[58] Kevin Matzen et al. “Low-cost 360 Stereo Photography and Video Capture”. In: *ACM Transactions on Graphics* 36.4 (2017), p. 148.

[59] Nikolaus Mayer et al. “A Large Dataset to Train Convolutional Networks for Disparity, Optical Flow, and Scene Flow Estimation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 4040–4048.

[60] Moritz Menze and Andreas Geiger. “Object Scene Flow for Autonomous Vehicles”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 3061–3070.

[61] Ivana Mikic, Slawomir Krucinski, and James D Thomas. “Segmentation and Tracking in Echocardiographic Sequences: Active Contours Guided by Optical Flow Estimates”. In: *IEEE Transactions on Medical Imaging* 17.2 (1998), pp. 274–284.

[62] “Model for the Extraction of Image Flow, author=Heeger, David J”. In: *Journal of the Optical Society of America* 4.8 (1987), pp. 1455–1471.

[63] Vinod Nair and Geoffrey E Hinton. “Rectified Linear Units Improve Restricted Boltzmann Machines”. In: *Proceedings of the International Conference on Machine Learning*. 2010, pp. 807–814.

[64] Jiahao Pang et al. “Cascade Residual Learning: A Two-stage Convolutional Neural Network for Stereo Matching”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 887–895.

[65] Shmuel Peleg, Moshe Ben-Ezra, and Yael Pritch. “Omnistereo: Panoramic Stereo Imaging”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 3 (2001), pp. 279–290.

[66] Ying Qiu et al. “EVM-CNN: Real-time contactless heart rate estimation from facial video”. In: *IEEE Transactions on Multimedia* (2018).

[67] Anurag Ranjan and Michael J Black. “Optical Flow Estimation using a Spatial Pyramid Network.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. Vol. 2. 4. 2017, p. 5.

[68] Jerome Revaud et al. “Epicflow: Edge-preserving Interpolation of Correspondences for Optical Flow”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 1164–1172.

[69] Stephan R Richter et al. “Playing for Data: Ground Truth from Computer Games”. In: *Proceedings of the European Conference on Computer Vision*. Springer. 2016, pp. 102–118.

[70] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-Net: Convolutional Networks for Biomedical Image Segmentation”. In: *Proceedings of the International Conference on Medical Image Computing and Computer-assisted Intervention*. Springer. 2015, pp. 234–241.

[71] Stefan Roth and Michael J Black. “Fields of Experts: A Framework for Learning Image Priors”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. Vol. 2. IEEE. 2005, pp. 860–867.

[72] Stefan Roth and Michael J Black. “On the Spatial Statistics of Optical Flow”. In: *International Journal of Computer Vision* 74.1 (2007), pp. 33–50.

[73] Daniel Scharstein and Richard Szeliski. “A Taxonomy and Evaluation of Dense Two-frame Stereo Correspondence Algorithms”. In: *International Journal of Computer Vision* 47.1-3 (2002), pp. 7–42.

[74] Christopher Schroers, Jean-Charles Bazin, and Alexander Sorkine-Hornung. “An Omnistereoscopic Video Pipeline for Capture and Display of Real-World VR”. In: *ACM Transactions on Graphics* 37.3 (2018), p. 37.

[75] Eero P Simoncelli, Edward H Adelson, and David J Heeger. “Probability Distributions of Optical Flow”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 1991, pp. 310–315.

[76] Nitish Srivastava et al. “Dropout: a Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[77] Deqing Sun, Erik B Sudderth, and Michael J Black. “Layered Image Motion with Explicit Occlusions, Temporal Consistency, and Depth Ordering”. In: *Proceedings of the Annual Conference on Advances in Neural Information Processing Systems*. 2010, pp. 2226–2234.

[78] Deqing Sun et al. “Learning Optical Flow”. In: *Proceedings of European Conference on Computer Vision*. Springer. 2008, pp. 83–97.

[79] Deqing Sun et al. “PWC-Net: CNNs for Optical Flow Using Pyramid, Warping, and Cost Volume”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 8934–8943.

[80] Jayant Thatte et al. “Depth Augmented Stereo Panorama for Cinematic Virtual Reality with Head-motion Parallax”. In: *Proceedings of IEEE International Conference on Multimedia and Expo*. IEEE. 2016, pp. 1–6.

[81] Jayant Thatte et al. “Stacked Omnistereo for Virtual Reality with Six Degrees of Freedom”. In: *Proceedings of IEEE Conference on Visual Communications and Image Processing*. IEEE. 2017, pp. 1–4.

[82] Régis Vaillant, Christophe Monrocq, and Yann Le Cun. “Original Approach for the Localisation of Objects in Images”. In: vol. 141. 4. IEEE, 1994, pp. 245–250.

[83] D.H. Warren and E.R. Strelow. *Electronic Spatial Sensing for the Blind: Contributions from Perception, Rehabilitation, and Computer Vision*. Nato Science Series E: Springer, 1985.

[84] Philippe Weinzaepfel et al. “DeepFlow: Large Displacement Optical Flow with Deep Matching”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2013, pp. 1385–1392.

[85] Manuel Werlberger et al. “Anisotropic Huber-L1 Optical Flow.” In: *Proceedings of the British Machine Vision Conference*. Vol. 1. 2. 2009, p. 3.

[86] Jonas Wulff and Michael J Black. “Efficient Sparse-to-dense Optical Flow Estimation using a Learned Basis and Layers”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 120–130.

[87] Li Xu, Jiaya Jia, and Yasuyuki Matsushita. “Motion Detail Preserving Optical Flow Estimation”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34.9 (2012), pp. 1744–1757.

[88] Zhaozhuo Xu et al. “Deformable ConvNet with Aspect Ratio Constrained NMS for Object Detection in Remote Sensing Imagery”. In: *Remote Sensing* 9.12 (2017), p. 1312.

[89] Fisher Yu and Vladlen Koltun. “Multi-scale Context Aggregation by Dilated Convolutions”. In: *arXiv:1511.07122* (2015).

[90] Jure Zbontar, Yann LeCun, et al. “Stereo Matching by Training a Convolutional Neural Network to Compare Image Patches.” In: *Journal of Machine Learning Research* 17.1-32 (2016), p. 2.

[91] Nikolaos Zioulis et al. “OmniDepth: Dense Depth Estimation for Indoors Spherical Panoramas”. In: *Proceedings of European Conference on Computer Vision*. Ed. by Vittorio Ferrari et al. Springer, 2018, pp. 453–471.