ABSTRACT

Title of dissertation:	LEARNING OF DENSE OPTICAL FLOW, MOTION AND DEPTH FROM SPARSE EVENT CAMERAS
	Chengxi Ye Doctor of Philosophy, 2019
Dissertation directed by:	Prof. Yiannis Aloimonos and Dr. Cornelia Fermüller Department of Computer Science

With recent advances in the field of autonomous driving, autonomous agents need to safely navigate around humans or other moving objects in unconstrained, highly dynamic environments. In this thesis, we demonstrate the feasibility of reconstructing *dense* depth, optical flow and motion information from a neuromorphic imaging device, called Dynamic Vision Sensor (DVS). The DVS only records sparse and asynchronous events when the changes of lighting occur at camera pixels. Our work is the first monocular pipeline that generates dense depth and optical flow from sparse event data only.

To tackle this problem of reconstructing dense information from sparse information, we introduce the Evenly-Cascaded convolutional Network (ECN), a bio-inspired multi-level, multi-resolution neural network architecture. The network features an evenly-shaped design, and utilization of both high and low level features.

With just 150k parameters, our self-supervised pipeline is able to surpass pipelines that are 100x larger. We evaluate our pipeline on the MVSEC self driving dataset and present results for depth, optical flow and and egomotion estimation in wild outdoor scenes. Due to the lightweight design, the inference part of the network runs at 250 FPS on a single GPU, making the pipeline ready for realtime robotics applications. Our experiments demonstrate significant improvements upon previous works that used deep learning on event data, as well as the ability of our pipeline to perform well during both day and night.

We also extend our pipeline to dynamic indoor scenes with independent moving objects. In addition to camera egomotion and a dense depth map, the network utilizes a mixture model to segment and compute per-object 3D translational velocities for moving objects. For this indoor task we are able to train a shallow network with just 40k parameters, which computes qualitative depth and egomotion.

Our analysis of the training shows modern neural networks are trained on tangled signals. This tangling effect can be imagined as a blurring introduced both by nature and by the training process. We propose to untangle the data with network deconvolution. We notice significantly better convergence without using any standard normalization techniques, which suggests us deconvolution is what we need.

LEARNING OF DENSE OPTICAL FLOW, MOTION AND DEPTH FROM SPARSE EVENT CAMERAS

by

Chengxi Ye

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2019

Acknowledgments

I would like to thank my advisors, Professor Yiannis Aloimonos and Dr Cornelia Fermüller for being so nice and supportive even in those unproductive times. Without them I would not have obtained my degree.

I would like to thank Prof. James A. Yorke, who has used his experience to reshape my way of doing research and help me finish my PhD.

I would like to thank Prof. Hector Corrada-Bravo for supporting me doing base-calling/blind deconvolution research. This has relation with what this thesis finishes up with.

I would also like to thank Prof. Mihai Pop for helping me to get into UMD, and for supporting me in the earlier period of my PhD.

I owe many thanks to my family who have waited for me for so long.

I thank my labmates and my friends, who have added lots of fun to the PhD grinding.

Table of Contents

Acknowledgements ii

List of Tables vi

List of Figures vii

1 Introduction 1
- 1.1 The Dynamic Vision Sensor 1
- 1.2 Structure from Motion 3
- 1.3 What is an Artificial Neural Network? 4
- 1.4 Why is it Hard to Train a Network? 4
- 1.5 A Missing Piece 5
- 1.6 Inspirations from Human Learning 7
- 1.7 The Organization of the Theis 8

2 Related Work 10
- 2.1 Network Architectures 10
- 2.2 Event-based Depth Estimation 11
- 2.3 Event-based Optical Flow 12
- 2.4 Independent Motion Detection 13
- 2.5 Learning in Structure from Motion 14

3 Evenly Cascaded Convolutional Networks 15
- 3.1 Methods 20
  - 3.1.1 Utilization of Multilevel Features 20
    - 3.1.1.1 From a Network Architectural Point of View 20
    - 3.1.1.2 From a Wavelet Point of View 22
    - 3.1.1.3 From an Optimization Point of View 23
  - 3.1.2 Fractional Scaling 24
  - 3.1.3 Convolution Block Design 25
- 3.2 Experiments 28
  - 3.2.1 Datasets 28
  - 3.2.2 Results 29

3.2.3 Comparison of Convolution Blocks over CIFAR10 and CIFAR100 . . . 31
3.3 Conclusion . . . 35

4 Unsupervised Learning of Dense Optical Flow, Depth and Egomotion from Sparse Event Data 37
4.1 Related Work . . . 39
4.1.1 Event-based Depth Estimation . . . 39
4.1.2 Event-based Optical Flow . . . 40
4.1.3 Self-supervised Structure from Motion . . . 41
4.2 Methods . . . 41
4.2.1 Ego-motion Model . . . 41
4.2.2 Input Data Representation . . . 42
4.2.3 The Pipeline . . . 43
4.2.4 Evenly Cascaded Network Architecture . . . 44
4.2.5 Depth Predictions . . . 46
4.2.6 Feature Decorrelation . . . 48
4.3 Experimental Evaluation . . . 49
4.3.1 Implementation Details . . . 50
4.3.2 Dataset Preparation . . . 51
4.3.3 Ablation Studies . . . 52
4.3.3.1 Testing on the *SfMlearner* . . . 52
4.3.3.2 Normalization Methods . . . 52
4.3.3.3 Visualizing a Shallow and Tiny Network . . . 53
4.3.3.4 Performance Versus Event Rate . . . 53
4.3.4 Qualitative Results . . . 54
4.3.5 Optical Flow Evaluation . . . 54
4.3.6 Depth Evaluation . . . 56
4.3.7 Egomotion Estimation . . . 56
4.3.8 Discussion and Failure Cases . . . 60
4.4 CONCLUSION . . . 61

5 EV-IMO: Motion Segmentation Dataset and Learning Pipeline for Event Cameras 67
5.1 The Architecture . . . 68
5.1.1 Network Input . . . 68
5.1.2 Overview of the Architecture . . . 70
5.1.3 Motion Model . . . 71
5.1.4 A Mixture Model for Ego-motion and Independently Moving Objects . . . 72
5.1.5 Loss functions . . . 72
5.1.5.1 Event Warping Loss . . . 73
5.1.5.2 Motion Mask Loss . . . 74
5.1.5.3 Depth Loss . . . 74
5.1.6 Evenly Cascading Network Architecture . . . 75

5.1.7 Prediction of Depth and Component Weights 75
5.2 EV-IMO Dataset 76
5.2.0.1 Methodology 76
5.2.0.2 Dataset Generation 77
5.2.1 Sequences 79
5.3 Experiments 80
5.3.0.1 Qualitative Evaluation 82
5.3.0.2 Segmentation and Motion Estimation 83
5.3.0.3 Comparison With Previous Work 84
5.4 Conclusions 86

6 Network Deconvolution 89
6.1 Introduction 89
6.2 Related Work 92
6.2.1 Normalization and Whitening 92
6.3 The Deconvolution Operation 93
6.3.1 Definition 93
6.3.2 Expected Isometry via Deconvolution 95
6.3.3 Accelerated Convergence 96
6.3.4 How Network Deconvolution is Applied in a CNN 97
6.3.5 Efficient Calculation of the Inverse Square Root of the Covariance Matrix 99
6.3.6 Sparse Representations 101
6.4 Experiments 102
6.5 Acknowledgement 107
6.6 Conclusion 107
6.7 Appendix 108
6.7.1 Acceleration via Subsampling 108
6.7.2 Regularizations 109
6.7.3 Sparse Representations for Convolution Layers 109
6.7.4 Accelerated Convergence 110

Bibliography 112

List of Tables

3.1 CIFAR accuracies for ECN-6 . . . 28
3.2 ECN-6 architecture . . . 28
3.3 Results on the ImageNet-32 datase . . . 30
3.4 Error rate comparison with state-of-the-art efficient architectures . . . 31
3.5 Error rate comparison with state-of-the-art architectures . . . 31
3.6 Comparison of convolution blocks on the CIFAR-10 dataset . . . 33
3.7 Comparison of convolution blocks on the CIFAR-100 dataset . . . 35

4.1 Evaluation of the optical flow pipeline . . . 50
4.2 Evaluation of the depth estimation pipeline . . . 57
4.3 Egomotion estimation results . . . 59

5.1 EV-IMO sequences . . . 81
5.2 Evaluation on segmentation and motion estimation . . . 85
5.3 Comparison of $EV - IMO$ with a classic method . . . 85
5.4 Evaluation of the depth estimation . . . 87

6.1 Results on CIFAR datasets . . . 104
6.2 Comparison of top-1 accuracy on ImageNet . . . 108

List of Figures

1.1 The DVS Camera 2
1.2 The human learning process 9

3.1 Evenly cascaded network design 16
3.2 Visualizations of feature maps in ECN. 18
3.3 Multilevel feature representations within ECN. 21
3.4 Evaluation of various convolution blocks 27
3.5 Relationship between network parameter count, block category, and classification accuracy. 34
3.6 Comparison between parameter count and classification accuracy over CIFAR-10 for ECN with blocks 4 and 6. 35
3.7 Comparison between parameter count and classification accuracy over CIFAR-100 for ECN with blocks 4 and 6 36

4.1 Optical flow and depth inference on sparse event data in night scene . 38
4.2 A three-channel DVS data representation 39
4.3 The unsupervised pipeline. 43
4.4 The encoder-decoder structure 45
4.5 Qualitative results from our evaluation 47
4.6 Comparison of Abs Rel Errors using different normalization methods 62
4.7 Visualization of feature maps of the depth network 63
4.8 Visualization of the pose network 63
4.9 Evaluation of average endpoint error 64
4.10 Estimation of trajectories 65
4.11 A typical failure case and a dataset artifact 66

5.1 Depth and per-pixel pose inference on sparse event data, on our *EV-IMO* dataset 68
5.2 Our depth and motion estimation pipeline 69
5.3 The main interface of the automatic annotation tool 78
5.4 Types of background geometry featured in the EV-IMO dataset . . . 80
5.5 Comparison of the full network inference quality with the small version 83
5.6 Qualitative results from our evaluation 84

6.1 Network deconvolution 91
6.2 im2col 98
6.3 visualization of deconvolution 102
6.4 Results on MNIST datasets 103
6.5 Results on CIFAR-10 105
6.6 Results on CIFAR-100 105
6.7 Results on ImageNet 107
6.8 The effects of weight decay w/w.o deconvolution 110
6.9 Visualization of intermediate feature maps 111

Chapter 1: Introduction

With recent advances in the field of autonomous driving, autonomous platforms are no longer restricted to research laboratories and testing facilities - they are designed to operate in highly dynamic environments and have to maneuver amongst other moving objects or humans.

This renders the classic structure from motion (SfM) pipeline, often implemented via a SLAM-like [1] algorithms, not only inefficient but also incapable of solving the problem of motion estimation. A truly autonomous agent should be able to instantly detect every independently moving object on the scene, estimate the distance to it and predict its trajectory, while also having the knowledge of its own egomotion.

Modern self-driving cars are often fitted with a sophisticated sensor rig, featuring a number of LiDARs, cameras and radars, but even those undoubtedly expensive setups are prone to misperform in difficult conditions - snow, fog, rain or at night.

1.1 The Dynamic Vision Sensor

Recently, there has been much progress in imaging sensor technology, offering alternative solutions to scene perception. A neuromorphic imaging device, called

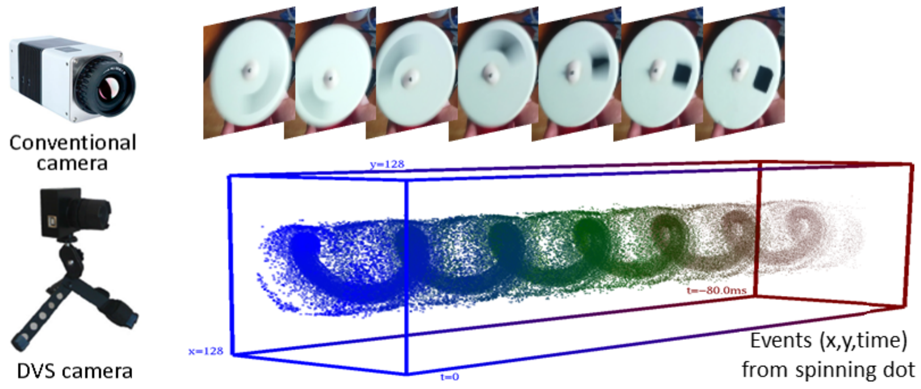


Figure 1.1: In contrast to conventional cameras which record image frames, DVS cameras record a stream of events, measured by log intensity changes of each pixel. (Source: http://www.umiacs.umd.edu/research/POETICON/DVSContours/)

Dynamic Vision Sensor (DVS) [2,3], inspired by the transient pathway of mammalian vision, can offer exciting alternatives for visual motion perception.

DVS cameras does not record image frames, but asynchronous temporal changes in the scene in form of a continuous stream of events, each of which is generated when a given pixel detects a change in log light intensity (Fig. 1.1). This allows the sensor to literally *see the motion* in the scene and makes it indispensable for motion processing and segmentation. The unique properties of event-based sensors - high dynamic range, high temporal resolution, low latency and high bandwidth allow these devices to function in the most challenging lighting conditions (such as almost complete darkness), while consuming an extremely small amount of power.

By its design, this sensor provides ideal properties for applications where high quality motion estimation and tolerance towards challenging lighting conditions are desirable. The DVS offers new opportunities for robust visual perception so much needed in autonomous robotics, but challenges associated with the sensor output, such as high noise, relatively low spatial resolution and sparsity, ask for different visual processing approaches.

1.2 Structure from Motion

On the algorithmic side, the estimation of 3D motion and scene geometry has been of great interest in Computer Vision and Robotics for quite a long time, but most works considered a scene to be static. Earlier classical algorithms studied the problem of Structure from Motion (SfM) [4] to develop "scene independent" constraints (e.g. the epipolar constraint or depth positivity constraint) and estimate 3D motion from image to facilitate subsequent scene reconstruction. In recent years, most works have adopted the SLAM philosophy [1], where depth, 3D motion and image measurements are estimated together using iterative probabilistic approaches, usually initialized with SfM estimates. Such reconstruction approaches are known to be computationally heavy and often fail in the presence of outliers.

To move away from the restrictions imposed by the classical visual geometry approaches, Computer Vision and Robotics community started to lean towards learning. Yet, while the problem of detecting moving objects has been studied both in the model-based and learning-based formulation, estimating object motion in addition to spatio-temporal scene reconstruction is still largely unexplored.

1.3 What is an Artificial Neural Network?

Following inspirations from neuroscience, researchers have constructed artificial neural networks as a composition of linear and non-linear mappings.

$$output = f_n \circ W_n \circ f_{n-1} \circ W_{n-1} ... f_1 \circ W_1 \circ input \tag{1.1}$$

A loss function is then calculated based on the output.

As an example, here the input can be a 1D vector, W_i's are linear transform matrices. As a special case, the W_i's can be convolution operations: note that a convolution operation can be written in Toeplitz matrix form.

f_i's are non-linear functions such:

$$sigmoid(x) = \frac{1}{1+e^{-x}}, \tag{1.2}$$

$$ReLU(x) = max(x, 0), \tag{1.3}$$

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{1.4}$$

.

The loss function can be the L_2 loss or cross entropy loss calculated with the output and some target values.

1.4 Why is it Hard to Train a Network?

At the time of this writing, neural networks are trained with gradient descent algorithms. The linear transform coefficients or network weights are adjusted in the negative gradient direction in order to minimize the Loss.

$$\frac{\partial Loss}{\partial W_i} = \frac{\partial Loss}{\partial y_n} \circ \frac{\partial y_n}{\partial y_{n-1}} \circ ... \circ \frac{\partial y_{i+1}}{\partial y_i} \circ \frac{\partial y_{i+1}}{\partial y_i} \circ \frac{\partial y_i}{\partial W_i} \tag{1.5}$$

Here let y_i be the output of layer i: $y_i = f_i \circ W_i \circ f_{i-1} \circ W_{i-1} ... f_1 \circ W_1 \circ input$.

A basic analysis using linear algebra/operator theory tells us the above calculations leads to numerical difficulties. Let the operator norm/ largest singular value of $\frac{\partial y_j}{\partial y_{j-1}}$ be σ_j. If $\sigma_j < 1$ for all layers, then $\frac{\partial Loss}{\partial W_i} \to 0, i \to 1$. This is the well-known vanishing gradient problem known in neural network community for decades. Because the gradient for shallow layers vanish exponentially, an local-optimal solution can not be found in reasonable finite time.

1.5 A Missing Piece

In the last chapter of this thesis, we will introduce a novel and simple fix to alleviate this issue. The motivations are briefly explained here.

We introduce a whitening transform that stretches the y_i's so that each dimension of the vector is independent and identically distributed (i.i.d) in the statistical sense. Given a batch of vectors, we calculate the mean subtracted vectors, then calculate the covariance matrix of the dimensions. Then we calculate an approximated inverse square root of the covariance matrix and multiply with the mean subtracted data to whiten the y_i's. Let this whitening/decorrelation transform be D_i, our network becomes:

$$output = f_n \circ W_n \circ D_{n-1} \circ f_{n-1} \circ W_{n-1} \circ D_{n-2} \circ ... \circ D_1 \circ f_1 \circ W_1 \circ D_0 \circ input \tag{1.6}$$

We investigate the properties of the whitening transform. For simplicity, we assume the activation function is $ReLU$. Note that this function, is multiplying the input vector by a diagonal matrix with entries either 1 or 0, based on the sign of the input. On a batch of data, the average effect of $ReLU$ is an attenuating effect and we assume the operator norm of it is smaller than 1. We denote the average $ReLU$ to be a new linear transform $\overline{ReLU}$. $\overline{ReLU}(x) = ax$ where $a < 1$ is a scalar number.

In each layer, the transform is:

$$y_i = D_i \circ ReLU \circ W_i \circ x_i \tag{1.7}$$

, here x_i is the input to the linear layer.

On a batch of data, the transform is approximated as a sequence of linear transforms:

$$y_i = D_i \circ a \circ W_i \circ x_i \tag{1.8}$$

.

If we assume the output dimension is the same as the input dimension, and if both x_i and y_i are i.i.d., then $D_i \circ a \circ W_i$ is having the effect of a *rotation*, which keeps the statistical properties of the signal unchanged both in forward and backward propagation. To emphasize on the back propagation: if $a \circ W_i$ has a diminishing effect on the gradients, then D_i has a raising effect to counteract with it.

$$\frac{\partial y_i}{\partial x_i} = D_i \circ \frac{\partial f_i}{\partial (W_i \circ x_i)} \circ \frac{\partial (W_i \circ x_i)}{\partial x_i} = D_i \circ f_i' \circ W_i = D_i \circ (aW_i) = Rotation \tag{1.9}$$

It is important to note that the procedure does not change the learning problem, as the standard network training is solving for $W_i \circ D_{i-1}$. Instead, this procedure removes the correlations between feature channels and improves the conditioning of the optimization problem.

To further explain the favorable property of the whitening transform, assume we are given a linear regression problem with L_2/min squared loss:

$$y = Xw, \tag{1.10}$$

$$loss = \frac{1}{2}\|y - \hat{y}\|^2 = \frac{1}{2}\|Xw - \hat{y}\|^2. \tag{1.11}$$

For an explicit solution we have $\frac{\partial loss}{\partial w} = X^t(Xw - \hat{y})$=0.

$$w = (X^tX)^{-1}X^t\hat{y} \tag{1.12}$$

. If the input is whitened, we have $w = X^t\hat{y}$. On the other hand, to conduct one iteration of gradient descent: we have $w_{new} = w_{old} - step\_len \times (X^tXw_{old} - X^t\hat{y})$. If $X^tX = I$ then $step\_len = 1$ is optimal and with one iteration we have $w_{new} = X^t\hat{y}$.

1.6 Inspirations from Human Learning

How do humans come to understand their world? Consider the learning of mathematics - we learn basic mathematics before proceeding to advanced mathematics, the understanding of basic math serving as the foundation on which to develop an understanding of advanced math(Fig. 1.2) . Once an advanced understanding of math is acquired, it in turn is used to better understand basic math, reinforcing an understanding of mathematics as a whole. Similarly in perception - higher levels of abstraction within our perceptual hierarchies are built upon the lower levels of abstraction. Additionally, using higher level interpretations to reinforce lower level interpretations can provide benefits to perception as a whole, in the same way that using a more expansive understanding of mathematics to understand elementary mathematics reinforces an understanding of mathematics as a whole.

In this thesis we will show how the intuition can be utilized to construct an artificial neural network (NN). We show favorable results in using variants of the network to address several computer vision tasks.

1.7 The Organization of the Theis

Following the human learning process, we develop the encoding part of ECN, in chapter 3 we introduce a novel neural network architecture - the Evenly-Cascaded convolutional Network (ECN) and show its favorable properties in image classification tasks. In chapter 4, we extend the network to an encoder-decoder architecture to address the problems of event data sparsity for depth, optical flow and egomotion estimation in a self driving car setting. Despite having just 150k parameters, our network is able to generalize well to different types of sequences. In chapter 5 we present a compositional neural network, which provides supervised up-to-scale depth and pixel-wise motion segmentation of the scene, as well as *unsupervised* 6 dof egomotion estimation and a per-segment linear velocity estimation using only monocular event data. This pipeline can be used in indoor scenarios for motion

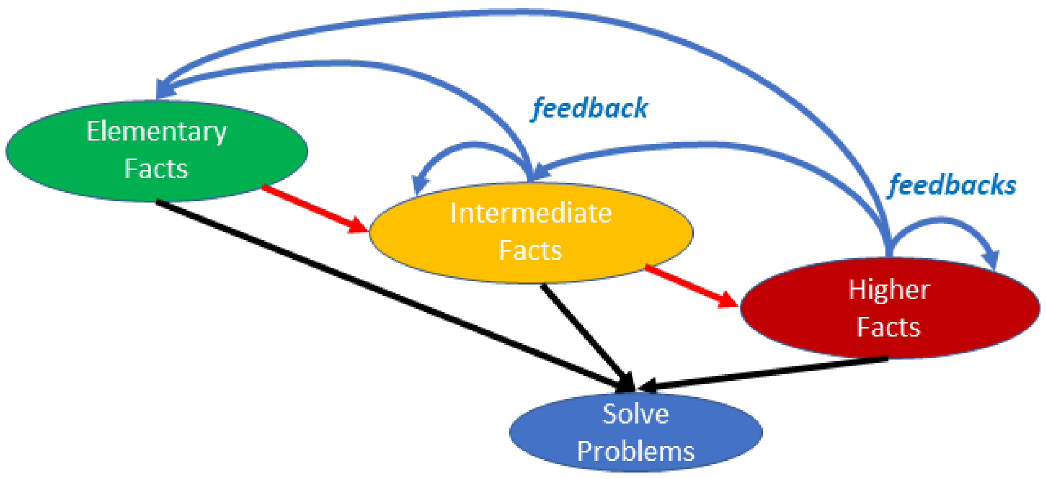


Figure 1.2: The human learning process.

estimation and obstacle avoidance. Finally, in chapter 6 we conclude the thesis with an improved way to train convolutional networks.

Chapter 2: Related Work

This chapter reviews related work of the learning tasks we addressed.

2.1 Network Architectures

Even though convolutional networks have led to many exciting breakthroughs in visual and language learning tasks [5], the mathematical understanding of convolutional networks is severely underdeveloped. It is intuitively clear that convolutional networks can be understood in part within the context of scale-space theory or multiresolution analysis [6]. While an excellent attempt has been made [7] to explain convolutional networks in wavelet terms, it remains unclear how the techniques in wavelet theory can be explicitly used to simplify the construction and training of convolutional networks.

Skip connections or identity maps in the networks [8–11] are a popular method for improving network performance. One intuitive understanding of the function of skip connections is that they pass lower level representations to deeper levels. In LSTM [8] and ResNet [9] networks the representations are adapted during this process. Whereas in DenseNet [10] shallow features are passed to the deeper levels without modification. One recent work, termed Dual Path Networks(DPN) [11], connects the two approaches using a high order recurrent neural network, and horizontally concatenates ResNet [9] and DensNet [10], and shows improved performance.

Presently, the best performing networks are typically 50 to 100 layers deep [9, 10, 12]. Translating an existing backbone architecture to a new application with different input and output sizes is a nontrivial task. The structuring of these standard designs are constrained by integer resizing operations such as pooling and strided convolution. Fractional pooling and strides have been explored previously, e.g. [13]. In our implementation of fractional pooling we employ bilinear interpolation, as is done in [14]. Furthermore, existing architectures merge different levels of features at the end of each block such that low- and high-level features become entangled and in-differentiable. ECN safely removes these limitations.

As the networks become wider and deeper, methods such as pruning [15–17], quantizing [18], and knowledge distillation [19] have been introduced to reduce the size of these networks. ECN uses recursion [20, 21] to re-use parameters across layers, allowing for greater compactness of network design. In the other direction, significant efforts have been spent on discovering more efficient network layers or overall architectures [10, 22–26].

2.2 Event-based Depth Estimation

The majority of event-based depth estimation methods [27, 28] use two or more event cameras. As our proposed approach uses only one event camera, we focus our discussion on monocular depth estimation methods. The first works on event-based monocular depth estimation were presented in [29] and [30]. Rebecq *et al.* [29] used a space-sweep voting mechanism and maximization strategy to estimate semi-dense depth maps where the trajectory is known. Kim *et al.* [30] used probabilistic filters to jointly estimate the motion of the event camera, a 3D map of the scene, and the intensity image. More recently, Gallego *et al.* [31] proposed a unified framework for joint estimation of depth, motion and optical flow. So far there has been no deep learning framework to predict depths from a monocular event camera.

2.3 Event-based Optical Flow

Previous approaches to image motion estimation used local information in event-space. The different methods adapt in smart ways one of the three principles known from frame-based vision, namely correlation [32, 33], gradient [34] and local frequency estimation [35] The most popular approaches are gradient based - namely, to fit local planes to event clouds [36] As discussed in [37], local event information is inherently ambiguous. To resolve the ambiguity Barranco et al. [37] proposed to collect events over a longer time intervals and compute the motion from the trace of events at contours.

Recently, neural network approaches have shown promising results in various estimation problems without explicit feature engineering. Orchard and Etienne-Cummings [38] used a spiking neural network to estimate flow. Most recently, Zhu *et al.* [39] released the MVSEC dataset [40] and proposed self-supervised learning algorithm to estimate optical flow. Unlike [39], which uses grayscale information as a supervision signal, our proposed framework uses only events and thus can work in challenging lighting conditions.

2.4 Independent Motion Detection

Many motion segmentation methods used in video applications are based on 2D measurements only [41, 42]. 3D approaches, such as the one here, model the camera's rigid motion. Thompson and Pong [43] first suggested detecting moving objects by checking contradictions to the epipolar constraint. Vidal *et al.* [44] introduced the concept of subspace constraints for segmenting multiple objects. A good motion segmentation requires both constraints imposed by the camera motion and some form of scene constraints for clustering into regions. The latter can be achieved using approximate models of the rigid flow or the scene in view, for example by modeling the scene as planar, fitting multiple planes using the plane plus parallax constraint [45], or selecting models depending on the scene complexity [46]. In addition constraints on the occlusion regions [47] and discontinuities [48] have been used. Recently, machine learning techniques have been used for motion segmentation [49, 50]. As discussed next, the well-known SfM learner acquires both, the depth map and the rigid camera motion, and thus the flow due to rigid motion is fully constrained.

2.5 Learning in Structure from Motion

In pioneering work, Saxena *et al.* [51] demonstrated that shape can be learned from single images, inspiring many other supervised depth learning approaches (e.g. [52]). The concept was recently adopted in the SfM pipeline, and used in stereo [53] and video [54]. Most recently, Zhou *et al.* [55] took it a step further, and showed how to estimate 3D motion and depth through the supervision of optical flow. Wang *et al.* [56] instead of predicting depth in a separate network component propose to incorporate a Direct Visual Odometry (DVO) pose predictor. Mahjourian *et al.* [57] in addition to image alignment enforce alignment of the geometric scene structure in the loss function. Yang *et al.* [58] added a 3D smoothness prior to the pipeline, which enables joint estimation of edges and 3D scene. Yin *et al.* [59] include a non-rigid motion localization component to also detect moving objects. Our architecture is most closely related to SfM-Net [60], which learns using supervised and non-supervised components depth and 3D camera and object motion. However, due to the lack of a dataset, the authors did not evaluate the object motion estimation. Our work detects, segments, and estimates the 3D motion of independently moving objects, and provides the means for evaluation.

Chapter 3: Evenly Cascaded Convolutional Networks

Disregarding the interplay between levels of abstraction in perception, the recent trend in deep neural networks is to simply make networks deeper. Since networks have become so deep, researchers have developed a now standard design that divides network layers into blocks of layers [5, 9, 10, 61]. Each block consists of layers of transforms that produce feature maps of the same shape. Cross-block transforms recombine the previous features and incorporate strided convolutions or pooling operations to reshape the feature maps. This strategy is an extension of the traditional design of the multilayer perceptron or fully-connected network [62]. The introduction of blocks has allowed a better organization of the evolving layers of abstraction within those networks. Overall, more high level features are abstracted through these transforms (Fig. 3.1, 3.3). However, the unstructured recombination of features in existing networks has made the investigation of deep neural networks nontrivial [7, 63].

Aside from deviating from basic intuitions about learning and perception, the now standard design has other apparent shortcomings. Since strided convolution or pooling are used to resize the feature maps by integer scales, researchers are compelled to create ad hoc network shapes, ones which are subject to awkward

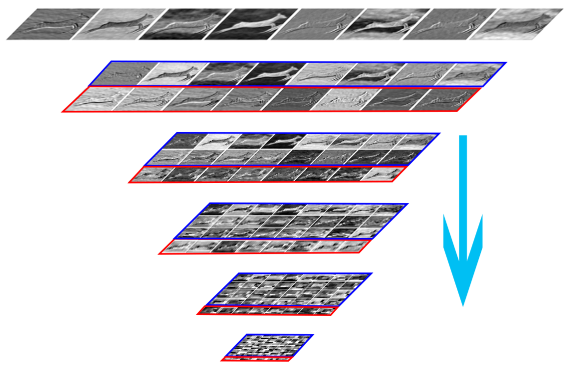


Figure 3.1: Our evenly cascaded design with 6 cascading layers using a scaling rate of 0.75. The growth rate (the number of high level feature channels generated in each subsequent layer) is set to 8. From the second layer on, the low level feature channels are framed in blue boxes, the newly generated high level feature channels are framed in red boxes.

constraints of integer pooling and strides. Some networks assign more layers in later stages, where the feature maps are very small [9]. Some other networks are wider but have fewer layers [64]. The specification of feature map dimensions or distribution of computational resources is highly engineered and uneven in these designs. It is difficult to determine which designs outperform others, given that the overall shape is different. To make things worse, when the task changes, the network shape needs to be handcrafted again.

Arguably the biggest shortcoming of existing designs is that they afford little intuition on the training process, and consequently make deep networks notoriously hard to train. Researchers have expended significant effort in developing better methods to train these networks.

In this chapter we introduce an architecture more closely in line with intuitions of biological learning and perception: Evenly Cascaded convolutional Network (ECN). ECN is 1) easier than existing architectures to adapt to new tasks, 2) produces internal representations which are more humanly interpretable, 3) performs robustly as parameter count is restricted, and 4) produces competitive performance when compared to other state-of-the-art methods.

ECN is structured around the insights that 1) maintaining low-level features through to the upper layers of a network is beneficial, 2) allowing multiple levels of features to interact with each other within the network is beneficial, 3) abrupt changes in feature map dimension could be less ideal than gradual changes, and 4) the manner in which features are conventionally combined within network blocks hampers the preservation of low-level features. Our architecture instantiates the first two of these insights through a "cascade" architecture - a two stream architecture, one stream for low-level features, one stream for successively higher level features, where these two streams interact at every layer. This differs from the existing method of using skip connections to introduce low-level features into upper level network layers in that low level features are maintained, and modulated appropriately, rather than simply jumping layers [10]. This approach differs from a conventional two-stream approach [65] in that both streams interact with each other. For the third insight, in both streams of our cascade architecture bilinear interpolation is employed for fractional pooling, allowing a gradual rather than abrupt decrease of feature map dimensions. Different from existing architectures, in ECN a scaling factor is introduced which simplifies the design of the shape of the network removing the need for handcrafting network shape. For the fourth insight we

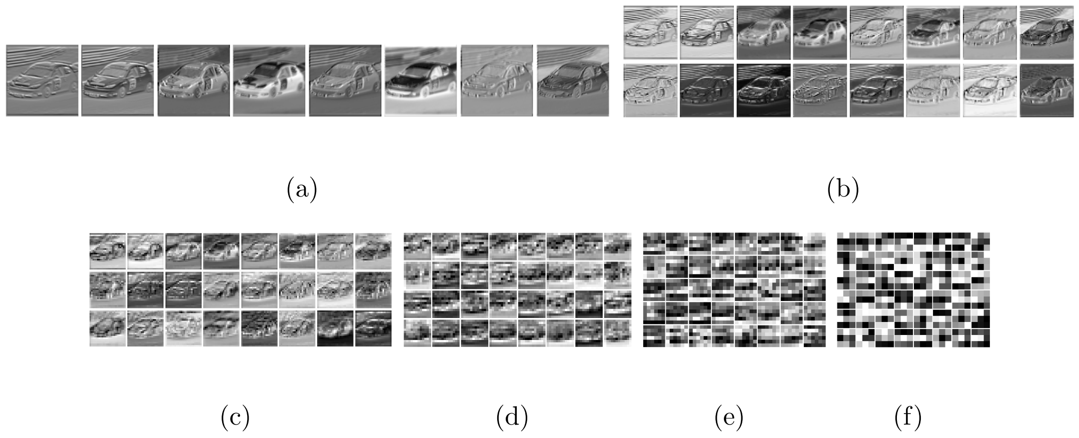


Figure 3.2: Visualizations of feature maps in ECN. The underlying network has 8 feature maps in the first hidden layer. The feature channel growth rate is set to 8. In this 6-hidden-layer network there are $8, 16, 24, 32, 40, 48$ feature maps in layers 1-6 respectively. In each deeper layer the newly generated higher level feature maps are appended as a new row. (a) Level 1 feature maps in the first hidden layer. (b) Level 1 features are adapted and tiled in the first row, level 2 features are newly generated using level 1 features, and are appended in the second row. (c) Both level 1 and level 2 features are adapted and tiled in the first two rows. Level 3 features are newly generated from level 1 and level 2 features, and are appended as the third row. (d) Level 4 features. (e) Level 5 features. (f) Level 6 features are used for recognition. The feature maps have been rescaled for better visualization.

remove the conventional combination of features across blocks in neural networks in order to preserve the low-level signal in ECN's two-stream cascade architecture. We demonstrate that preserving multilevel features enhances training by providing easy-to-train shortcuts.

Our evaluation of ECN resulted in several intriguing results: 1) With ECN's principled structuring, shallow networks (Fig. 3.1) seem to perform competitively well when compared to extremely deep networks. 2) Complex high level tasks such as image classification can be approached through evenly downsampling and adapting a set of highly structured features (Fig. 3.2). 3) Low level features may be of critical importance in high level tasks such as image classification: not only do these features remain similar in the deeper adaptation process, but they may have provided major convenience for the training of high level features.

Finally, we evaluate ECN using multiple convolution block designs and find that recurrent and recursive designs lead to improved efficiency and accuracy. Without additional treatment, a standard convolution block design combined with recurrent connections leads to state-of-the-art accuracy in benchmark image classification tasks. Another block design using a recursive filter gives rise to state-of-the-art efficiency. Our 6-cascading-layer design with under 500k parameters achieves 95.24% and 78.99% accuracy on CIFAR-10 and CIFAR-100 datasets, respectively, outperforming the current state-of-the-art on small parameter networks, and our 3 million parameter version is competitive to the state-of-the-art.

3.1 Methods

ECN is based on a simple cascading layer design. Intuitively, a cascading layer incorporates multilevel features and gradually resizes those features before providing them to the next layer. ECN consists of iteratively stacked cascading layers. This iterative construction process ends when a stopping condition is met - e.g., when feature map dimensions fall below a predefined size.

3.1.1 Utilization of Multilevel Features

We now follow the intuition that allowing an interplay between level of abstraction within perception can give rise to greater perceptual synergy in introducing ECN's usage of multilevel features. To illustrate ECN's relation to work in the literature we also provide three additional viewpoints of our utilization of multilevel features: one from the network architecture, one from wavelet analysis, and one from optimization of neural networks.

3.1.1.1 From a Network Architectural Point of View

Our cascading design can be viewed as evolving from the designs of several well-performing networks [9–11]. Our first design can be viewed as a ResNet-inspired DenseNet, or a DenseNet-inspired ResNet. We make an extension to DenseNet [10] when passing low level features to deeper layers by allowing them to be modulated by higher level features. The modulation of low level features can be viewed as a *feedback* mechanism, where high level knowledge is providing "advanced viewpoints"

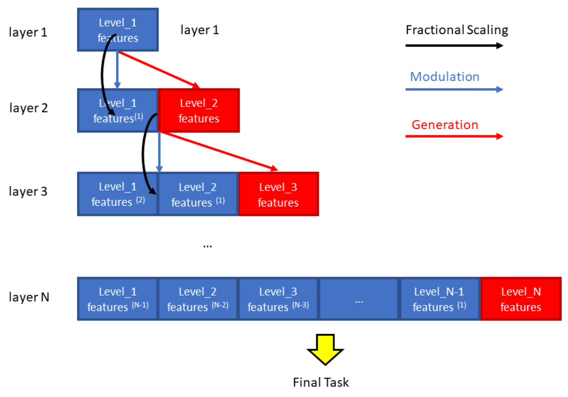


Figure 3.3: Multilevel feature representations within ECN. For the i-th layer (i-th row in the figure), fractional scaling is used to downsample the features from layer $i-1$ (row $i-1$ in the figure). A convolution block is applied over these downsampled features to generate the modulation signal, and the level i features (red block). The modulation signal is then added to the features downsampled from the previous layer. The superscripts in each block represent the number of modulations the features have undergone.

to improve the low level knowledge, effected through a link from the higher level stream to the lower level stream of the network. The modulation signal is generated using a convolution block, and is added to the original features as in residual learning [9]. The generation of high level features is also achieved using the convolution block - however, the results are appended to the existing channels (Figs. 3.1, 3.3). Similar to our design, but more costly, Dual Path Network [11] concatenates ResNet and DenseNet layers. By not passing low-level features through a 1×1 convolution, such as is done in DPN, ECN better preserves low-level features, making them more directly available at higher layers.

As a consequence of this ECN is able to pass features throughout the *whole network*, rather than only within each *block*. With this design we explicitly enforce that the shallowest level features be preserved throughout the whole network, with adaptations made only if they improve the final task (Figs. 3.1, 3.2).

3.1.1.2 From a Wavelet Point of View

It is noteworthy that the cascading layer is a more general form of the cascade algorithm used in wavelet packet decomposition [6, 65], where previous level signals are decomposed into a low frequency branch and a high frequency branch, usually followed by downsampling by a factor of 2. In wavelet packet decomposition, the original signal is decomposed into a binary tree. The difference between this and ECN is that in ECN the two siblings are merged before the next level of decomposition, deviating from a tree structure. The extensions we made in our cascading design derive from the adaptations and downsampling of features which we introduce later. Due to the cascading design, the evolution of feature maps within ECN closely resembles the evolution of modulus maxima in scale-space theory [6]. This relation suggests a path for further mathematical investigation of neural networks.

3.1.1.3 From an Optimization Point of View

ECN's construction facilitates training by providing easily trainable shortcuts to the optimization: we speculate that as a consequence of the two stream architecture, ECN allows the decomposition of f, the mapping from network input to output, into a series of progressively hierarchically deeper, and therefore harder to train, functions: $f = f_1 + f_2 + ... + f_N$. See figure 3.3 for a visualization of function decomposition within and across layers. In figure 3.3 layer N contains features of levels 1 through N - this provides a direct link from the training (gradient) signal not only to higher level features, as is the case in other architectures, but to mid- and low-level features as well. This relaxes the interdependence in training between f_i (low f_i can be trained with less dependence on high f_i), and has the potential to facilitate training by allowing a training of f component-wise. The result of this is the capability to train a complex function, f, by training the simpler functions of which it is composed, f_i.

Decompositions analogous to this kind have been proven to be helpful in multiscale signal analysis. Complex operations are simplified by conducting them at multiple scales, either (1) from coarse-to-fine or (2) in parallel. Here, we take the second approach to train different levels in parallel: the easier to train, or coarse, components can serve as a backbone model for the harder components, where the harder-to-train layers have only to compensate for the residuals of the learning problem. We conjecture that ECN's construction facilitates the progressive training of neural networks. Note that ECN's construction is a revival of the layer-wise pretraining technique [66] which triggered the era of deep learning. This classic approach followed the correct intuition of progressively developing high level representations. But the layer-wise training, which belongs to the first approach, lacks proper supervision signal.

3.1.2 Fractional Scaling

To facilitate the passing of low level features throughout the whole network, and to evenly resize the feature maps in a network, we propose to use fractional scaling, performed using bilinear interpolation, replacing the classic integer pooling and striding operations. Bilinear interpolation was first introduced in Spatial Transformer Networks [14] to continuously deform feature maps for recognition tasks. Here we use bilinear interpolation to replace all the size change operations in the network.

Using bilinear interpolation, the outputs of the previous layer are fractionally scaled to the desired shape, and then serve as inputs to the next layer. Since bilinear interpolation is a locally smooth operation, subgradients can be calculated for backpropagation. In a network with a decreasing feature map dimension, having a constant scaling factor close to 1 leads to a deep network. Similarly, having a constant scaling factor close to 0 leads to a shallow network.

We adopt a simple uniform sampling when scaling the feature maps for obtaining the sample grid for bilinear interpolation. Non-uniform sampling is a natural extension [67] and we leave it for future work. Here, the output feature map dimension can be either calculated from the scaling factor or manually specified. The uniformly spaced sampling grid is then calculated to sample from the previous feature map.

3.1.3 Convolution Block Design

Many researchers have proposed different convolution block designs for use within layers of convolutional networks. However, in most cases different block designs are associated with different network architectures. As a consequence, it is difficult to draw conclusions about the relative merits of different block designs. However, with ECN's evenly cascaded design, it is straightforward to incorporate different block designs. In this chapter we propose and evaluate multiple block designs. More advanced block designs than are presented here can be evaluated in subsequent works and potentially result in improved performance. In this chapter we include six basic blocks:

Block 1: Single convolution (Fig. 3.4(a)). The convolutional layer has one single convolution operation, combined with standard techniques such as batch normalization [68] and the ReLU activation function [5]. We order these operations as

$BN \rightarrow ReLU \rightarrow Conv$.

Block 2: Double convolution (Fig. 3.4(b)). Two single convolution layers are chained together, the number of intermediate output channels is set to be the larger of the input and output channels.

Block 3: Recurrent convolution (Fig. 3.4(c)). We iteratively reuse the weights in Block 1 through recurrent connections. We made alterations to the previously reported approach [20]. Firstly, the number of input channels do not usually match the number of output channels. Here we propose a simple solution: when the output channels (OUT) are more than the input channels (IN), we *slice* the matching channels from the output (e.g. the first IN channels) to reuse as input. Similar to an $LSTM$ [8] we add the outputs of the later iterations to the initial outputs. It is important to point out that our implementations of this recurrent design lead to worse results in ECN. The output signals usually have very different statistical properties from the input signals, and this inconsistency interfered with training. Our solution to this is that we insert an individual batch normalization operation for each iteration, leaving the convolution kernel weights shared in all iterations. This design significantly improves performance with very little increase in number of parameters (Tables 3.6, 3.7, Fig. 3.5).

Block 4: Recurrent double convolution (Fig. 3.4(d)). Similar to Block 3 we iteratively reuse Block 2.

Block 5: Recursive convolution (Fig. 3.4(e)). To make the convolutions more efficient we adopt separable convolution operations [23] to replace standard convolution. Cross-channel 1×1 convolution is applied in the first stage to change the

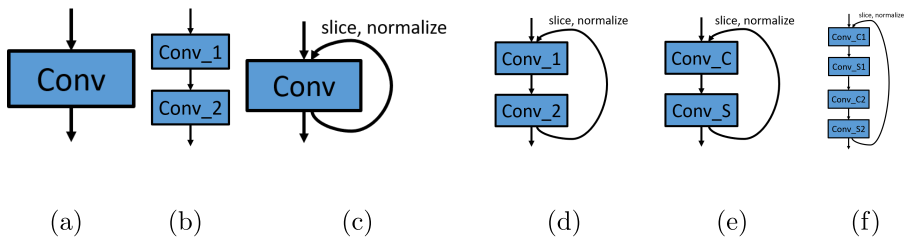


Figure 3.4: Evaluation of various convolution blocks: (a) Single convolution, (b) Double convolution, (c) Recurrent convolution, (d) Recurrent double convolution, (e) Recursive convolution. $Conv\_C$ stands for cross-channel 1×1 convolution, $Conv\_S$ stands for channel-wise spatial convolution. (f) Recursive quadruple convolution.

number of input channels to match the number of output channels, followed by the channel-wise convolution in the second stage. Compared to standard convolution which involves more parameters, separable convolutions are usually more efficient but are weaker than directly applying standard convolutions. One fix for this weakness is to iteratively apply the filtering as in Blocks 3 and 4. Traditionally this technique is called recursive filtering. We adopt this name and include this class of filtering in our comparison.

Block 6: Recursive quadruple convolution (Fig. 3.4(f)). Each recurrent double convolution (Block 4) is replaced by four separable convolutions.

Table 3.1: CIFAR accuracies for ECN-6

Conv Block	**Initial Channels**	**Scaling**	**CIFAR-10**	**Parameters**	**CIFAR-100**	**Parameters**
type 4	16	3/4	93.49%	214330	68.15%	220180
type 4	32	3/4	95.47%	849130	76.05%	860740
type 4	64	3/4	96.20%	3380170	79.80%	3403300
type 4	128	3/4	96.68%	13488010	81.75%	13534180
type 6	64	3/4	95.24%	421770	78.99%	444900

Table 3.2: ECN-6 architecture

ECN-6
3x3 Conv
$\begin{bmatrix} CascadedConvBlock \\ FractionalScaling \end{bmatrix} \times 6$
global average pooling
Softmax

3.2 Experiments

3.2.1 Datasets

We evaluate ECN over three datasets: CIFAR-10, CIFAR-100 [69], and ImageNet-32 [70]. These datasets consist of 32x32 pixel images. CIFAR-10 has 10 classes, CIFAR-100 has 100 classes, and ImageNet-32 has 1000 classes. We employ standard methods of data augmentation, including horizontal image flips, and random

32x32 crops of zero padded images, with 4 pixel padding. CIFAR-10 and CIFAR-100 each contains 50,000 training samples, and 10,000 testing samples. ImageNet-32 contains images of ImageNet [71], downsampled to 32x32 pixels; it contains 1.2 million training samples, and 50 thousand validation samples.

3.2.2 Results

The ECN network we use has a shape where feature map dimensions consistently decrease in size; it is constructed by iteratively stacking cascading layers until the feature map size is below a preset threshold (4 pixels). In our experiments we fix the number of iterations in Blocks 3, 4, 5, and 6 to 3.

In table 3.1 we employed a scaling factor of $\frac{3}{4}$, resulting in an evenly cascaded structure with 6 cascading layers (ECN-6). We report results for differently scaled structures using block 4, and one more result using a more efficient design using block 6. At the beginning of the network, a convolution is used to transform the channel count of the input to $init\_channels$, which takes the value 16, 32, 64, or 128. In each consecutive layer, we generate channels of high level features with a growth rate of $init\_channels \times 2 \times (1 - scaling\_factor)$, corresponding to 8, 16, 32, 64 respectively in the four networks. The overall network architecture can be found in table 3.2. Here a cascaded convolution block [6] represents using one convolution block and grouping the results into a low level branch and a high level branch. Global average pooling is used to convert the final feature map into a vector for classification. To avoid overfitting we also insert dropout after ReLU activations for

Table 3.3: Results on the ImageNet-32 dataset, for WideResNet [64], and ECN (channel count)

ImageNet32	**Params**	**Top-1 error**	**Top-5 error**
WRN-28-1 [64]	0.44M	67.97%	42.49%
WRN-28-2 [64]	1.6M	56.92%	30.92%
WRN-28-5 [64]	9.5M	45.36%	21.36%
WRN-28-10 [64]	37.1M	40.96%	18.87%
ECN-6, block 6 (32)	0.24M	63.91%	38.50%
ECN-3, block 6 (64)	0.48M	59.05%	33.68%
ECN-6, block 6 (64)	0.68M	55.51%	30.26%
ECN-6, block 6 (128)	2.1M	46.29%	21.92%
ECN-3, block 4 (128)	7.8M	45.10%	21.06%
ECN-6, block 4 (128)	14.0M	41.87%	18.61%

the 3 largest networks in table 3.1. The dropout rates range from 0.03-0.25. We use stochastic gradient descent to train the network for 2000 epochs with a batch size of 512, for CIFAR10 and CIFAR100, and for 50 epochs and a batch size of 512 for ImageNet-32. The training is scheduled with an initial learning rate of 0.1 and followed by cosine annealing learning rates. The results can be found in table 3.1.

Our studies on ImageNet-32 demonstrate ECN's potential to generalize to larger scale datasets. We evaluate ECN-3 and ECN-6, corresponding to scaling rates of $\frac{1}{2}$ and $\frac{3}{4}$, over ImageNet-32. We use initial channel counts of 32, 64, and 128, and report results in table 3.2.2, with channel counts given in parentheses. ECN-

Table 3.4: Error rate comparison with state-of-the-art efficient architectures

Model	Params	CIFAR-10	CIFAR-100
VGG-16 pruned [16]	5.4M	6.60	25.28
VGG-19 pruned [17]	2.3M	6.20	-
VGG-19 pruned [17]	5M	-	26.52
Resnet-56 pruned [16]	.73M	6.94	-
Resnet-110 pruned [16]	1.68M	6.45	-
Resnet 164-B pruned [17]	1.21M	5.27	23.91
DenseNet-40-pruned [17]	.66M	5.19	25.28
CondenseNet-94 [25]	.33M	5.00	24.08
CondenseNet-86 [25]	.52M	5.00	23.64
ECN, Block 6	.42M	4.76	21.01

Table 3.5: Error rate comparison with state-of-the-art architectures

Model	Params	CIFAR-10	CIFAR-100
ResNet-1001 [72]	16.1M	4.62	22.71
Stochastic-Depth-1202 [73]	19.4M	4.91	-
Wide-ResNet-28 [64]	36.5M	4	19.25
ResNeXt-29 [74]	68.1M	3.58	19.25
DenseNet-BC-190 [10]	25.6M	3.46	17.18
NASNet-A* [25]	3.3M	3.41	-
CondenseNet*light-160 [25]	3.1M	3.46	17.55
CondenseNet-182 [25]	4.2M	3.76	18.47
ECN, Block4	3.3M	3.8	20.2
ECN, Block4	13.3M	3.32	18.25

6 is more efficient than the strong baseline results reported in WideResNet [64]. An ECN-6 network using only 2 million parameters shows comparable results to a WideResNet architecture using 9.5 million parameters. ECN with block 4 produces competitive results using smaller number of parameters than WideResNet. ECN-3, which has 3 cascading layers and 7 million parameters outperforms the 9.5 million parameter WRN-28-5 result. A larger ECN-6 network using block 4 with 14 million parameters achieves accuracy that is comparable to WRN-28-10, which contains 37.1 million parameters.

3.2.3 Comparison of Convolution Blocks over CIFAR10 and CIFAR100

We have compared the six block designs over CIFAR10 and CIFAR100 using various sized networks. The networks are trained for 500 epochs. We tested scaling factors $\frac{1}{2}$, $\frac{3}{4}$, and $\frac{7}{8}$, and the corresponding networks have $3, 6$, and 12 cascading layers. The growth rates are calculated using the same strategy as explained above. For these experiments we use 3-stage learning rate scheduling, decreasing the learning rate at 40% and 80% total epoch count by a factor of 10. We set batch size to 512 for CIFAR-10. For CIFAR-100 a batch size of 128 usually leads to better performance, and we report the better of size 128 and size 512 batches. The results over CIFAR-10 and CIFAR-100 can be found in tables 3.6 and 3.7, respectively.

By comparing Blocks 1 and Blocks 3 in tables 3.6 and 3.7, and Fig. 3.5, we found that reusing the convolution weights via recurrent connections significantly improves performance, while maintaining a small network size. When the convolution block becomes powerful, and especially when the model gets large, the improvement due to recurrent connections becomes smaller (Blocks 2 vs Blocks 4). Still, we find a surprise here that through recurrent connections even the single convolution can perform competitively with the widely used double convolution, using the same number of parameters (Fig. 3.5). The optimal balance between depth and width varies from block to block. Our most efficient convolution block is block 6, which uses recursive quadruple convolutions. We reach state-of-the-art efficiency and the best results are reported in the last row of table 3.1. It is noteworthy that although using separable convolution [23] reduces the number of parameters, the gain in efficiency also comes with a decrease in accuracy. The effective reduction in parameters enabled by using separable convolutions in ECN blocks 5 and 6 is around 2 fold to 4 fold.

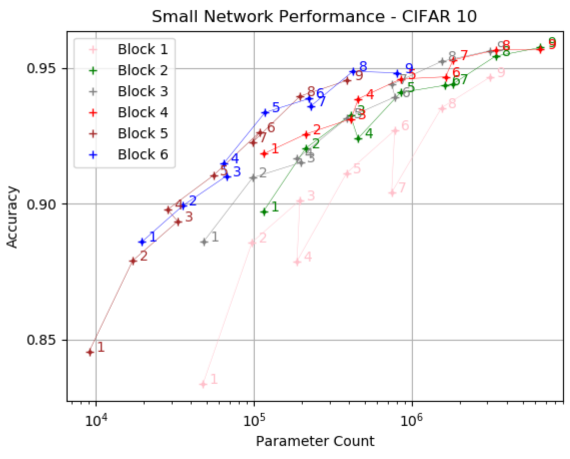
When compared to other state-of-the-art efficient architecture designs, listed in table 3.4, ECN using block 6 achieves the lowest error rate without using any pruning

Table 3.6: Comparison of convolution blocks on the CIFAR-10 dataset

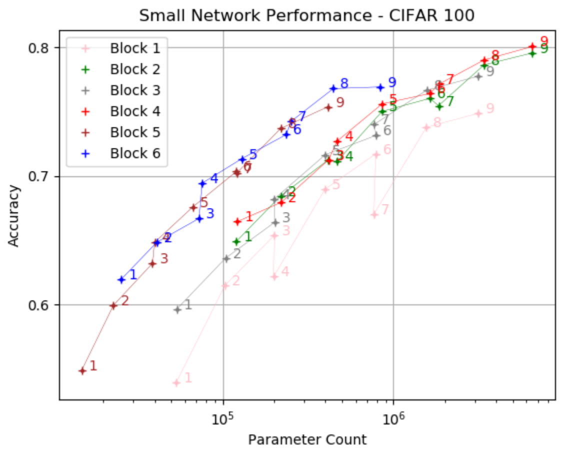
CIFAR-10		**Block 1**		**Block 2**		**Block 3**		**Block 4**		**Block 5**		**Block 6**	
Ch	**Sc**	**Acc**	**Params**	**Acc**	**Params**	**Acc**	**Params**	**Acc**	**Params**	**Acc**	**Params**	**Acc**	**Params**
16	1/2	83.35%	47482	89.71%	114586	88.59%	47866	91.84%	115546	84.56%	9066	88.59%	19514
16	3/4	88.58%	97258	92.04%	212410	90.96%	98122	92.57%	214330	87.90%	17090	89.92%	35370
16	7/8	90.12%	195082	93.26%	407194	91.51%	196906	93.12%	411034	89.35%	32946	91.02%	66986
32	1/2	87.88%	187114	92.40%	454954	91.66%	187882	93.86%	456874	89.78%	28362	91.47%	64106
32	3/4	91.13%	385738	94.09%	845290	93.15%	387466	94.59%	849130	91.04%	55418	93.36%	117450
32	7/8	92.70%	776074	94.36%	1622506	93.93%	779722	94.67%	1630186	92.64%	108762	93.87%	223754
64	1/2	90.41%	742858	94.41%	1813066	94.39%	744394	95.29%	1816906	92.25%	97674	93.59%	228554
64	3/4	93.53%	1536394	95.43%	3372490	95.24%	1539850	95.66%	3380170	93.96%	195818	94.87%	421770
64	7/8	94.65%	3095818	95.75%	6477514	95.63%	3103114	95.68%	6492874	94.54%	389034	94.82%	806666

methods. This is significant, as a simple and principled architecture design is proving to be better than sophisticated methods such as pruning described in [17], [16] and even better than [25] with smaller parameter count (Figs. 3.6, 3.7). On the other hand, ECN block 4 does relatively well compared to other architectures listed in table 3.5 that are using more advanced designs than ours.

We have shown that there are avenues for improving the performance of convolutional networks by using principled designs like ECN. Even the simplest designs can reach state-of-the-art performance. Due to limitations in space and computational resources, only the 6 basic block designs are evaluated. More advanced block designs can modularly replace our basic block designs and potentially produce even better numbers.



(a)



(b)

Figure 3.5: Relationship between network parameter count, block category, and classification accuracy on (a) CIFAR-10 and (b) CIFAR-100 datasets. The results are labeled according to the rows in which they appear in tables 3.6 and 3.7.

Table 3.7: Comparison of convolution blocks on the CIFAR-100 dataset

CIFAR-100		**Block 1**		**Block 2**		**Block 3**		**Block 4**		**Block 5**		**Block 6**	
Ch	**Sc**	**Acc**	**Params**	**Acc**	**Params**	**Acc**	**Params**	**Acc**	**Params**	**Acc**	**Params**	**Acc**	**Params**
16	1/2	53.93%	53332	64.93%	120436	59.64%	53716	66.45%	121396	54.88%	14916	61.95%	25364
16	3/4	61.50%	103108	68.45%	218260	63.62%	103972	67.95%	220180	59.96%	22940	64.86%	41220
16	7/8	65.40%	200932	71.23%	413044	66.43%	202756	71.26%	416884	63.21%	38796	66.71%	72836
32	1/2	62.17%	198724	71.12%	466564	68.19%	199492	72.72%	468484	64.85%	39972	69.44%	75716
32	3/4	68.98%	397348	75.02%	856900	71.66%	399076	75.62%	860740	67.61%	67028	71.34%	129060
32	7/8	71.72%	787684	76.04%	1634116	73.16%	791332	76.42%	1641796	70.36%	120372	73.22%	235364
64	1/2	67.02%	765988	75.47%	1836196	74.00%	767524	77.13%	1840036	70.23%	120804	74.25%	251684
64	3/4	73.79%	1559524	78.64%	3395620	76.67%	1562980	79.03%	3403300	73.70%	218948	76.80%	444900
64	7/8	74.87%	3118948	79.57%	6500644	77.76%	3126244	80.07%	6516004	75.38%	412164	76.93%	829796

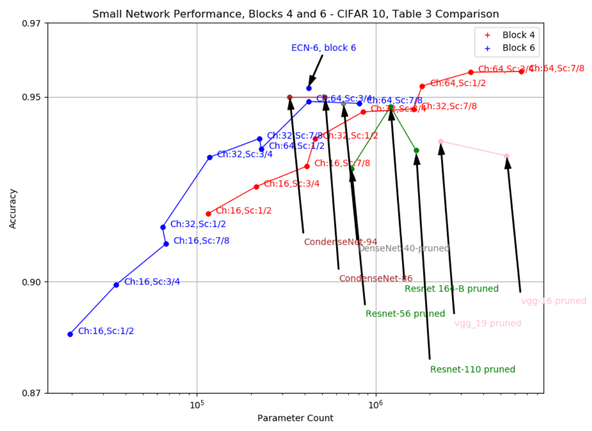


Figure 3.6: Comparison between parameter count and classification accuracy over CIFAR-10 for ECN with blocks 4 and 6. Results are based on architectures listed in table 3.4. Additionally, our best result using ECN-6 with block 6, attained through longer training, is plotted as "ECN-6, block 6".

3.3 Conclusion

Taking inspiration from cascading methods in wavelet packet decomposition, we have developed Evenly Cascaded convolutional Networks (ECN) for image tasks.

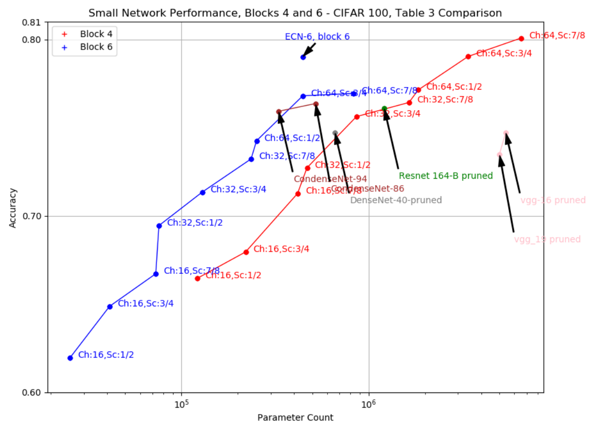


Figure 3.7: Comparison between parameter count and classification accuracy over CIFAR-100 for ECN with blocks 4 and 6. Results are based on architectures listed in table 3.4. Additionally, our best result using ECN-6 with block 6, attained through longer training, is plotted as “ECN-6, block 6”.

ECN differs from other networks in the use two interacting streams - a high-level feature stream and a low-level feature stream. ECN’s two streams allow for the promulgation of low-level features throughout the entire network, as well as the modulation of those low-level features using advanced perspectives from high-level features. The explicit use of multilevel features not only leads to highly capable networks but provides shortcuts for the training process. Additionally, ECN is structured such that feature map dimensions decrease in a consistent manner, removing burdens of ad hoc architecture design, and potentially improving feature preservation and utility. We have evaluated ECN over CIFAR-10 and CIFAR-100, obtaining state-of-the-art performance, for both datasets, for small network settings; and over ImageNet-32 ECN obtains competitive results.

Chapter 4: Unsupervised Learning of Dense Optical Flow, Depth and Egomotion from Sparse Event Data

In this work we introduce a novel lightweight encoding-decoding neural network architecture - the Evenly-Cascaded convolutional Network (ECN) to address the problems of event data sparsity for depth, optical flow and egomotion estimation in a self driving car setting. Despite having just 150k parameters, our network is able to generalize well to different types of sequences. The simple nature of our pipeline allows it to run at more than 250 inferences per second on a single NVIDIA 1080 Ti GPU. We perform ablation studies using the SfMlearner architecture [55] as a baseline and evaluate different normalization techniques (including our novel *feature decorrelation*) to show that our model is well suited for event data.

We demonstrate superior generalization to low-light scenes. Fig. 4.1 shows an example featuring night driving - the network trained on a day light scene was able to predict both depth and flow even with a low event rate and abundance of noise. This is facilitated by our event-image representation: instead of the latest event timestamps, we use the average timestamp of the events generated at a given pixel. The averaging helps to reduce the noise without losing the timestamp information. Moreover, we use multiple slices as input to our model to better preserve the 3D

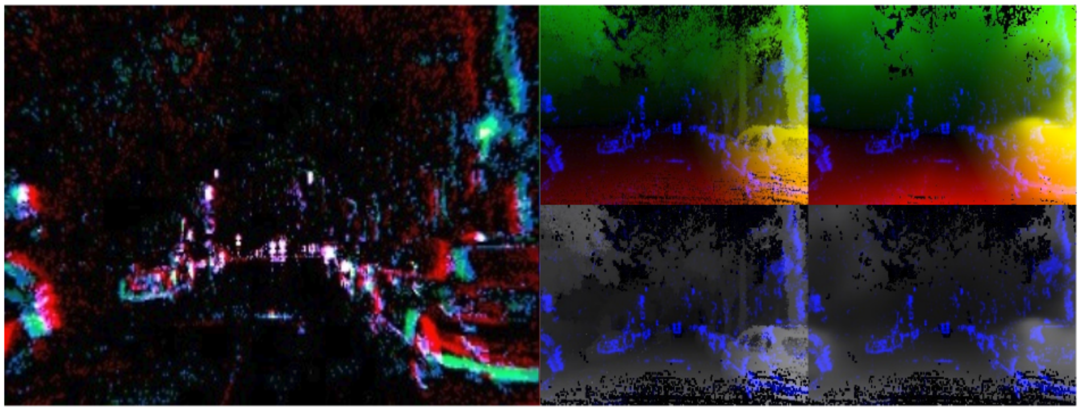


Figure 4.1: Optical flow and depth inference on sparse event data in night scene: event camera output (left), ground truth (middle column), network output (right) (top row - flow, bottom row - depth). The event data is overlaid on the ground truth and inference images in blue. Note, how our network is able to 'fill in' the sparse regions and reconstruct the car on the right.

structure of the event cloud and more robustly estimate egomotion. The main contributions of our work can be summarized as:

- The first unsupervised learning-based approach to structure from motion using monocular DVS input. Our pipeline trains on data in day but transfers well to night.
- Demonstrating that dense, meaningful scene and motion information can be recovered from sparse event data.
- A new lightweight high-performance encoder-decoder network that extends a recent cascaded design [75].
- A new alternative to normalization techniques, called *feature decorrelation*,

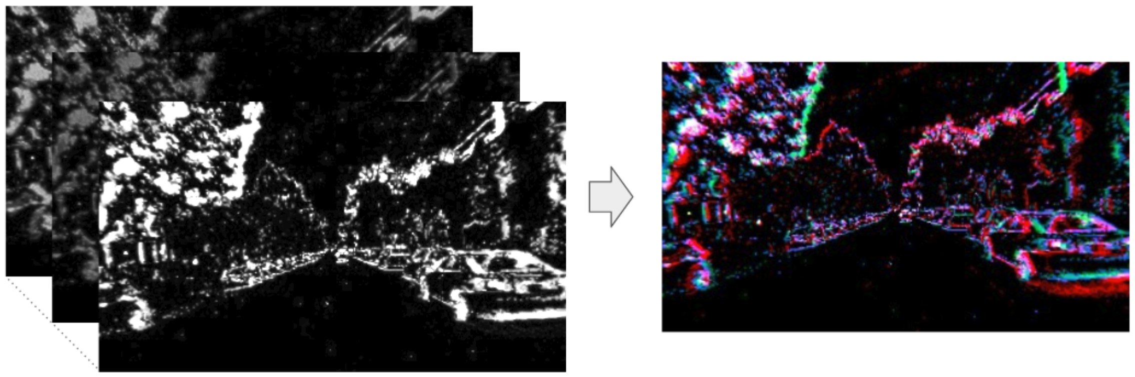


Figure 4.2: *A three-channel DVS data representation. The first channel represents the time image described in [76]. The second and third channels represent the per-pixel positive and negative event counts. Best viewed in color.*

which significantly improves training time and inference quality.

- Quantitative evaluation on the MVSEC dataset [40] of dense and sparse depth, optical flow and egomotion.
- A pre-processesed MVSEC [40] dataset facilitating further research on event-based SfM.

4.1 Related Work

4.1.1 Event-based Depth Estimation

The majority of event-based depth estimation methods [27,28] use two or more event cameras. As our proposed approach uses only one event camera, we focus our discussion on monocular depth estimation methods. The first works on event-based monocular depth estimation were presented in [29] and [30]. Rebecq *et al.* [29] used a space-sweep voting mechanism and maximization strategy to estimate semi-dense depth maps where the trajectory is known. Kim *et al.* [30] used probabilistic filters to jointly estimate the motion of the event camera, a 3D map of the scene, and the intensity image. More recently, Gallego *et al.* [31] proposed a unified framework for joint estimation of depth, motion and optical flow. So far there has been no deep learning framework to predict depths from a monocular event camera.

4.1.2 Event-based Optical Flow

The most popular approaches to optical flow computation on event data are gradient based - namely, to fit local planes to event clouds [36]. As discussed in [37], local event information is inherently ambiguous. To resolve the ambiguity Barranco et al. [37] proposed to collect events over a longer time intervals and compute the motion from the trace of events at contours.

Recently, neural network approaches have shown promising results in various estimation problems without explicit feature engineering. Orchard and Etienne-Cummings [38] used a spiking neural network to estimate flow. Most recently, Zhu *et al.* [39] released the MVSEC dataset [40] and proposed self-supervised learning algorithm to estimate optical flow. Unlike [39], which uses grayscale information as a supervision signal, our proposed framework uses only events and thus can work in challenging lighting conditions.

4.1.3 Self-supervised Structure from Motion

The unsupervised learning framework for 3D scene understanding has recently gained popularity in frame-based vision research. Zhou et. al [55] pioneered this line of work. The authors followed a traditional geometric modeling approach and built two neural networks, one for learning pose from single image frames, and one for pose from consecutive frames, which were self-supervised by aligning the frames via the flow. Follow-up works (e.g. [77] have used similar formulations with better loss functions and networks, and recently [28] proposed SfM learning from stereo DVS data.

4.2 Methods

4.2.1 Ego-motion Model

We assume that the camera is moving with rigid motion with translational velocity $v = (v_x, v_y, v_z)^T$ and rotational velocity $\omega = (\omega_x, \omega_y, \omega_z)$, and that the camera intrinsic parameters are provided. Let $\mathbf{X} = (X, Y, Z)^T$ be the world coordinates of a point, and $\mathbf{x} = (x, y)^T$ be the corresponding pixel coordinates in the calibrated camera. Under the assumption of rigid motion, the image velocity $\mathbf{u} = (u, v)^T$ at $(x, y)^T$ is given as:

$$\begin{pmatrix} u \\ v \end{pmatrix} = \frac{1}{Z}\begin{pmatrix} -1 & 0 & x \\ 0 & -1 & y \end{pmatrix}\begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} + \begin{pmatrix} xy & -1-x^2 & y \\ 1+y^2 & -xy & -x \end{pmatrix}\begin{pmatrix} \omega_x \\ \omega_y \\ \omega_z \end{pmatrix} = A\mathbf{p} \quad (4.1)$$

As such, given the inverse depth and the ego-motion velocities $\mathbf{v}, \omega$, we can calculate the optical flow at a point using a matrix multiplication (Equation 4.1) Here $\mathbf{p}$ is used to denote the pose vector $(\mathbf{v}, \omega)^T$, and A is a 2×6 matrix. Due to scaling ambiguity in this formulation, depth Z and translation (v_x, v_y, v_z) are computed up to a scaling factor.

4.2.2 Input Data Representation

The raw data from the DVS sensor is a stream of events, which we treat as data of 3 dimensions. Each event encodes the pixel coordinate (x, y) and the *timestamp* t. In addition, it also carries information about its *polarity* - a binary value that disambiguates events generated on rising light intensity (positive polarity) and events generated on falling light intensity (negative polarity).

The 3D (x, y, t) event cloud (within a small time slice), called event slice, is projected onto a plane and converted to a 3-channel image. An example of such image can be seen in Fig. 4.2. Two of the channels are the per-pixel counts of positive and negative events. The third channel is the *time image* as described in [76] - each pixel consists of the average timestamp of the events generated on this pixel, because the averaging of timestamps provides better noise tolerance. The neural network input consists of up to 5 such consecutive slice images to better preserve the timestamp information of the event cloud.

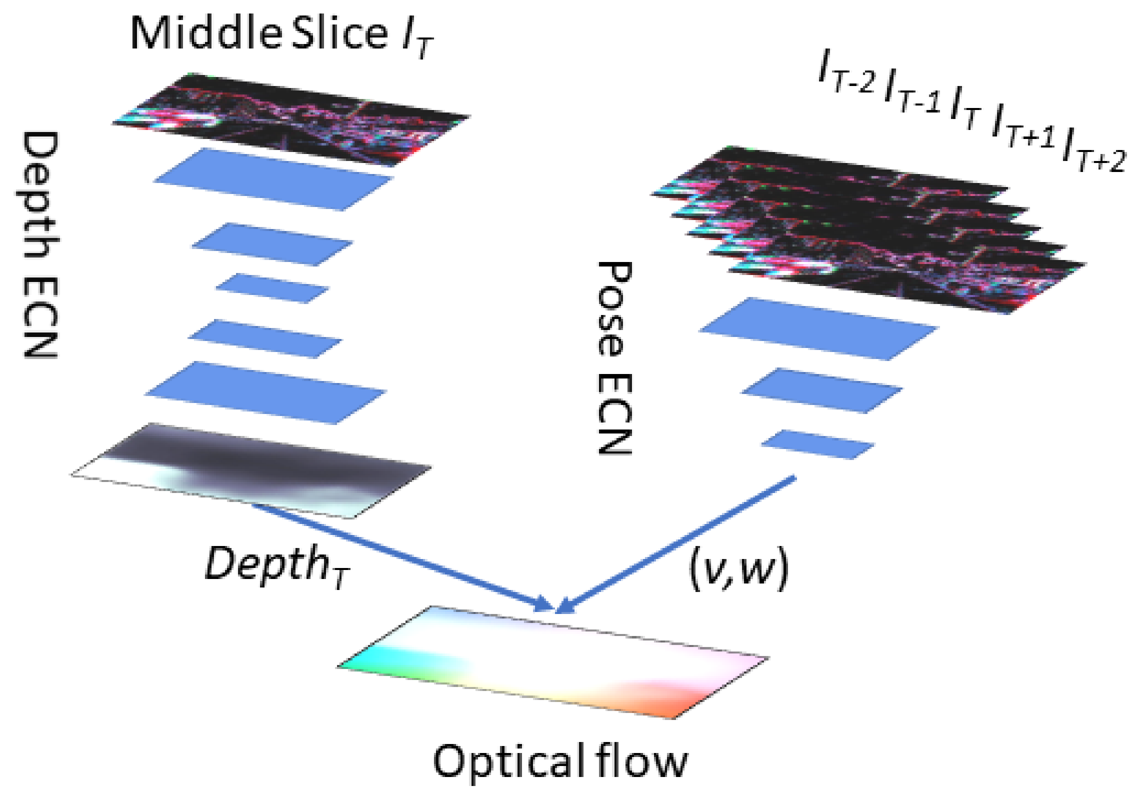


Figure 4.3: *The depth network (top) with an encoder-decoder architecture is used to estimate scene depth. The pose network (bottom) takes consecutive frames to estimate the translational velocity and rotational velocity with respect to the middle frame. Given the poses of neighboring frames and the depth of the middle frame, we calculate the optical flow. The neighboring frames are inversely warped to the middle frame and the warping difference provides the supervision loss. In the networks lighter/darker colors represents lower/higher level features.*

4.2.3 The Pipeline

The global network structure is similar to the one proposed in [55]. It consists of a depth prediction component and a parallel pose prediction component, which both feed into a optical flow component to warp successive event slices. The loss from the warping error is backpropagated to train flow, inverse depth, and pose.

Our network components, instead of the standard CNNs, are based on our ECN network structure. An (ECN based) encoding-decoding architecture is used to estimate scaled inverse depth $\frac{1}{Z}$ from a single slice of events. To address the data sparsity, we use bilinear interpolation, which propagates local information and fills in the gaps between events. A second network, which takes consecutive slices of signals, is used to derive v and ω. Then, using the rigid motion and inverse depth to predict the optical flow, neighboring slices at neighboring time instances $T+1, T+2$ and $T-1, T-2$ are warped to the slice at T (Fig. 4.3). The l^1 loss is used to measure the difference between the warped events and the middle slice as

$$Loss_{warp} = \sum_{T-2 \le n \le T+2, n \ne T} |I_n^{warpped} - I_T| \tag{4.2}$$

It is worth pointing out that the outputs of our networks are multi-scale. The loss functions are weighted by the number of pixels to calculate the total loss.

4.2.4 Evenly Cascaded Network Architecture

Our transform of features takes biological inspiration from multi-stage information distillation, and incorporates feedback [75]. In our architecture, the encoding layers split the (layer) input into two streams of features (Fig. 4.4): one encodes the features from the previous layer at lower resolution; the other directly generates a set of higher level features, as in CNN architectures [78]. At the end of the encoding

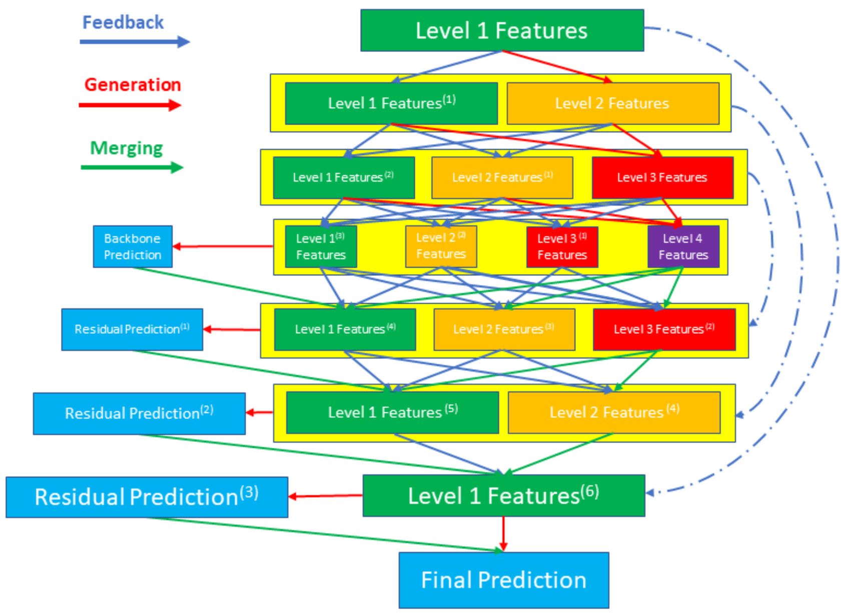


Figure 4.4: *The encoder-decoder structure. Only the generation and merging of features are shown.*

stage, the network has a multi-scale feature representation. This representation is used in our pose prediction.

In this work we added to the encoder [75] a decoder, which works as follows: In each decoding layer, we use the higher level features from the previous layer as a feedback signal to improve the lower level features, and combine them with the features from the corresponding encoding layer as in the U-Net [79] architecture. At the end of the decoding stage, the network has acquired a set of modulated high resolution low-level features.

Our design facilitates training because residual learning is conducted throughout the network for each level of features, while in comparison, the original ResNet [80] does that only in design blocks. In the encoder, we generate higher level features similar to DenseNet [81], but we use residual learning. In the decoder part, the generation process is inverted into a merging process: in each layer the highest level features are gradually merged back into the lower level features and finally to the backbone pathway to improve them.

To tackle the challenges raised by sparse event data and evenly resize the features, we use bilinear interpolation. In the encoding layers, our network evenly downscales the feature maps by a scaling factor of $(s < 1)$ to get coarser and coarser features. In the decoding layers, the feature maps are reversely upscaled by a factor of $1/s$. The network construction is automatic and is controlled by the scaling factor. Bilinear interpolation propagates the sparse data spatially, facilitating dense prediction of depth and optical flow.

4.2.5 Depth Predictions

In the decoding stage, we make predictions from features at different resolutions and levels (Fig. 4.4). Initially, both high and low-level coarse features are used to predict a backbone depth map. The depth map is then up-sampled with bilinear interpolation for refinement. Then the enhanced lower level features are used to estimate the prediction residue, which are added to the backbone estimation to refine it.

Our pipeline is based on monocular vision and predicts depth up to a scale. In real world driving scenes, the mean depth value stays relatively stable. Taking advantage of this observation, we use batch normalization before making the depth prediction so the predicted depths have similar range.

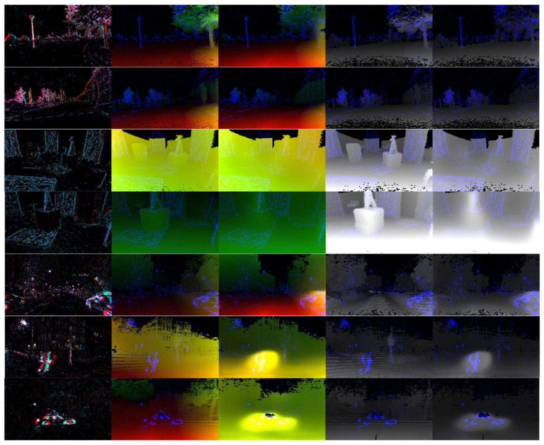


Figure 4.5: *Qualitative results from our evaluation. The table entries from left to right: DVS input, ground truth optical flow, network output for flow, ground truth for depth, network output for depth. The event counts are overlaid in blue for better visualization. Examples were collected from sequences of the MVSEC [40] dataset: (top to bottom) outdoor day 1, outdoor day 1, indoor flying 1, indoor flying 2, outdoor night 1, outdoor night 2, outdoor night 3. Note that on the 'night' sequences the ground truth is occasionally missing due to Lidar limitations but the pipeline performs well. Best viewed in color.*

To further address the sparsity in data, we utilize a sparsity constraint that promotes non-local information propagation [82]:

$$\begin{aligned} Loss_{smooth}(I) &= \sum_i \sum_{j \in N(i)} |I_j - I_i|^p \\ &= \sum_i \sum_{j \in N(i)} |I_j - I_i|^{p-2} |I_j - I_i|^2 = \sum_i \sum_{j \in N(i)} w_{ij} |I_j - I_i|^2 \end{aligned} \quad (4.3)$$

Here the loss is applied on the first-order derivatives of the depth estimation in a neighborhood $N(i)$, and we use a sparse penalty where $0 < p \leq 1$. The complete loss of our pipeline is hence defined as:

$$Loss = Loss_{warp} + \lambda Loss_{smooth} \tag{4.4}$$

4.2.6 Feature Decorrelation

Gradient descent training of neural networks can be challenging if the features are correlated. Researchers have proposed normalization strategies [68, 83, 84] to account for scale inconsistency. However, it is important to point out, scaling the data without decorrelating it does not guarantee good conditioning. Without correction, the covariance matrix of the data matrix X is ill-conditioned. The gradient descent algorithms take many iterations due to this ill-conditioning. Uisng decorrelation, the data matrix will be transformed to have I as covariance matrix, and the condition number will be 1.

We compute the inverse square root of the covariance between the feature channels. Using the Denman-Beavers square root iteration [85], we can calculate the inverse square root in a simple and forward fashion. Given a symmetric positive definite covariance matrix C, Denman-Beavers iterations start with initial values $Y_0 = C$, $Z_0 = I$. The iteration is defined as: $Y_{k+1} = \frac{1}{2}Y_k(3I - Z_kY_k), Z_{k+1} = \frac{1}{2}(3I - Z_kY_k)Z_k$, and $Z_k \to C^{-\frac{1}{2}}$ [84]. In our implementation, we evenly divide the features into groups [83, 86], and reduce the correlation between the groups by performing a few (1-10) Denman-Beavers iterations. We notice that a few iterations lead to significantly faster convergence and better results.

To further explain the favorable property of the whitening transform, assume we are given a linear regression problem with L_2 loss: $y = Xw$, $Loss = \frac{1}{2}\|y - \hat{y}\|^2 = \frac{1}{2}\|Xw - \hat{y}\|^2$.

For an explicit solution we have $\frac{\partial Loss}{\partial w} = X^t(Xw - \hat{y}) = 0$,

$$w = (X^t X)^{-1} X^t \hat{y}. \tag{4.5}$$

If the input is decorrelated, we have $w = X^t\hat{y}$. On the other hand, to conduct one iteration of gradient descent: we have $w_{new} = w_{old} - step\_len \times (X^t X w_{old} - X^t \hat{y})$. If $X^t X = I$ then $step\_len = 1$ is optimal and with one iteration we have $w_{new} = X^t\hat{y}$.

Following this spirit, we insert the decorrelation operation before the linear/convolution layers. In non-linear regression problem, the solution is found iteratively. In each layer, the transform is:

$$y_i = ReLU \circ W_i \circ D_i \circ x_i, \tag{4.6}$$

here x_i is the input to the $i-th$ layer, D_i is the decorrelation operation. It is important to note that the decorrelation procedure does not change the learning problem, as the standard network training is solving for $W_i \circ D_i$. Instead, this procedure removes the correlations between feature channels and improves the conditioning of the optimization problem.

4.3 Experimental Evaluation

Our our self-supervised learning framework can infer both dense optical flow and depth from sparse event data. We evaluate our work on the MVSEC [40] event

Table 4.1: Evaluation of the optical flow pipeline

	outdoor day 1		outdoor night 1		outdoor night 2		outdoor night 3		indoor flying[1]	
	AEE	% Outlier	AEE	% Outlier	AEE	% Outlier	AEE	% Outlier	AEE	% Outlier
ECN	0.35	0.04	**0.49**	**0.82**	**0.43**	**0.79**	**0.48**	**0.80**	0.21	**0.01**
ECN_{masked}	**0.30**	0.02	0.53	1.1	0.49	0.98	0.49	1.1	**0.20**	**0.01**
$Zhu18$ [28]	0.32	**0.0**	-	-	-	-	-	-	0.84	2.50
$EV\text{-}FlowNet_{best}$ [39]	0.49	0.20	-	-	-	-	-	-	1.45	10.26
$SfMlearner$	0.58	0.89	0.59	1.01	0.78	1.32	0.59	1.38	0.55	0.29
ECN_{erate}	0.28	0.02	0.46	0.67	0.40	0.53	0.43	0.67	0.20	0.01

camera dataset which, given a ground truth frequency of 20 Hz, contains over 40000 ground truth images.

The MVSEC dataset, inspired by KITTI [87], features 5 sequences of a car on the street (2 during the day and 3 during the night), as well as 4 short indoor sequences shot from a flying quadrotor. MVSEC was shot in a variety of lighting conditions and features low-light and high dynamic range frames which are often challenging for an analysis with classical cameras.

4.3.1 Implementation Details

Our standard network architecture has scaling rates of 0.5 and 2.0 respectively for the encoding and decoding layers, and results in 5 encoding/decoding layers. Our depth network has 8 initial hidden channels and expands with a growth rate of 8. We halve these settings to 4 for our pose network. The pose network takes

5 consecutive event slices and predicts $6d$ pose vectors. We use 3×3 convolutions, and the combined network has $150k$ parameters. We train the network with a batch size of 32 and use the Adam optimizer with a learning rate of 0.01. Interestingly, we notice that compared to the standard architecture of the SfMlearner, the learning rate is higher. Thus, the new design allows us to learn at a faster rate. The learning rate is annealed using cosine scheduling, and we let the training run for 30 epochs. Our training takes 7-minutes for each epoch using a single Nvidia GTX 1080Ti GPU. We set the smoothness loss weight $\lambda = 0.1$. Our model using batch normalization runs at 250 FPS at inference as it has been heavily GPU optimized. The model using feature decorrelation runs at 40 FPS. The slow down is mainly due to matrix multiplications in our customized layer which is not optimized for the GPU.

4.3.2 Dataset Preparation

In the experiments with the outdoor sequences, we trained the network using only the *outdoor day 2* sequence with the hood of the car cropped. Our experiments demonstrate that our training generalizes well to the notably different *outdoor day 1* sequence, as well as to the *night* sequences. For the *indoor* sequences, since they were too short to create a representative training set, we used 80% of each sequence for training and evaluated on the remaining 20%. We would like to note that the *outdoor night* sequences have occasional errors in the ground truth (see for example Fig. 4.5 last three rows, or Fig. 4.11). All incorrect frames had to be manually removed for the evaluation.

We have noticed, that although adaptive time windows, based on the event rate, are sometimes employed in the literature ([76], [28]), these approaches may not tolerate noise, which is particularly present during night sequences. In our experiments, we use fixed-width time slices of 1/40-th of a second.

4.3.3 Ablation Studies

4.3.3.1 Testing on the *SfMlearner*

As baseline we use the state-of-the-art *SfMlearner* [55] on our data (event images). *SfMlearner* has a fixed structure of 7 encoding and 7 decoding layers. It has 32 initial hidden channels and expands to 512 channels. Overall the model contains 33M parameters. SfMlearner is trained using Adam optimizer with a learning rate of $2e^{-4}$ and a batch size of 4. We replace the inputs with our event slices, and we include the evaluation results for flow and egomotion in tables 4.1 and 4.3.

4.3.3.2 Normalization Methods

We compare two normalization methods and our decorrelation method on the validation set portion of the outdoor day 2 sequence. We apply 5 Denman-Beavers iterations in the decorrelation procedure. Compared with normalization methods,

[1]Due to the size of the indoor dataset, we trained on 80% of the sequences and tested on 20% - this was only the case for *indoor* sequences. We would like to note, that *EV-FlowNet* requires classical frames for training and Zhu18 uses a stereo camera pair. To compute aggregate results for [28, 39], we prorate the results they have presented by the number of usable pixels in the sequence and recompute the average.

decorrelation leads to more thorough data whitening, and we have noticed this layer-wise whitening lead to faster convergence and lower evaluation loss (Fig. 4.6).

4.3.3.3 Visualizing a Shallow and Tiny Network

Our lightweight multi-level, multi-resolution design allows us to construct networks of any depth and size. As a preliminary attempt, we set the scaling rate to 1/3 and 3.0 for encoding and decoding layers respectively, so the network has only 3 encoding/decoding layers. As the network is small, we can directly visualize all its internal feature maps. A deeper and wider network would produce a higher quality output but also more feature maps, which we do not list here. In Fig. 4.7 we have listed all the feature maps in the small depth network. The row number corresponds to the level number of the features for each figure. We notice the encoder seems to play a feature extraction role in the network and the decoder starts to produce semantically meaningful representation corresponding to the desired output (depth). By scrutinizing the pose network outputs, we notice the network is intelligent enough to aggregate information corresponding to different time periods of the events in the first layer (Fig. 4.8). Otherwise, mixing up the events at different time period would make the pose estimation harder.

4.3.3.4 Performance Versus Event Rate

Since the event data is inherently sparse, we investigate the performance of the pipeline in relation to the data sparsity.

We measure the data sparsity as a percentage of the pixels on the input images with at least one event. Fig. 4.9 demonstrates how the event rate is inversely proportional to the average endpoint error for the optical flow (we have observed similar behavior for depth and egomotion). The *outdoor day 1* sequence is used to minimize the influence of the noise.

We find the inverse correlation between event rate and inference quality to be a useful observation, as this property could be efficiently used in sensor fusion in a robotic system. Motivated by that, we provide an additional row to the Table 4.1: ECN_{erate}, and report our error metrics once again only for the frames with *higher than average* number of event pixels across the datasets.

4.3.4 Qualitative Results

In addition to the quantitative evaluation, we present a number of samples for qualitative analysis in Fig. 5.6. The last three rows of the table show the night sequences, and how the pipeline performs well even when only a few events are available. The third and the fourth rows show indoor scenes. The indoor sequences were relatively small and it is highly possible that the quality of the output would increase given a larger dataset.

4.3.5 Optical Flow Evaluation

We evaluate our optical flow results in terms of Average Endpoint Error ($AEE = \frac{1}{n}\sum \|\vec{y} - \vec{y}^{*}\|_2$ with y^* and y the estimated and ground truth value, and n the number of pixels for which flow was estimated) and compare our results against two state-of-the-art optical flow methods for event-based cameras: $EV - FlowNet$ [39] and a recent stereo method [28] (in the tables - $Zhu18$).

Because our network produces flow and depth values for every image pixel, our evaluation is not constrained by pixels which did not trigger a DVS event. Still, for consistency reasons, we report both numbers for each of our experiments (for example, ECN and ECN_{masked}, where the latter has errors computed only on the pixels with at least one event). Similar to KITTI and EV-FlowNet, we report the percentage of outliers - values with error more than 3 pixels or 5% of the flow vector magnitude.

To compare against [39] and [28], we account for the difference in the frame rates (for example, EV-FlowNet uses the frame rate of the DAVIS classical frames) by scaling our optical flow. We also provide aggregated results for the indoor scenes (split on a train and a test set 80/20 as described above), although these are not the main focus of our study. Our main results are presented in the Table 4.1.

We show that our optical flow performs well during both day and night, all on unseen sequences. The results are typically better for the experiments with event masks except for the *outdoor night*. One possible explanation for that is that this sequence is much noisier with events being generated not only on the edges, which leads to suboptimal masking.

4.3.6 Depth Evaluation

Since there are currently no monocular event-based methods for the depth estimation based on unsupervised learning, we provide the classical scale-invariant depth metrics, used in many works such as [88], [55], [89]:

$$Accuracy : \% of y_i \; s.t. \; max(\frac{y_i}{y_i^*}, \frac{y_i^*}{y_i}) = \delta < th \tag{4.7}$$

$$SILog : \frac{1}{n}\sum d_i^2 - \frac{1}{n^2}(\sum d_i)^2, d_i = \log y_i - \log y_i^* \tag{4.8}$$

$$AbsoluteRelativeDifference : \frac{1}{n}\sum \frac{|y - y^*|}{y^*} \tag{4.9}$$

$$LogarithmicRMSE : \sqrt{\frac{1}{n}\sum \|\log y - \log y^*\|^2} \tag{4.10}$$

Our results are presented in Table 4.2 for both event count-masked depth values and full, dense depth. Since the night driving scenes have similar depth geometries, we aggregate all 3 sequences in one table entry.

Applying an event mask during the evaluation increases accuracy for all scenes - this is expected, as the inference is indeed more accurate on the pixels with event data. On the contrary, the error rate increases on the outdoor scenes and decreases on the indoor scenes. This is probably due to higher variation of the outdoor scenes and also faster motion of the car.

4.3.7 Egomotion Estimation

Our pipeline is capable of inferring egomotion on both day and night sequences, and transfers well from *outdoor day 2* onto *outdoor day 1* and *outdoor night 1,2,3*.

Table 4.2: Evaluation of the depth estimation pipeline. Results on masked, sparse depth are separated by "/", followed by the results on *SfMLearner* in braces.

	outdoor day 1	outdoor night	indoor flying
Abs Rel	0.29 / 0.33 (0.55)	0.34 / 0.39 (0.53)	0.28 / 0.22 (0.49)
RMSE log	0.29 / 0.33 (0.54)	0.38 / 0.42 (0.51)	0.29 / 0.25 (0.50)
SILog	0.12 / 0.14 (0.28)	0.15 / 0.18 (0.32)	0.11 / 0.11 (0.12)
$\delta < 1.25$	0.80 / 0.97 (0.65)	0.67 / 0.95 (0.56)	0.75 / 0.98 (0.57)
$\delta < 1.25^2$	0.91 / 0.98 (0.78)	0.85 / 0.98 (0.75)	0.91 / 0.99 (0.79)
$\delta < 1.25^3$	0.96 / 0.99 (0.89)	0.93 / 0.99 (0.87)	0.96 / 1.00 (0.88)

Since our pipeline is monocular, we predict the translational component of the velocity up to a scaling factor, while the rotational velocity does not need scaling. Despite our network outputs full 6 degree of freedom velocity, we did not achieve high quality on indoor sequences. This is likely due to highly more complicated motion types and a small size of the indoor dataset. We further discuss this in sec. 4.3.8.

For the driving scenarios we can *make an important observation* - the mean distance of the road in respect to the camera is often a constant. We crop the lower middle part of the inferred depth image and apply a scaling factor such that the mean depth value (corresponding to the road location) is constant. Consequently, only a single extrinsic value (camera height on the car) is needed to reconstruct the scaled motion. In our experiments, we report egomotion with translational scales taken both from ground truth (AEE_{tr}^{gt}) and using the depth constancy constraint (AEE_{tr}^{depth}), with a global scale taken from ground truth. The qualitative results are presented in Fig. 4.10.

Unlike [28], we train *SfMlearner* on the event images, and not on the classical frames to allow for evaluation on the night sequences. We provide comparison to the work in [28], although it uses a stereo pipeline and reports results only on the *outdoor day 1* sequence.

To be consistent with [28], we report our trajectory estimation relative pose and relative rotation errors as $RPE = arccos(\frac{t_{pred} \cdot t_{gt}}{\|t_{pred}\|_2 \cdot \|t_{gt}\|_2})$ and $RRE = \|logm(R_{pred}^T R_{gt})\|_2$. Here $logm$ is matrix logarithm and R are Euler rotation matrices. The RPE essentially amounts to the angular error between translational vectors (ignoring the scale), and RRE amounts to the total 3-dimensional angular rotation error. To account for translational scale errors, we report classical Endpoint Errors - computed as a magnitude of the differences in translational component of the velocities. Our quantitative results are presented in Table 4.3.

Table 4.3: *Egomotion estimation results on driving sequences - 'outdoor day 1' and 'outdoor night 1,2,3'. The ARPE and ARRE are reported in degrees and radians respectively [28], AEE is in m/s.* AEE_{tr}^{gt} *- translational endpoint error with ground truth normalization.* AEE_{tr}^{depth} *- normalized using depth prediction and a global scaling factor (see sec. 4.3.7).*

		$ARPE$	$ARRE$	AEE_{tr}^{gt}	AEE_{tr}^{depth}
ECN	outdoor day 1	3.98	0.00267	0.70	1.29
$Zhu18$ [28]	outdoor day 1	7.74	0.00867	-	-
$SfMlearner$	outdoor day 1	16.99	0.00916	1.59	2.39
ECN	outdoor night 1	3.90	0.00139	0.42	1.26
$SfMlearner$	outdoor night 1	9.95	0.00433	1.04	2.47
ECN	outdoor night 2	3.44	0.00202	0.80	1.34
$SfMlearner$	outdoor night 2	13.51	0.00499	1.66	3.15
ECN	outdoor night 3	3.28	0.00202	0.49	1.03
$SfMlearner$	outdoor night 3	1.053	0.00482	1.42	2.74

4.3.8 Discussion and Failure Cases

A monocular pipeline tends infer more information from the shape of the contours on depth estimation and hence would transfer poorly on completely different scenarios. Nevertheless, we were able to achieve good generalization on night sequences and demonstrate promising results for depth and flow for indoor scenes (trained separately on parts of indoor sequences).

We observe a relatively small angular drift on trajectory estimation (Fig. 4.10). Despite our model predicting a full 6 degree of freedom motion we admit that in the car scenario only 2 motion parameters play a meaningful role and the network may tend to overfit. For this reason, training on the indoor scenes, featuring more complicated motion would require a notably larger dataset than presented in MVSEC. We still achieve results superior to *SfMlearner* and the stereo method [28], while for the comparison with the latter we must attribute some of our success to the fact that our translational velocity prediction is only up to scale. A common shortcoming of event-based sensors in the lack of data when the relative motion is not present. Fig. 4.11 shows such an example. This issue (although it does not affect egomotion) can be solved only by fusing data from other visual sensors or by moving the event-based sensor continuously. Because of the smoothness constraint used to combat data sparsity, the network tends to blur object boundaries. Still, for the driving environment the contours of obstacles, people and cars are clearly visible, as can be seen in Fig. 5.6.

4.4 CONCLUSION

We have presented a novel low-parameter pipeline for generating dense optical flow, depth and egomotion from sparse event camera data. We also have shown experimentally that our new neural network architecture using multi-level features improves upon existing work.

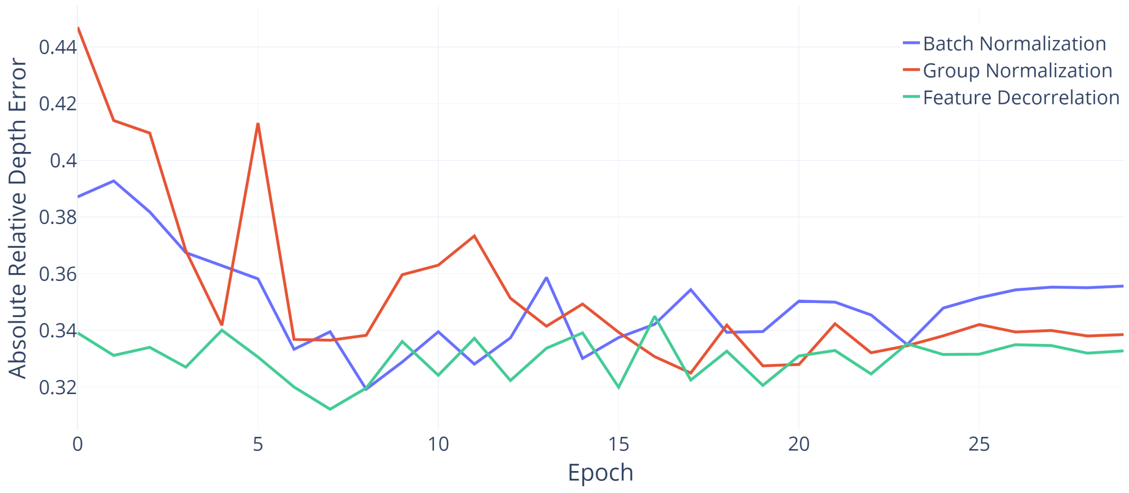


Figure 4.6: *Comparison of Abs Rel Errors using different normalization methods on outdoor day 1 sequence (less is better).*

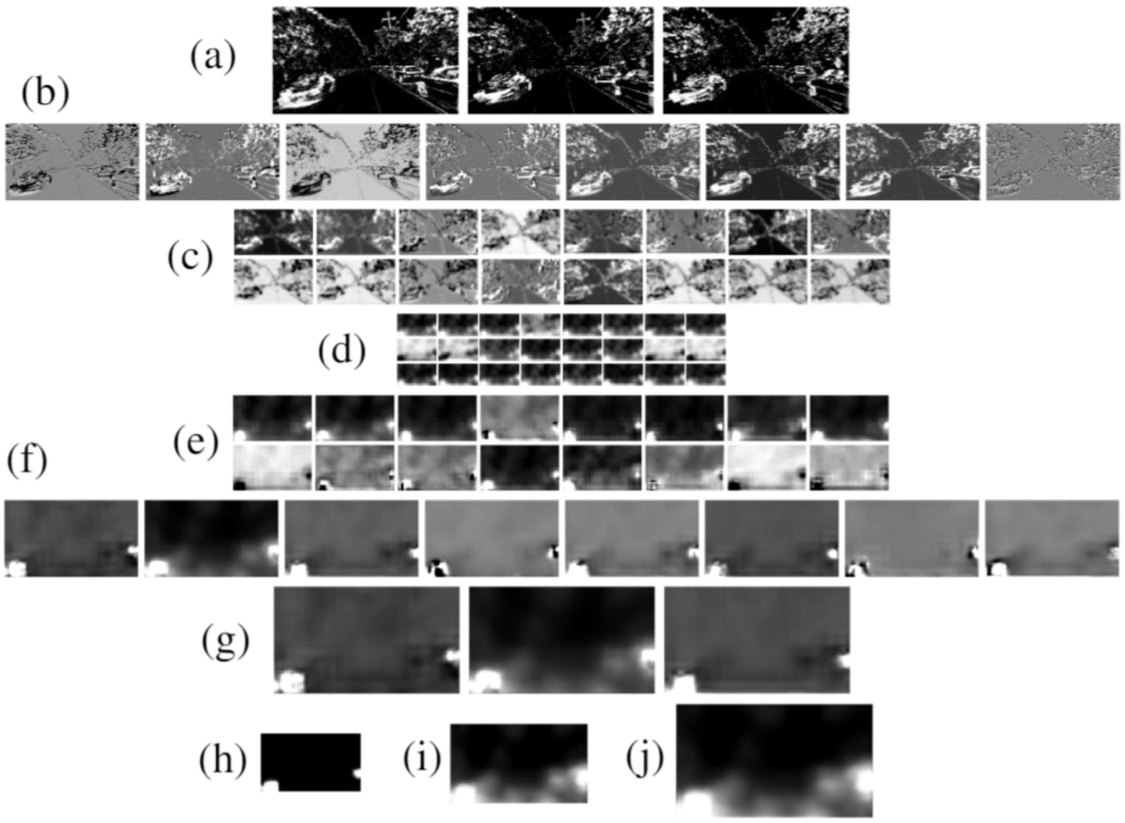


Figure 4.7: *Visualization of feature maps of the depth network. (a) Input channels. (b-g) Feature maps of the encoder-decoder network. (h-j) Multiscale predictions by layers (e-g).*

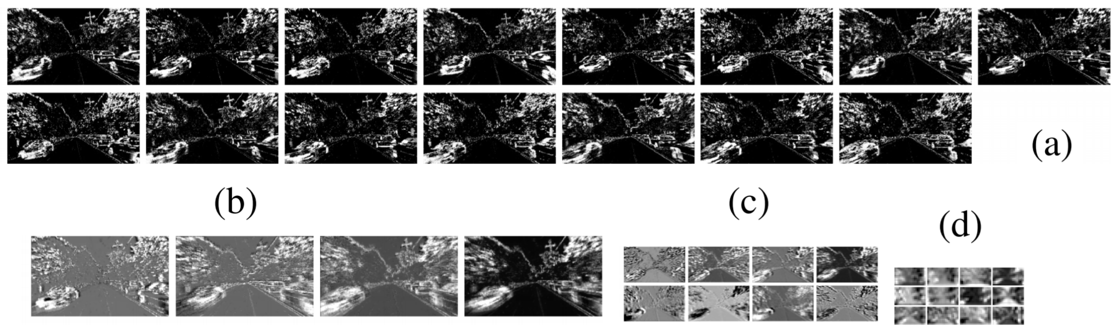


Figure 4.8: *Visualization of the pose network. (a) Input channels. (b-d) Feature maps of the pose network.*

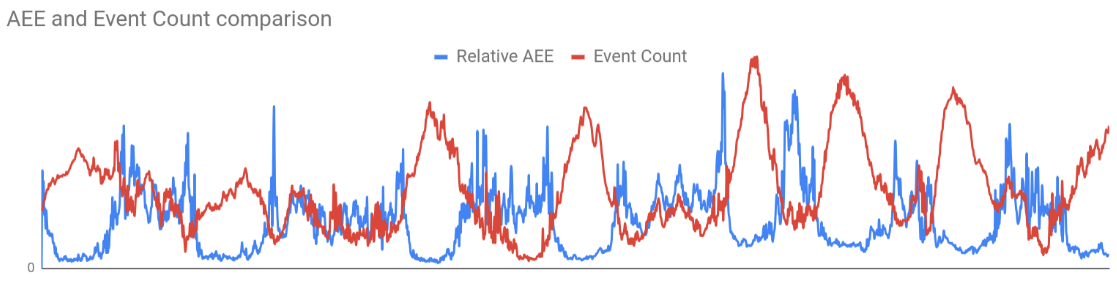


Figure 4.9: The Average Endpoint Error (blue) and the number of pixels with at least one event (red) for the first 1500 frames of 'outdoor_day1' sequence of the MVSEC dataset. Both plots are normalized so that the mean value is 0.5 for easier comparison.

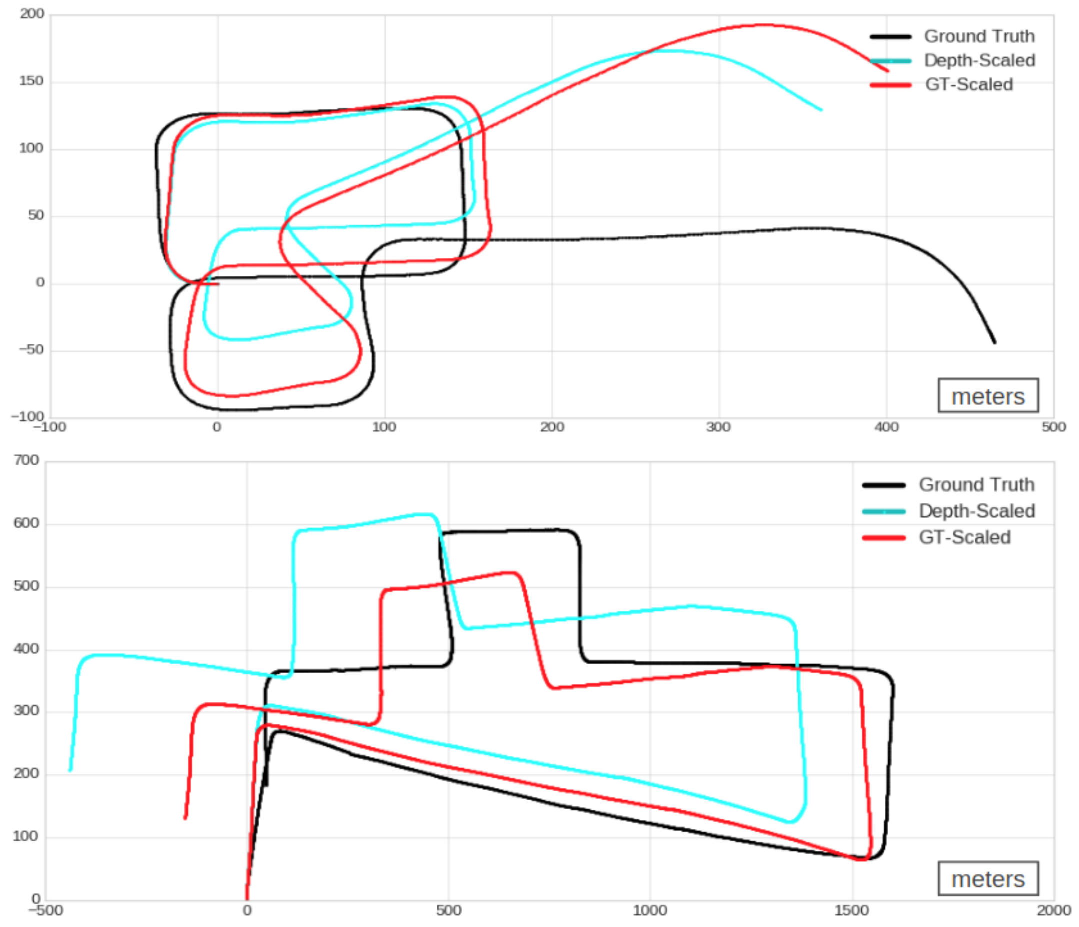


Figure 4.10: *Estimated trajectories on 'outdoor day 1' (top) and 'outdoor night 2' (bottom) sequences, acquired by integrating the egomotion predictions. The network was trained only on 'outdoor day 2'. Black: ground truth. Red: network prediction with translational scale applied from ground truth. Cyan: result by assuming the mean depth is fixed throughout the sequence (sec. 4.3.7) and by applying a single global scaling to the translational pose.*

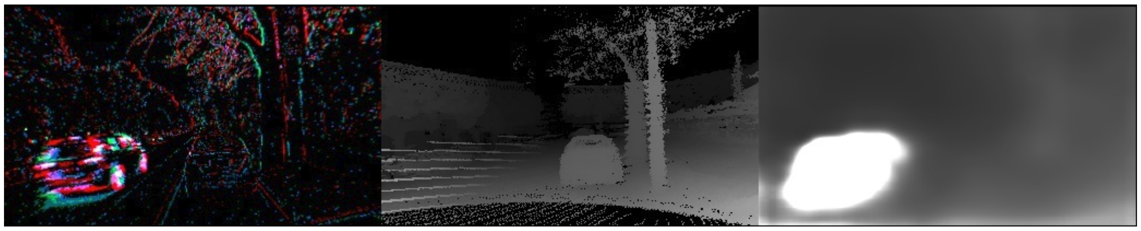


Figure 4.11: *A typical failure case and a dataset artifact: A non-moving car (visible in the middle ground truth inverse depth image) is not visible on the DAVIS camera (left image) which prevents ECN from inferring optical flow or depth correctly (right image is the inference inverse depth image). On the contrary, the moving car on the left side of the road is clearly visible in the event space and its depth inference is correct, but due to the Lidar limitations the depth ground truth is completely missing. This frame is taken from the 'outdoor_night 1' MVSEC sequence.*

Chapter 5: EV-IMO: Motion Segmentation Dataset and Learning Pipeline for Event Cameras

In this chapter we introduce a compositional neural network (NN) pipeline, which provides supervised up-to-scale depth and pixel-wise motion segmentation of the scene, as well as *unsupervised* 6 dof egomotion estimation and a per-segment linear velocity estimation using only monocular event data (see Fig. 5.1). This pipeline can be used in indoor scenarios for motion estimation and obstacle avoidance.

We also create a dataset, *EV-IMO*, which includes 32 minutes of indoor recording with multiple independently moving objects shot against a varying set of backgrounds and featuring different camera and object motions. To our knowledge, this is the first dataset for event-based cameras to include accurate pixel-wise masks for independently moving objects, apart from depth and trajectory ground truths.

To summarize, our contributions are:

- The first NN for estimating both camera and object 3D motion using event data;
- The first dataset – *EV-IMO* – for motion segmentation with ground truth depth, per-object mask, camera and object motion;

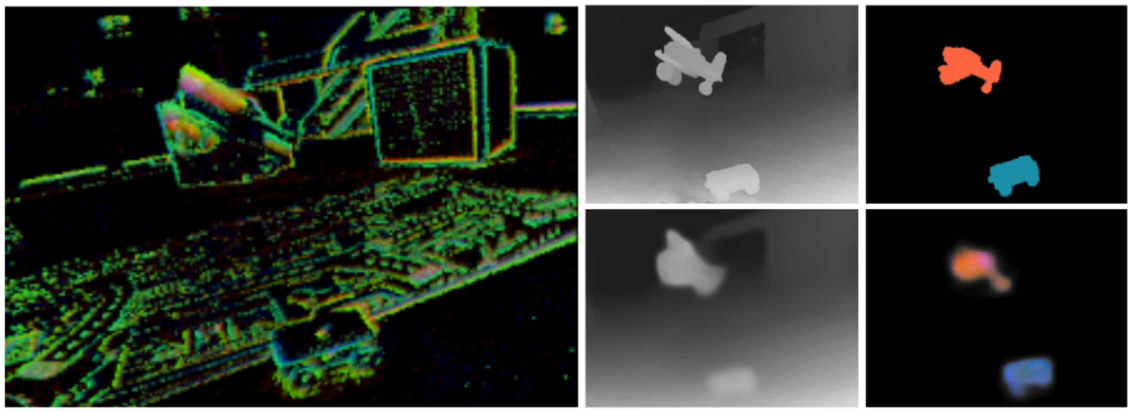


Figure 5.1: Depth and per-pixel pose inference on sparse event data, on our *EV-IMO* dataset. The top, left row is the ground truth depth and pose (the color corresponds to object's linear velocity), the bottom left row is the predicted network output. Camera egomotion is also estimated but not visualized. Best viewed in color.

- A novel loss function tailored for event alignment, measuring the profile sharpness of the motion compensated events;
- Demonstration of the feasibility of using a shallow low parameter multi-level feature NN architecture for event-based segmentation while retaining similar performance with the full-sized network;

5.1 The Architecture

5.1.1 Network Input

The raw data from the DVS is a continuous stream of events. Each event, $e(x, y, t, p)$ is encoded by its pixel position (x, y), timestamp t, accurate to microsec-

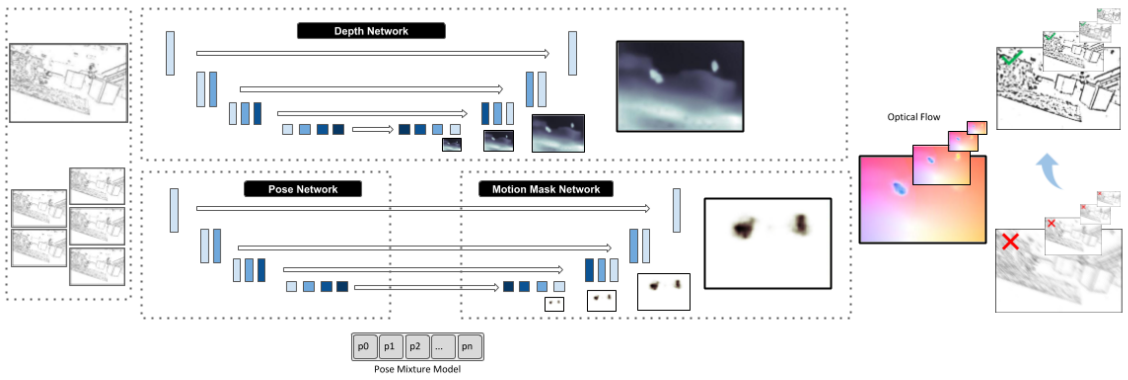


Figure 5.2: *A depth network (top) using an encoder-decoder architecture trained in supervised mode is used to estimate scene depth. A pose network (bottom left) takes consecutive event slices to generate a mixture model for the pixel-wise pose. A mixture of poses and mixture probabilities (bottom right) are outputs of this network. The outputs of the two networks are multi-scale and used to generate the optical flow, then to inversely warp the inputs at various resolutions.*

onds, and binary polarity, $p \in \{-1, 1\}$, indicating whether the intensity decreased or increased.

In the (x, y, t) space, the event stream represents a 3D pointcloud. To leverage the maximum information from this representation and pass it down to the network, we subdivide the event stream into consecutive *time slices* of size δt (in our implementation - 25 ms). Every time slice is projected on a plane with a representation as in the previous chapter.

We then feed these 2D maps to the neural networks in our pipeline. The benefit of the 2D input representation is the reduction of data sparsity, and a resulting increase in efficiency compared to the 3D learning approaches. Yet, the 2D input may suffer from motion blur during fast motions. We tackle this problem by using a fine scale warping loss (sec. 5.1.5.1), which uses 1 ms. slices to compute the loss.

5.1.2 Overview of the Architecture

Our pipeline (see Fig. 5.2) consists of a depth prediction network and a pose prediction network. Both networks are low parameter [90] encoder-decoder networks [79]. Our depth network performs a prediction on a single slice map. A supervision loss $Loss_{depth}$ comes by comparing with the ground truth as we describe in subsection 5.1.5.3. Our pose network uses up to 5 consecutive maps, to better account for the 3D structure of the raw event data. The pose network utilizes a mixture model to estimate pixel-wise 3D motion (relative pose) and corresponding motion masks from consecutive event slices. The masks are learned in supervised mode. We introduce a $Loss_{mask}$ on the motion mask. Finally, the two network outputs are used to generate the optical flow (Fig. 5.2). Successive event slices within a small period of time are then inversely warped. Perfectly motion compensated slices should stack into a sharp profile, and we introduce a two-stage $Loss_{warp}$ to measure the warping quality. The sum of the losses $Loss = Loss_{warp} + w_{depth}Loss_{depth} + w_{mask}Loss_{mask}$ is backpropagated to train flow, inverse depth, and pose.

5.1.3 Motion Model

For each pixel, given the inverse depth, there is a linear relation between the optical flow and the 3D motion parameters (Eq. 4.1). We model the motion of individual moving objects as 3D translation (without rotation), since most objects have relatively small size. The motion (pose) of any object is modeled as the sum of the rigid background motion and the object translation. Our network uses a mixture model for object segmentation - the 3D motion p^i at a pixel (x_i, y_i), is modeled as the sum of the camera motion p_{ego} and weighted object translations, where the weights are obtained from motion masks as:

$$p^i = p_{ego} + \sum_{j=1}^{C} m_j^i t_j, \tag{5.1}$$

In the above equation m_j^i are the motion mask weights for the $i-th$ pixel and t_j the estimated translations of the C objects.

5.1.4 A Mixture Model for Ego-motion and Independently Moving Objects

The pose network utilizes a mixture model to predict pixel-wise pose. At the end of the encoder part, the network outputs a set of poses $(p_0, t_1, ..., t_C)$ in parallel. p_0 is the ego-motion pose p_{ego}, and $(t_1...t_C)$ are interpreted as the translations *with respect to the background* or residual translations. The residual translations are added to the ego-motion pose as in Eq. 5.1 to get the candidate poses of objects relative to the camera.

In the decoding part, the network predicts pixel-wise mixture weights or motion masks for the poses. We use the mixture weights and the pose candidates to generate pixel-wise pose. The mixture weights sum to 1 for each pixel. We found experimentally that allowing a pixel to belong to multiple rigid motions as opposed to only one, leads to better results. This is because soft assignment allows the model to explain more directions of motions. However, since during training, the object masks are provided, qualitatively sharp object boundaries are learned.

Using the mixture model representation allows us differentiate object regions, moving with relatively small difference in 3D motion.

5.1.5 Loss functions

We describe the loss functions used in the framework. It is noteworthy that the outputs of our networks are multi-scale. The loss functions described in this section are also calculated at various scales. They are weighted by the number of pixels and sum up to calculate the total loss.

5.1.5.1 Event Warping Loss

In the training process, we calculate the optical flow and inversely warp events to compensate for the motion. This is done by measuring the warping loss at two time scales, first for a rough estimate, between slices, then for a refined estimate within a slice where we take full advantage of the timestamp information in the events.

Specifically, first using the optical flow estimate, we inversely warp neighboring slices to the center slice. To measure the alignment quality at the coarse scale, we take 3-5 consecutive event slices, where each consists of 0.05 seconds of motion information, and use the absolute difference in event counts after warping as the loss:

$$Loss_{coarse} = \sum_{-K \leq n \leq K, n \neq 0} |I_n^{warpped} - I^{middle}|,$$

where I denotes the three maps of positive events, negative events and average timestamps, and K is either 1 or 2. To refine the alignment, we process the event point clouds and divide the slices into smaller slices of $1ms$. Separately warping each of the small slices allows us to fully utilize the time information contained in the continuous event stream.

We stack warped slices and use the following sharpness loss to estimate the warping quality. Intuitively speaking, if the pose is perfectly estimated, the stacking of inversely warped slices should lead to a motion-deblurred sharp image. Let $S = \sum_{n=-N}^{N} |I_n^{warped}|$ be the stacking of inversely warped event slices, where n represents the n-th slice in a stack of $2N+1$ slices. Our basic observation is that the sparse quasi-norm $||\cdot||_p$ for $0 < p < 1$ favors a sharp non-negative image over a blurred one. That is, $\sum_i |x_i|^p \geq (\sum_i |x_i|)^p$ for $0 < p < 1$. Based on this observation, we calculate the quasi-norm of S to get the fine scale loss: $Loss_{fine} = ||S||_p, 0 < p < 1$.

5.1.5.2 Motion Mask Loss

Given the ground truth motion mask, we apply a binary cross entropy loss on the mixture weight of the ego-motion pose component to constrain that our model applies the ego-motion pose in the background region: $Loss_{mask} = -\sum_{i \in background} log(m_0^i)$ To enforce that the mixture assignment is locally smooth, we also apply a smoothness loss on the first-order gradients of all the mixture weights.

5.1.5.3 Depth Loss

With ground truth depth available, we enforce the depth network output to be consistent with the ground truth. We adjust the network output and the ground truth to the same scale, which we denote as *predict* and *truth* and apply the following penalty on their deviation: $Loss_{depth} = max(\frac{truth}{predict}, \frac{predict}{truth}) + \frac{|predict - truth|}{truth}$. Additionally, we apply a smoothness penalty on the second-order gradients of the prediction values, $Loss_{depth\_smooth} = ||\Delta predict||_1$.

5.1.6 Evenly Cascading Network Architecture

We adopt the low parameter evenly cascaded convolutional network (ECN) architecture as our backbone network design [90]. The ECN network aggregates multilevel feature streams to make predictions. The low level features (Fig. 5.2, light blue blocks) are scaled with bilinear interpolation and improved throughout the whole encoding-decoding structure via residual learning. Along that, the network also progressively generates high level features (Fig. 5.2, darker blue blocks) in the encoding stage. The decoding stage proceeds reversely, the high level features are transformed by convolution and progressively merged back to the low level features to enhance them. Skip links (white arrows) are also used in the network as in the original U-Net [79].

5.1.7 Prediction of Depth and Component Weights

In the decoding stage, we make predictions using features at different resolutions and levels (Fig. 5.2). Initially, both high and low-level coarse features are used to predict a backbone prediction map. The prediction map is then upsampled and merged into existing feature maps for refinements in the remaining decoding layers. In the middle stage, high level features as well as features in the encoding layers are merged into the low level features to serve as modulation streams. The enhanced lower level features are used to estimate the prediction residue, which are usually also low-level structures. The residue is add to the current prediction map to refine it. The final prediction map is therefore obtained through successive upsamplings and refinements.

5.2 EV-IMO Dataset

One of our contributions is the collection of the *EV-IMO* dataset - the first event camera dataset to include multiple independently moving objects and camera motion (at high speed motion), while providing accurate depth maps, per-object masks and trajectories at over 200 frames per second. The next sections describe our automated labeling pipeline, which allowed us to record more than 30 high quality sequences with a total length of half an hour. The source code for the dataset generation will be made available, to make it easier to expand the dataset in the future. A sample frame from the dataset is shown in Fig. 5.3.

5.2.0.1 Methodology

Event cameras such as the DAVIS are designed to capture high speed motion and work in difficult lighting conditions. For such conditions classical methods of collecting depth ground truth, by calibrating a depth sensor with the camera, are extremely hard to apply - the motion blur from the fast motion would render such ground truth unreliable. Depth sensors have severe limitations in their frame rate as well. Furthermore it would be impossible to automatically acquire object masks - manual (or semi-automatic) annotation would be necessary. To circumvent these issues we designed a new approach:

1. A static high resolution 3D scan of the objects, as well as 3D room reconstruc-

tion is performed before the dataset recording takes place.

2. The VICON® motion capture system is used to track both the objects and the camera during the recording.

3. The camera center as well as the object and room scans are calibrated with respect to the the VICON® coordinate frame.

4. For every pose update from the VICON motion capture, the 3D point clouds are transformed and projected on the camera plane, generating the per-pixel mask and ground truth depth.

This method allows to record accurate depth at very high frame rate, avoiding the problems induced by frame-based collection techniques. While we acknowledge that this approach requires expensive equipment, we argue that our method is superior for event-based sensors, since it allows to acquire the ground truth at virtually any event time stamp (by interpolating poses provided at 200 Hz) - a property impossible to achieve with manual annotation.

5.2.0.2 Dataset Generation

Each of the candidate objects (up to 3 were recorded) were fitted with VICON® motion capture reflective markers and 3D scanned using an industrial high quality 3D scanner. We use RANSAC to locate marker positions in the point cloud frame and using acquired point correspondences we transform the point cloud to the world frame at every update of the VICON. To scan the room, we place reflective markers

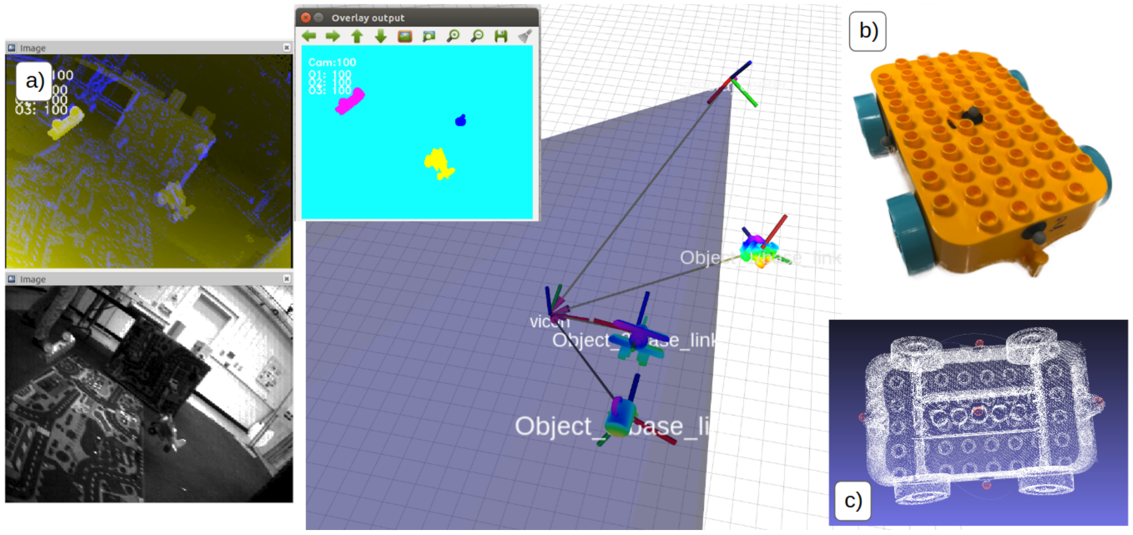


Figure 5.3: a) - The main interface of the automatic annotation tool. Camera cone of vision, depth and motion masks are visible. b) - Example object used in the dataset. c) 3D scan of the object.

on the Asus Xtion RGB-D sensor and use the tracking as an initialization for global ICP alignment.

To compute the position of the DAVIS camera center in the world frame we follow a simple calibration procedure, using a wand that is tracked by both VICON and camera. The calibration recordings will be provided with the dataset. The static pointcloud is then projected to the pixel coordinates (x, y) in the camera center frame following equation 5.2:

$$(x, y, 1)^T = KCP_{davis}^{-1}P_{cloud}X_i \tag{5.2}$$

Here, K is the camera matrix, P_{davis} is a 4×4 transformation matrix between reflective markers on the DAVIS camera and the world, C is the transformation between reflective markers on the DAVIS and DAVIS camera center, P_{cloud} is the transformation between markers in the 3D pointcloud and reflective markers in the world coordinate frame, and X_i is the point in the 3D scan of the object.

Or dataset provides high resolution depth, pixel-wise object masks and accurate camera and object trajectories. We additionally compute, for every depth ground truth frame, the instantaneous camera velocity and the per-object velocity in the camera frame, which we use in our evaluations. We would like to mention, that our dataset allows to set varying ground truth frame rate - in all our experiments we generated ground truth at 40 frames per second.

5.2.1 Sequences

A short qualitative description of the sequences is given in Table 5.1. We recorded 6 sets, each consisting of 3 to 19 sequences. The sets differ in the background (in both depth and the amount of texture), the number of moving objects, motion speeds and lighting conditions.

A note on the dataset diversity: it is important to note, that for event-based cameras (which capture only edge information of the scene) the most important factor of diversity is the variability on *motion*. Different motions create 3D event clouds which vary significantly in their structure, even with similar backgrounds. Nevertheless, we organize our sequences into four background groups - *'table'*, *'boxes'*, *'plain wall'* and *'floor'* (see Fig 5.4), with the latter two having varying amounts of texture - an important factor for event cameras. We also include several tabletop

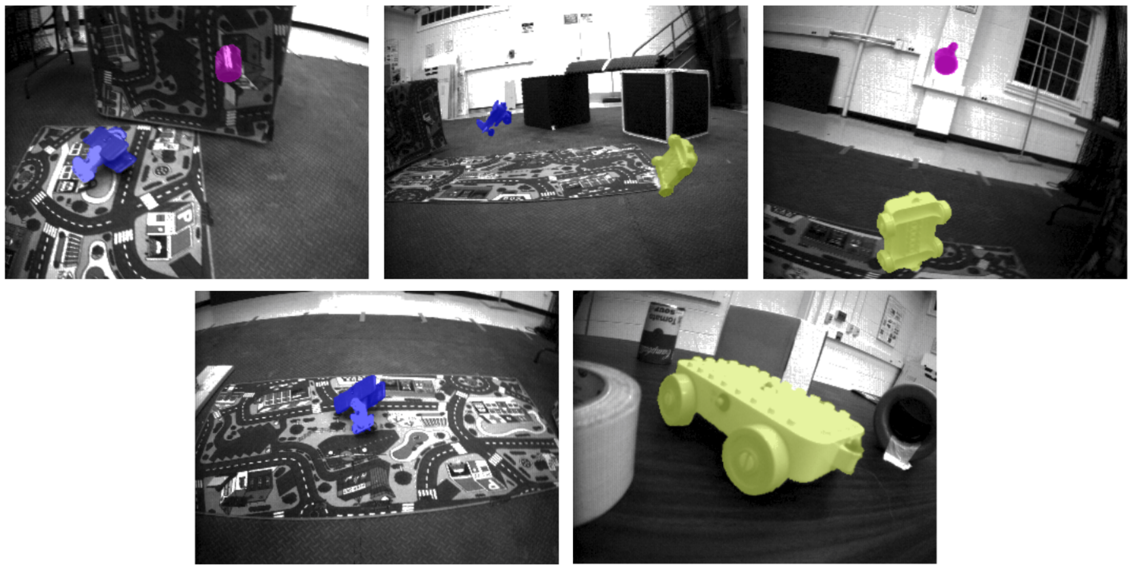


Figure 5.4: Types of background geometry featured in the EV-IMO dataset (from left to right): ’table’, ’boxes’, ’plain wall’, ’floor’ and ’tabletop’.

scenes, with clutter and independently moving objects.

5.3 Experiments

We present the first method for motion segmentation on event-based data using neural networks. Learning for this task is challenging because the data from event-based sensors is extremely sparse (coming only from object edges). Nevertheless, we were able to estimate the full camera egomotion, a dense depth map, and the 3D linear velocities of the independently moving objects in the scene.

We trained our networks with the Adam optimizer using a starting learning rate of 0.01 with cosine annealing for 50 epochs. The batch size was 32. We distributed the training over 4-Nvidia GTX 1080Ti GPUs and the training finished within 24 hours. Inference runs at over 100 fps on a single GTX 1080Ti.

Table 5.1: EV-IMO sequences

	background	speed	texture	occlusions	objects	light
Set 1	boxes	low	medium	low	1-2	normal
Set 2	floor/wall	low	low	low	1-3	normal
Set 3	table	high	high	medium	2-3	normal
Set 4	tabletop	low	high	high	1	normal
Set 5	tabletop	medium	high	high	2	normal
Set 6	boxes	high	medium	low	1-3	dark / flicker

In all experiments, we trained on *'box'* and *'floor'* backgrounds, and tested on *'table'* and *'plain wall'* backgrounds (see Table 5.1 and Fig. 5.4). For the Intersection over Union (IoU) scores, presented in Table 5.2 the inferenced object mask was thresholded at 0.5.

Our baseline architecture contains approximately 2 million parameters. It has 32 initial hidden channels and a growth rate of 32. The feature scaling factors are $\frac{1}{2}$ and 2 for the encoding and decoding. Overall the networks have 4 encoding and 4 decoding layers.

However, for many applications (such as autonomous robotics), precision is less important than computational efficiency and speed. We train an additional shallow network with just 40 thousand parameters. In this setting we have 8 initial hidden channels and a growth rate of 8. The feature scaling factors are $\frac{1}{3}$ and 3 respectively. The resulting networks have only 2 encoding and 2 decoding layers. We found that the 40k network is not capable of predicting object velocity reliably, but it produces reasonable camera egomotion, depth and motion masks, which can be tracked to extract the object translational velocities.

5.3.0.1 Qualitative Evaluation

Apart from the quantitative comparison we present a qualitative evaluation in Figs. 5.6 and 5.5. The per-object pose visualization (Fig. 5.6, columns 4 and 5) directly map the 3D linear velocity to RGB color space. The network is capable of predicting masks and pixel-wise pose in scenes with different amount of motion,

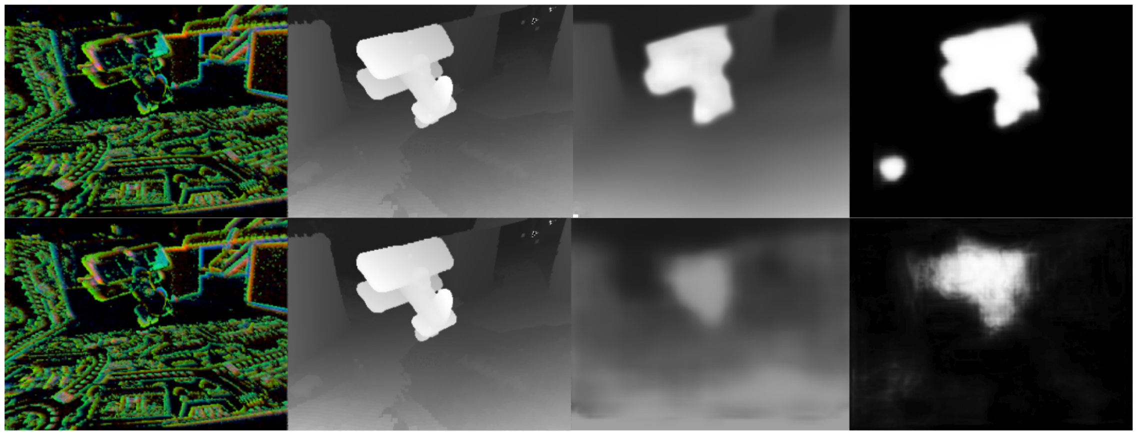


Figure 5.5: Comparison of the full network inference quality (2M parameters, top row) with the small version (40k parameters, bottom row)

number of objects or texture.

Fig. 5.5 shows how the quality of the depth and motion mask output is affected by reducing the size of the network. While the background depth is affected only to a small degree, the quality of the object mask and depth suffers notably.

5.3.0.2 Segmentation and Motion Estimation

To evaluate the linear components of the velocities, for both egomotion and object motion, we compute the classical Average Endpoint Error (AEE). Since our pipeline is monocular, we apply the scale from the ground truth data in all our evaluations. To account for the rotational error of the camera (which does not need scaling) we compute the Average Relative Rotation Error $RRE = \|logm(R_{pred}^T R_{gt})\|_2$. Here $logm$ is the matrix logarithm, and R are Euler rotation matrices. The RRE essentially amounts to the total 3-dimensional angular rotation error of the camera. We also extract several sequences featuring fast camera motion and evaluate them

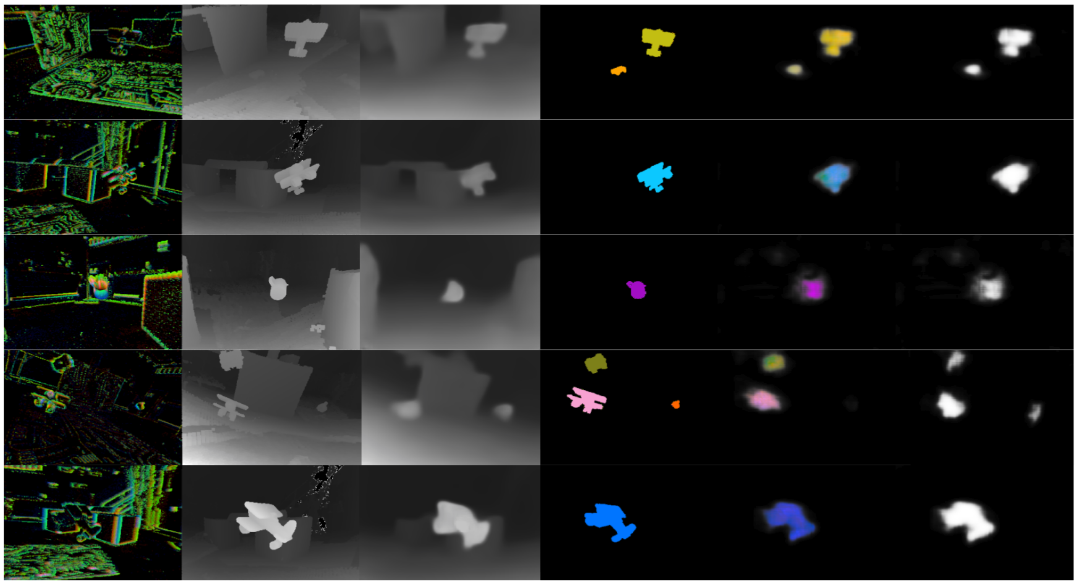


Figure 5.6: Qualitative results from our evaluation. The table entries from left to right: DVS input, ground truth for depth, network output for depth, ground truth pixel-wise pose, predicted pixel-wise pose, predicted motion mask. Examples were collected from *EV-IMO* dataset. Best viewed in color.

separately. We present AEE in m/s, and $ARRE$ in radians/s in Table 5.2.

We compute the averaged linear velocity of the independently moving objects within the object mask (since it is supplied by the network per pixel) and then also compute AEE. To evaluate the segmentation we compute the commonly used Intersection over Union (IoU) metric. Our results are presented in Table 5.2.

5.3.0.3 Comparison With Previous Work

As there is no public code available for monocular SfM on event-based data, we evaluate on a 4-parameter motion-compensation pipeline [76]. We evaluated the

Table 5.2: Evaluation on segmentation and motion estimation. The numbers in braces are values for the 40k version of the network. *AEE* is in m/s, *ARRE* is in rad/s.

	Cam *AEE*	Cam *ARRE*	Obj AEE	IOU
table	0.07 (0.09)	0.05 (0.08)	0.19	0.83 (0.63)
plainwall	0.17 (0.23)	0.16 (0.24)	0.38	0.75 (0.58)
fastmotion	0.23 (0.28)	0.20 (0.26)	0.43	0.73 (0.59)

egomotion component of the network on a set of sequences without IMOs and with no roll/pitch egomotion and with planar background found in *'plain wall'* scenes, to make [76] applicable ([76] does not account for depth variation). Table 5.3 reports the results in m/s for the translation and in rad/s for the rotation. We were not able to achieve any meaningful egomotion results on scenes with high depth variation for [76].

Table 5.3: Comparison of $EV-IMO$ with [76].

	AEE	*ARRE*
EV-IMO	0.024	0.095
Classical [76]	0.031	0.134

We also evaluate our approach against a recent method [90] - *ECN* network, which estimates optical flow and depth on the event-based camera output. The method was originally designed and evaluated on a road driving sequence (which features a notably more simple and static environment, as well as significantly rudimentary egomotion). Still, we were able to tune [90] and train it on *EV-IMO*. We provide the comparison for the depth for our baseline method, the smaller version of our network (with just 40k parameters) and *ECN* in Table 5.4.

We conducted the experiments on sequences featuring a variety of backgrounds and textures (the lack of texture is a limiting factor for event-based sensors). Even though *ECN* [90] was not designed to segment independently moving objects, the comparison is valid, since it infers depth from a single frame. Instead, we attribute the relatively low performance of [90] to a significantly more complex motion present in EV-IMO dataset, as well as more diverse depth background.

5.4 Conclusions

Event-based sensing promises advantages over classic video processing in applications of motion processing because of the data's unique properties of sparseness, high temporal resolution, and low latency. In this chapter, we presented a compositional NN pipeline, which uses a combination of unsupervised and supervised components and is capable of generalizing well across different scenes. We also presented the first ever method of event-based motion segmentation with evaluation of both camera and object motion, which was achieved through the creation of a

Table 5.4: Evaluation of the depth estimation

	Error metric			Accuracy metric		
	Abs Rel	RMSE log	SILog	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
			Baseline Approach			
plain wall	0.16	0.26	0.07	0.87	0.95	0.97
cube background	0.13	0.20	0.04	0.87	0.97	0.99
table background	0.31	0.32	0.12	0.74	0.90	0.95
			40k Network			
plain wall	0.24	0.33	0.11	0.75	0.90	0.95
cube background	0.20	0.26	0.07	0.77	0.92	0.97
table background	0.33	0.34	0.15	0.65	0.87	0.95
			ECN			
plain wall	0.67	0.59	0.33	0.27	0.52	0.80
cube background	0.60	0.56	0.30	0.29	0.53	0.78
table background	0.47	0.48	0.23	0.45	0.69	0.86

new state of the art indoor dataset - *EV-IMO*, recorded with the use of a VICON® motion capture system.

Future work will delve into a number of issues regarding the design of the NN and usage of event data. Specifically, we consider it crucial to study event stream augmentation using partially or fully simulated data. We also plan to investigate ways to include tracking and connect the estimation over successive time slices, and investigate different alternatives of including the grouping of objects into the pipeline.

Chapter 6: Network Deconvolution

Convolution is a central operation in Convolutional Neural Networks (CNNs), which applies a kernel or mask to overlapping regions shifted across the image. In this work we show that the underlying kernels are trained with highly correlated data, which leads to co-adaptation of model weights. To address this issue we propose what we call *network deconvolution*, a procedure that aims to remove pixel-wise and channel-wise correlations before the data is fed into each layer. The deconvolution can be efficiently calculated at a fraction of the cost of a convolution layer. We show that by removing this correlation we are able to achieve better convergence rates during model training with superior results *without* the use of batch normalization on the CIFAR-10, CIFAR-100, MNIST, Fashion-MNIST datasets, as well as against reference models from ”model zoo” on the ImageNet standard benchmark.

6.1 Introduction

Images of natural scenes depict homogeneous color and texture regions delineated by edges, and adjacent pixels are statistically highly correlated [91,92]. We can imagine that this correlation is somehow induced by an external process: in the same way correlations are introduced when a Gaussian kernel *blurs* an input image, we can think of the natural images themselves as being blurred and correlated by some unknown operator, which we call the *tangling* by nature (Figure 6.1). This tangling complicates object recognition tasks as adjacent pixels contain highly redundant information, and convolutional networks endure the cost of processing this information without accruing substantial benefits. As applying a Gaussian blur complicates human recognition (leading to the need for corrective lenses), the tangling effect in natural images may also complicate machine learning in neural networks, requiring its own method of correcting the image, a method we call *network deconvolution*.

Convolutional Neural Networks (CNNs) [93], the most widely used networks in visual learning, have demonstrated superior performance in a variety of tasks ranging from Computer Vision [94], over Natural Language Processing [95], to Reinforcement Learning [96], owing to their ability to learn their own convolutional kernels that extract meaningful features from the input.

In CNNs, resulting from a combination of the tangling effect and the fact that the kernel shifts only slightly between each receptive field, layers of the neural network are in reality re-learning much of the same information over and over, a factor that we believe slows down learning. We refer to the correlation between the raw pixels in a single image or feature map as the *pixel-wise correlation*. Similarly, in the case of different channels of a hidden layer of the network, there is a strong correlation or "cross-talk" between these channels because these channels are provided with statistically very similar inputs; we refer to this as *channel-wise correlation*. The goal of this paper is to remove redundant features that will not lead to effective learning. In this paper we propose two methods for attacking this problem:

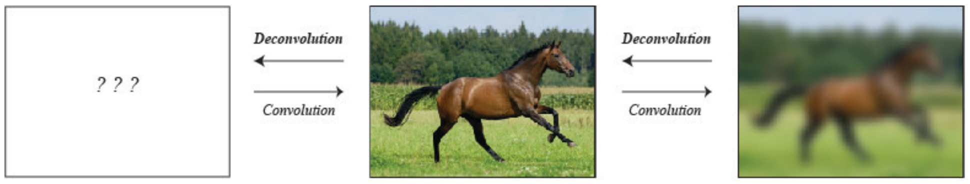


Figure 6.1: A "real world" image that the retina might see, and a convolution applied. Performing convolution on this image using some kind of filter, such as a typical Gaussian kernel here, adds in correlations to the resulting image, giving the typical "blur" phenomenon, which makes object recognition more difficult. Removing this blur is the process of *deconvolution*. Neural networks iteratively learn their own convolutional kernels, but many of these iterations are applied on highly correlated image patches, which can slow down learning. Just like glasses help a human see better by removing blur, the proposed deconvolutional operators decorrelate features so that neural networks "see" better.

Full deconvolution is the removal of both pixel-wise and channel-wise correlations, whereas *channel deconvolution* is the removal of just channel-wise correlations. We define *network deconvolution* as the application of full deconvolution to the input at every layer of a network.

Our contributions are the following:

- We introduce a *pixel-wise* decorrelation at each layer of the network, which we call *network deconvolution*, that aims to ensure that at every step only sparse and discriminative data is used for learning.

- We propose an novel subsampling based acceleration so that the deconvolution can be carried out at a cost fractional to the corresponding convolution layer.

- We demonstrate consistently superior accuracy of our approach on the CIFAR-10, CIFAR-100, Fashion-MNIST, and ImageNet datasets compared to batch normalization, showing that deconvolution can be used as a generic procedure in a variety of architectures.

6.2 Related Work

6.2.1 Normalization and Whitening

Since its introduction, batch normalization has been the main normalization technique [97] to facilitate the training of deep networks using stochastic gradient descent (SGD). Many techniques have been introduced to address cases for which batch normalization does not perform well. These include training with a small batch size [83], and on recurrent networks [98]. However, to our knowledge, none of these methods has demonstrated superior performance on the ImageNet dataset.

In the signal processing community, our network deconvolution is what would be referred to as a *whitening* mechanism. There have been previous attempts to whiten the feature channels and to utilize second-order information. For example, in [99,100] the authors approximate second-order information using the Fisher information matrix. However, we found that implementations of this form are unstable for deep networks, a fact that was already noted by the authors of [100]. One reason is that these methods directly compute a matrix inversion and that is excluded from the back propagation. We point out in Eq. 6.4 that there is a first-order correction term that needs to be included to ensure accurate computation of gradients. More importantly, previous methods only addressed the correlation of different channels (that is, across features), but did not consider correlation between nearby pixels (within a single feature). This is critical, because kernels need to be trained from the *tiny discrepancies* between the highly correlated neighboring pixels.

The most related work is [86,101–103], which attempts to decorrelate the data being fed into each layer, but it only considers correlation across the channels. As far as we are aware, our work is the first to perform *pixel* decorrelation.

6.3 The Deconvolution Operation

6.3.1 Definition

In this section we describe the deconvolution operation, and give some intuition for why it helps the network learn. The task of the deconvolution operation is to stretch a vector of random variables such that each random variable is independent and identically distributed (i.i.d) in the statistical sense. The goal of this is to remove the correlation that a variable has with another variable. We will cover two types of deconvolutions, *pixel* and *channel* deconvolution. *Network Deconvolution* is thus the application of these at each layer of the neural network.

Given a batch of vectors $x_1, x_2, ...x_N$, organized as rows of column vectors, we calculate their covariance matrix $Cov_x = E(x-\mu_x)^T(x-\mu_x) = \frac{1}{N}\sum_{j=1}^{N}(x_j - \mu_x)^T(x_j-\mu_x) = \frac{1}{N}X^TX$. Here X is the centered matrix, such that the mean of each column is 0. We then calculate an approximated inverse square root of the covariance matrix $D_x = Cov_x^{-\frac{1}{2}}$ and multiply this with the centered vectors $(x - \mu_x) \cdot D_x$; this decorrelates each dimension (pixel, channel) from other dimensions. In a sense, we are deconvolving the *tangling* originating from the statistics of the raw image or resulting from the training process. The untangling operator is this inverse square root of the covariance matrix. If computed perfectly, this would turn the covariance of the resulting covariance matrix, $\frac{1}{N}X^T X$, into the identity matrix I.

When dealing with multiple feature channels in a convolution layer, the data matrix is constructed by flattening (large) image patches in each channel into columns and then concatenating these columns. As noted previously, there are two types of correlation that affect the learning of weights: one is the cross-channel correlation, the other is the more prominent autocorrelation between nearby pixels.

In Fig. 6.2 (top right) we show the calculated covariance matrix of the data matrix X in the first layer of a VGG network [104] from ImageNet. The first layer is a 3×3 convolution that mixes RGB channels. The total dimension of the weights is 27, the corresponding covariance matrix is 27×27. The diagonal blocks correspond to the pixel-wise correlation within 3×3 neighborhoods. The off diagonal blocks correspond to correlation of pixels across different channels. Generally natural images demonstrate stronger pixel-wise correlation than cross-channel correlation, as the diagonal blocks are significantly brighter than the off diagonal blocks.

We denote the deconvolution operation as D_i, the input to next layer x_{i+1} can be written as:

$$x_{i+1} = f_i \circ W_i \circ D_i \circ x_i \tag{6.1}$$

where $\circ$ is the (right) matrix multiplication operation, x_i is the input coming from the $i-$th layer, D_i is the deconvolution operation on that input, and W_i is the weights in the layer. In general, the deconvolution operation removes the correlations between the columns.

6.3.2 Expected Isometry via Deconvolution

For simplicity, we assume on all layers the activation function is a sample-variant matrix multiplication. The popular *ReLU* [105] activation falls into this category. It is important to note that essentially this function multiplies the input vector by a diagonal matrix with entries either 1 or 0, based on the sign of the input sample. Because of this the average effect of *ReLU* on a batch of data is an attenuating effect, in that it can only either allow values to pass or turn them off, (setting them to 0), and we assume that its operator norm is smaller than 1. We investigate the transform of inputs to the next linear/convolution layer.

$$z_{i+1} = D_{i+1} \circ f_i \circ W_i \circ z_i, \tag{6.2}$$

where z_i is the (deconvolved) input to the linear layer and z_{i+1} is the (deconvolved) input to the next linear layer.

If we assume the output dimension and the input dimension to be the same, and if both z_i and z_{i+1} are i.i.d., then $F_i = D_{i+1} \circ f_i \circ W_i$ is an *isometry* on expectation, which keeps the statistical properties of the signal unchanged in the forward propagation. An ideal isometry also keeps properties of gradients unchanged in the backward propagation.

$$\frac{\partial Loss}{\partial z_i} = \frac{\partial Loss}{\partial z_{i+1}} \frac{\partial z_{i+1}}{\partial z_i} \tag{6.3}$$

$$\frac{\partial z_{i+1}}{\partial z_i} = D_{i+1} \circ f_i \circ W_i + \frac{\partial D_{i+1}}{\partial z_i} \circ f_i \circ W_i \approx D_{i+1} \circ f_i \circ W_i \approx Rotation \tag{6.4}$$

To emphasize this property on the backward propagation of the error signal: if $f_i \circ W_i$ has a diminishing effect on the gradients, then D_{i+1} is expected to raise the values to counteract.

It is important to note that the procedure does not change the learning problem, as the standard network training is solving for $W_i^* = W_i \circ D_i$. Instead, this *change of variables* procedure removes the correlations between feature columns and improves the conditioning of the optimization problem.

6.3.3 Accelerated Convergence

Another favorable property of deconvolution is that after applying it, optimal solutions may be computed in one iteration. Assume we are given a linear regression problem with L_2 loss:

$$y = Xw, \tag{6.5}$$

where y is the output, X is the input, and w is the weight matrix we are trying to solve for. Then, we have as an L_2/min loss function:

$$Loss_{L_2} = \frac{1}{2}\|y - \hat{y}\|^2 = \frac{1}{2}\|Xw - \hat{y}\|^2. \tag{6.6}$$

An explicit solution is given as $\frac{\partial Loss_{L_2}}{\partial w} = X^t(Xw - \hat{y}) = 0$

$$w = (X^tX)^{-1}X^t\hat{y} \tag{6.7}$$

If the input is deconvolved, we have $\frac{1}{N}X^tX = I$, then $w = \frac{1}{N}X^t\hat{y}$. On the other hand, to conduct one iteration of gradient descent on Eq. 6.6, we have:

$$w_{new} = w_{old} - step\_len \times \frac{1}{N}(X^tXw_{old} - X^t\hat{y}). \tag{6.8}$$

If the input is deconvolved then $step\_len = 1$ is optimal and with one iteration we will have $w_{new} = \frac{1}{N}X^t\hat{y}$.

In our experiments section, we show that the deconvolution operation does in fact significantly speed up training on a variety of standard benchmark tasks.

6.3.4 How Network Deconvolution is Applied in a CNN

In an effort to simplify and understand the calculation of a single kernel with the input data in a batch, we can rewrite the convolution operation in matrix form as: $x * kernel = Xw$. In essence, we are converting the entire process of convolution / shifting over the image into one large matrix multiplication. In the 2-dimensional case, w is the flattened 2D $kernel$. The first column of X corresponds to the flattened image patch of $x[1 : H-k, 1 : W-k]$. Neighboring columns correspond to shifted patches of x: $X[:, 2] = vec(x[1 : H-k, 2 : W-k+1]), ..., X[:, k^2] = vec(x[k : H, k : W])$. This is done using the commonly used function *im2col* (See Fig. 6.2). When formulated in this manner, the resulting combined matrix is extremely ill-posed. This ill-posedness slows down the training algorithm, and cannot be addressed by normalization methods [97]. Solving for the kernel given the input data X and the output data y, is known as a kernel estimation problem [106]. It takes tens or hundreds of gradient descent iterations to converge to a practically close enough

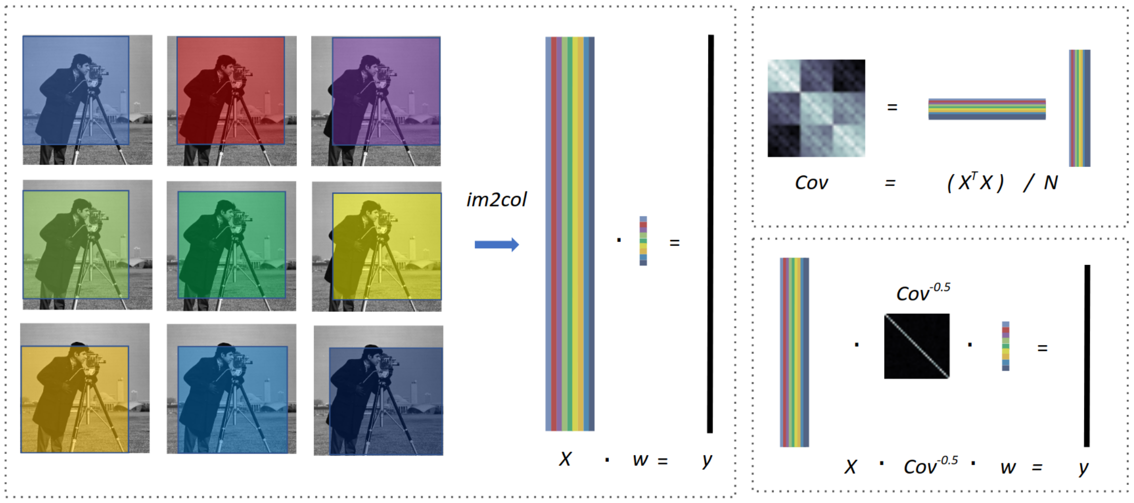


Figure 6.2: (Left) Given a single channel image, and a 3×3 kernel, the kernel is first flattened into a 9 dimensional vector w. The 9 image patches, corresponding to the image regions each kernel entry sees when overlaying the kernel over the image and then shifting the kernel one pixel each step, are flattened into a tall matrix X. It is important to note that because the patches are shifted by just one pixel, the columns of X are highly correlated. The output y is calculated with matrix multiplication Xw, which is then reshaped back into a $2D$ image. (Top Right) In a convolution layer the matrix X and Cov is calculated from Algorithm 1. (Bottom Right) The pixel-wise and group-wise correlation is removed by multiplying this X matrix with with $Cov^{-\frac{1}{2}}$, before the weight training.

solution. However, we emphasize that a close form solution exists as given by Eq. 6.7.

For convolutional networks, we usually have multiple input feature channels and multiple kernels in a layer. We vectorize and concatenate all the kernels to get w and follow Algorithm 1 to construct X and $D \approx (Cov + \epsilon \cdot I)^{-\frac{1}{2}}$. Here $\epsilon \cdot I$ is introduced to improve stability. We then apply the deconvolution operation D to X to remove the correlation between neighboring pixels and across different channels. The deconvolved data is then multiplied with w. The full equation becomes $y = X \cdot D \cdot w$(Fig. 6.2). The output matrix y is then reshaped into the output shape of the layer.

6.3.5 Efficient Calculation of the Inverse Square Root of the Covariance Matrix

We compute the approximate inverse square root of the covariance matrix at low cost using the Denman-Beavers iteration method [85] in a simple and straightforward fashion. This method is important because there is a first-order correction (Eq. 6.4) term that needs to be included to avoid accumulating errors when training deep networks. Given a symmetric positive definite covariance matrix Cov, Denman-Beavers iterations start with initial values $Y_0 = Cov$, $Z_0 = I$. The iteration is defined as: $Y_{k+1} = \frac{1}{2}Y_k(3I - Z_kY_k)$, $Z_{k+1} = \frac{1}{2}(3I - Z_kY_k)Z_k$, and $Z_k \rightarrow Cov^{-\frac{1}{2}}$ [84]. It is important to point out a practical implementation detail: when we have Ch_{in} input feature channels, and the kernel size is $k \times k$, then the size of the covariance matrix is $(Ch_{in} \times k \times k) \times (Ch_{in} \times k \times k)$. The covariance matrix becomes large in deeper layers of the network. Inverting such a matrix is slow and highly unstable. In our implementation, we evenly divide the feature channels Ch_{in} into smaller G groups [83, 86, 101]. Usually we set $G \sim 16$. The mini-batch covariance of a small groups has a manageable size of $(G \times k \times k) \times (G \times k \times k)$. Denman-Beavers iterations are therefore conducted on small matrices. We notice that only a few (~ 5) iterations are necessary to deconvolve both the pixel correlation and the (grouped) channel correlation, leading to fast convergence and better results. Solving for the inverse square root takes $O((k \times k \times G)^3)$.

The computation of the covariance matrix has complexity $O(H \times W \times k \times k \times G \times G \times \frac{Ch_{in}}{G}) = O(H \times W \times k \times k \times Ch_{in} \times G)$. In comparison, the computational complexity of a regular convolution layer has a similar complexity of $O(H \times W \times k \times k \times Ch_{in} \times Ch_{out})$. However, we note that in a direct implementation, the runtime of our training using deconvolution is slower than convolution using wallclock as a metric. This is due to the lack of support in implicit calculation of the matrix in existing libraries. Here we propose a simple $S = 4\times$ subsampling technique that speeds up the operation by $16\times$ while maintaining the training quality. With this acceleration, the computational complexity of deconvolution operation is reduced to only a fraction of the corresponding convolution layer. (More details can be found in the supplementary material.) Without further optimization, our training speed is roughly the same with training a network using batch normalization on the ImageNet dataset. Note that the testing stage, deconvolution is faster than batch normalization because the deconvolution can be merged with the kernels using the associate rule of matrix multiplication.

The overall complexity is $O(\frac{H\times W\times k\times k\times Ch_{in}\times G}{S\times S} + (k\times k\times G)^3)$, which is smaller than the convolution operation in practice.

Algorithm 1 Computing the Deconvolution Matrix

```
1: **Input:** C channels of input features $[x_1, x_2, ..., x_C]$
2: **for** $i \in \{1, ..., C\}$ **do**
3:     $X_i = im2col(x_i)$
4: **end for**
5: $X = [X_1, ..., X_C]$ %Vertically Concatenate
6: $X = Reshape(X)$ %Divide columns into $G \times k \times k$ groups
7: $Cov = \frac{1}{N}X^tX$
8: $D \approx (Cov + \epsilon \cdot I)^{-\frac{1}{2}}$
```

6.3.6 Sparse Representations

Our deconvolution applied at each layer removes the pixel-wise and channel-wise correlation and transforms the original dense representations into sparse representations, without losing information. This is a desired property of the input data, and there is a whole field developed around sparse representations [91, 92]. In Fig. 6.3, we visualize the deconvolution operation on an input and show how the resulting representations (6.3(d)) are much sparser than the original image (6.3(b)). This also holds true for hidden layer representations. We show in the supplementary material that sparse representations has made classic regularizations more effective.

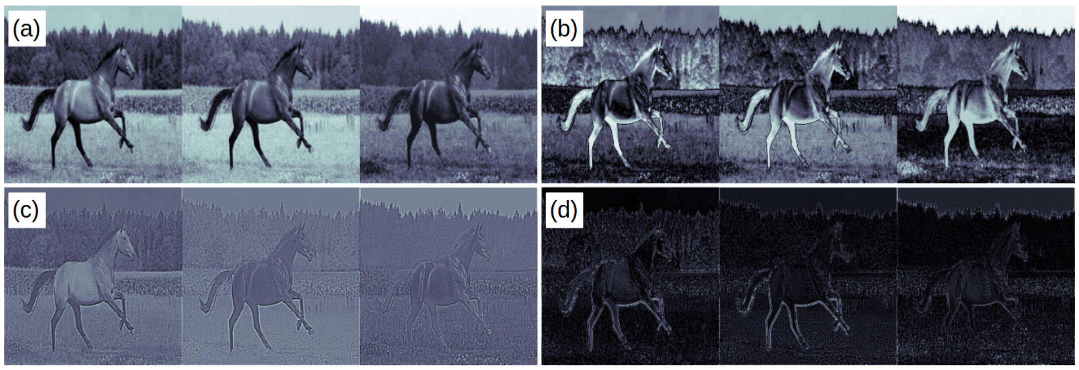


Figure 6.3: As giving the image zero-mean is commonly done as a preprocessing step, we visualize the three RGB channels of the zero-meaned image transformation on an example input image. (a) The input image (min-max normalized for visualization). (b) The absolute value of the zero-meaned input image. (c) The deconvolved input image (min-max normalized, so gray areas stands for 0). (d) The absolute value of the deconvolved image.

6.4 Experiments

We now describe experimental results validating that network deconvolution is a powerful and successful tool for sharpening the data. In fact, our experiments show that it outperforms identical networks using batch normalization [97], a widely used method for input normalization. As we will see across all experiments, deconvolution not only improves the final accuracy but also decreases the amount of iterations it takes to learn a reasonably good set of weights in a small number of epochs.

We make a note to plot and compare against previous work [101,102] (see Related Work) that only applied per-channel decorrelation and show that our network deconvolution technique outperforms only a network channel-wise deconvolution.

Linear Regression with L_2 loss and Logistic Regression: As a first experiment, we ran network deconvolution on a simple linear regression task to show its efficacy. We select the Fashion-MNIST dataset, which contains 60000 28×28 article images for training and 10000 for testing. The dataset has 10 categories. It is noteworthy that with binary targets and the L_2 loss, the problem has an explicit solution if we feed the whole dataset as input. This problem is the classic kernel estimation problem, where we need to solve for 10 optimal 28×28 kernels to convolve with the inputs and minimizes the L_2 loss with the binary targets. During our experiment, we notice it is important to use a small learning rate $0.02 - 0.1$ for vanilla SGD training to prevent divergence. However, we notice with deconvolution, it is possible to use the optimal learning rate 1.0 and get high accuracy as well. It takes ~ 5 iterations to get to a low cost under the mini-batch setting (Fig. 6.4(a)). This even holds if we change the loss to logistic loss(Fig. 6.4(b)).

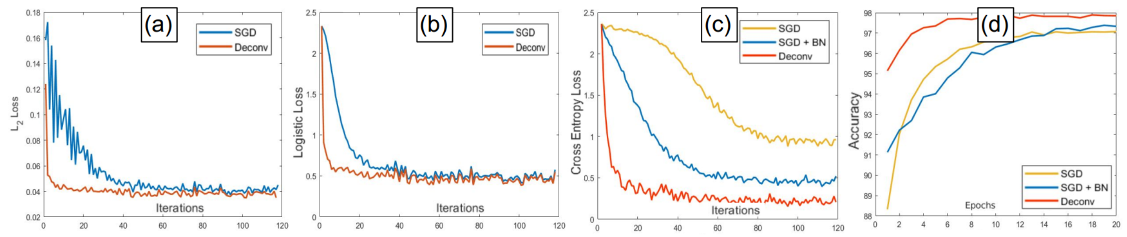


Figure 6.4: (a-b): Regression Losses on the Fashion-MNIST dataset showing the effectiveness of deconvolution versus batch normalization on a non-convolutional layer type. (a) One layer, linear regression model with L_2 loss. (b) One layer, linear regression model with logistic loss. (c-d): Results of a 3-hidden-layer Multi Layer Perceptron (MLP) network on the MNIST dataset.

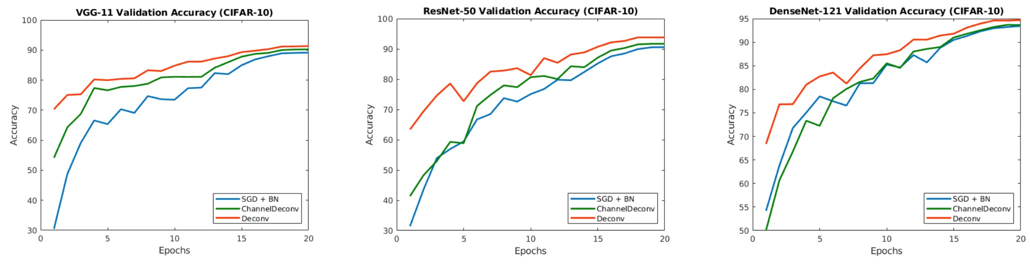
CIFAR-10	DC	BN	CD	CIFAR-100	DC	BN	CD
VGG-11	**91.33**	89.15	90.23	VGG13	**74.74**	70.57	74.31
DenseNet-121	**94.71**	93.45	93.65	Densenet121	**80.27**	79.2	79.09
ResNet-50	**94.05**	90.6	91.7	Resnet50	**80.43**	77.78	76.62

Table 6.1: (Left) Comparison on the CIFAR-10 dataset of final validation accuracy percentage after 20 epochs of VGG-11, ResNet50, and DenseNet-121 comparing full pixel and channel network deconvolution (DC), network channel deconvolution only (CD), and batch normalization (BN). (Right) same comparison but with VGG-13 instead of 11, on the CIFAR-100 dataset, ran for 100 epochs.

Convolutional Networks on CIFAR-10/100: We ran deconvolution on the CIFAR-10 (Fig. 6.5, Table 6.1(left)) and CIFAR-100 (Fig. 6.6, Table 6.1(right)) datasets, where we again compare the use of network deconvolution versus the use of batch normalization and the use of network channel-only deconvolution. Across different network architectures for both datasets, deconvolution significantly improves the final accuracy on these well-known datasets. We find that deconvolution leads to faster convergence, On the CIFAR-10 dataset, with 20 epochs of training leading to results that were only achievable using standard training for over 100 epochs.

As settings we remove all batch normalization in the networks and replace them with deconvolution before each convolution/fully-connected layer. For convolutional layers, we split the feature channels into 16 groups before calculating the covariance matrix. For fully-connected layers, we split the channels into 512 groups.

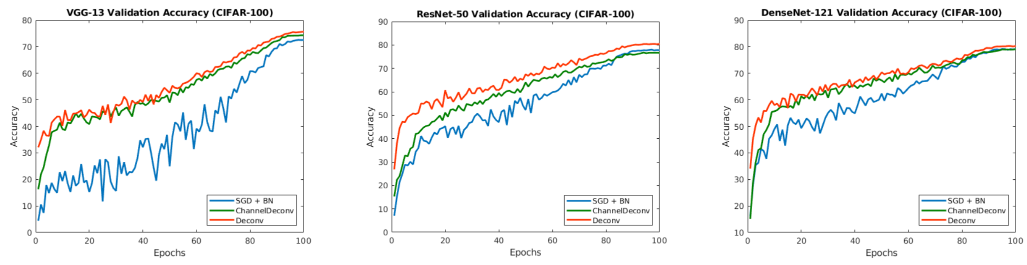
Here, we showcase the generalizability of deconvolution across a variety of different CNN architectures. We report results using some of the most popular architectures, ResNet [107], VGGNet [104], and DenseNet [78], where the standard batch normalization procedure has been replaced with pixel and channel deconvolution. For the CIFAR-10 experiments, we used a batch size of 128, and a weight decay of .001, to demonstrate the speed of convergence. For CIFAR-100, we used a batch size of 256 and weight decay of 005, with 100 epochs. **Convolutional Networks on Ima-**



(a) (b) (c)

Figure 6.5: Results of training various networks on the CIFAR-10 dataset.

geNet: We tested two widely-used model architectures (VGG-11, ResNet-18) from the PyTorch model zoo and find significant improvements on both networks over



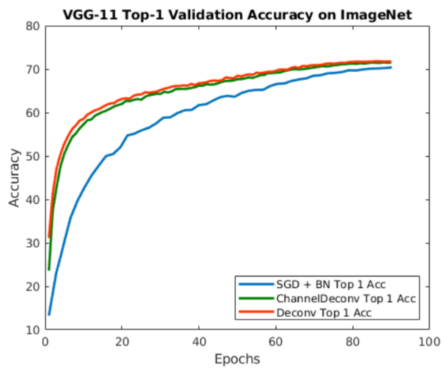
(a) (b) (c)

Figure 6.6: Results of training various networks on the CIFAR-100 dataset. The red curve, full deconvolution (pixel and channel) performs better across all architectures.

the reference models in the model zoo. Notably, for the VGG-11 network, we notice our method has led to significant improved accuracy, the top-1 accuracy is even higher than 71.55%, reported by the reference VGG-13 model trained with batch normalization. The improvement introduced by network deconvolution (+2.72%) is twice as large as the introduction of batch normalization (+1.36%). This fact also suggests us improving the training methods may lead to more improvements than improving the architecture.

As settings we keep most of the default settings to train the two models. For deconvolution, we use the settings as described above with only one modification. For deconvolution of the fully-connected layers, we split the features into 32 groups. Our conjecture is that for this complex dataset, the full feature covariance structure of the whole dataset is under-represented with a small batch size. However, dividing these feature into 32 groups alleviates this issue. The networks are trained for 90 epochs with batch size 256, weight decay 0.0001. The initial learning rates are 0.1, 0.01 respectively for ResNet-18 and VGG-11 as described in the paper. We used cosine annealing to smoothly decrease the learning rate to compare the curves.

Non-Convolutional, Multi-layer Perceptron Networks: Finally, we ran experiments to confirm that the network deconvolution procedure can extend to non-convolutional layers via channel deconvolution, and is thus capable of improving classification on datasets not just important to computer vision but also to the broader machine learning community. We constructed a 3-layer fully-connected network that has 128 hidden nodes in each layer. For the activation function, we use the *sigmoid*. As with the other experiments, we compare the use of batch normalization, channel-only deconvolution, and full deconvolution (pixel and channel). Indeed, applying deconvolution to MLP networks outperforms batch normalization, as shown in Fig. 6.4.

(a) 

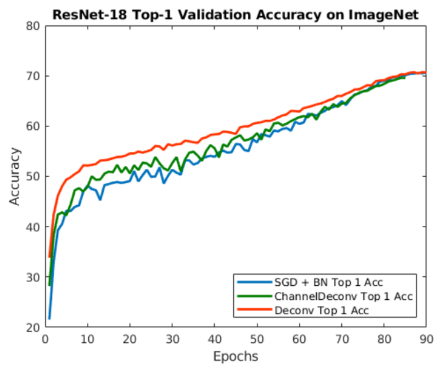
(b) 

Figure 6.7: Results of training the VGG-11/ResNet-18 network on the ImageNet dataset.

6.5 Acknowledgement

We would like to express our gratitude to Prof. Brian Hunt for his insightful comments.

6.6 Conclusion

In this paper we presented network deconvolution, a novel normalization method for pixel-wise decorrelation. The method was evaluated extensively and shown to improve the optimization efficiency over standard Batch Normalization. We provided a thorough analysis regarding its performance and demonstrated consistent performance improvements of the deconvolution operation on multiple major benchmarks.

ImageNet	No Norm.	REF BN	CD	DC
VGG-11	69.02	70.38	71.45	**71.74**
Resnet-18	N/A	69.76	69.80	**70.65**

Table 6.2: Comparison of top-1 accuracy of full deconvolution with the model zoo implementation of VGG-11 and ResNet-18 on ImageNet between: Reference implementation on PyTorch with no normalization (No Norm.), reference implementation with batch norm (REF BN), Channel Deconvolution only (CD) and Full Pixel and Channel Deconvolution (DC).

Our proposed deconvolution operation is straightforward in terms of implementation and can serve as a good alternative to Batch Normalization.

6.7 Appendix

6.7.1 Acceleration via Subsampling

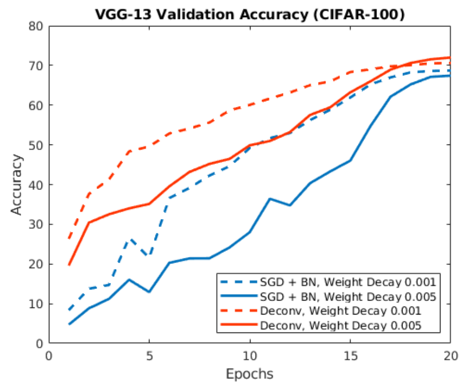
We have demonstrated the computation of the covariance matrix has complexity $O(H \times W \times k \times k \times Ch_{in} \times G)$, which is usually lower than the complexity of a convolution layer in the real settings. However, at the time of this writing, these operations have not been incorporated into any existing deep learning packages. Direct computations incur unnecessary memory copy operations which significantly slow down the training. One key observation is that the covariance matrix is usually very small compare to the number of pixels involved in the computation. As an example, it is unnecessary to use 1 million samples to calculate a 9×9 covariance. Here we propose a simple subsampling technique to siginificantly reduce the computational cost. When calling the *im2col* function, we specify a sampling stride to uniformly subsample from the original image. With subsampling, the cost is reduced to $O(\frac{H \times W \times k \times k \times Ch_{in} \times G}{S^2})$. Experimentally we have noticed sampling strides of $S = 3 \sim 5$ reduces the computational cost by $10\times \sim 20\times$, without sacrificing training accuracy. With this acceleration, the training wall time is close to a network using batch normalization on the ImageNet.

6.7.2 Regularizations

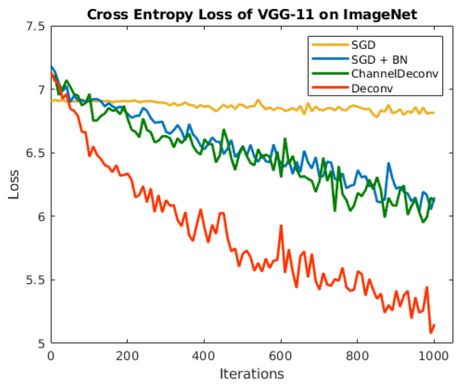
If two features correlate, weight decay regularization is less effective. If X_1, X_2 are strongly correlated features, but differ in scale, and if we look at: $w_1 X_1 + w_2 X_2$. The weights is likely to co-adapt during the training and weight decay is likely to be more effective on the larger coefficient. The other, small coefficient is left less penalized. Network deconvolution reduces the co-adaptation of weights, weight decay has become less ambiguous and more effective (Fig. 6.8(a)).

6.7.3 Sparse Representations for Convolution Layers

Fig. 6.9 shows the inputs to the 5-th convolution layer in $VGG-11$. This input is the output of a $ReLU$ activation function. The deconvolution operation first subtract the mean, and then remove the correlation between nearby pixels, resulting in a sharper and sparser representation.



(a)

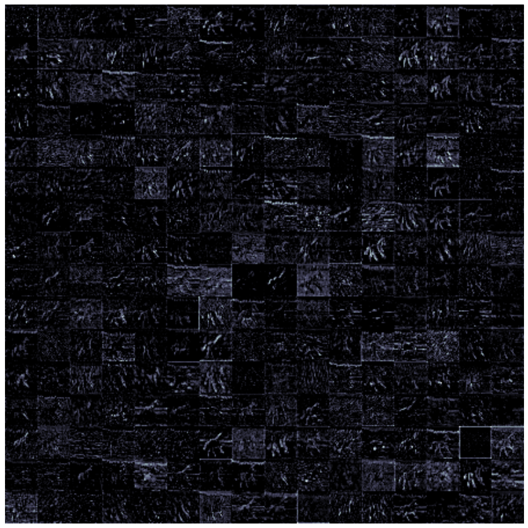


(b)

Figure 6.8: (a)The effects of weight decay on stochastic gradient descent (SGD) and batch normalization (BN) versus SGD and deconvolution (Deconv) with 5 iterations done to produce the inverse covariance matrix, training on the CIFAR-100 dataset on the VGG-13 network. Here we notice that increased weight decay leads to worse results for standard training. However in our case with deconvolution, the final accuracy actually improves with increased weight decay (.001 to .005).(b)The training loss of the VGG-11 network on the ImageNet dataset. Only the first 1000 iterations are shown. Comparison is made among SGD, SGD with batch normalization, channel deconvolution only, and full deconvolution.

6.7.4 Accelerated Convergence

We demonstrate the loss curves using different settings when training the VGG-11 network on the ImageNet dataset(Fig. 6.8(b)). We can see network deconvolution leads to significantly faster decay in training loss.


(a)

(b)

Figure 6.9: (a)Input features to the 5-th convolution layer in VGG-11. (b) Taking the absolute value of the deconvolved features. The features have been min-max normalized for visualization.(Best view on a display.)

Bibliography

[1] Andrew J Davison, Ian D Reid, Nicholas D Molton, and Olivier Stasse. Monoslam: Real-time single camera slam. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (6):1052–1067, 2007.

[2] P. Lichtsteiner, C. Posch, and T. Delbruck. A 128 x 128 at 120db 15 micros latency asynchronous temporal contrast vision sensor. *Solid-State Circuits, IEEE Journal of*, 43(2):566–576, 2008.

[3] C. Brandli, R. Berner, M. Yang, S. Liu, and T. Delbruck. A 240 180 130 db 3 s latency global shutter spatiotemporal vision sensor. *IEEE Journal of Solid-State Circuits*, 49(10):2333–2341, Oct 2014.

[4] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[6] Stphane Mallat. *A Wavelet Tour of Signal Processing, Third Edition: The Sparse Way*. Academic Press, Inc., Orlando, FL, USA, 3rd edition, 2008.

[7] S. Mallat. Understanding deep convolutional networks. *Philosophical Transactions of the Royal Society of London Series A*, 374:20150203, April 2016.

[8] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[10] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*, 2016.

[11] Yunpeng Chen, Jianan Li, Huaxin Xiao, Xiaojie Jin, Shuicheng Yan, and Jiashi Feng. Dual path networks. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4467–4475. Curran Associates, Inc., 2017.

[12] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *CoRR*, abs/1505.00387, 2015.

[13] Benjamin Graham. Fractional max-pooling. *arXiv preprint arXiv:1412.6071*, 2014.

[14] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial transformer networks. *CoRR*, abs/1506.02025, 2015.

[15] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR*, abs/1510.00149, 2015.

[16] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *CoRR*, abs/1608.08710, 2016.

[17] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. *CoRR*, abs/1708.06519, 2017.

[18] Matthieu Courbariaux and Yoshua Bengio. Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1. *CoRR*, abs/1602.02830, 2016.

[19] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015.

[20] Ming Liang and Xiaolin Hu. Recurrent convolutional neural network for object recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[21] Richard Socher, Brody Huval, Bharath Bath, Christopher D Manning, and Andrew Y Ng. Convolutional-recursive deep learning for 3d object classification. In *Advances in neural information processing systems*, pages 656–664, 2012.

[22] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR*, abs/1602.07360, 2016.

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

[24] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *CoRR*, abs/1707.01083, 2017.

[25] Gao Huang, Shichen Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Condensenet: An efficient densenet using learned group convolutions. *CoRR*, abs/1711.09224, 2017.

[26] Ting Zhang, Guo-Jun Qi, Bin Xiao, and Jingdong Wang. Interleaved group convolutions for deep neural networks. *CoRR*, abs/1707.02725, 2017.

[27] Yi Zhou, Guillermo Gallego, Henri Rebecq, Laurent Kneip, Hongdong Li, and Davide Scaramuzza. Semi-dense 3d reconstruction with a stereo event camera. *European Conference on Computer Vision(ECCV)*, 2018.

[28] Alex Zihao Zhu, Liangzhe Yuan, Kenneth Chaney, and Kostas Daniilidis. Unsupervised Event-based Learning of Optical Flow, Depth, and Egomotion. *arXiv e-prints*, page arXiv:1812.08156, Dec 2018.

[29] Guillermo Gallego Henri Rebecq and Davide Scaramuzza. Emvs: Event-based multi-view stereo. In E. R. Hancock R. C. Wilson and W. A. P. Smith, editors, *Proceedings of the British Machine Vision Conference (BMVC)*, pages 63.1–63.11, September 2016.

[30] Hanme Kim, Stefan Leutenegger, and Andrew J. Davison. Real-time 3d reconstruction and 6-dof tracking with an event camera. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, pages 349–364, Cham, 2016. Springer International Publishing.

[31] Guillermo Gallego, Henri Rebecq, and Davide Scaramuzza. A unifying contrast maximization framework for event cameras, with applications to motion, depth, and optical flow estimation. *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2018.

[32] T. Delbruck. Frame-free dynamic digital vision. In *Proceedings of Intl. Symposium on Secure-Life Electronics, Advanced Electronics for Quality Life and Society, Tokyo, Japan,*, pages 21–26, March 2008.

[33] Min Liu and Tobi Delbruck. Block-matching optical flow for dynamic vision sensors: Algorithm and fpga implementation. In *Circuits and Systems (ISCAS), 2017 IEEE International Symposium on*, pages 1–4. IEEE, 2017.

[34] Ryad Benosman, Sio-Hoi Ieng, Charles Clercq, Chiara Bartolozzi, and Mandyam Srinivasan. Asynchronous frameless event-based optical flow. *Neural Netw.*, 27:32 – 37, March 2012.

[35] Stephan Tschechne, Tobias Brosch, Roman Sailer, Nora von Egloffstein, Luma Issa Abdul-Kreem, and Heiko Neumann. On event-based motion detection and integration. In *Proceedings of the 8th International Conference on*

Bioinspired Information and Communications Technologies, pages 298–305, 2014.

[36] R. Benosman, C. Clercq, X. Lagorce, Sio-Hoi Ieng, and C. Bartolozzi. Event-based visual flow. *Neural Networks and Learning Systems, IEEE Transactions on*, 25(2):407–417, 2014.

[37] Francisco Barranco, Cornelia Fermüller, and Yiannis Aloimonos. Contour motion estimation for asynchronous event-driven cameras. *Proceedings of the IEEE*, 102(10):1537–1556, 2014.

[38] G. Orchard and R. Etienne-Cummings. Bioinspired visual motion estimation. *Proceedings of the IEEE*, 102(10):1520–1536, Oct 2014.

[39] Alex Zihao Zhu, Liangzhe Yuan, Kenneth Chaney, and Kostas Daniilidis. Ev-flownet: Self-supervised optical flow estimation for event-based cameras. *Robotics: Science and Systems*, 2018.

[40] A. Z. Zhu, D. Thakur, T. Özaslan, B. Pfrommer, V. Kumar, and K. Daniilidis. The multivehicle stereo event camera dataset: An event camera dataset for 3d perception. *IEEE Robotics and Automation Letters*, 3(3):2032–2039, July 2018.

[41] Deqing Sun, Erik B Sudderth, and Michael J Black. Layered segmentation and optical flow estimation over time. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1768–1775. IEEE, 2012.

[42] J-M Odobez and Patrick Bouthemy. Mrf-based motion segmentation exploiting a 2d motion model robust estimation. In *Proceedings., International Conference on Image Processing*, volume 3, pages 628–631. IEEE, 1995.

[43] William B. Thompson and Ting-Chuen Pong. Detecting moving objects. *International Journal of Computer Vision*, 4(1):39–57, Jan 1990.

[44] René Vidal, Yi Ma, and Shankar Sastry. Generalized principal component analysis (gpca). In *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, volume 1, pages I–I. IEEE, 2003.

[45] Michal Irani and P Anandan. A unified approach to moving object detection in 2d and 3d scenes. *IEEE transactions on pattern analysis and machine intelligence*, 20(6):577–589, 1998.

[46] Philip HS Torr. Geometric motion segmentation and model selection. *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 356(1740):1321–1340, 1998.

[47] Abhijit S Ogale, Cornelia Fermuller, and Yiannis Aloimonos. Motion segmentation using occlusions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(6):988–992, 2005.

[48] Katerina Fragkiadaki, Geng Zhang, and Jianbo Shi. Video segmentation by tracing discontinuities in a trajectory embedding. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1846–1853. IEEE, 2012.

[49] Katerina Fragkiadaki, Pablo Arbelaez, Panna Felsen, and Jitendra Malik. Learning to segment moving objects in videos. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[50] Pia Bideau, Aruni RoyChowdhury, Rakesh R Menon, and Erik Learned-Miller. The best of both worlds: combining cnns and geometric constraints for hierarchical motion segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 508–517, 2018.

[51] Ashutosh Saxena, Sung H Chung, and Andrew Y Ng. Learning depth from single monocular images. In *Advances in neural information processing systems*, pages 1161–1168, 2006.

[52] David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. In *Advances in neural information processing systems*, pages 2366–2374, 2014.

[53] Ravi Garg, Vijay Kumar BG, Gustavo Carneiro, and Ian Reid. Unsupervised cnn for single view depth estimation: Geometry to the rescue. In *European Conference on Computer Vision*, pages 740–756. Springer, 2016.

[54] Junyuan Xie, Ross Girshick, and Ali Farhadi. Deep3d: Fully automatic 2d-to-3d video conversion with deep convolutional neural networks. In *European Conference on Computer Vision*, pages 842–857. Springer, 2016.

[55] Tinghui Zhou, Matthew Brown, Noah Snavely, and David G. Lowe. Unsupervised learning of depth and ego-motion from video. In *CVPR*, 2017.

[56] Chaoyang Wang, José Miguel Buenaposada, Rui Zhu, and Simon Lucey. Learning depth from monocular videos using direct methods. *CoRR*, abs/1712.00175, 2017.

[57] Reza Mahjourian, Martin Wicke, and Anelia Angelova. Unsupervised learning of depth and ego-motion from monocular video using 3d geometric constraints. *CoRR*, abs/1802.05522, 2018.

[58] Zhenheng Yang, Peng Wang, Yang Wang, Wei Xu, and Ram Nevatia. LEGO: learning edge with geometry all at once by watching videos. In *Proc. IEEE Computer Vision and Pattern Recognition Conference*, 2018.

[59] Zhichao Yin and Jianping Shi. Geonet: Unsupervised learning of dense depth, optical flow and camera pose. *CoRR*, abs/1803.02276, 2018.

[60] Sudheendra Vijayanarasimhan, Susanna Ricco, Cordelia Schmid, Rahul Sukthankar, and Katerina Fragkiadaki. Sfm-net: Learning of structure and motion from video, 2017.

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

[62] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. chapter Learning Internal Representations by Error Propagation, pages 318–362. MIT Press, Cambridge, MA, USA, 1986.

[63] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013.

[64] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *CoRR*, abs/1605.07146, 2016.

[65] R. R. Coifman and M. V. Wickerhauser. Entropy-based algorithms for best basis selection. *IEEE Trans. Inf. Theor.*, 38(2):713–718, September 2006.

[66] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In B. Schölkopf, J. C. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 153–160. MIT Press, 2007.

[67] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei. Deformable Convolutional Networks. *ArXiv e-prints*, March 2017.

[68] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[69] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images, 2009.

[70] Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. A downsampled variant of imagenet as an alternative to the CIFAR datasets. *CoRR*, abs/1707.08819, 2017.

[71] J. Deng, W. Dong, R. Socher, L. J. Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, June 2009.

[72] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. *CoRR*, abs/1603.05027, 2016.

[73] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q. Weinberger. Deep networks with stochastic depth. *CoRR*, abs/1603.09382, 2016.

[74] Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. *CoRR*, abs/1611.05431, 2016.

[75] Chengxi Ye, Chinmaya Devaraj, Michael Maynord, Cornelia Fermüller, and Yiannis Aloimonos. Evenly cascaded convolutional networks. In *IEEE International Conference on Big Data, Big Data 2018, Seattle, WA, USA, December 10-13, 2018*, pages 4640–4647, 2018.

[76] Anton Mitrokhin, Cornelia Fermuller, Chethan Parameshwara, and Yiannis Aloimonos. Event-based moving object detection and tracking. *IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, 2018.

[77] Reza Mahjourian, Martin Wicke, and Anelia Angelova. Unsupervised learning of depth and ego-motion from monocular video using 3d geometric constraints. *CoRR*, abs/1802.05522, 2018.

[78] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016.

[79] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI (3)*, volume 9351 of *Lecture Notes in Computer Science*, pages 234–241. Springer, 2015.

[80] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[81] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4700–4708, 2017.

[82] Chengxi Ye, Dacheng Tao, Mingli Song, David W. Jacobs, and Min Wu. Sparse norm filtering. *CoRR*, abs/1305.3971, 2013.

[83] Yuxin Wu and Kaiming He. Group normalization. *CoRR*, abs/1803.08494, 2018.

[84] Tsung-Yu Lin and Subhransu Maji. Improved bilinear pooling with cnns. *CoRR*, abs/1707.06772, 2017.

[85] Eugene D. Denman and Alex N. Beavers, Jr. The matrix sign function and computations in systems. *Appl. Math. Comput.*, 2(1):63–94, January 1976.

[86] Chengxi Ye, Yezhou Yang, Cornelia Fermüller, and Yiannis Aloimonos. On the importance of consistency in training deep neural networks. *CoRR*, abs/1708.00631, 2017.

[87] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? The KITTI Vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361, June 2012.

[88] David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. In *NIPS*, 2014.

[89] Ravi Garg, Vijay Kumar B. G, and Ian D. Reid. Unsupervised CNN for single view depth estimation: Geometry to the rescue. *CoRR*, abs/1603.04992, 2016.

[90] Chengxi Ye, Anton Mitrokhin, Cornelia Fermüller, James A. Yorke, and Yiannis Aloimonos. Unsupervised learning of dense optical flow, depth and egomotion from sparse event data. *CoRR*, abs/1809.08625v2, 2018.

[91] Bruno A. Olshausen and David J. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609, 1996.

[92] Aapo Hyvrinen, Jarmo Hurri, and Patrick O. Hoyer. *Natural Image Statistics: A Probabilistic Approach to Early Computational Vision.* Springer Publishing Company, Incorporated, 1st edition, 2009.

[93] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[94] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[95] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537, 2011.

[96] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[97] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[98] Tim Salimans and Diederik P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *CoRR*, abs/1602.07868, 2016.

[99] James Martens and Roger B. Grosse. Optimizing neural networks with kronecker-factored approximate curvature. *CoRR*, abs/1503.05671, 2015.

[100] Guillaume Desjardins, Karen Simonyan, Razvan Pascanu, and koray kavukcuoglu. Natural neural networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2071–2079. Curran Associates, Inc., 2015.

[101] Chengxi Ye, Anton Mitrokhin, Cornelia Fermüller, James A. Yorke, and Yiannis Aloimonos. Unsupervised learning of dense optical flow and depth from sparse event data. *CoRR*, abs/1809.08625, 2019.

[102] Lei Huang, Dawei Yang, Bo Lang, and Jia Deng. Decorrelated batch normalization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 791–800, 2018.

[103] Lei Huang, Yi Zhou, Fan Zhu, Li Liu, and Ling Shao. Iterative normalization: Beyond standardization towards efficient whitening. *CoRR*, abs/1904.03441, 2019.

[104] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[105] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[106] Chengxi Ye, Chiaowen Hsiao, and Hctor Corrada Bravo. BlindCall: ultra-fast base-calling of high-throughput sequencing data by blind deconvolution. *Bioinformatics*, 30(9):1214–1219, 01 2014.

[107] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.