Unsupervised Training of Deep Neural Networks for Motion Estimation

Aria Ahmadi

Submitted in partial fulfillment of the requirements of the Degree of Doctor of Philosophy

Supervisor: Prof. Ioannis Patras

School of of Electronic Engineering and Computer Science

Queen Mary University of London

United Kingdom

Nov 2018

Statement of originality

I, Aria Ahmadi, confirm that the research included within this thesis is my own work or that where it has been carried out in collaboration with, or supported by others, that this is duly acknowledged below and my contribution indicated. Previously published material is also acknowledged below.

I attest that I have exercised reasonable care to ensure that the work is original, and does not to the best of my knowledge break any UK law, infringe any third party's copyright or other Intellectual Property Right, or contain any confidential material.

I accept that the College has the right to use plagiarism detection software to check the electronic version of the thesis.

I confirm that this thesis has not been previously submitted for the award of a degree by this or any other university.

The copyright of this thesis rests with the author and no quotation from it or information derived from it may be published without the prior written consent of the author.

Signature: Aria Ahmadi

Date: 29/11/2018

Acknowledgments

First, I would like to thank my supervisor Ioannis Patras, who trusted me from the beginning, and whose support and advice were significantly important for the realization of this thesis.

I'd like to thank my family, who always supported me in the pursuit of this project, and to whom this thesis is dedicated.

There is no word that can describe my gratitude towards my friends Mina Adel Thabet, Petar Palasek, and Ioannis Marras, who have been always there for me and have helped me above and beyond.

I will always be thankful to my friends Thomas Cuvillier, Juan Abdon Miranda, Christos Tzelepis, Oya Celiktutan, Faranak Sobhani, and Andrej Satnik.

To all my colleagues in the MMV group, and all the friends that shared time with me and have encouraged me to fulfill this objective.

Abstract

This thesis addresses the problem of motion estimation, that is, the estimation of a field that describes how pixels move from a reference frame to a target frame, using Deep Neural Networks (DNNs). In contrast to classic methods, we don't solve an optimization problem at test time. We train DNNs once and apply it in one pass during the test which reduces the computational complexity. The major contribution is that in contrast to a supervised method, we train our DNNs in an unsupervised way. By unsupervised, we mean without the need for ground truth motion fields which are expensive to obtain for real scenes.

More specifically, we have trained our networks by designing cost functions inspired by classical optical flow estimation schemes and generative methods in Computer Vision. We first propose a straightforward CNN method that is trained to optimize the brightness constancy constraint and we embed it in a classical multiscale scheme in order to predict motions that are large in magnitude (GradNet). We show that GradNet generalizes well to an unknown dataset and performed comparably with state-of-the-art unsupervised methods at that time. Second, we propose a convolutional Siamese architecture wherein is embedded a new soft warping scheme applied in a multiscale framework and is trained to optimize a higher-level feature constancy constraint (LikeNet). The architecture of LikeNet allows a trade-off between the computational load and memory and is 98% smaller than other SOA methods in terms of learned parameters. We show that LikeNet performs on par with SOA approaches and the best among uni-directional methods, methods that calculate motion field in one pass. Third, we propose a novel approach to distill slower LikeNet in a much faster regression neural network without losing much of the accuracy (QLikeNet).

The results show that using DNNs is a promising direction for motion estimation, although further improvements are required as classical methods yet perform the best.

Contents

List of Figures

List of Tables

List of Abbreviations

CNN - convolutional neural network

CPU - central processing unit

DNN - deep neural network

GPU - graphics processing unit

OF - optical flow

MF - Motion Field

ME - Motion Estimation

RGB - red green blue

MRF - Markov Random Field

CRF - Conditional Random Field

Chapter 1

Introduction

By the tremendous advances in electronics and communication, video data can be easily captured and is ubiquitous. Motion in a video is as a result of moving objects and/or moving camera, which is the essence of a video. Accurate and efficient analysis of motion information serves many purposes.

Motion analysis refers to computing or analysing the motion pattern of the camera or the scene. Several kinds of motion measurement problems exist in computer vision three examples of which we present here. First, Video tracking which is the process of locating a moving object over time using a camera and thus deals with the analysis of the motion information. Second, Camera ego-motion estimation which is the problem of estimating the motion of camera in a static or partially-dynamic scene. It aims at recovering the 3D rigid motion (i.e., rotation and translation) of the camera, or, equivalently, the 3D rigid transformation of camera coordinate systems, using the color or depth/range data captured by the camera. Third, Dense motion estimation which aims to compute the pixel movement vectors on a 3D spatiotemporal plane, in the presence of a dynamic scene and/or a moving camera. The goal of (dense) motion estimation is to estimate the motion field, a field that describes how pixels move from a reference frame to a target frame. The problems of tracking/camera ego-motion estimation and dense motion estimation are closely related to each other: e.g. camera

motion can be derived from dense motion estimates.

The measured motion can be used in several applications such as in video coding and video understanding. In the case of video coding, motion information is used to remove the temporal redundancy [27, 53]. Almost all video coding standards use block-based motion estimation and compensation such as the MPEG series including the most recent HEVC. There are a wide variety of applications such as TV broadcasting, video streaming, DVD and Blue-ray discs in which direction researches have been conducted [27, 53]. In the context of computer vision and video analysis, an accurate estimation of the motion in a scene is fundamental for video understanding. This is because motion is one of the basic concepts that convey a considerable portion of the information required for video analysis. For example, motion information alone has been shown to be effective for action recognition [46, 75]. The idea comes from the fact that the human visual cortex has two pathways: ventral and dorsal stream. The first one is responsible for object recognition while the second one recognizes motion. Among other applications are surveillance, robotics, segmentation, activity recognition, registration, video search, and retrieval. Many researches have been conducted in these directions [47, 26, 35, 55, 43, 89, 16, 17, 70, 25, 41, 95].

1.1 Challenges

Motion estimation is a complex task for which there are multiple challenges such as illumination variation in consecutive frames (including self-shadowing [12]), estimation of large motions, occluded area, motion discontinuities, transparent surfaces, and estimation of motion for low texture surfaces. We attempted to address the illumination variation by augmenting the input data by adding artificial intensity variation while the supervision is based on the original input. Furthermore, we propose to use features

that are more robust towards intensity variation compared to intensity features [1] which improved the results. To estimate large motions, most of the motion estimation methods in the literature are embedded in a multiscale schemes [1, 2, 17, 38, 16, 93]. most of the methods minimize the difference between the features extracted from the corresponding points in the reference frame and target frame. That is while one of the corresponding points does not exist if occluded. In the literature, the occluded area is usually modeled through forward-backward estimation of motion field [59, 90] and checking the consistency of both estimations. This approach needs to be improved though. Another challenge is when a motion field consists of regions where the motion varies smoothly, divided by the boundaries where the motion field changes abruptly. Most of the energy minimization frameworks assume that the flow is continuous which makes capturing the sharp discontinuities challenging [94]. In the literature, motion boundaries are respected by applying an edge-aware smoothness term [59, 90] which works to some extent, but yet needs to be improved. Estimation of large displacements is another challenge which is tried to be addressed using multiscale schemes by most of the classic [16, 17, 8] and even DNN-based methods [67, 82] in the literature. All our unsupervised approaches have been embedded in the classic multiscale scheme as well. In two of our approaches, we also try to improve the correspondence matching by using more sophisticated features rather than just intensity features in the motion compensated error. Still there are challenges that are not addressed much in the literature such as, the estimation of motion for transparent surfaces and low-texture area. The researches that have been conducted to overcome these problems have lead to both classic methods, which do not use DNNs, and DNN-based methods.

1.2 Previous Works

Classic motion estimation started with the method proposed by Horn and Schunk [38] wherein the objective function penalizes the deviation from the intensity constancy assumption and also the assumption that the estimated motion field has to be smooth.

Since then, the objective function for motion estimation has evolved over time to several variations which improve the estimation. For example, including the gradient term which penalizes the deviation from the constant gradient of the intensity assumption improves the estimation considerably [16]. The estimation further improves by integrating descriptor matching in the variational formula [17, 93]. These methods rely on a multiscale motion estimation scheme which helps with the estimation of large displacements. One drawback with the multiscale scheme is that the error propagates across scales. The reason is that each scale of the multiscale scheme uses the resized and scaled motion field estimated in the previous scale. This way, the wrong estimations grow across scales. Furthermore, in the lower scales, although larger motions and surfaces are smaller, most of the details are lost in the downsampling operation. Later on, some methods proposed to estimate motion in single scale by densification of a sparse motion field obtained from a feature matching process [70]. Apart from the improvements over time, all classic methods have to solve an optimization problem for each sample, thus are slower. Also, the classical methods make very approximate assumptions about the image feature changes and the spatial structure of the flow. The key advantage of learning to compute flow from data, as DNN-based methods do, is that supervised methods do not make an approximate assumption and instead learn to estimate motion from data. Regarding unsupervised methods, in chapter 3, we will show that, because of the inner regularization effect of DNNs, our unsupervised trained network performs better than Horn and Schunk method [38], in a multiscale scheme, although both methods use similar loss functions. Still, there is a gap between classic and DNN-based methods which will reduce by further researches.

Deep Neural Networks (DNNs) have shown to have promising performance in different applications [52, 75, 76]. It was the beginning of a wave of works on the application of DNNs in different computer vision tasks. DNNs are capable of learning high to low-level abstract representations which enables modeling a complex relationship between

their input and output. DNNs have also been used for motion estimation [25, 41] wherein DNNs are trained in a supervised way. Although these methods work fairly well, they are still bound to a limitation DNNs face in solving problems. For them to achieve a high performance, they require many training data samples. Unlike other areas of computer vision, such as action and object recognition, motion datasets still lack enough properly labeled data for motion in real scenes. The reason is that labeling is pixel level and thus too expensive to have for many samples. The first alternative is to use limited synthetic datasets [25, 41] which do not completely characterize the dynamics of real scenes. The other alternative is to train DNNs in an unsupervised way. Unsupervised methods are mainly trained based on classic objective functions. The core of the exploited objective functions is to minimize the feature constancy constraint which can be linearized in different ways [2, 69] for back-propagation purposes. Similar to classic methods, adding extra terms to the objective function improves the performance. Furthermore, some methods calculate both forward and backward passes to model the occlusion and/or consistency check [59, 90] which is reported to improve the performance significantly.

1.3 Overview of the Proposed Methods

The aim of our research is to propose a DNN-based framework that realizes training a DNN for motion estimation without the need for labeled training data, that is unsupervised. Furthermore, to use the unsupervised trained network as a baseline for fine-tuning the DNN for an object specific motion estimation task.

In the first proposed training scheme, a fully convolutional DNN, CNN, with hourglass architecture is trained in an unsupervised way for motion estimation. The input is the pair of reference and target frames and the output is the motion field. The architecture is a fully convolutional hourglass neural network with skip connections at the

encoder side to the corresponding resolution at the decoder side. The training loss is the widely used motion compensated intensity error linearized based on the first-order Taylor expansion. Further experiments where the linearization is based on second-order Taylor expansion show more stability. The training data is pairs of consecutive frames randomly drawn from human action recognition dataset, UCF101. The classic multiscale scheme is adopted to help with the estimation of large displacements. The evaluations show good generalization to unknown synthetic and real datasets.

In the second proposed method, a Siamese CNN was trained in an unsupervised way for motion estimation, likeNet. Unlike other DNN-based motion estimation methods, LikeNet solves motion estimation as a classification problem. Each specific displacement vector is assigned a label and the goal is to predict a label that represents the displacement of each pixel in the reference frame to the target frame. LikeNet receives as input the pair of reference and target frames and calculates a distribution over the displacement labels. Each branch of the Siamese architecture is responsible for calculating a score map for each specific label and receives as input the reference frame and the target frame which is shifted along the displacement vector corresponding to that label. These displacement vectors have integer components and in our experiments cover a uniform grid area around centre. The output of each branch is a similarity map which expresses how successful the shift has been in aligning similar pixels. During training we aim to maximize the probability of the case where similar pixels are aligned. During the test, the estimated motion vector for each pixel equals to the displacement vector under which shifting yields the highest probability. The classic multiscale scheme is adopted to help with the estimation of large displacements and also reduce the computational complexity. The similarity measure is based on the low level features calculated by the first layer of an object recognition DNN, VGG. A conditional Random Field (CRF) implemented as an RNN [100] is adopted to improve the quality of the estimated motion field in the lowest scale to prevent the error from being propagated across scales.

The main drawback of LikeNet is its slow run time. The reason is that although the LikeNet architecture is fully parallelizable, the available parallel computing resources are limited and as a result, a considerable part of the computation runs in serial which slows down LikeNet. In the third proposed method, we propose a method to use LikeNet as a teacher to train a quicker CNN, QLikeNet. The exploited CNN has an architecture similar to what is used in the first chapter. During the training, one branch of LikeNet stacks on top of the CNN. It receives as input the reference frame and the warped version of the target frame, warped based on the output of the primary CNN. Its mission is to evaluate the motion field calculated by the primary CNN by scoring how successful has been the calculated motion field in aligning similar pixels. For the loss function, we propose to maximize the similarity map calculated by the branch of the LikeNet. The idea is to maximize the cases where warping operation aligns similar pixels. We also study the case which the task is object-specific motion estimation. In our application, we focus on motion estimation of motorcycle helmet in real scenes.

1.4 Major contributions

We focus on unsupervised training of Deep Neural Networks (DNNs) for motion estimation. We were among the first to propose an unsupervised training scheme for DNNs for motion estimation with [45] being proposed at roughly the same period. We assume that an unsupervised training scheme has to provide good generalization to an unknown dataset. Unlike other methods, for evaluations, we focus on the performance without fine tuning on the target dataset. In this thesis, we make several contributions, which are summarized below. We consider the evaluation of our contributions as an important aspect. Therefore, we performed extensive experiments on publicly available datasets.

- In Chapter 3, for the first time, we present an unsupervised training scheme

for motion estimation. In such an approach, the training data are randomly drawn from a real dataset, UCF101 human action recognition dataset, without any information about the underlying motion distribution. The network trained using the proposed approach has shown to generalize well to an unknown dataset. We also proposed to exploit classic multiscale approach which has shown to be effective in the whole literature, in combination with our new DNN-based technique. The idea of combining with classic principals was later on adopted by some other state-of-the-art supervised methods [67]. Our proposed method is concurrent with the method in [45], which is similar to [69], although our approach and the approaches in [45, 69] are different in architecture, training loss, and training data.

- In Chapter 4, in contrast to all other unsupervised DNN methods solve motion estimation as a regression problem we propose to solve motion estimation as a classification problem. Although solving motion estimation as a classification problem limits the resolution of the estimation, LikeNet performs better than the state-of-the-art regression-based methods. The architecture of the proposed method is 98% smaller than other unsupervised methods in terms of learned parameters. In this method we propose to use more sophisticated VGG features instead of commonly used intensity features only for training. Also, for the first time, we propose to improve the quality of the estimation in lower scales by adopting conditional random fields (CRFs) implemented as a recurrent neural network [100]. This approach was at the same time put into practice in one of the supervised DNN-based method [82].

- In Chapter 5, we propose a new way of training DNNs for motion estimation that is both fast and accurate. Although the method proposed in Chapter 4 showed to perform well in comparison with other unsupervised methods, the slower runtime is a drawback. We address this shortcoming by proposing a new method for squeezing that network in a faster architecture that solves motion

estimation as a regression problem. Furthermore, we also study the performance of training a DNN for an object specific motion estimation task, in our case motion estimation of motorcycle helmet on a rider. For this purpose, we first train a DNN in an unsupervised way for motion estimation and then fine tune it in a supervised way on a synthetic bespoke dataset designed for our specific task.

1.5 Organisation of the thesis

In Chapter 2, we review and compare works related to the problem of motion estimation, then we also provide a brief review of the application of the graphical models for improving the pixel-level prediction tasks. In Chapter 3, we present our firstly proposed unsupervised method, GradNet. In Chapter 4, we present our second motion estimation method, LikeNet, that reformulates the motion estimation as a classification task and also benefits from the application of graphical models, more specifically CRFs, for improving the prediction. In Chapter 5, we present our third unsupervised training technique that is based on squeezing LikeNet in a fully convolutional neural network. In Chapter 6, final conclusions are drawn, and a discussion of the future work is given.

Publications

Ahmadi, Aria, and Ioannis Patras. "Unsupervised convolutional neural networks for motion estimation." 2016 IEEE international conference on image processing (ICIP). IEEE, 2016.

Ahmadi, Aria, Ioannis Marras, and Ioannis Patras. "LikeNet: A Siamese Motion

Estimation Network Trained in an Unsupervised Way." 2018 British Media and Vision Conference (BMVC)

Chapter 2

Related Work

In Chapter 1, we introduced the field of motion estimation. In this chapter, we will present an overview of the related works in the literature addressing the problem of motion estimation. It all started with the simultaneous works proposed by Horn and Schunk [38], and Lucas and Kanade [57]. Horn and Schunk in [38] proposed a classical optical flow formulation based on which several methods were later proposed to improve the estimated motion field. We divide motion estimation methods into two major groups: DNN-based methods and other methods which we refer to as classic methods.

The chapter is organized as follows. In Section. 2.1 we discuss some of the methods that do not use DNNs for motion estimation, i.e. classic methods. In Section 2.2, we review the supervised and unsupervised DNN-based techniques proposed for the task of motion estimation in the literature. In Section. 2.2.2, we review uni-directional unsupervised methods for motion estimation which calculate the motion field only in a forward pass describing how pixels move from the reference frame to the target frame. In Section 2.2.2, we review bi-directional unsupervised methods for motion estimation which benefits from calculating the motion field both in the forward and backward passes. A motion field calculated in a backward pass describes how pixels move from

the target frame to the reference frame. Bi-directional motion estimation methods improve the estimation by excluding the occluded area from the training. Finally, we conclude the related works in Section 2.3.

2.1 Classic Methods

To tackle the problem of motion estimation, one of the main assumptions is that the dense features, E, describing pixels in the reference frame should not change by a displacement in the target frame. This is referred to as the feature constancy constraint, Fig. 2.1. The constraint states that,

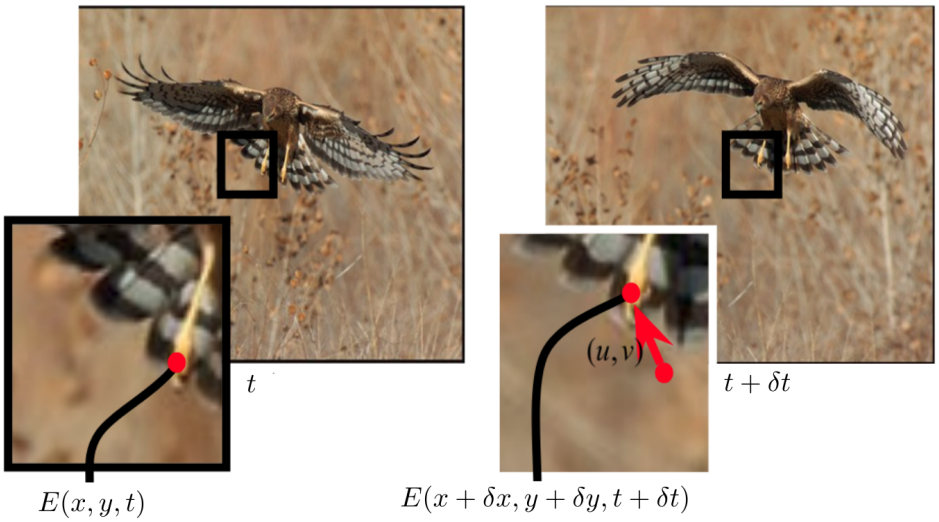

Figure 2.1: The illustration of the feature constancy constraint.

$$E(x, y, t) = E(x + \delta x, y + \delta y, t + \delta t) \tag{2.1}$$

where,

$$\begin{aligned} \delta x &= u\delta t \\ \delta y &= v\delta t \end{aligned} \tag{2.2}$$

where δx and δy are vertical and horizontal velocities and u and v are respectively horizontal and vertical displacements. The first approach is to penalize the deviation from the intensity constancy assumption. This is implemented by minimizing the L_{Data},

$$L_{Data}(u, v) = (E_x u + E_y v + E_t)^2 \tag{2.3}$$

where E refers to intensity features and E_x, E_y, and E_t are respectively horizontal, vertical and temporal derivatives of the intensity features. Equation 2.3, is an approximation based on the first-order Taylor expansion of $E(x + \delta x, y + \delta y, t + \delta t)$ around (x, y, t) in the motion compensated intensity error. The major problem with solving equation 2.3 is that this equation is only one constraint to solve for two variables. Such an under-determined problem has infinite solutions. The first solution to this problem is to assume that neighboring pixels in a small neighborhood have similar motion vectors. Then the optical flow constraint is evaluated with respect to all pixels in this small neighborhood. Lucas and Kanade [57] first proposed to use quadratic deviations in a least square approach, equation 2.4.

$$\min_{u,v} \sum_{x' \in N(x)} (E_x(x')u + E_y(x')v + E_t(x'))^2 \tag{2.4}$$

where $N(x)$ denotes a neighborhood around x. Solving this equation results in a flow field that is subpixel accurate and the effect of higher order terms in the Taylor expansion can be ignored. More specifically, Lucas Kanade method follows a gradient descent method to minimize the difference between grey values in the reference frame and their correspondences in the target frame.

At the same time, Horn and Schunk [38] proposed a method to address the under-determined problem. Horn and Schunk also assume that the motion field is smooth.

The smoothness assumption also takes care of the situation where there is no gradient. Smoothness is applied by minimizing the term, L_{Smooth},

$$L_{Smooth}(u, v) = (\partial u/\partial x)^2 + (\partial u/\partial y)^2 + (\partial v/\partial x)^2 + (\partial v/\partial y)^2 \tag{2.5}$$

Based on the Horn and Schunk method, the total loss to minimize is a weighted sum of L_{Data} and L_{Smooth},

$$L = \int (L_{Data} + \gamma L_{Smooth}) dX \tag{2.6}$$

where $X = \binom{x}{y}$ and γ weights the smoothness term. Choosing a quadratic penalty function, $f(x) = x^2$, as in [57, 38] makes the optimization much easier, although it has a strong influence on outliers. The next option would be to use robust statistics [40] to reduce the influence of outliers. L_1 norm [16], the Tukey function [64], the Lorentzian norm [13], and the Leclerc's function [60] are robust substitutions to quadratic penalty function. Horn and Schunk minimize their proposed energy using variational approaches [38].

According to [9], variational methods have been dominant until the application of deep neural networks. There are several advantages to variational methods. One is that in variational methods, it is possible to have several assumptions in one optimization problem. The other is that variational methods yield dense flow field, whilst many of other methods need interpolation as a post-processing step to densify the sparse estimated flow field. The estimation of motions with large magnitude was yet an issue which was later solved by embedding a motion estimation method in a multiscale coarse-to-fine scheme. The history of using spatial pyramid goes back to [19] and was first used for motion estimation in [30]. Spatial pyramids are also used

in other computer vision applications, more specifically by deep neural networks such as in [23] where generative image models are learned.

Since Horn and Schunk, some methods have been proposed to improve the data term by introducing a pre-processing step using Gaussian filtering [16, 18, 101]. Also, some other methods were proposed to add to the robustness against illumination changes. These methods can be categorized into three main schemes:

Structure texture decomposition. Based on the idea in [7], Wedel et al. [92] used the Rudin-Usher-Fatemi technique [74] to separate the texture from the structure. The idea is that illumination change affects mainly the structure although dropping the structure means to loose an information that can be useful.

Color space. Methods in this scheme estimate the motion field in color spaces that are more robust towards the illumination changes. Some examples in this direction are to use HSI color space [87, 31], the normalized RGB channels [31], and the HSV space [101, 62].

More robust constancy constraints. Brox et al [16] propose to add an extra constraint, gradient term, to the loss function which makes the method robust against intensity value changes. Gradient term penalizes the deviation from the constant gradient of the intensity. Accordingly, the data loss updates to,

$$L = \int \psi(|E(X+F,t+\delta t) - E(X,t)|^2 + \gamma|\nabla E(X+F,t+\delta t) - \nabla E(X,t)|^2)dX \tag{2.7}$$

where $\psi(x) = \sqrt{x^2 + \epsilon}$ is the Charbonier penalty function which is the robust version of $L1$ norm. Minimizing the energy function proposed in [16] allows for estimation of accurate dense motion field for small motions. Papenberg et al. [66] suggested that constancy of the Hessian and the Laplacian are also useful. Zimmer et al. [102] used similar constraints [16] but in HSV space to tackle the problems caused by the illumination changes. Mohammed et al. [63] suggested using a texture constancy constraint.

Recently, there has been considerable interest in using feature matching to add robustness to the estimated motion field. The SIFT-flow method [54] employs SIFT [56] features to estimate the motion field. Although, using SIFT features do not allow for estimation of small displacements. Brox et al. [17] and Weinzaepfel et al. [93] use descriptors matched between two frames integrated into a variational approach. Brox et al. in [17] combines the advantage of both energy minimization methods which yield dense motion field for small motions and descriptor matching which allows finding large displacements. Brox et al. [17] , compared to [16], propose an extra term in the smoothness loss that preserves the high frequencies - edges. Brox et al. [17] use a segmentation method to find regions in the frame and produce region descriptors that are later used to find a sparse set of hypothesis for correspondences. These hypotheses, initial matches, are then integrated into a variational approach. Their energy function is similar to [16] but, with one additional term which integrates the correspondence information. Weinzaepfel et al. in [93] first propose a new non-rigid matching algorithm which can retrieve smooth dense correspondence and then they suggest a method for combining the matches with a variational approach for motion estimation. The method proposed in [70] has three steps: First, to find the matched features between the two frames which forms a sparse set. Second, to perform densification of this sparse set of matches by computing a sparse-to-dense edge-aware interpolation. Third, they perform one step of variational energy minimization using the dense interpolation as initialization. Although this method does not suffer from

previous shortcomings, like the other, it is still relaxing handcrafted constraints. There is a drawback with the methods which use feature matching and that is they rely on salient points and false matches are unavoidable. Stoll et al. [79] suggest an adaptive method for the integration of feature matching and variational part to mitigate the effect of false matches. Zin [15] applied segment matching to improve the matching component. Revaud et al. [71] used HOG features to reduce false matches.

Model-based approaches dominated the motion estimation field for years. Even though they are very successful in some cases, their performance is limited by the approximate assumptions they make about image brightness and spatial structure of the flow. In parallel, machine learning techniques have been used before in the estimation of optical flow. For example, local statistics of optical flow were modeled using Mixtures of Gaussian models [73]. In [81], in addition to studying Statistics of optical flow, regularizers were learned using a mixture of Gaussians. The method proposed in [14] assumes a motion field is a combination of some principal components and learn the coefficients in the combination. Kennedy and Taylor [48] classify the motion field among several poorly estimated flow fields. Recently, to overcome the limitations of model-based approaches, DNN-based algorithms were exploited to learn to estimate motion fields from the data. As shown by our experiments in the next chapter, we show that if a DNN is trained in an unsupervised way using a classic loss function, Horn and Schunk method [38], it performs better than the classic method. Still, there is a gap between the performance of more recent classic and unsupervised DNN-based methods. This gap can be addressed by introducing a more sophisticated loss function and more expressive and practical architecture.

2.2 DNN-based Methods

Recently, Deep Neural Networks (DNNs) have shown to have promising performance in several Computer Vision problems [52]. The exploited DNNs, more specifically Convolutional Neural Networks (CNNs), are high-capacity models that approximate the complex, non-linear transformation between input imagery and the output. Success with CNNs has relied almost exclusively on fully-supervised schemes, where the target value (i.e., the label) is provided during training, although unsupervised methods are growing more and more. Several supervised and unsupervised training techniques have been proposed for the training of the DNNs for motion estimation which will be briefly reviewed in this section. Supervised methods have shown that a DNN, with a well-designed architecture, can achieve promising performance if trained on a populated enough dataset with ground truth [41, 28, 82]. Supervised methods mainly minimize the Average End-point Error (AEE) for training. On the other hand, unsupervised methods minimize a proxy loss function which minimization, ideally, minimizes the average end-point error. A well-designed cost function is one of the key factors in the unsupervised training of DNNs for motion estimation.

2.2.1 Supervised Methods

Training a CNN for motion estimation in a supervised way means that given a set of frame pairs and their ground truth motion fields, a learning-based method for motion estimation can be learned that can estimate the motion field for an unknown pair of frames. Let $\{I_{1k}, I_{2k} \in \mathbb{R}^{w \times h \times 3}, F_k \in \mathbb{R}^{w \times h \times 2}\}_{k=1}^{N}$ represent the training dataset where I_{1k} and I_{2k} are the input pair of frames, first frame and the second frame, and F_k is their corresponding ground truth motion field. The aim is to learn a model H such that $H(I_{1k}, I_{2k}) = F_k$ which can estimate the motion field for an unseen frame pair during the test time. Mainly, Average End-point Error (AEE) is used to evaluate the performance of a motion estimation algorithm. This measure is also minimized to

train a DNN. AEE is defined in the following,

$$AEE = \sum_{k=1}^{N} |\hat{F}_k - F_k|^2 \tag{2.8}$$

where $|.|^2$ is the mean square error and $\hat{F}_k$ is the motion field estimated by the DNN. To the best of our knowledge, all of the DNN-based supervised methods solve motion estimation as a regression problem and rely on minimization of Eq. 2.8 for training. Since obtaining ground truth motion field for real scenes is not easy, supervised methods rely on synthetically made datasets. The first and One of the commonly used datasets for training is the flying chairs dataset [25]. This dataset is explained here as it is used by most of the supervised an unsupervised methods for training. There are other datasets which are mainly used for evaluation and are described in section 3.3. This dataset is obtained by overlaying the randomly parameterized affine transformed version of a rendered set of 3D chair models [6] on random background drawn from Flickr dataset [36]. Figure 2.2 and Fig. 2.3 respectively illustrate samples from flying chairs dataset [25] and the general scheme most of the DNN-based methods follow for motion estimation. For visualization purposes, the direction and magnitude in the motion field are encoded according to a color wheel.

The first supervised method was proposed by Dosovitskiy et al. in [25] for motion estimation. FlowNet is very similar to DeepFlow [93] as both methods aggregate features from fine to coarse using convolution and max-pooling, although in DeepFlow no parameter is learned. FlowNet has shown a performance that was close to the state-of-the-art classic methods in a number of synthetically generated image sequences. Dosovitskiy et al. in [25] propose two Convolutional Neural Networks (CNN) trained for motion estimation which differs in their early convolutional layers.

FlowNet Simple, FlowNetS, is composed of conventional convolutional layers which

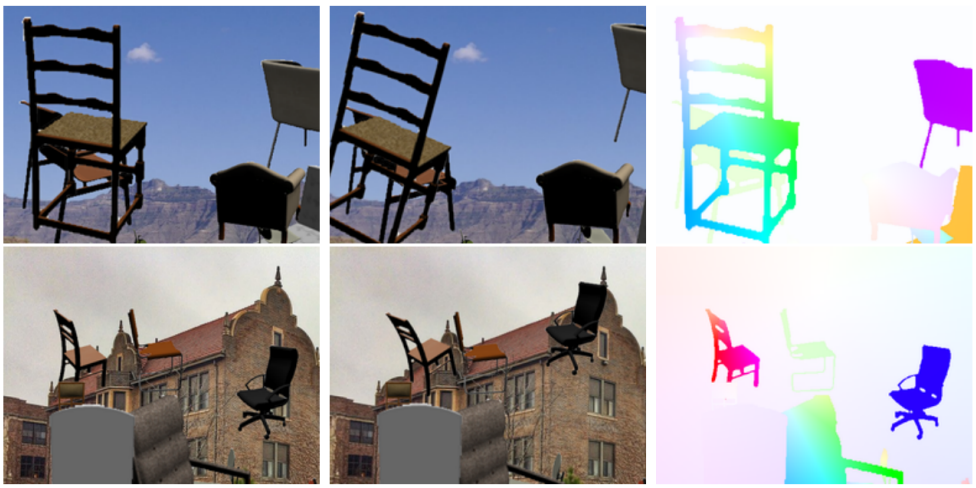

Figure 2.2: Two samples from Flying Chairs dataset [25]. The pair of images and the visualization of their corresponding groundtruth motion field.

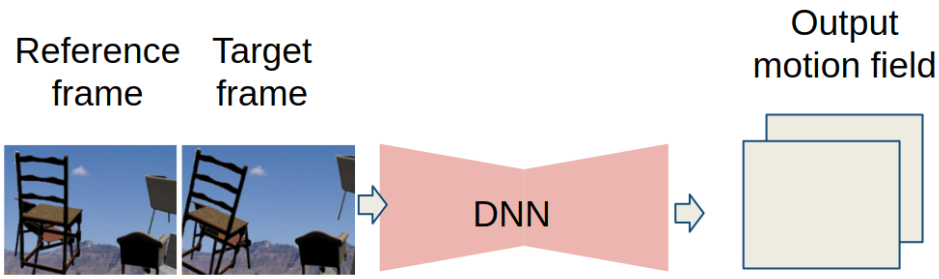

Figure 2.3: The general scheme for motion estimation using DNNs.

receive the input pair of images as 2×3 channels of data and calculates 2 channels of motion field in the output. The architecture is hourglass-like and is composed of two parts, the encoder and the decoder, Fig. 2.4 upper architecture. The encoder builds abstract representations through 9 convolutional layers with 6 down-sampling steps of factor 2. The decoder builds towards high-resolution motion field layer by layer through 5 convolutional layers with up-sampling. Each layer of the decoder receives information through skip connections from the same scale on the encoder side. Each layer of the decoder side outputs a motion field for which it receives supervision and then is up-sampled and concatenated with the representations calculated by its next layer.

FlowNet Correlation, FlowNetC, is very similar to FlowNetS. The difference is that the two input frames are fed into two Siamese streams, each composed of 3 convolutional layers that share parameters. The idea is that first, meaningful representations are first extracted from the input frames. The computed representations are then joined through a correlation layer and the rest is similar to FlowNetS architecture, Fig. 2.4 lower architecture. Both FlowNetS and FlowNetC have skip connections from the encoding side of their architecture to the decoding side, which helps to import lower level features to higher layers to mainly keep the higher frequency information. The idea behind this kind of architecture is to learn strong features at multiple scales and abstractions and to ease finding the actual correspondences based on these features. Most of the supervised and unsupervised DNN-based methods are inspired by this architecture. FlowNet is reported to outperform some of the classical state-of-the-art methods, table 3.1. FlowNet uses FlyingChair dataset [6] for training and minimize AEE as training loss. As reported, FlowNetC overfits easier to the kind of data it is provided for training.

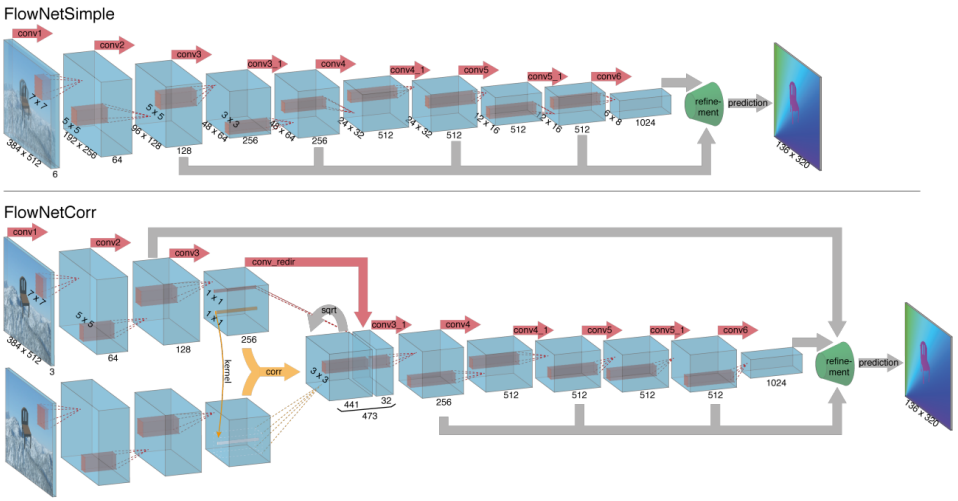

Figure 2.4: The two FlowNet architectures: FlowNetS (top) and FlowNetC (bottom). The figure is from [25].

Flownet 2.0 [41] is an improvement to FlowNet and tries to address the shortcomings of FlowNet. More specifically, FlowNet 2.0 improves the estimation of small displacements and the quality of the estimated motion field. These improvements are realized by stacking multiple networks that are specialized in the estimation of both large and small displacements. A cascade of FlowNetC followed by two FlowNetS is trained to estimate large displacements, each one improving the estimation of the previous one. The first one receives, as input, a pair of frames and the latter ones additionally receive the output of the previous network, the motion compensated intensity error, and the warped version of the second frame. Also, a variation of FlowNetS is trained to estimate small displacements. Another network is trained to fuse the output of the cascade and the small displacement network. The whole framework is end-to-end trainable and significantly increases the quality of the estimated motion field, of course by slightly sacrificing the speed. A dataset scheduling is followed that improves the results. FlowNets are initially trained on FlyingChairs [6] and then fine-tuned on FlyingThings3D [58] and then fine-tuned on a mixture of both datasets. They show that this schedule and order of using the datasets is important as FlyingChairs is simpler and helps to learn basic matching concepts before making any confusing priors on more complicated displacements in 3D space. Finetuning on FlyingThings3D which is more realistic complements the already learned concepts. Although these datasets do not characterize real-world data but allow for generating arbitrary amounts of samples with custom properties. Figure 2.5 illustrates the architecture used by FlowNet 2.0.

Another method has been proposed by Ranjan and Black [67] which deals with large displacements by employing DNNs in a classic coarse-to-fine scheme. This way the number of the learned parameters reduces significantly compared to FlowNet yet achieving a higher performance compared to FlowNet. The coarse-to-fine scheme uses a spatial pyramid. The idea is that in the lower resolution, the displacements between the input pair are smaller and though the context can be captured easily by convolutional

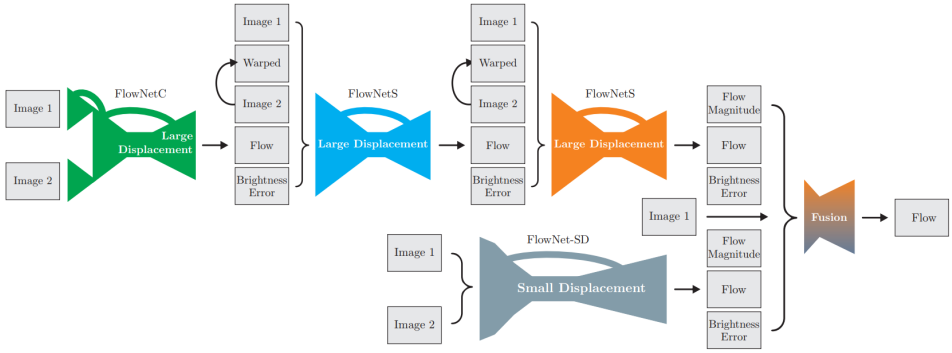

Figure 2.5: Schematic view of complete FlowNet2.0 architecture: To compute large displacement optical flow multiple FlowNets are combined. Braces indicate concatenation of inputs. Brightness Error is the difference between the first image and the second image warped with the previously estimated flow. To optimally deal with small displacements, smaller strides are introduced in the beginning and convolutions between up-convolutions into the FlowNetS architecture. Finally, a small fusion network is applied to provide the final estimate [41]. "Image 1" and "Image 2" are respectively reference frame and target frame.The figure is from [41].

filters in a ConvNet. The motion is estimated at each level of the pyramid, scaled and up-sampled to be used in the next scale. Following the classic approaches [80], the motion field obtained from the previous level is used to warp the second frame towards the first frame which is used with the first frame as a pair to be fed again to the motion estimation DNN. The estimated motion field is used to update the motion field from the previous level before moving to the next level. This continues until the motion field in the main resolution is calculated. The diagram of the multiscale scheme is illustrated in Fig. 2.6.

The difference between SpyNet [67] and other supervised methods up to this point is that instead of estimating a full motion field, the DNN is meant to predict the flow increment at that level. More specifically, the network learns to estimate residual flow at each pyramid level. The network is trained from coarse to fine to learn the flow correction at each level and add this to the flow output of the network in the previous level. This way, the estimated displacements stay small in each level. Figure 2.7 illustrates the SpyNet framework.

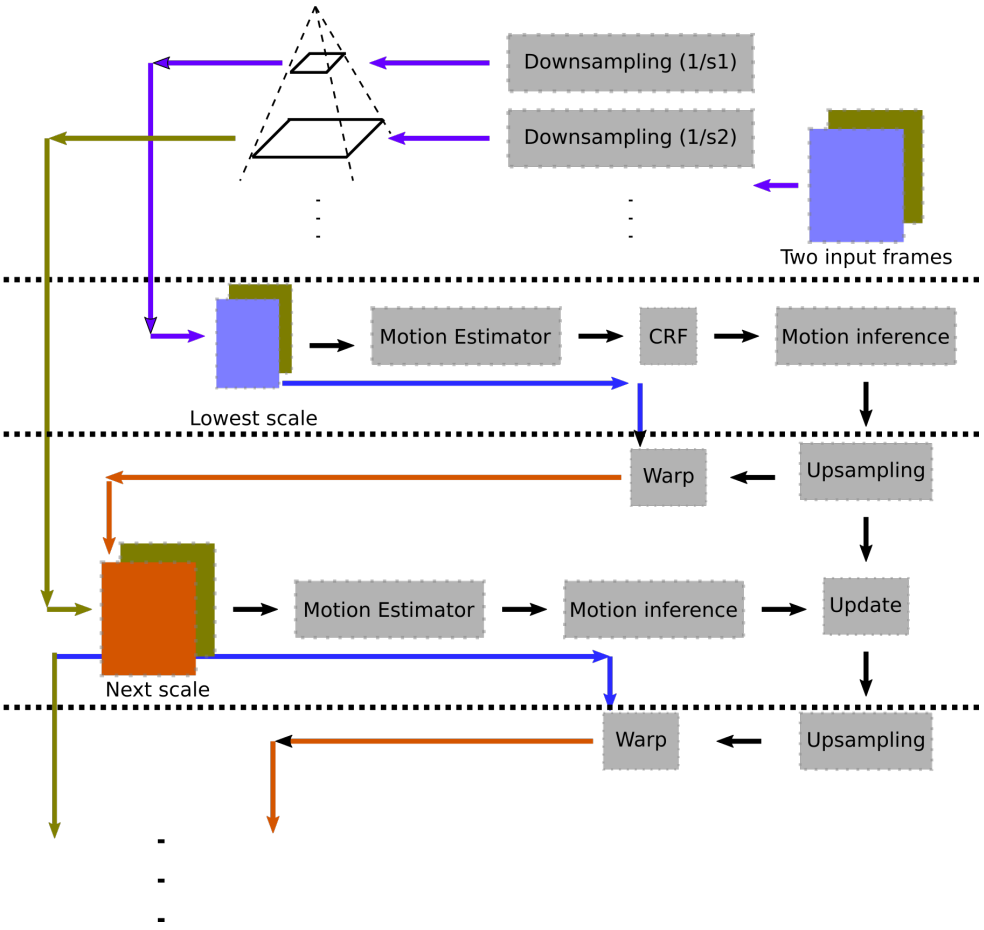

Figure 2.6: The schematic for classic multiscale scheme for motion estimation. *Sk* represents the scaling factor in the pyramid and k indexes the level.

SpyNet shows that using classic principles in combination with DNNs is potentially beneficial. However, some important classic principles have been missing, one of which is that the DNN-based methods all receive as input the raw images. Most of the classic methods, preprocess the input and operate based on the extracted features that are robust towards the shadow, or light changes [91, 9].

PWC-Net [82] addresses these shortcomings by making use of classic principles. The method follows the classic multiscale scheme. In each scale, first, a convolutional neural network calculates more descriptive representations in different scales for each of the input pair of frames. These representations are then used to calculate a cost

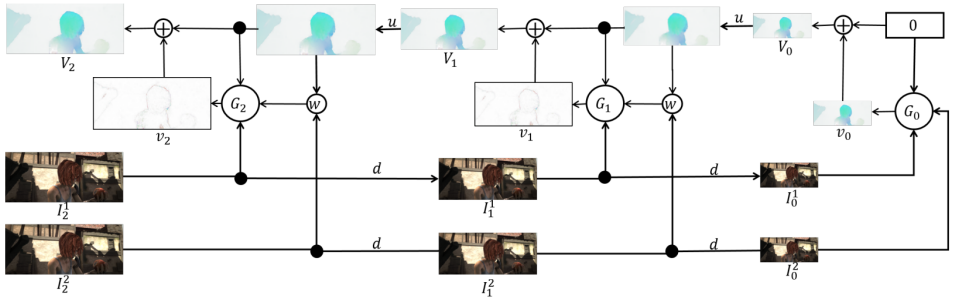

Figure 2.7: The network G_0 computes the residual flow v_0 at the highest level of the pyramid (smallest image) using the low resolution images $\{I_0^1, I_0^2\}$. At each pyramid level, the network G_k computes a residual flow v_k which propagates to each of the next lower levels of the pyramid in turn, to finally obtain the flow V_2 at the highest resolution. The figure is from [67].

volume arguing that the cost volume is a more discriminative representation of the disparity than raw intensity features. A cost volume stores the data matching costs for relating corresponding pixels in the two input frames [39]. Cost volumes have been used for motion estimation before as well [97, 21], however, the cost volume has been used in a single scale which adds significantly to the computational load. The method proposed in [82] exploits the cost volume in several levels of a pyramid to avoid the computational load while keeping the benefits of using a cost volume. The computed cost volume is then fed to a convolutional neural network which calculates a motion field. The motion field is then refined by another network. the refined motion field is up-sampled and up-scaled for further refinement in the next scale. Figure 2.8 summarizes the key components of PWC-Net and compares it beside the classic coarse-to-fine approaches [16, 38, 80, 13].

In Fig. 2.8, the cost volume is generated by calculating the correlation between the features from the first frame and the warped version of the features from the second frame. The CNN that estimates the motion field, receives as input the cost volume, the features from the first frame and the up-sampled motion field from the previous level of the pyramid and outputs the flow field. The number of feature channels at each convolutional layer is 128, 128, 96, 64, and 32. This architecture is kept fixed for

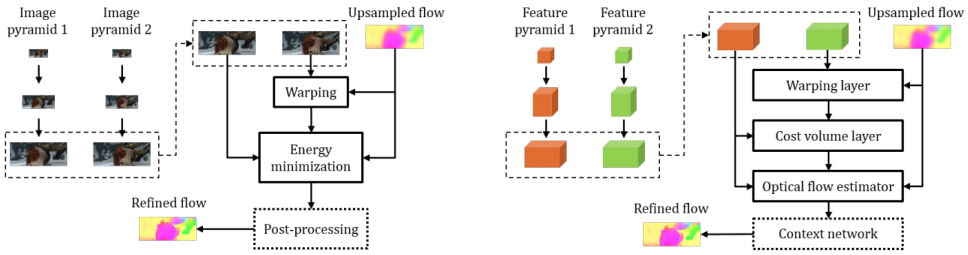

Figure 2.8: On the left, classic coarse-to-fine scheme and the related energy minimization. On the right, feature pyramid and refinement at one pyramid level by PWC-Net. The figure is from [82].

all levels of the pyramid.

2.2.2 Unsupervised Methods

Supervised training of the DNNs requires a lot of training data. Since it is difficult to obtain dense motion field for real data, all the supervised DNN-based methods rely on synthetically generated datasets, as it is easy to generate samples in large amount [58, 41, 28]. Some of the datasets that are commonly used for training and evaluation will be explained in details in the next chapter. However, the synthetic dataset does not characterize real data and generalizing to real datasets is still a challenge. Several unsupervised trained networks have been proposed that try to address this issue.

Although the evaluation is still based on the AEE, unsupervised methods, unlike the supervised methods, minimize a proxy loss function that ideally minimizes the AEE. The proxy loss functions mainly penalize the deviation from the classic feature constancy assumption, Eq. 2.9.

$$MCIE = \sum_{k=1}^{N} |warp(I_{2k}, \hat{F}_k) - I_{1k}|^2 \tag{2.9}$$

where $warp(I_{2k}, \hat{F}_k)$ returns the warped version of I_{2k} with respect to the motion field F_k. In most of the unsupervised methods, warping is implemented using the spatial transformer layer [42]. This method estimates the intensity value in the destination

based on the bilinear interpolation of the surrounding neighbors. Eq. 2.10 expresses how the interpolation is performed.

$$V_i^C = \sum_n^H \sum_m^W I_{nm}^c max(0, 1 - |x_i^s - m|) max(0, 1 - |y_i^s - n|) \tag{2.10}$$

where (x_i^s, y_i^s) defines the spatial location in the input where a sampling kernel is applied to get the value at a particular pixel in the output V.

Unsupervised training of DNNs for motion estimation started with uni-directional methods which estimate the motion field in only one pass. The bidirectional methods which estimate the motion field in forward and backward passes are more accurate as two-pass estimation allows for motion consistency check as well as taking into consideration the occluded area. Forward pass estimates the displacements from the reference frame to the target frame. Backward pass estimates the displacements from the target frame to the reference frame.

Uni-directional Training Techniques

The first unsupervised DNN-based methods were proposed in [2, 45]. Our method [2] minimizes the classic intensity constancy constraint and linearisation of the warped term is based on the first order Taylor expansion. The DNN learns to estimate motion during the training and embeds in a multiscale scheme during the test. However, this method is not easy to train and there is no mechanism to improve the quality of estimation in the lower resolutions to prevent the propagation of the errors across scales. [45] adopts a similar approach, however instead of linearisation of the motion compensated intensity differences, they utilise a spatial transformer layer [42].

[69] adopt the FlowNetS architecture [25] and train it by penalizing the deviation from the intensity constancy assumption linearized using the method proposed in [42]. The loss function also considers penalizing the deviation from the gradient constancy

assumption and also assumes that the motion field has to be smoothness. The architecture, similar to FlowNetS, has skip connections between the encoder side and the decoder side. The motion field is estimated in lower scales of the architecture where the supervision applies during the training and passed to the next scale after being upscaled. This helps with the estimation of large displacements. However, the up-scaling is embedded in the architecture and therefore any modification in the number of up-scaling layers requires further fine-tuning. All of the methods above treat motion estimation as a regression problem, where the estimation output is a map with 2 channels, corresponding to horizontal and vertical displacements, with spatial dimensions equal to those of the input images.

Bi-directional Training Techniques

Although, several techniques have been proposed for unsupervised training of DNNs for motion estimation, yet there is a large gap between the supervised and unsupervised methods. To further improve the unsupervised training, some methods are proposed which try to take the occlusion into consideration during the training [90, 59]. All of them first try to identify the occluded area and train only based on the gradients from the non-occluded regions. Otherwise, the DNN would falsely learn to move pixels to the occluded area. Addressing this has shown to improve the accuracy during testing [90, 59].

The end-to-end unsupervised approach proposed in [59] builds on recent optical flow CNNs [25, 41] and trains the architectures in an unsupervised way using the photometric loss similar to [45]. To mitigate the occlusion effect during the training, the optical flow is estimated bidirectional in forward and backward direction, see Fig. 2.9. The difference between the forward and backward passes is the order in which the reference frame and the target frame are fed to the network. The occlusion detection is based on the forward-backward consistency assumption proposed in [83]. For occlusions in

forward direction, the occlusion flag o_x^f is defined to be 1 whenever the constraint,

$$|w^f(x) + w^b(x + w^f(x))|^2 < \alpha_1(|w^f(x)|^2 + |w^b(x + w^f(x))|^2) + \alpha_2 \tag{2.11}$$

is violated, and 0 otherwise. In equation 2.11, w^f and w^b respectively denote the forward and backward motion fields. For the backward direction, the backward flag o_x^b is defined in the same way as in equation 2.11, having w^f and w^b exchanged. By integrating the occlusion flags in the training phase, the conventional data loss turns into,

$$\begin{aligned} E_D(w^f, w^b, o^f, o^b) = \sum_{x \in P} & (1 - o_x^f).\rho(f_D(I_1(x), I_2(x + w^f(x)))) + \\ & (1 - o_x^b).\rho(f_D(I_2(x), I_1(x + w^b(x)))) + \\ & o_x^f \lambda_p + o_x^b \lambda_p \end{aligned} \tag{2.12}$$

where $f_D(I_1(x), I_2(x'))$ measures the photometric difference between two corresponding pixels x and x' in I_1 and I_2, and $\rho(x)$ is the robust Charbonier penalty function. In equation 2.12, λ_p represents a constant penalty which penalizes all pixels being considered as occluded, trivial solution. A major difference with the method proposed by Jason et al. [45] is that their loss function penalizes the deviation from the constant intensity assumption which is invariant to illumination changes, whereas illumination changes are very common in a natural scene [88]. Thus, I in equation 2.12 represents the ternary census transform [98, 78]. The census transform has shown to be robust towards additive and multiplicative illumination changes [33]. Another difference of UnFlow [59] with [45] is that the smoothness term in UnFlow is based on second-order smoothness constraint [86, 99] which provides more effective regularization. The schematic of UnFlow [59] is illustrated in figure 2.9.

Wang et al. [90] also proposed a method that models the occlusion during the

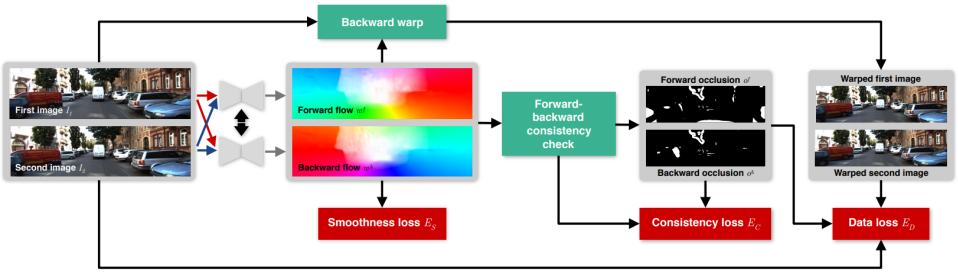

Figure 2.9: Schematic of the proposed unsupervised loss for training UnFlow [59]. The data loss compares flow-warped images to the respective original images and penalizes their difference. The figure is from [59].

unsupervised training. The occlusion identification works based on the fact that in the backward pass, there are some pixels in the target frame that have no source pixel due to occlusion. A reversed bilinear sampling is used to calculate the distribution of the displaced pixel to its nearest neighbors, range map V [4]. Accordingly, the occlusion map, $O(x, y)$, is calculated as $O(x, y) = min(1, V(x, y))$. The core of the training loss function consists of two components: a photometric loss (L_P) and an edge-aware smoothness loss (L_S). The integration of the occlusion map into the training function is done as in Eq. 2.13 as the intensity term and in Eq. 2.16 as the gradient term.

$$L_P^1 = \left[\sum_{i,j} \psi(\tilde{I}_1(i,j) - I_1(i,j)).O(i,j)\right] / \left[\sum_{i,j} O(i,j)\right] \tag{2.13}$$

$$L_P^1 = \left[\sum_{i,j} \psi(\nabla\tilde{I}_1(i,j) - \nabla I_1(i,j)).O(i,j)\right] / \left[\sum_{i,j} O(i,j)\right] \tag{2.14}$$

where $\tilde{I}$ denotes the warped version of I. Multiplying the occlusion map, $O(i, j)$, with the motion compensated intensity and gradient of intensity errors encourages the backpropagation of the error during the training to more focus on the non-occluded areas. This relieves the network from finding correspondences for the areas that are occluded in any of the input frames which is said [90] to improve the accuracy.

Two smoothness terms are exploited where one encourages smoothness through first-order $\delta_d F_{12}$ and the other one through the second-order derivatives, $\delta_d^2 F_{12}$, of the motion field. The exponents suppress the smoothness where there is an edge in the spatial domain in order to respect high frequencies, edges, in the estimated motion field.

$$L_S^1 = \sum_{i,j} \sum_{d \in x,y} \psi(|\delta_d F_{12}(i,j)| e^{-\alpha |\delta_d I_1(i,j)|}) \tag{2.15}$$

$$L_S^2 = \sum_{i,j} \sum_{d \in x,y} \psi(|\delta_d^2 F_{12}(i,j)| e^{-\alpha |\delta_d I_1(i,j)|}) \tag{2.16}$$

where ψ is the Charbonnier penalty formula $\psi(s) = \sqrt{s^2 + 0.001^2}$ over the non-occluded regions with both image brightness and image gradient. The final training loss is the sum of the above four terms,

$$L = \gamma_1 L_P^1 + \gamma_2 L_P^2 + \gamma_3 L_S^1 + \gamma_4 L_S^2 \tag{2.17}$$

A major difference between [90] and other unsupervised techniques that use [42] is that the interpolation is developed to search in a larger area when back-propagating the training error through the warping operation.

The adopted architecture is a modified version of the FlowNetS. In FlowNetS, in the decoder side of the architecture, each layer receives the up-sampled flow field estimated in the previous scale concatenated with the deconvoluted representations of the previous scale and the representations from the corresponding encoder side. In the modified version, each decoder-side layer receives similar input replacing the up-sampled motion field from the previous scale with some auxiliary representations. The

auxiliary representations are the output of a 5 layer CNN, all layers stride 1, which receives as input the down-scaled reference frame, target frame, target frame warped with the motion field estimated in the previous scale and the motion compensated intensity error. The auxiliary representations consist of 2 channels. The estimated motion field in each scale is then added to the up-sampled version of the motion field estimated in the previous scale to form a residual block. Other scales are modified accordingly. The modification is shown in Fig. 2.10. $Image1_6$ and $Image2_6$ are input images down-sampled 64 times.

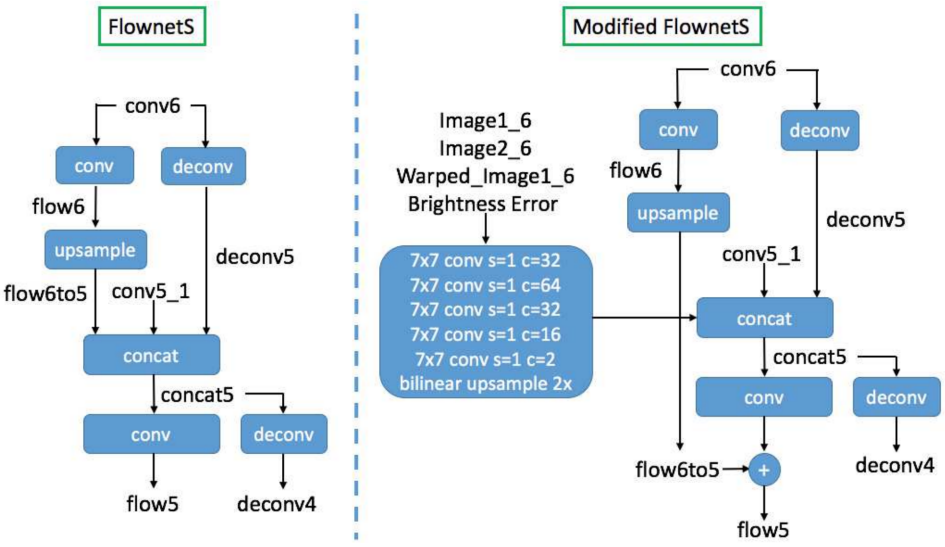

Figure 2.10: The modification to the FlowNetS structure at one of the decoding stage - stage 6. On the left, the original FlowNetS structure is shown. On the right, the modification of the FlowNetS structure is shown. $conv6$ and $conv5_1$ are features extracted in the encoding phase and named after [25]. The figure is from [90]

The training schematic is shown in Fig. 2.11. It contains two FlowNetS [25] architectures with shared parameters. One estimates forward and the other estimates backward optical flow respectively F_{12} and F_{21}. The forward warping module generates an occlusion map from the backward flow. The backward warping module generates the warped image that is used to compare against the original frame 1 over the non-occluded area. The part that calculates forward optical flow has the aforementioned smoothness term in its training loss function in addition to the photometric loss func-

tion. Although the supervision that the backward CNN receives comes from the fact that its output is used for calculating the occlusion map.

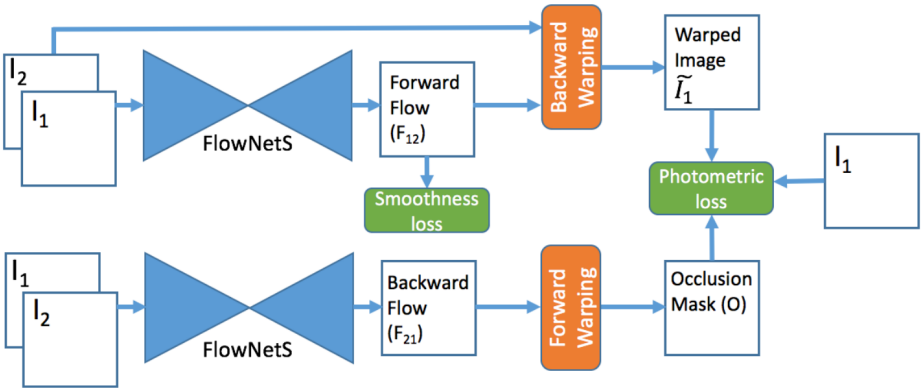

Figure 2.11: The network architecture used for occlusion aware technique, composed of two FlowNetS. The figure is from [90]

Although supervised DNN-based algorithms have shown to have good performance, they require a large amount of ground truth optical flow for training the network's parameters in order to get reasonable accuracy. Labeling real data is expensive and not easy and synthetic data does not fully characterize real data. Provided data, still there is no guarantee that the trained model would perform well enough in an unknown scenario. The unsupervised methods have recently shown the possibility to achieve promising performance. Although they do not need ground truth for training, there is still a large gap between them and their supervised counterpart in terms of performance.

2.3 Conclusion

So far, many classic methods, methods that do not use DNNs, have been proposed for motion estimation. All the classic methods, solve an optimization problem for each sample. The minimized loss function is composed of a few constraints and thus can be under-constrained. Developing an approach that can learn motion estimation directly

from the data is desired. Deep Neural Networks have shown to have promising performance in different applications, more specifically, pixel level prediction tasks such as motion estimation. CNNs are high capacity DNNs that are able to model the nonlinear relationship between the input and the output. The problem with supervised training of DNNs is that a large amount of labeled training data is required and obtaining ground truth motion field for many real scenes is difficult. Most of the unsupervised methods rely on synthetically generated datasets for which the ground truth can be obtained in a large number. Although synthetic data does not characterize real data, primary supervised DNNs trained on synthetic data has shown to perform close to state-of-the-art classic methods. More recent supervised methods benefit from classic principles to improve performance. In parallel, to address the problem of the labeled data, unsupervised motion estimation techniques have been proposed. Unsupervised methods minimize a proxy loss function whose minimization ideally minimizes the AEE and is mainly based on the classic principles. The loss functions used for unsupervised training of DNNs mainly penalize the deviation from the feature constancy assumption. Unsupervised techniques started with uni-directional methods which estimate the motion field in one pass. Recently, bi-directional methods have shown to have promising performance. Bi-directional methods estimate the motion field in both forward and backward passes which help with consistency check and modeling the occlusion.

Chapter 3

GradNet: Gradient-based Unsupervised Training of Deep Neural Networks for Motion Estimation

In this chapter, we present our firstly proposed method for unsupervised training of CNNs for motion estimation. We realize our unsupervised training scheme by exploiting a classical cost function which builds on the widely used optical flow constraint proposed by Horn-Schunk [38]. Our major difference to Horn-Schunk based methods is that the cost function is used only during training and without regularization. Once trained, given a pair of frames as input the CNN gives at its output layer an estimation of the motion field. The cost function is differentiable with respect to the unknown motion field and, therefore, allows the backpropagation of the error and the end-to-end training of the CNN. Furthermore, we improve the way the intensity constancy constraint introduced by Horn and Schunk is relaxed and show how our improvement connects with the spatial transformation technique proposed by Jaderberg et al. [42] used by other unsupervised methods [69, 90].

In order to help with the estimation of motions large in magnitude, we embed the proposed trained network in a classical coarse-to-fine multiscale scheme. Merging DNNs with long proven classical principles was adapted by supervised methods as well later in [67, 82]. We train our CNN using randomly chosen pairs of consecutive frames from the real dataset UCF101 [77] with no information on groundtruth motion. The trained network is then evaluated on unknown real and synthetic datasets. Since our loss function works based on the gradients of the input, we name our trained network GradNet. Figure 3.1 illustrates the overview of the training and test of our first proposed DNN-based motion estimator, GradNet.

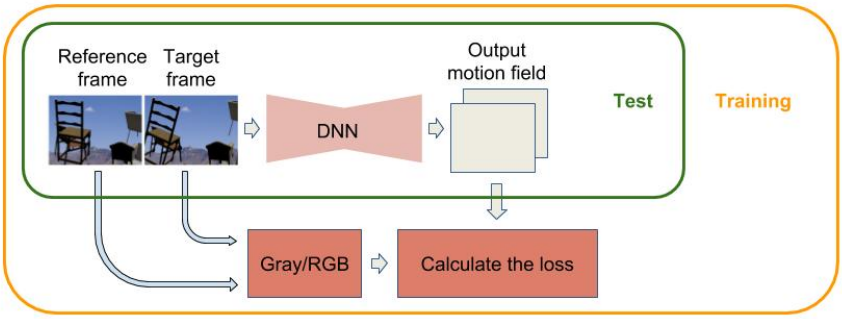

Figure 3.1: The overview of the training and test of GradNet. The green rectangle encloses what is involved during the test. The yellow rectangle encloses what is involved during the training.

3.1 Method

At the heart of all motion estimation methods is the minimization of the difference between features extracted at a certain location (x, y) in the reference frame at t and its correspondence in the target frame at $t + dt$. The classic Horn and Schunck [38] method penalizes the deviation from the assumption of constant intensity, that states that the intensity at a pixel in the reference frame at time t and the intensity at its correspondence at time $t + dt$ are the same. Accordingly, the goal is the minimization

of the Motion Compensated Intensity Error (MCIE), that is,

$$E_D(F) = \sum_{x,y=1}^{M} |I_{u(x,y),v(x,y),\Delta t}(x,y,t) - I(x,y,t)|^2 \tag{3.1}$$

where M is the number of pixels in each of the input frames and,

$$I_{u(x,y),v(x,y),\Delta t}(x,y,t) \triangleq I(x+u(x,y), y+v(x,y), t+\Delta t) \tag{3.2}$$

In Eq. 3.1, $I(x,y,t)$ is the intensity at pixel (x,y) at frame t, and $F(x,y) \triangleq \begin{bmatrix} u(x,y) \\ v(x,y) \end{bmatrix}$ is the unknown motion vector at pixel (x,y). Clearly, F has two components $u(x,y)$ and $v(x,y)$ that are respectively the horizontal and vertical displacements of the pixel with coordinates (x,y). From now on, for convenience, we will remove the summation and write the equations for one pixel, unless mentioned otherwise.

We propose to train GradNet by minimizing Eq. 3.1 in a way that the input to the network is the pair of $I_{u(x,y),v(x,y),\Delta t}$ and I and the output is F. For a computationally feasible backpropagation of error during the training, the loss function has to be differentiable with respect to F. This would be possible if the loss is linear with respect to the network's output. We first study the case where Eq. 3.1 is linearized using a first-order Taylor expansion and then we study the case which the expansion is of a higher order. We also show the connection between the higher order Taylor expansion and the widely used interpolation technique proposed by Jaderberge et al. [42].

3.1.1 First-order Taylor expansion

We first linearize Eq. 3.1 using the first-order Taylor expansion of the warped target frame $I_{u,v,1}$. The Taylor expansion is performed with respect to the horizontal, vertical

and temporal displacements. Based on the literature, the first-order Taylor expansion leads to an approximation of Eq. 3.1, $\hat{E}_D^{1st} = |uI_x + vI_y + I_t|^2$, where I_x, I_y and I_t are respectively the horizontal, vertical and temporal intensity derivatives of the first frame. Minimizing this equation penalizes the deviation from the intensity constancy assumption. In the equation above and later on, we may omit the pixel coordinates for notation simplicity. During the training we use the more robust Charbonnier penalty function, $\rho(x) = \sqrt{x^2 + \epsilon}$, which is a differentiable variant of the robust convex function $L1$ norm. ϵ represents a small number, in our experiments $\epsilon = 0.001$. The final loss function would be,

$$\hat{E}_D^{1st} = \sqrt{(uI_x + vI_y + I_t)^2 + \epsilon} \tag{3.3}$$

While there are many formulas for approximate differentiation [5, 34], we use the kernels that are used in the method proposed by Horn and Schunck [38]. Figure 3.2 illustrates the two-channel kernels that perform on the inputs, which are grayscale, to calculate the horizontal derivatives, I_x, and vertical derivatives, I_y respectively on the left and on the right-hand side. The temporal derivative I_t is calculated by simply subtracting the reference frame from the target frame.

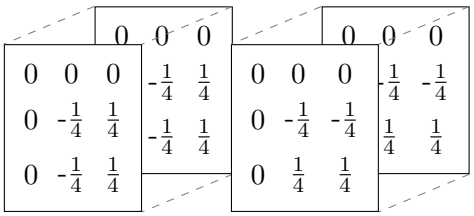

Figure 3.2: On the left is the kernel that is used to calculate the horizontal derivatives and on the right is the kernel that is used for calculation of the vertical derivatives.

Derivative of the Loss with Respect to the Weights

The minimization of the loss is computationally tractable only if the derivative of the loss with respect to F can be calculated in a closed form. Since, the motion field F is a function of the weights w of the CNN the loss, $\hat{E}_D^{1st}$, is also a function of the CNN weights. More importantly, our loss function allows us to calculate the derivatives of it with respect to the network weights. Specifically, using the chain rule,

$$\frac{\partial \hat{E}_D^{1st}}{\partial w} = \frac{\partial \hat{E}_D^{1st}}{\partial F}\frac{\partial F}{\partial w}. \tag{3.4}$$

The second part, that is $\frac{\partial F}{\partial w}$, is the partial derivatives of the output F of the CNN with respect to its weights w. This can be calculated in a classical manner using the standard form of the backpropagation algorithm.

$$\frac{\partial \hat{E}_D^{1st}}{\partial F} = \begin{bmatrix} \frac{\partial \hat{E}_D^{1st}}{\partial u} \\ \frac{\partial \hat{E}_D^{1st}}{\partial v} \end{bmatrix} = \begin{bmatrix} \sum_{x,y=1}^{M} \frac{I_x(uI_x+vI_y+I_t)}{\sqrt{(uI_x+vI_y+I_t)^2+\epsilon}} \\ \sum_{x,y=1}^{M} \frac{I_y(uI_x+vI_y+I_t)}{\sqrt{(uI_x+vI_y+I_t)^2+\epsilon}} \end{bmatrix}. \tag{3.5}$$

3.1.2 Second-order Taylor expansion

The first-order Taylor expansion has been the most widely used technique in classical methods. Since the classical methods solve an optimization problem for each sample, using a more accurate higher-order expansion is not computationally beneficial. Although it is not a problem in case of DNNs as once trained, only the DNN is applied during the test time. We will also show later, Fig. 3.4, that a DNN trained using a higher-order expansion performs better compared to when it is trained using the first-order expansion. In this section, we suggest using a higher-order Taylor expansion when training GradNet so that the calculation of the derivatives of Eq. 3.1 with respect to unknowns u and v is feasible. Furthermore, in the case of the first-order expansion, expanding around x and y requires the u and v to be small. In this section, we propose the expansion to be around (x, y, t) to remove the small motion constraint

in the case of first-order expansion. We propose a second-order expansion along spatial axis, x, y, and a first-order expansion along the time axis, t. To describe our higher order expansion, we first rewrite Eq. 3.1 as follows,

$$E_D(F) = |I(x + \lfloor u(x,y) \rfloor + \alpha, y + \lfloor v(x,y) \rfloor + \beta, t + \gamma) - I(x,y,t)|^2 \tag{3.6}$$

where

$$\begin{aligned} \forall \alpha, \beta \in [0,1] \\ \alpha(x,y) \triangleq u(x,y) - \lfloor u(x,y) \rfloor \\ \beta(x,y) \triangleq v(x,y) - \lfloor v(x,y) \rfloor \\ \gamma = 1 \end{aligned} \tag{3.7}$$

The approximation of E_D becomes,

$$\hat{E}_D^{2nd}(F) = |T(F)|^2 \tag{3.8}$$

where,

$$\begin{aligned} T(F) = \alpha I_x + \beta I_y + I_t + \\ \frac{1}{2!}(\alpha\beta I_{xy} + \alpha I_{xt} + \beta I_{yt} + \alpha^2 I_{xx} + \beta^2 I_{yy}) \end{aligned} \tag{3.9}$$

In Eq. 3.9,we calculate I_x, I_y, I_{xy}, I_{xx}, I_{yy} by using the following derivative filters respectively, $\frac{1}{2}\begin{bmatrix} 0 & -1 & 1 \end{bmatrix}$, $\frac{1}{2}\begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix}$, $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & -1 & 1 \end{bmatrix}$, $\begin{bmatrix} 1 & -2 & 1 \end{bmatrix}$, and $\begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix}$ at point

(x, y, t). I_t is calculated by subtracting I_2 from I_1. I_{xt} and I_{yt} are calculated using the combination of the already defined operations, I_x, I_y, and I_t .

Derivative of the Loss with Respect to the Weights

Similar to in the case of first-order Taylor expansion, the derivatives of Eq. 3.8 with respect to F can also be calculated in a closed form,

$$\frac{\partial \hat{E}_D^{2nd}}{\partial w} = \frac{\partial \hat{E}_D^{2nd}}{\partial F}\frac{\partial F}{\partial w}. \tag{3.10}$$

The second part of Eq. 3.10, that is $\frac{\partial F}{\partial w}$, is the partial derivative of the output F of the CNN with respect to its weights w. This can be calculated in a classical manner using the standard form of the backpropagation algorithm. The first term, that is $\frac{\partial \hat{E}_D^{2nd}}{\partial F}$, removing the constant terms in equation 3.6, can be calculated in closed form as

$$\begin{aligned}\frac{\partial \hat{E}_D^{2nd}}{\partial F} &= \begin{bmatrix} \frac{\partial \hat{E}_D^{2nd}}{\partial \alpha} \\ \frac{\partial \hat{E}_D^{2nd}}{\partial \beta} \end{bmatrix} = \\ &\begin{bmatrix} \sum_{x,y=1}^{M} 2(I_x + \frac{1}{2!}(\beta I_{xy} + \gamma I_{xt} + 2\alpha I_{xx})(T(x, y, F)) \\ \sum_{x,y=1}^{M} 2(I_y + \frac{1}{2!}(\alpha I_{xy} + \gamma I_{yt} + 2\beta I_{yy})(T(x, y, F)) \end{bmatrix}.\end{aligned} \tag{3.11}$$

Double Expansion

When using a Taylor expansion, for a more accurate approximation of the target point, the expansion should be around a node that is as close as possible to the target point. For this purpose, we divide the subpixel area into two regions $\alpha < 1 - \beta$ and $\alpha > 1 - \beta$ and depending on where the target point is, try to expand around the closest node. We propose to expand along the opposite points $(x + \lfloor u(x, y) \rfloor, y + \lfloor v(x, y) \rfloor)$

if $\alpha < 1 - \beta$ and $(x + \lceil u(x,y) \rceil, y + \lceil v(x,y) \rceil)$ if $\alpha > 1 - \beta$. Figure 3.3 illustrates the aforementioned opposite points respectively in blue and green color.

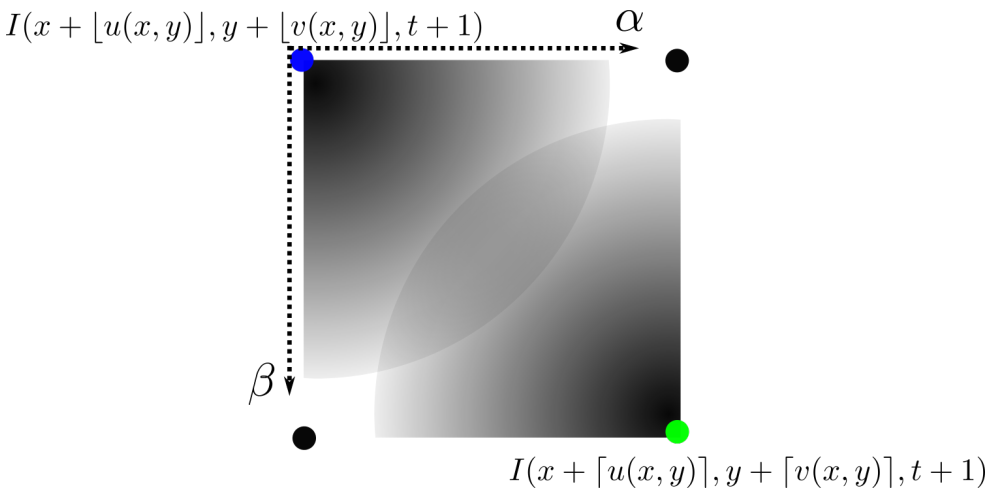

Figure 3.3: Expansion around opposite pixels

3.1.3 Taylor Expansion - Interpolation, Connection

In this section, we show that if a specific polynomial is fitted to a specific set of pixels in the spatiotemporal space of the input frames, the second-order Taylor expansion in Eq. 3.8 when the derivatives are calculated the specific way proposed in Sec. 3.1.2, can be obtained. The higher-order expansion is meant to approximate the intensity values in the target frame. We implement this by fitting a polynomial of degree 2, to a specific set of neighboring pixels. For this purpose, we draw the terms of the higher-order Taylor expansion in Eq. 3.9 from the following general formula of polynomials,

$$I_{\alpha(x,y),\beta(x,y),\gamma}(x,y,t) = \sum_{k=0}^{n_k} \sum_{i=0}^{n_i} \sum_{j=0}^{n_j} C_{ijk} \alpha^i \beta^j \gamma^k \tag{3.12}$$

which would be:

$$\begin{aligned} I_{\alpha,\beta,\gamma}(x,y,t) = C_{000} + C_{100}\alpha + C_{010}\beta + C_{001}\gamma + \\ \frac{1}{2!}(C_{110}\alpha\beta + C_{101}\alpha\gamma + C_{011}\beta\gamma \\ + C_{200}\alpha^2 + C_{020}\beta^2) \end{aligned} \tag{3.13}$$

The polynomial with the higher-order Taylor expansion terms, Eq. 3.13, has 9 unknowns, $C_{ijk}(x,y,t)$, that can be calculated by evaluating the polynomial at 9 points, blue pixels in Fig.3.5-c. For further details on how to calculate the unknowns, we refer to Appendix A. By calculating the unknowns, C_{ijk}s, Eq. 3.13 can be written as in Eq. 3.14. This way, we arrive at the second order Taylor expansion around $(x + \lfloor u \rfloor, y + \lfloor v \rfloor, t)$ when the derivatives are calculated point-wise similar to in Sec. 3.1.2. Equation 3.14 shows how the expansion looks like.

$$\begin{aligned} I_{\alpha,\beta,1}(x + \lfloor u \rfloor, y + \lfloor v \rfloor, t) = I_{000} + \alpha I_x + \beta I_y + I_t + \\ \frac{1}{2!}(\alpha\beta I_{xy} + \alpha\gamma I_{xt} + \beta\gamma I_{yt} + \alpha^2 I_{xx} + \beta^2 I_{yy}) \end{aligned} \tag{3.14}$$

3.1.4 Connection to Spatial Transformation Networks

Now if the following terms are picked from the general formulation of the polynomials in Eq. A.1,

$$I_{\alpha,\beta,\gamma}(x,y,t) = C_{000} + C_{100}\alpha + C_{010}\beta + C_{001}\gamma + C_{110}\alpha\beta \tag{3.15}$$

by fitting it to the specific set of red pixels shown in Fig. 3.5-b, the bilinear interpolation proposed by jaderberg et al. [42] can by obtained. For further details on fitting

the polynomial to the specific set of pixels we would refer to the Appendix. A. The fitted polynomial has the following form:

$$I_{\alpha,\beta,\gamma}(x,y,t) = I_{0,0,\gamma} + \alpha I_x + \beta I_y + \alpha\beta I_{xy} \tag{3.16}$$

where I_x, I_y, and I_{xy} are respectively calculated by the following derivative filters $\begin{bmatrix} 0 & -1 & 1 \end{bmatrix}$, $\begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix}$, and $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & -1 & 1 \end{bmatrix}$. By expanding and rearranging the terms, the bilinear interpolation terms can be obtained.

$$I_{\alpha,\beta,1}(x,y,t) = (1-\alpha)(1-\beta)I_{0,0,1} + \alpha(1-\beta)I_{1,0,1} + \beta(1-\alpha)I_{0,1,1} + \alpha\beta I_{1,1,1} \tag{3.17}$$

3.1.5 First-order vs. Second-order Expansion

In this section, we study how the first-order expansion performs compared to second-order expansion. We plot how the training loss, $\hat{E}$, and the validation Motion Compensated Intensity Error (MCIE) varies in both cases when the training and validation samples are drawn from UCF101 [77] dataset. Figure 3.4 illustrates the comparison.

As can be seen, the training loss, $\hat{E}$ reduces monotonously during the training for both cases. In the case of first-order, the validation MCIE reduces up to some gradient steps and then slightly starts increasing. That is while for the case of second-order expansion, MCIE significantly drops in comparison to the first-order case with no increase. The second-order expansion shows to lead into a better performance, we consider it as the main approach for training GradNet.

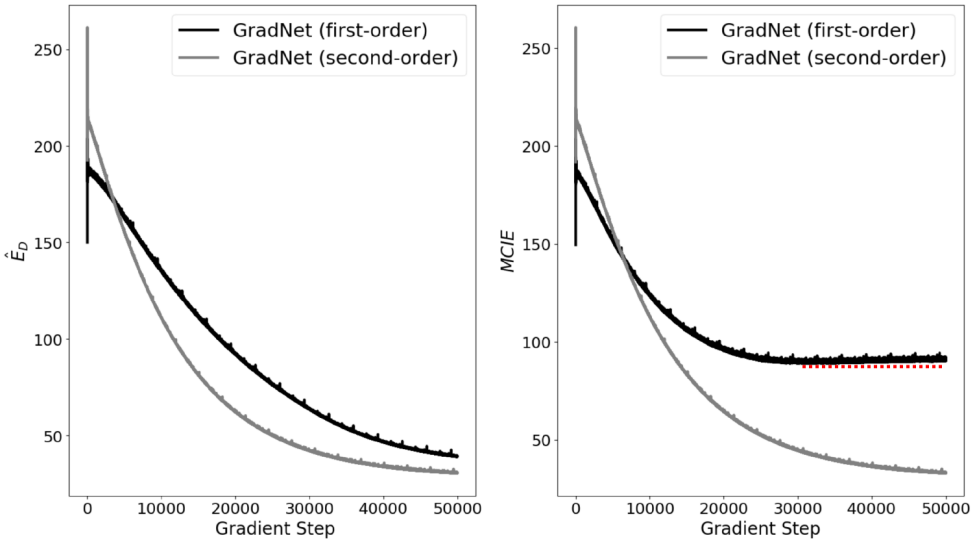

Figure 3.4: The left plot shows how the training loss varies for both cases where the loss function is based on a first-order expansion and a second-order expansion. The right plot shows how the validation MCIE varies during the training.

3.1.6 Loss Augmentation

Following the literature, we also assume that the motion field has to be smooth. We penalize the deviation from the smoothness assumption by adding the term in equation 3.18 to the training loss.

$$E_S(F) = (|\nabla u|^2 + |\nabla v|^2) \tag{3.18}$$

Putting all together, the final energy function that can be used to train GradNet is:

$$E_{tot}(F) = \hat{E}_D(F) + \zeta E_S(F) \tag{3.19}$$

To estimate motions of large magnitude, following the dominant paradigm in the field, we embed our method in a coarse-to-fine multiscale scheme. At the test time, that is once trained, given a pair of frames as input the CNN gives as output the motion field that describes how pixels move from the reference frame to the target

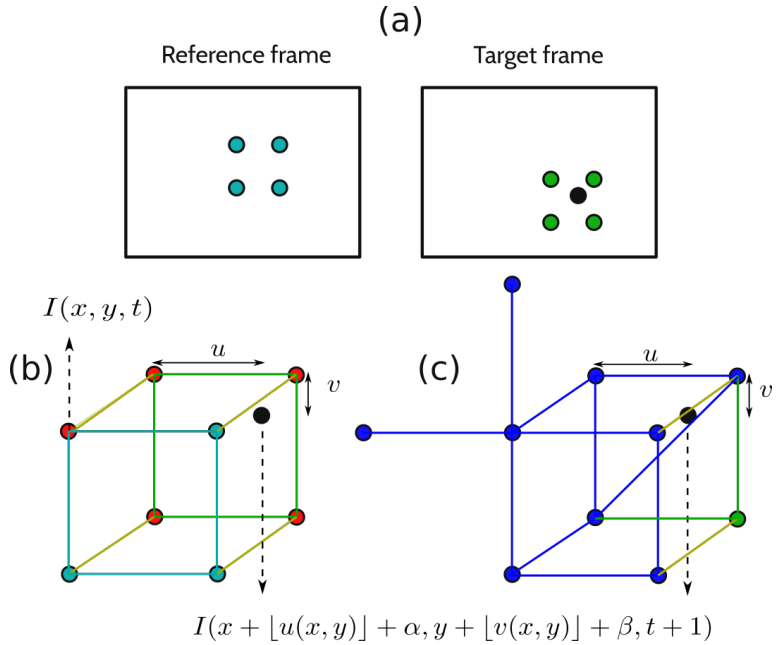

Figure 3.5: The neighboring pixels involved in fitting the polynomials. (b): Fitting a polynomial using the neighbors in red results in the bilinear interpolation proposed by Jaderberg et al. [42]. (c): Fitting a polynomial using the neighboring pixels in blue results in the second-order Taylor expansion.

frame. The estimation at each level updates the estimation from the previous level. The updated motion field is up-scaled and then used to warp the target frame towards the reference frame. The reference frame and the warped target frame are then given as input to GradNet to calculate another update on the motion field. Several scales are exploited in the multiscale scheme. After each update, and similar to other methods in the literature [80], the calculated motion field in each iteration is smoothened (by a Gaussian filter). The proposed algorithm at test time is summarized in Algorithm 1.

3.2 Architecture and Training

We propose a fully convolutional neural network with 8 convolutional layers. The activation function of all layers is ReLu except for the last layer's which is Linear. The full architecture is illustrated in figure 3.6. The architecture could be imagined as two parts. The CNN makes a compact representation of motion information in the first

Algorithm 1 The algorithm of our proposed framework during the test time.

procedure
  I_1,I_2: Two input frames
  F_{tot}: The desired motion field
  I_1^n,I_2^n: The downsampled versions of I_1 and I_2 by a factor of $n = 0.7^k$
  $F_{tot} = 0$
  $I_{2_w}^n \leftarrow I_2^n$
  while $n \geq 1$ **do**
    $\Delta F \leftarrow CNN(I_1^n, I_{2_w}^n)$: Calculate the update on the motion field
    $\Delta F \leftarrow GaussFilt(\Delta F)$: Gaussian filter the motion field
    $F_{tot} \leftarrow F_{tot} + \Delta F$: Update F_{tot} using the motion field
    Up-sample F_{tot} by a factor of $\frac{1}{0.7}$
    $n = \frac{n}{0.7}$
    $I_{2_w}^n \leftarrow warp(I_2^n, F_{tot})$: Warp I_2^n towards I_1^n using the motion field
  end while
  Return F_{tot}

part, encoder. This compact representation is then used to reconstruct the motion field in the second part, decoder. For allowing the decoder to have access to the lost information in the strides of the encoder, there are connections between the layers of the encoder and the corresponding layers in the decoder. Figure 3.6 illustrates these connections. To update the CNN weights during the training phase, we used ADAM [49] with a batch size of 8, random samples from random requences, and with an initial learning rate of 0.0001 and parameters $\beta 1 = 0.9$ and $\beta 2 = 0.999$. The samples in each batch are not from same sequence and are randomly selected from all the samples available in the dataset. The learning rate drops after $25k$ gradient steps by a factor of 2 every $5k$ iterations. GradNet was trained this way for $295k$ iterations.

3.3 Dataset

We train GradNet using a natural dataset, UCF101 [77], in an unsupervised way. The training set consists of $100k$ pairs of consecutive frames drawn randomly from about 1

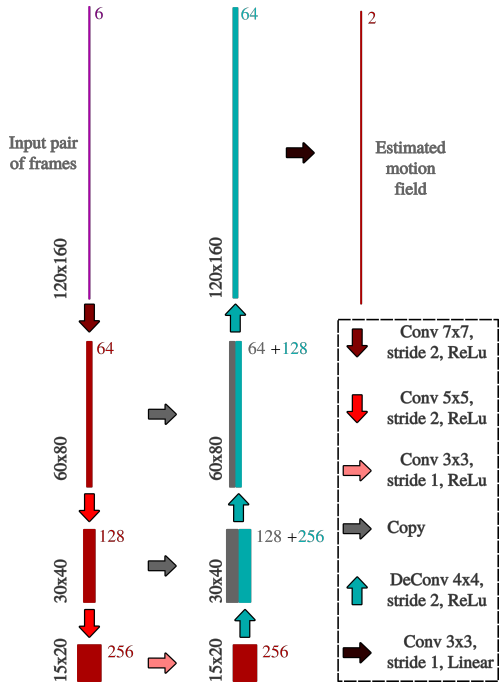

Figure 3.6: GradNet architecture, inspired by U-Net [72]. Each box corresponds to a multi-channel feature map. The number of channels is denoted at the top right corner of the box. The number denoted at the bottom left corner of the boxes is the height×width of the featuremap. Grey boxes represent copied feature maps. The purple box represents the input featuremap which consists of 6 RGB channels of the input pair of frames. The arrows denote the different operations.

million frames re-sized to half of the original width and height. Since no ground truth is available, no data scheduling can be assumed in the training phase. Though, a shade affect is added to the input as an augmentation. The spatial location of the shade is chosen randomly. Figure 3.7-a shows an example frame from UCF101 and Fig. 3.7-b shows the example with the augmentation. This kind of augmentation provides a level of robustness against intensity variations.

We evaluate GradNet on both the synthetic and real datasets. The MPI Sintel [20] dataset, training split, includes 1041 training samples, rendered artificial scenes for which the groundtruth is available. In MPI Sintel the attention has been to meet the properties of realistic images as much as possible. Two renders are available: the Final version contains motion blur and atmospheric effects, such as fog, while the Clean

Figure 3.7: (a) A sample drawn from UCF101 (b) The shade added as an augmentation.

version does not include realistic effects. The KITTI dataset [61] contains 200 training image pairs and includes displacements of large magnitude. KITTI contains only a specific motion type, as the dataset is acquired from a camera on a car. The captured frames are real world scenes and the ground truth is obtained from simultaneously recording the data from a 3D laser scanner. The scenes are rigid and the motion is from a moving observer. Furthermore, the motion ground truth is sparse as it was not possible to measure the motion of some objects, e.g. sky or distant objects. Middlebury is a low population, 8 samples, dataset commonly used for evaluation. In the making of Middlebury, a computer controlled system makes sure that the scenes are moved in small steps in a way that no scene point moves by more than 2 pixels. A fine spatter pattern of fluorescent paint is applied to all surfaces in the scene. High-resolution images are taken both under ambient lighting and UV lighting. After capturing the reference frame, the scene is moved slightly to capture the target frame. The ground-truth flow is computed by tracking the patterns that were applied by fluorescent paint and are highly visible under UV lighting. For further details, we would refer to [9]. Examples from the evaluation datasets are illustrated in Fig. 3.8.

Figure 3.8: Examples drawn from the evaluation dataset.

3.4 Experiments

In order to evaluate the performance of GradNet, we report its results on 4 most commonly used datasets, namely the real KITTI 2015 and the synthetically generated MPI-Sintel [20] clean and final, and the widely used synthetic Middlebury dataset and compare them with the results of other state-of-the-art methods. MPI-Sintel final

		Method	Fine tuned	Sintel clean		Sintel final		KITTI 2015	Middlebury
				train	test	train	test	train	train
Classic		DeepFlow	N	2.66	5.38	3.57	7.21	10.63	0.25
		LDOF (CPU) [17]	N	4.64	7.56	5.96	9.12	18.19	0.44
		LDOF (GPU) [83]	N	4.76	-	6.32	-	18.20	0.36
		EpicFlow [70]	N	2.27	4.12	3.56	6.29	9.27	0.31
		FlowFields [8]	N	1.86	3.75	3.06	5.81	8.33	0.27
		PCA-Layers [96]	N	3.22	5.73	4.52	7.89	12.74	0.66
		PCA-Flow [96]	N	4.04	6.83	5.18	8.65	14.01	.70
Deep neural network based	Sup	FlowNetS [25]	N	4.50	6.96	5.45	7.52	-	1.09
	Sup	FlowNetC [25]	N	4.31	6.85	5.87	8.51	-	1.15
	Sup	SpyNet [67]	N	4.12	6.69	5.57	8.43		0.33
	Sup	FlowNet2 [41]	N	2.02	3.96	3.14	6.02	10.06	0.35
	Unsup	DSTFlow [69]	**N**	6.93	10.40	7.82	11.11	24.30	-
	Unsup	DSTFlow(KITTI) [69]	Y	7.10	10.95	7.95	11.8	16.79	-
	Unsup	DSTFlow(Sintel) [69]	Y	6.16	10.41	7.38	11.28	23.69	-
	Unsup	DSTFlow(C+K) [69]	Y	7.51	-	8.29	-	22.93	-
	Unsup	DSTFlow(C+S) [69]	Y	6.47	10.84	6.81	11.27	25.98	-
	Unsup	**GradNet-FO (First-Order)**	**N**	8.94	12.4	10.60	13.78	14.94	2.79
	Unsup	**GradNet-SO (Second-Order)**	**N**	**6.65**	**10.30**	**7.81**	11.25	**16.28**	**1.14**

Table 3.1: Performance comparison. AEE stands for Average End-point Error (in pixels). Upper section reports the performance for classical methods while lower section reports the performance for DNN-based methods.

is a version of MPI-Sintel that includes more realistic effects and is one of the most realistic synthetic datasets for which ground truth is available. Worthy of mentioning that finetunning did not improve the results and in all the evaluations, GradNet-FO and GradNet-SO are not finetuned on any of the evaluation datasets. Experimental results show that although GradNet is not finetuned, it can generalize well to unknown datasets. Specially the performance on KITTI, which is a real dataset, shows that GradNet-SO, that is trained on a real dataset, performs much better than DSTFlow that is trained on a synthetic dataset.Yet, there is a gap between the performance of the classic methods and the DNN-based methods.

For further evaluation, the motion field estimated by different methods and the error map are visualized, respectively, in figures 3.9 and 3.10. The illustrated samples are chosen from MPISintel training split of the final pass. Figure 3.9 illustrates the visualization of the output of some of the classic methods including the Horn and Schunk method [38] embedded in a multiscale scheme, denoted by HS. At first glance, it can be seen that HS performs considerably worse than GradNet-FO, although their loss

	Method	d < 10	10<d<40	d>40
Classic	EpicFlow	2.71	6.98	36.76
	DeepFlow	2.19	6.16	40.51
	HAOF	2.69	8.14	50.29
	LDOF	2.19	6.46	41.35
	HS	7.07	16.46	62.97
DNN-based	GradNet-FO	4.68	11.42	52.02
	GradNet-SO	3.89	9.15	43.74

Table 3.2: AEE for different ranges of d, $d = \sqrt{u_{gt}^2 + v_{gt}^2}$.

function is similar. This shows that although using similar loss function, the internal regularization effect of deep neural networks helps to get better performance. Still, GradNet-SO performs better than GradNet-FO as it makes a more accurate approximation of the warped frame in the motion compensated intensity error whilst training. The quantitative results in table 3.2 confirm the conclusion from the visualized results. In table 3.2, d is the magnitude of groundtruth motion, $d = \sqrt{u_{gt}^2 + v_{gt}^2}$. Also, the numbers are in terms of AEE and show how different methods perform in the estimation of small to large motions. Figure 3.9 also shows that LDOF [17] respects the edges much better than HAOF [16] due to the extra edge respecting term in the loss function. Although EpicFlow performs the best, it is prone to high error when estimating motion field for a large low-texture area, sample in 5^{th} row in figure 3.9. The reason is that EpicFlow interpolates between the corresponding matched features and if the features are matched wrong, the interpolation calculates wrong motion field for the whole area.

Figure 3.10 illustrates the Endpoint-Error map (EE-map) calculated for several methods on the MPISintel training split of the final pass. This figure shows that all methods have difficulty in estimation at motion discontinuities, occluded area, low-texture area, and where the intensity variation is significant. These shortcomings are addressed to some extent by supervised methods which will be illustrated and discussed in chapter 6.

Figure 3.9: The visualized motion fields calculated by Deepflow [93], EpicFlow [70], High Accuracy Optical Flow (HAOF) [16], Large Displacement Optical Flow [17], Horn and Schunk method [38], GradNet First Order (GradNet-FO), and GradNet Second Order (GradNet-SO) on MPISintel dataset [20].

Figure 3.10: The AEE maps calclualted for several samples from several methods on MPISintel the final-training split. Each sample is normalized by a factor of $\frac{255}{max(d)}$, where $d = \sqrt{u_{gt}^2 + v_{gt}^2}$. Values more than 255 are rounded to 255.

3.5 Computational Complexity

The runtime of GradNet is $156ms$ in comparison with the runtime of DSTFlow [69] which is $30ms$. DSTFlow has the same architecture as FloNetS [25]. To measure the runtime, both algorithms are implemented using Lasagne in Theano and the exploited GPU is GeForce GTX 1080. Table 3.3 contains the runtime breakdown of GradNet.

In terms of learned parameters, GradNet has the lowest number of learned parameters in comparison with other unsupervised motion estimation methods. Taking the supervised methods into the consideration, GradNet is still one of the smallest DNN-based motion estimators in term of learned parameters. Table 4.1 contains a comparison with other CNN architectures trained for motion estimation.

3.6 Evaluation on MPI Sintel, Test Split

As shown in Table 3.1, compared to the training split, the accuracy is lower on the test split. The groundtruth motion for the test split of MPI Sintel dataset is not publicly available and the evaluation is done automatically by the Sintel official server. The right column, middle column, and the left column fo Figure. 3.11, respectively, depict the reference frames, motion fields calculated by GradNet and error fields of some of

	GradNet (CNN)	Smoothening	Warping	**Sum**
scale 1	0.0012	0.0001	0.0008	0.0022
scale 2	0.0012	0.0001	0.0009	0.0023
scale 3	0.0015	0.0001	0.0012	0.0029
scale 4	0.0023	0.0002	0.0016	0.0042
scale 5	0.0036	0.0005	0.0026	0.0068
scale 6	0.0066	0.0010	0.0046	0.0123
scale 7	0.0124	0.0021	0.0087	0.0233
scale 8	0.0247	0.0041	0.0166	0.0454
scale 9	0.0479	0.0088	-	0.0568
Sum	0.1019	0.0173	0.0372	**0.1560**

Table 3.3: The runtime breakdown of GradNet in multiscale scheme in second. Scale 1 refers to the lowest resolution, scale 9 refers to the main resolution.

Method	Number of learned parameters
FlowNetS	32,070,472
FlowNetC	32,561,032
SpyNet	1,200,250
GradNet	**3,666,562**

Table 3.4: Compared to other DNN-based methods, GradNet is the smallest among unsupervised methods and in general one of the smallest in terms of learned parameters. DSTFlow [69] follows the FlowNetS architecture.

the samples provided by the MPISintel server. Investigating these visualizations give an idea of how the major errors happen. We point out major error regions using a color-coded guide at the bottom of Fig. 3.11. The error field in Fig. 3.11 is obtained by normalizing the AEE to $[0, 255]$. The AEE reported under each error field gives an idea of how large the error is in bright areas of the error field. Figure 3.11 depicts how the estimated motion field looks like in low and high error regions.

Visualization of the Learned Filters. Figure 3.12 illustrates the visualization of one of the filters learned by the first layer of GradNet. From top to bottom, each row shows the filter channel that operates on each of the R, G and B channels. Left column operates on the first frame and the right one operates on the second frame. Note that the actual filters are 7×7 pixels and here are upsampled using bilinear interpolation for visualization. Unlike the visualization of filters in [67], our unsupervised trained filter shows complex derivations filters that are independent for each of RGB channels.

3.7 Conclusions

In this work, we propose estimating dense motion fields with CNNs. We show that surprisingly perhaps, a simple cost function that relies on the optical flow equation can be used successfully for training a deep convolutional network in a completely unsupervised manner and without the need of any regularization or other constraints performing better than classic counterpart, Horn and Schunk method. The cost function is based on first-order Taylor expansion. We also show that training the CNN

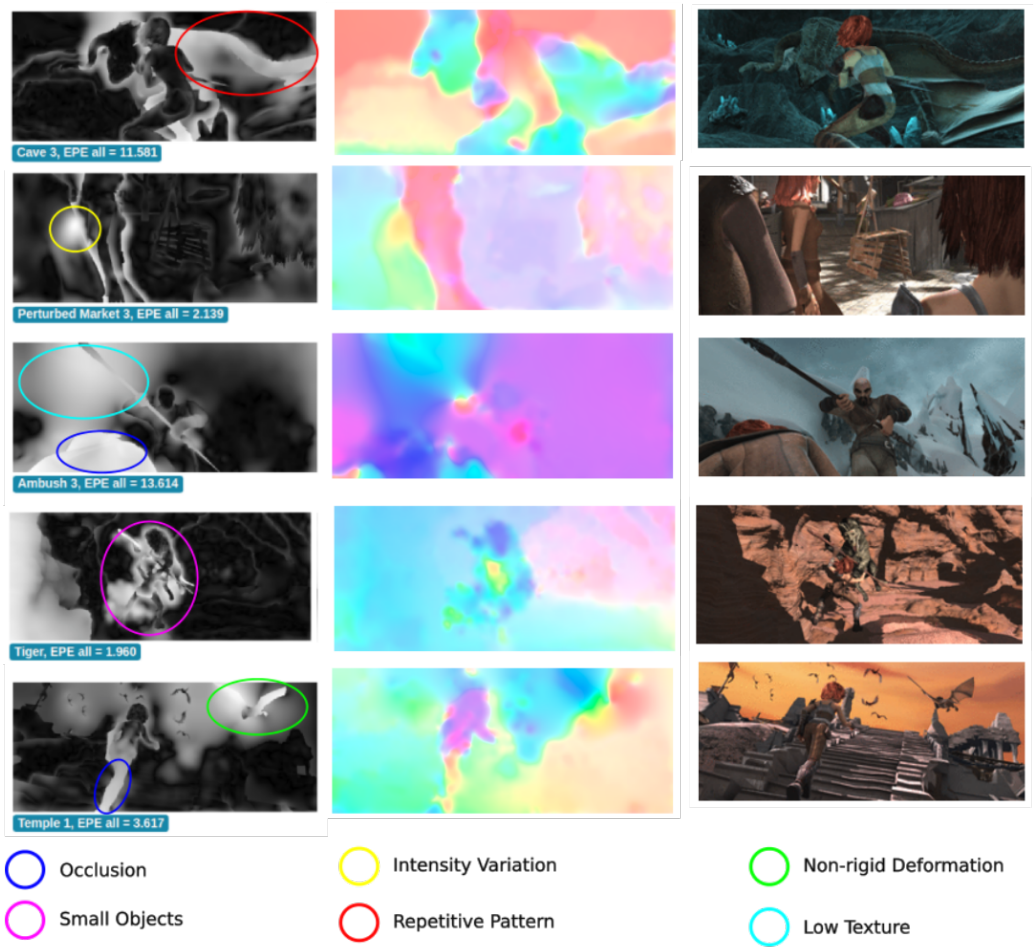

Figure 3.11: The reference frames of some of the samples from test split of MPISintel, the motion fields estimated by GradNet for each of the samples, and the error fields calculated for the estimated motion fields.

using a cost function that is based on the second-order Taylor expansion significantly improves the performance. Furthermore, we show the connection between our proposed loss function and the interpolation-based loss function used by other unsupervised methods. Similar to classic methods which estimate motion in a multiscale scheme to improve the estimation of large motions with large magnitude, GradNet is also embedded in a multiscale scheme. We observed that finetuning on the target datasets does not improve the performance. We show that our CNN has a performance that is comparable to other state-of-the-art methods and that it can generalize very well to an unknown dataset, MPI-Sintel, without the need for refinement. GradNet has

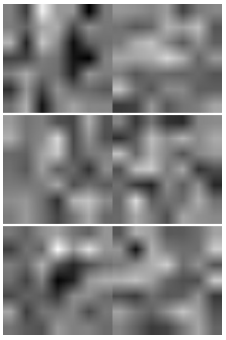

Figure 3.12: Visualization of one of the filters in the first layer of GradNet

the least number of learned parameters among other unsupervised methods and one of the least among supervised methods. The proposed method in this chapter is among the very few studies conducted on the application of DNNs for motion estimation.

Chapter 4

LikeNet: A Siamese Motion Estimation Network Trained in an Unsupervised Way

To the best of our knowledge, all the DNN-based methods solve motion estimation as a regression problem, $F \subset \mathbb{R}^2$. In this section, we present in detail our approach that addresses motion estimation as a classification problem,$F \in \mathbb{Z}^2$. We propose a CNN for pixel-level motion class prediction, LikeNet. LikeNet treats motion estimation as a dense labeling problem. We propose an unsupervised trained Deep Network by adopting a Siamese architecture, with as many branches as motion labels. Each branch of the architecture, receives as input the reference frame and the target frame translated by the motion label in question, and produces the (not normalised) probability map for the motion label in question - that is the (unnormalized) probability that a pixel is translated by the motion vector corresponding to the motion label in question. More general, at test time, LikeNet receives as input a pair of consecutive frames, I, and outputs a pixel-level distribution over K motion labels/classes, P. Figure 4.1 illustrates the overview of LikeNet. To the best of our knowledge, this is the first time the problem of motion estimation is treated as a classification problem in a DNN-based

framework.

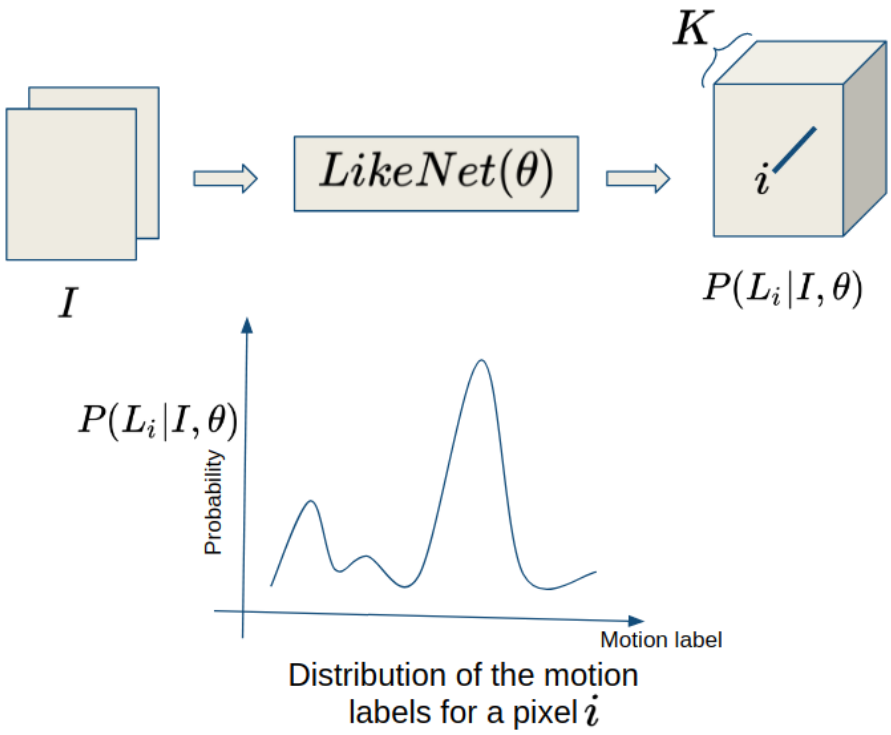

Figure 4.1: The overview of LikeNet.

In order to deal with motions with large magnitude, our network is embedded in a classical multiscale scheme. A major issue in multiscale methods is that errors at lower resolutions are propagated to higher ones - for this reason, significant gains can be made by improving the quality of the estimation at the lower levels. To this end, we use Conditional Random Fields (CRFs) at the lowest resolution, implemented as an RNN similar to in [100], so that one can form an end-to-end trainable framework for motion estimation which combines the strengths of deep learning and graphical modelling. More specifically, we employ the CRF to improve the estimated motion field at the lowest resolution of our multi-scale scheme. To the best of our knowledge, this is the first time that CRFs are integrated into a DNN-based framework for motion estimation. Figure 4.2 describes the training order of LikeNet and the CRF.

Our network is trained on a simple cost function, without explicit smoothness or

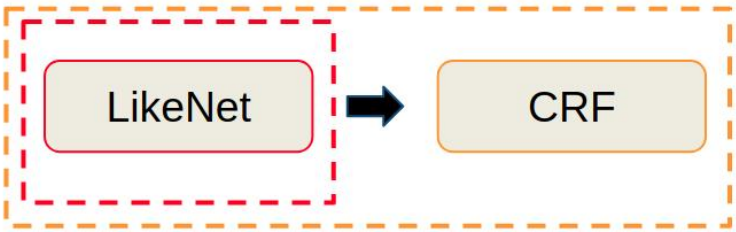

Figure 4.2: The schematic describing the training order of LikeNet and the CRF

other constraints - those are implicitly modelled in the filters of the CNN and are learned from training data. Random, consecutive pairs of frames drawn from videos of the UCF-101 dataset [77] were used for training, while for evaluation both synthetic and real datasets were used. Although finetunning LikeNet for a target dataset, does not improve the results for that dataset, we show that the unsupervised trained LikeNet performs better or in par with other unsupervised trained DNN-based methods on both synthetic and real data, even in the case that the other methods are finetuned (in an unsupervised manner) on the target dataset. In addition, when compared to other DNN-based methods, LikeNet model is the smallest in terms of learned parameters - respectively 98% and 42% smaller than FlowNet [28, 69] and SpyNet [67] architectures.

In this chapter, we first give a brief review of the application of the CRF implemented as RNN for pixel level prediction, then we explain the methodology. We present the conclusion after the experimental results are presented.

4.1 LikeNet: a CNN for Motion Estimation

In this section, we present the details of LikeNet. As similar to in classification methods [100], let L be a Random Field defined over a set of N discrete variables $\{L_1, \cdots, L_N\}$, where N is the number of image pixels. The domain of each variable L_i is a set of motion labels $M = \{\mathbf{m}_1, \cdots, \mathbf{m}_K\}$, each label corresponding to a motion vector. Clearly, an instantiation of the label field L corresponds to a dense motion field and in

this chapter, the term m_k will be used to denote both the k^{th} label and the k^{th} motion vector - the interpretation should be clear from the context. Also, let $I \in R^{2\times N}$ denote an input pair of consecutive frames, each of size N. LikeNet receives I as input and outputs a pixel-level distribution over the motion classes, $P(L|I;\theta)$, where θ denotes the model parameters, that is the parameters of the network. The mode of $P(L|I;\theta)$ could be then used as a point estimate of the motion/label field. That is, $L^* = \underset{L}{\operatorname{argmax}}\, P(L|I;\theta)$.

4.1.1 Architecture

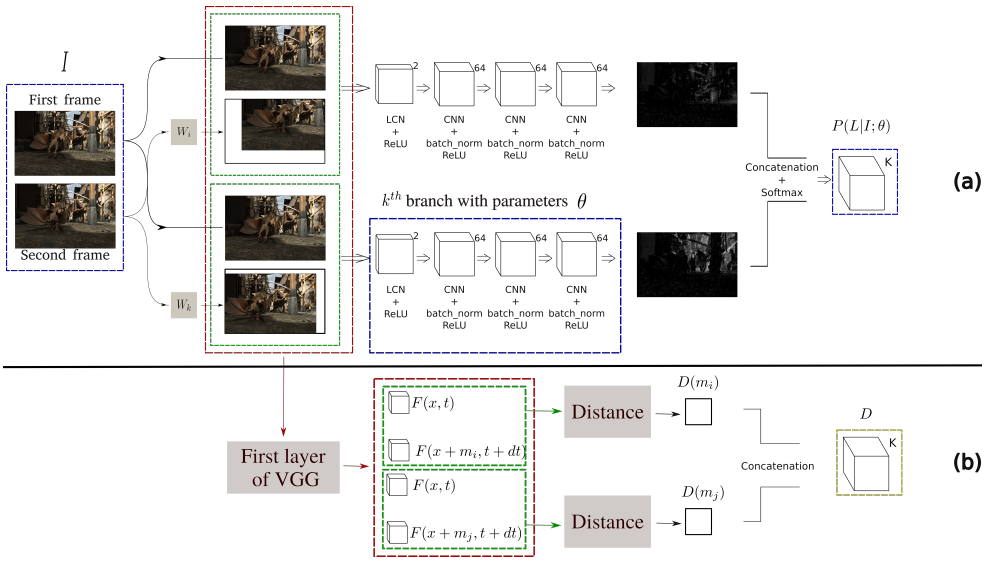

Figure 4.3: The proposed architecture of LikeNet. For simplicity, only two branches out of K branches (motion classes) are illustrated. The input to the k^{th} branch is the concatenation of the first frame and the second frame shifted with the corresponding motion vector $\mathbf{m}_k$. Block W_k warps its input along with motion vector $\mathbf{m}_k$. LikeNet outputs a pixel-level distribution over the motion classes, $P(L|I;\theta)$.

An overview of the proposed architecture of LikeNet at test time is given in Fig. 4.3(a). More specifically, to calculate $P(L|I;\theta)$, we propose a Siamese CNN with number of branches equal to K. The input to the k^{th} branch is the concatenation of the first frame and the second frame shifted by the corresponding motion vector $\mathbf{m}_k$. Each branch consists of a Local Contrast Normalization (LCN) layer [44] and three convolutional layers, and calculates a single channel, pixel-level heat-map, which

indicates whether the pixels at the same location in the reference and the shifted target frames, match. Since the target frame has been shifted by the motion vector $\mathbf{m}_k$ this heat-map can be interpreted as the probability (not normalized) that a pixel has moved by $\mathbf{m}_k$. In order to obtain a normalized distribution over all motion classes/labels, the concatenation of the heat-maps, calculated by all branches, are passed through a Softmax layer. In other words, the k^{th} branch is responsible for calculating pixel-level probability map $P(L = \mathbf{m}_k|I;\theta) \in [0,1]^N$ that expresses the probability that each pixel in I is displaced by $\mathbf{m}_k$. An arbitrary k^{th} branch and $P(L|I;\theta)$ are highlighted in blue dotted squares in Fig. 4.3(a).

During training we would like to learn parameters such that at pixel i the probability $P(L_i = \mathbf{m}_k|I;\theta)$ produced by LikeNet is high for the true motion label $\mathbf{m}_k$ and low for the other labels. Our assumption is that, under an appropriate distance measure, the feature differences/distances under the correct motion vector will be lower than the feature distance under an arbitrary motion vector. That is, the features $\mathbf{F}(x_i,t)$ extracted at location x_i (pixel i) in the reference frames, will be more similar to their correspondences $\mathbf{F}(x_i+\mathbf{m}_k,t+dt)$ in the target frame shifted by the correct $\mathbf{m}_k$ (Fig. 4.3). Formally, we train the network so as to minimize the following cost function:

$$C(I;\theta) = \sum_i \sum_{\mathbf{m}_k \in M} P(L_i = \mathbf{m}_k|I,\theta) D(i,\mathbf{m}_k), \tag{4.1}$$

where $D(i,\mathbf{m}_k)$ is the distance between $\mathbf{F}(x_i,t+dt)$ and its corresponding aligned pixel in the shifted target frame $\mathbf{F}(x_i+\mathbf{m}_k,t+dt)$. The distance that we use in this chapter is:

$$D(i,\mathbf{m}_k) = JSD(\mathbf{F}(x_i+\mathbf{m}_k,t+dt)\|\mathbf{F}(x_i,t)) \tag{4.2}$$

where JSD denotes the Jensen-Shannon divergence. In our experiments Jensen-Shannon divergence showed to be, slightly, a better measure than Euclidean distance. Also, we chose $\mathbf{F}$ in Eq. (4.2) to be the features calculated by the first convolutional layer of VGG-16 [76]. While raw intensities/colour could also be used, we have found

that those features perform better. These distances are calculated by the branch of the network depicted in Fig. 4.3(b), which clearly, is used only during training.

Intuitively, we would like that the network outputs higher probability $P(L = \mathbf{m}_k|I;\theta)$ at pixels where, under a shift by $\mathbf{m}_k$, the distance between the corresponding features is smaller. Given that the probability of the motions for each pixel sums to one, minimizing Eq. (4.1) forces the network to increase the probability of the motion classes for which their corresponding values in D are small. Deeper architectures for LikeNet did not provide with better results.

The proposed formulation relies on a quantisation of the motion vectors to motion labels. With a quantization of each of the horizontal/vertical components to integers, in order to be able to estimate motions of magnitude V, $K = V^2$ branches are needed at test time. For large V this is not practical. For this reason, at test time we embed the trained network in a multi-scale scheme as described in Algorithm 2. In our experiments we use 5 scales with 121, 169, 49, 9, 9, branches from the lowest resolution to the highest resolution, respectively.

Algorithm 2 The algorithm of our proposed framework during the test time.

procedure
  I_1,I_2: The two input frames
  Υ: The estimated motion field
  β: The number of image pyramid levels
  I_1^n,I_2^n: The downsampled versions of I_1 and I_2 by a factor of 2^n
  $I_{2_w}^n$: The warped version of the second frame
  $\Upsilon \leftarrow 0$, $I_{2_w}^n \leftarrow I_2^n$, $n \leftarrow \beta$
  while $n > 0$ **do**
    $\Delta\Upsilon \leftarrow CNN(I_1^n, I_{2_w}^n)$: Calculate the update on the motion field
    if $n = \beta$ **then**
      $\Delta\Upsilon \leftarrow CRF(I_1^n, \Delta\Upsilon)$: Correcting the motion field in the lowest resolution
    $\Delta\Upsilon \leftarrow GaussFilt(\Delta\Upsilon)$: Gaussian filtering of the motion field
    $\Upsilon \leftarrow \Upsilon + \Delta\Upsilon$: Update Υ using the motion field
    Up-sample Υ by a factor of $\frac{1}{0.5}$
    $n \leftarrow n - 1$
    $I_{2_w}^n \leftarrow warp(I_2^n, \Upsilon)$: Warp I_2^n towards I_1^n using the motion field
  end while
  Return Υ

4.2 CRF for Motion Estimation

To avoid the propagation of errors across scales, we improve the quality of the estimated motion at the lowest resolution of the multi-scale scheme by using a graphical model, CRF (Fig. 4.4). In this Section, we provide a brief overview of the CRFs, how we learn their parameters and how we use them at inference for pixel-wise labeling. The Random Variable L that models pixel motion labels form a Markov Random Field when conditioned upon the observation I. Given a graph $G = (V, E)$, where $V = \{L_1, L_2, ..., L_N\}$ and the observation I, the pair (I, L) can be modeled as a CRF characterized by a Gibbs distribution $P(L|I) = \frac{1}{Z(I)} exp(-E(L|I))$. Here $E(L)$ and $Z(I)$ are the energy and partition function, respectively. For convenience, we will drop the conditioning on I. In our CRF model, similar to [100], the energy of the label assignment L is given by:

$$E(L) = \sum_{i=1}^{N} \psi_u(L_i) + \sum_{j \in n(i) \setminus i} \psi_p(L_i, L_j) \tag{4.3}$$

where $\psi_u(L_i)$ is the unary energy of the pixel i taking the label L_i, and the pairwise energy component $\psi_p(L_i, L_j)$ measures the cost of assigning labels L_i, L_j to pixels i,j simultaneously. Also, $n(i)$ represents the neighborhood of pixel i. The unary is obtained from LikeNet which predicts the labels without any smoothness or consistency assumption. The pairwise energies provide an image data-dependent smoothing term that encourages assigning similar motion labels to pixels with similar properties. As in [51], we model pairwise potentials as weighted Gaussians:

$$\psi_p(L_i, L_j) = \mu(L_i, L_j) \sum_{m=1}^{M} \mathbf{w}^{(m)} \gamma_G^{(m)}(f_i, f_j), \tag{4.4}$$

where each $\gamma_G^{(m)}, m = 1, ..., K$, is a Gaussian kernel applied on feature vectors. The feature vector at pixel i, denoted by f_i, are spatial location and intensity values. The function $\mu(.,.)$, called the label compatibility function, captures the compatibility between different pairs of labels. Minimizing the above CRF energy $E(L)$ yields the

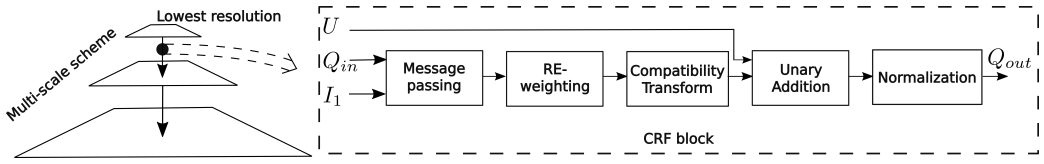

Figure 4.4: A CRF block at the lowest resolution during test time.

most probable motion label assignment L. To obtain the parameters, we follow the RNN-like training scheme proposed in [100]. We set $w^{(m)}$ to 1 and use a compatibility matrix $\mu(L_i, L_j) = \frac{1}{2}e^{-\frac{(L_i - L_j)^2}{2\sigma_c^2}}$ that is parametrized by a single parameter σ_c. This is different to [100] that addresses a labeling problem where there is no natural order/structure in the labels and learns a pairwise compatibility matrix $\mu(.,.) \in \mathbb{R}^{K \times K}$ (where K is the number of labels) – in our case, the labels are structured. An overview of the CRF block, drawn from [100], is illustrated in Fig. 4.4.

Table 4.2 provides a comparison between the performance of LikeNet when the CRF is used or not used with other classic and DNN-based methods. Based on the results, using a CRF improves the accuracy slightly, although the computational complexity increases significantly. In comparison with DTSFlow [69] and GradNet which minimize the motion compensated intensity error during the training, LikeNet has a considerably lower AEE. Based on table 4.2, LikNet performs the best among unidirectional unsupervised methods. Although, the supervised and classic methods still perform better. Investigating the output of LikeNet with and without the CRF shows that the CRF contribution is mainly to remove the erroneous estimations at the lowest scale and preventing them from propagating across scales. Figure 4.5 shows how the CRF visually affects the estimated motion field for some samples drawn from the MPI Sintel final.

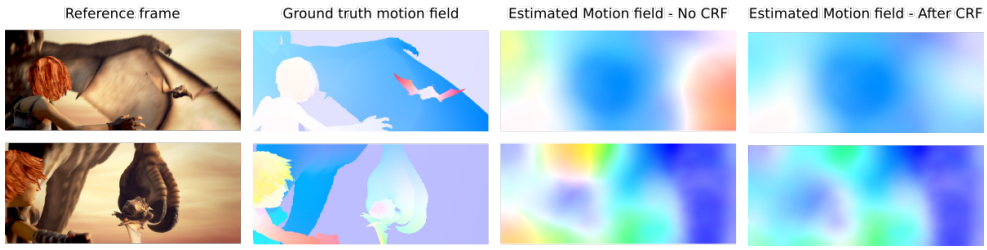

Figure 4.5: Visualization of how the application of CRF affects the output of LikeNet.

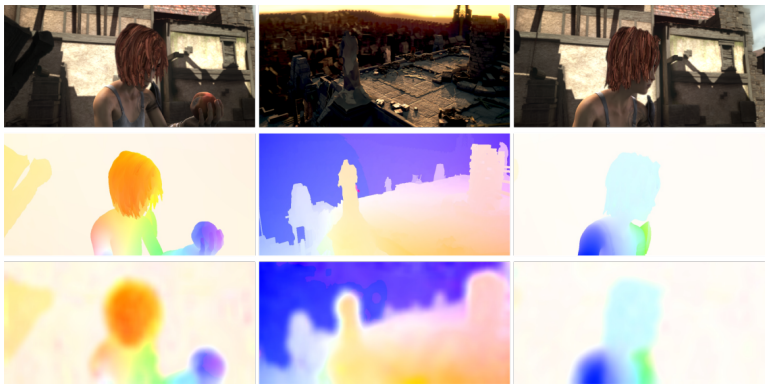

Figure 4.6: MPI-Sintel examples. Top-to-bottom, input reference frames, groundtruth flows, and predicted flows from LikeNet.

4.3 Experiments

In order to evaluate our work, we compare LikeNet with a number of state-of-the-art classical and DNN-based methods, supervised and unsupervised, on a number of benchmarks. Likenet is trained for 7k iterations on samples drawn from the real action recognition dataset UCF101 and is not fine-tuned on any synthetic dataset. We trained for 7k iterations on samples drawn from the action recognition dataset named UCF101, and optimized its parameters by adopting Nesterov momentum method with momentum 0.9 [84]. The learning rate starts from 0.1 and drops by a factor of 2 every 1000 iterations. In our experiments, during training, K is set to 121 to deal with horizontal and vertical motions of maximum magnitude 5 pixels. The motion distribution in the training data can be controlled by resizing the drawn frames, we resized the frames to 96×128 for training. The architecture of LikeNet is the smallest

Method	Number of learned parameters
FlowNetS	32,070,472
FlowNetC	32,561,032
SpyNet	1,200,250
LikeNet	**697,028**

Table 4.1: Compared to other DNN-based methods, LikeNet is the smallest in terms of learned parameters. DSTFlow [69] follows the FlowNetS architecture. UnFlow-C [59] follows the FlowNetC architecture and UnFlow-CS is a FlowNetS architecture stacked on top of a FlowNetC. UnFlow-CSS architecture is composed of a FlowNetS stacked on top of the UnFlow-CS.

possible that would end in an acceptable performance. Due to the Siamese nature of the architecture, a deeper network would considerably add to the computational complexity whilst we observed that a deeper architecture does not improve the final performance much.

Visualization of the motion field of some examples from MPI-sintel, estimated by LikeNet, are illustrated in Fig. 4.6. To quantitatively evaluate our method we report the Average End-point Error (AEE) on both synthetic and real datasets in Table 4.2. All measures include the occluded areas. We find that LikeNet is performing better than other unsupervised methods that do not use bidirectional schemes even without being fine-tuned on any other synthetic or real dataset, and close to UnFlow, that adopts a bidirectional scheme that helps significantly with occlusions.

4.4 Flexible Architecture; Memory-Speed Trade-offs

To best of our knowledge, the architectures of all other DNNs trained for motion estimation, both supervised and unsupervised, enforce a fixed computational load and memory requirement. A fixed architecture might not be appropriate when the computation power and/or memory is limited. The Siamese architecture of LikeNet allows for a trade-off between the required memory and the computational load. The architecture of LikeNet can be configured to run perfectly in parallel (P-configuration

	Method	Fine tuned	Sintel clean		Sintel final		KITTI 2015	Middlebury
			train	test	train	test	train	train
Classic	DeepFlow	N	2.66	5.38	3.57	7.21	10.63	0.25
	LDOF (CPU) [17]	N	4.64	7.56	5.96	9.12	18.19	0.44
	LDOF (GPU) [83]	N	4.76	-	6.32	-	18.20	0.36
	EpicFlow [70]	N	2.27	4.12	3.56	6.29	9.27	0.31
	FlowFields [8]	N	1.86	3.75	3.06	5.81	8.33	0.27
	PCA-Layers [96]	N	3.22	5.73	4.52	7.89	12.74	0.66
	PCA-Flow [96]	N	4.04	6.83	5.18	8.65	14.01	.70
Deep neural network based – Sup	FlowNetS [25]	N	4.50	6.96	5.45	7.52	-	1.09
	FlowNetC [25]	N	4.31	6.85	5.87	8.51	-	1.15
	SpyNet [67]	N	4.12	6.69	5.57	8.43		0.33
	FlowNet2 [41]	N	2.02	3.96	3.14	6.02	10.06	0.35
	UnFlow-CS-ft-(KITTI supervised) [59]	Y	-	-	11.99	-	(2.25)	0.64
	UnFlow-CSS-ft(KITTI supervised) [59]	Y	-	-	13.65	-	(1.86)	0.64
Deep neural network based – Unsup	UCNNME [2]	**N**	8.94	12.4	10.60	13.78	14.94	2.79
	DSTFlow [69]	**N**	6.93	10.40	7.82	11.11	24.30	-
	DSTFlow(KITTI) [69]	Y	7.10	10.95	7.95	11.8	16.79	-
	DSTFlow(Sintel) [69]	Y	6.16	10.41	7.38	11.28	23.69	-
	DSTFlow(C+K) [69]	Y	7.51	-	8.29	-	22.93	-
	DSTFlow(C+S) [69]	Y	6.47	10.84	6.81	11.27	25.98	-
	UnFlow-C-Cityscapes [59]	N	-	-	8.23	-	10.78	0.85
	UnFlow-C [59]	N	-	-	8.64	-	8.80	0.88
	UnFlow-CS [59]	N	-	-	7.92	-	8.14	0.65
	UnFlow-CSS [59]	N	-	-	7.91	10.22	8.10	0.65
	GradNet (Second-Order)	**N**	6.65	10.30	7.81	11.25	16.28	1.14
	LikeNet (No CRF)	**N**	**6.03**	-	**6.78**	-	**14.76**	0.75
	LikeNet	**N**	**5.7**	**10.02**	**6.49**	**10.69**	**14.66**	0.78

Table 4.2: Average End-point Error (in pixels) of classic and DNN-based methods. LikeNet performs better or in par with other unsupervised methods although it is not finetuned on any of the evaluation datasets and its capacity is considerably smaller that all other DNN-based methods.

which enforces the lowest computational load), partially parallel (PS-configuration) or fully in serial (S-configuration which enforces the lowest required memory).

In this section, the computational load and the required memory by each branch of LikeNet are respectively denoted by C and M and the LikeNet is supposed to predict for K motion labels. In P-configuration, the computational load would be C and the total required memory would be $K \times M$. In S-configuration, the computational load would be $K \times C$ and the total required memory would be M.

In PS-configuration, the branches are divided to b blocks of branches, where blocks are in S-configuration and branches in each block are in P-configuration. Ideally, the

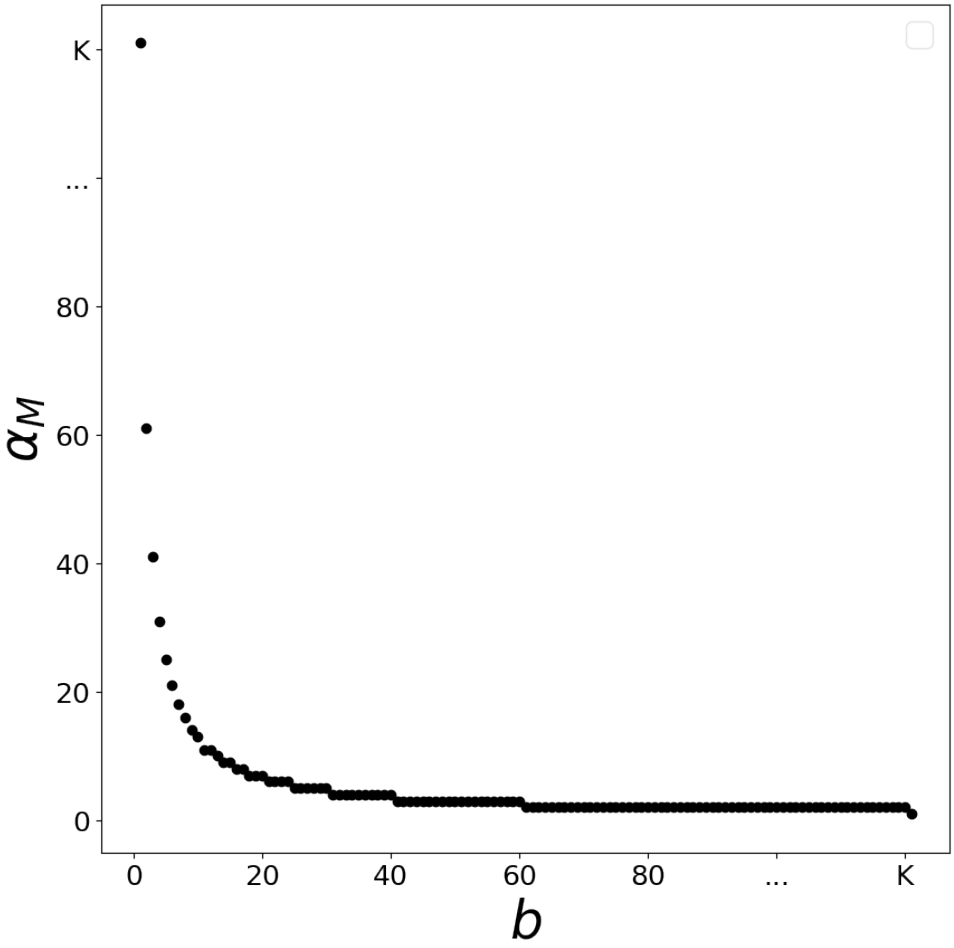

Figure 4.7: The plot illustrates how (b, α_M) vary in different configurations as b varies from 1 to K.

branches are distributed equally between the blocks to minimize the required memory. Figure 4.7 depicts how the computation/memory trade-off looks like. We denote the computation-memory pair as $(b \times C - \alpha_M \times M)$, where $\alpha_M = \lceil \frac{K}{b} \rceil$. In Fig. 4.7, b is changed from 1 to K and the plots depicts how (b, α_M) vary in different configurations.

As can be seen in Fig. 4.7, α_M can be reduced significantly by a slight increase in b. The point $b = \sqrt{K}$ is where there is a balance between the computational load and the required memory. Given that each branch of LikeNet is composed of only 4 convolutional layers, M and C are small for each branch, LikeNet is highly flexible to

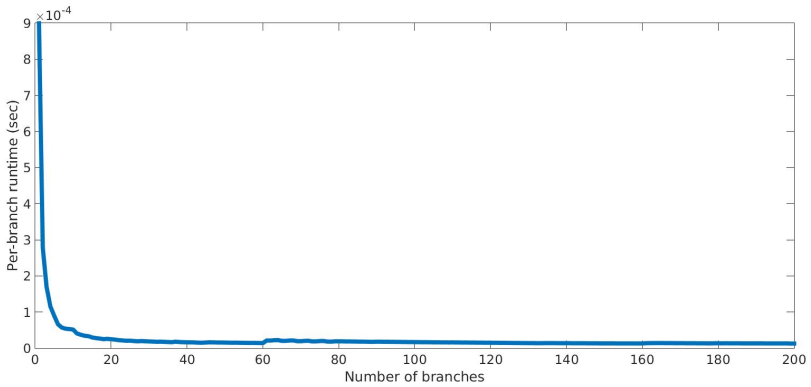

Figure 4.8: The plot of the number of branches against the per-branch runtime. This plot shows how far a GPU, in our case GEFORCE GTX 1080, can parallelize our architecture.

meet many available memory/computational requirements.

There is one problem and that is that the parallelization capacities of GPUs are limited by their number of processors that can run in parallel. We designed an experiment to show how limited the parallelization capacity of a GPU is. We feed LikeNet with a very small, 5×5, input pair of frames and increase the number of branches from 1 to $K = 121$ and inspect the processing time of each branch, Figure 4.8. By increasing the number of branches, if the process is parallelized in the GPU, the per-branch runtime has to reduce. Figure 4.8 shows that per-branch runtime reduces up to some extend, however it stays fixed at some point onward. It shows that that some point is where the GPU is serializing any additional computations. Because of the limited parallelization capacity of the available GPUs, we do not expect LikeNet to be very fast in practice.

4.4.1 Computational Complexity

The runtime of LikeNet is $19.208s$ in comparison with the runtime of GradNet which is $0.156s$ and DSTFlow [69] which is $0.030s$. DSTFlow has the same architecture as FloNetS [25]. To measure the runtime, all algorithms are implemented using Lasagne in Theano and the exploited GPU is GeForce GTX 1080. Table 4.3 contains the runtime breakdown of LikeNet.

	Data preparation	LikeNet (CNN)	CRF	Inference	Smoothening	Warping	**Sum**
scale 1	0.0514	0.1069	9.5840	0.0011	0.0002	0.0011	9.7449
scale 2	0.0520	0.5787	0.0000	0.0062	0.0006	0.0021	0.6396
scale 3	0.0934	1.0493	0.0000	0.0099	0.0022	0.0057	1.1604
scale 4	0.2153	2.3599	0.0000	0.0279	0.0076	0.0181	2.6287
scale 5	0.5009	4.4288	0.0000	0.0647	0.0401	0.0000	5.0344
Sum	0.9129	8.5236	9.5840	0.1098	0.0507	0.0270	**19.2082**

Table 4.3: The runtime breakdown of LikeNet in multiscale scheme in second. Scale 1 refers to the lowest resolution, scale 5 refers to the main resolution.

4.4.2 Number of Parameters

Let us note that in our Siamese architecture, each branch does the relatively easy task of computing the similarity between corresponding pixels in two frames. This considerably reduces the model complexity of LikeNet, each branch of which consists of only 4 layers. The number of parameters that are learned is $697,028$ compared to $1,200,250$, and $32,070,472$ and $32,561,032$ parameters SpyNet [67], FlowNetS and FlowNetC respectively learn, Table 4.1. LikeNet is about 42% smaller than SpyNet and 98% smaller than FlowNet. While the number of the parameters are not explicitly reported, the DSTFlow method proposed in [69] uses the FlowNetS architecture, and the UnFlow methods build on the FlowNet architectures, in some cases of considerable complexity (e.g., UnFlow-CSS architecture is a FlowNetC followed by a two FlowNetS).

The visualization of the filter weights in the first layer of LikeNet is given in Fig. 4.9. The visualization shows that most of the spatio-temporal filters are not equally sensitive to all color channels. This is different to what was shown in case of a supervised trained network [67]. A guess can be that objects in the natural scenes reflect mainly the sunlight and different colors have different distributions in the sunlight [10]. This might indicate that because the supervision of LikeNet is from natural images, the filters respond differently to different colors. In Fig. 4.9, most of the filters respond more to respectively grey, blue, green, yellow and then red. Confirmation requires further researches and experiments which is out of the scope of this thesis.

Figure 4.9: Visualization of the first layer filters of LikeNet.

4.5 Summary

In this chapter, we proposed a new Siamese CNN which solves the motion estimation as a classification problem, named LikeNet. We showed that a feature constancy constraint can be used for successfully training LikeNet in an unsupervised manner and without the need for any handcrafted regularization or other constraints. Our CNN is trained on the real UCF101 dataset. We show that it performs better than the other state-of-the-art unsupervised methods that do not use bi-directional constraints, and that it can generalize very well to unknown datasets without the need for finetunning. The architecture of the network allows for computational flexibility and prediction of as many motion classes as required. For future work, we intend to incorporate bi-directional constraints (i.e., perform motion estimation based on three frames) and investigate on computationally efficient schemes.

Chapter 5

Quick LikeNet (QLikeNet) : Distilling LikeNet in a Fast Regression CNN

Experiments in the previous section showed that using the higher level features for training a Siamese architecture with a little number of learned parameters, leads into impressive accuracy when generalizing to unknown datasets. Despite the high accuracy, LikeNet is computationally demanding during test time. The reason is that although the LikeNet's architecture can be fully parallelized, the number of cores are limited whilst LikeNet has many branches. Another drawback with LikeNet is that the current design of LikeNet does not allow for the estimation of subpixel displacements due to the classifier nature of the algorithm which performs based on a grid of integer displacements. The method proposed by Hinton et al. [37] suggests distillation of a cumbersome model in a smaller model, however the small model generalizes the same way as the large model.

In this section, we attempt to address these drawbacks by using LikeNet as a teacher to train a much faster CNN, Quick LikeNet (QLikeNet). Unlike LikeNet that solves

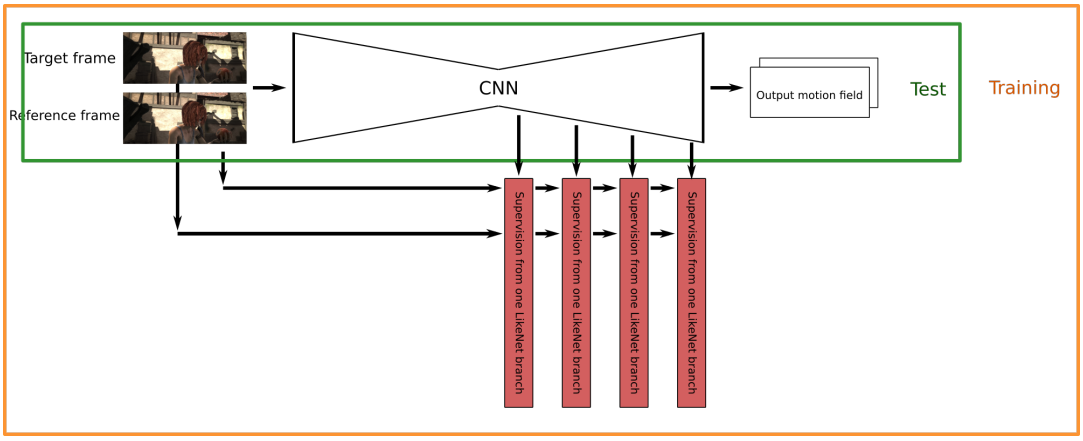

Figure 5.1: The overview of QLikeNet during the test and the training.

motion estimation as a classification problem, QLikeNet addresses motion estimation as a regression problem. More specifically, at test time, QLikeNet receives as input a pair of consecutive frames and calculates a pixel-level motion field, $(u, v) \subset \mathbb{R}^2$. QLikeNet is presented in a separate chapter apart from LikeNet as the nature of the two approaches, architectures, and training and test schemes are very different. The difference between QLikeNet and GradNet is the backpropagation from extra middle supervisions and that each supervision is formulated to come through one branch of LikeNet. Figure 5.1 depicts the overview of the proposed DNN during the test and training. Similar to our proposed methods in the previous chapters, we embed QLikeNet in a multiscale scheme to help with the estimation of large displacements. The idea is that a small model that is trained to generalize the same way as the cumbersome model, will perform better on the test data than the case that it is trained the normal way. In this chapter, inspired by the distillation idea, we propose a technique that uses one branch of already trained cumbersome LikeNet to train a faster network. Although distillation was the inspiration, we cannot call our method distillation as in distillation the direct output of the cumbersome model is used to train the smaller model, whilst we use the output of only one branch of LikeNet for training. As the smaller network is trained using likenet and is quicker, in the following we refer to it as QLikeNet.

5.1 Methodology

As mentioned in the previous chapter, each branch of LikeNet, given the input pair of frames, calculates a similarity map. The input pair of frames is the reference frame and the warped version of the target frame. The target frame is warped based on a constant motion field, a field with the same displacement vector for each pixel. Each value in the calculated similarity map represents a measure of how similar the pixels in corresponding locations in the two input frames are. More specifically, how successful the warp operation has been in aligning similar pixels. Concatenation of the similarity maps calculated by all branches forms a distribution over motion labels, $P(L|I;\theta)$. The mode of $P(L|I;\theta)$ could then be used as a point estimate of the motion/label field, that is $L^* = \underset{L}{\operatorname{argmax}}\, P(L|I;\theta)$.

In order to train QLikeNet, the motion field $F \subset \mathbb{R}^2$ calculated by the CNN, QLikeNet, with parameters ω, is used to warp the target frame towards the reference frame. Given the pair of the reference frame alongside the target frame, I, to a single branch of LikeNet, it calculates a similarity map (or P, as only one branch is used). We propose to train QLikeNet by maximizing $P(F|I;\theta)$. Maximizing $P(F|I;\theta)$ encourages QLikeNet to calculate a motion field that is most successful in aligning the corresponding pixels in reference and target frames. Figure 5.2 illustrates the process. Maximizing the similarity map corresponds to minimizing the minus similarity map. The loss function we minimize during the training of QLikeNet is,

$$L(F) = -|S(I_1, \hat{I}_2(F); \omega)|_1 \tag{5.1}$$

where S, N, and ω are respectively the similarity map, the number of pixels in the similarity map and the parameters of QLikeNet. As mentioned before, the similarity map is the output of one branch of LikeNet. Similar to [25, 69], we apply the supervision at each scale of the decoder side in our architecture, Fig. 5.3. The loss

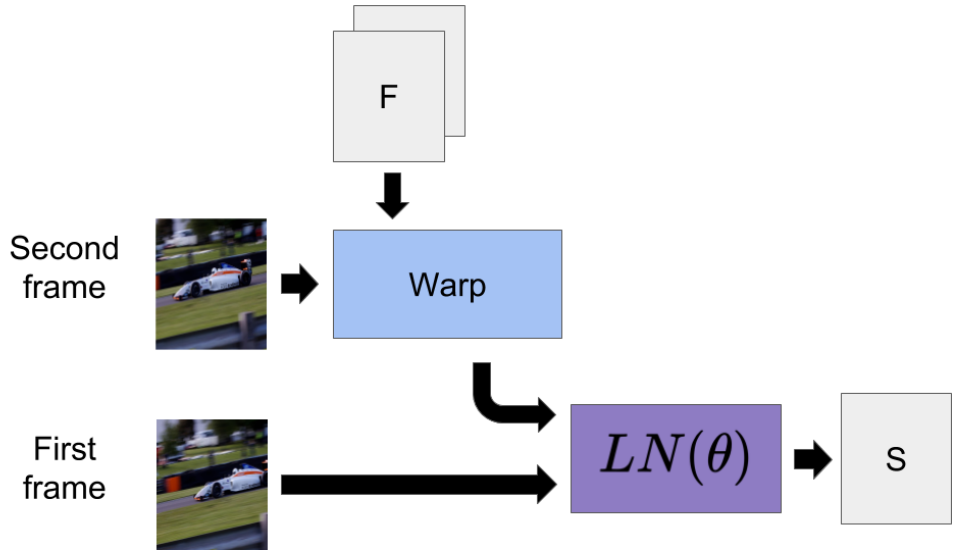

Figure 5.2: F represents the estimated motion field. $LN(\theta)$ represents one branch of LikeNet. S represents the similarity map which is the output of one branch of LikeNet.

function we minimize during the training in each layer, is a combination of the loss in the current scale and the sum of the losses in the previous scales weighted by ω. In all of our experiments we set $\eta = 0.1$. The loss function that we use for training the network in the final scale is:

$$L_{tot} = \frac{1}{N}\sum_{x}^{N}(L_4 + \eta L_3 + \eta L_2 + \eta L_1) \tag{5.2}$$

5.2 Architecture

Following the literature, similarly to most of the other unsupervised methods, we adopt an architecture inspired from FlowNetS. Our architecture is smaller in terms of learned parameters and also incorporates less down/up scaling in the architecture which is meant to estimate smaller displacements in one go. Using a bigger architecture does not improve the performance. To help with the estimation of large displacements, we adopt the classic multiscale scheme.

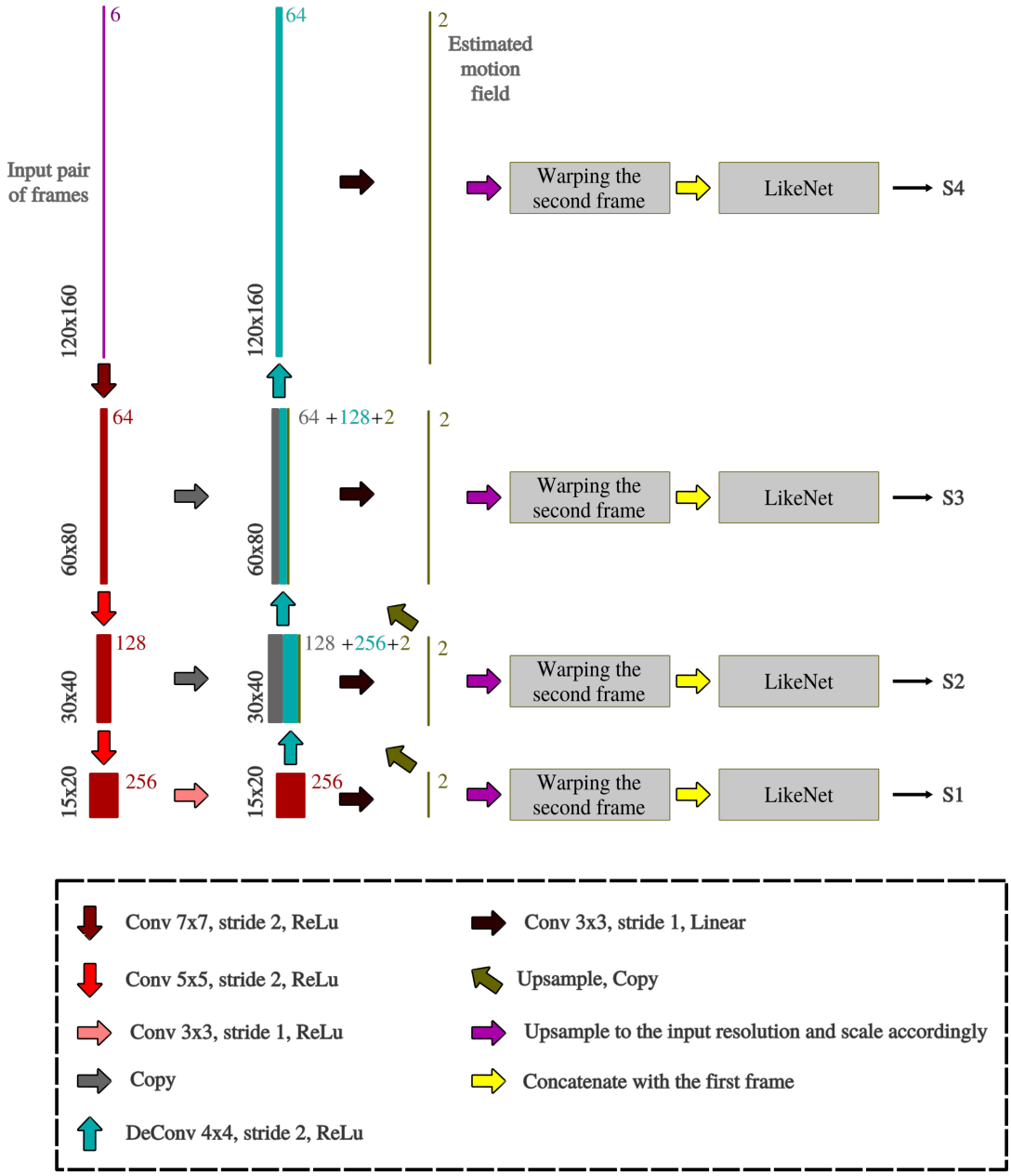

Figure 5.3: QLikeNet architecture and the training block diagram.

The calculation of the gradient of the loss with respect to the QLikeNet's weights is computationally feasible, Eq. 5.3. The gradient of the first term in Equation 5.3 can be calculated in closed form as LikeNet is a conventional CNN. The second term is also

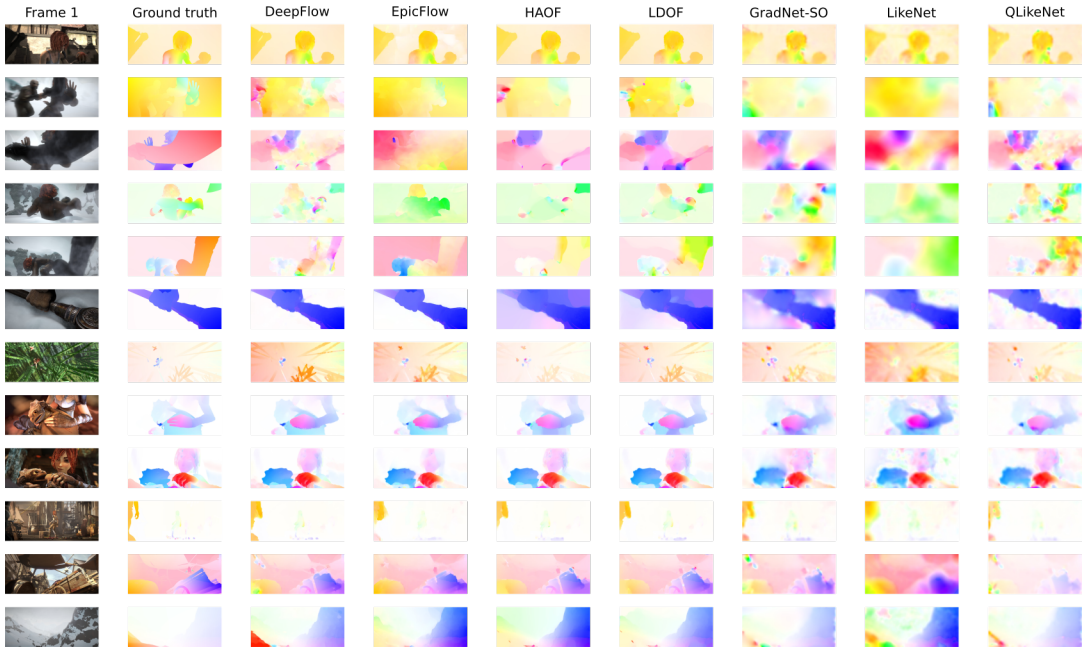

Figure 5.4: Comparison with classic methods. The visualized motion fields calculated by Deepflow [93], EpicFlow [70], High Accuracy Optical Flow (HAOF) [16], Large Displacement Optical Flow [17], GradNet Second Order (GradNet-SO), LikeNet, QLikeNet on MPISintel dataset [20].

computationally feasible as the warping operation is proven [42] to be differentiable.

$$\frac{\partial L}{\partial F} = \frac{-\partial S}{\partial I(x+F, t+\delta t)} \frac{\partial I(x+F, t+\delta t)}{\partial F} \tag{5.3}$$

5.3 Experimental Results

Table 5.2 reports the performance of QLikeNet for both cases when the training is based on VGG features or tracking features [11]. The results show that QLikeNet can perform better than LikeNet on some datasets although the training is based on the same features.

Similar to in chapter 3, for further evaluation, the motion field estimated by different methods are visualized, respectively, in figures 5.4 to compare with classic methods and

	Method	d<10	10<d<40	d>40
Classic	EpicFlow	2.71	6.98	36.76
	DeepFlow	2.19	6.16	40.51
	HAOF	2.69	8.14	50.29
	LDOF	2.19	6.46	41.35
	HS	7.07	16.46	62.97
DNN-based	GradNet-FO	4.68	11.42	52.02
	GradNet-SO	3.89	9.15	43.74
	LikeNet	3.21	9.26	45.27
	QLikeNet	3.79	8.91	40.36
	UnFlow	2.98	8.22	45.32
	SpyNet	2.52	7.15	35.62

Table 5.1: AEE for different ranges of d, $d = \sqrt{u_{gt}^2 + v_{gt}^2}$.

		Method	Fine tuned	Sintel clean		Sintel final		KITTI 2015	Middlebury
				train	test	train	test	train	train
Classic		DeepFlow	N	2.66	5.38	3.57	7.21	10.63	0.25
		LDOF (CPU) [17]	N	4.64	7.56	5.96	9.12	18.19	0.44
		LDOF (GPU) [83]	N	4.76	-	6.32	-	18.20	0.36
		EpicFlow [70]	N	2.27	4.12	3.56	6.29	9.27	0.31
		FlowFields [8]	N	1.86	3.75	3.06	5.81	8.33	0.27
		PCA-Layers [96]	N	3.22	5.73	4.52	7.89	12.74	0.66
		PCA-Flow [96]	N	4.04	6.83	5.18	8.65	14.01	.70
Deep neural network based	Sup	FlowNetS [25]	N	4.50	6.96	5.45	7.52	-	1.09
		FlowNetC [25]	N	4.31	6.85	5.87	8.51	-	1.15
		SpyNet [67]	N	4.12	6.69	5.57	8.43		0.33
		FlowNet2 [41]	N	2.02	3.96	3.14	6.02	10.06	0.35
		UnFlow-CS-ft-(KITTI supervised) [59]	Y	-	-	11.99	-	(2.25)	0.64
		UnFlow-CSS-ft(KITTI supervised) [59]	Y	-	-	13.65	-	(1.86)	0.64
	Unsup	UCNNME [2]	**N**	8.94	12.4	10.60	13.78	14.94	2.79
		DSTFlow [69]	**N**	6.93	10.40	7.82	11.11	24.30	-
		DSTFlow(KITTI) [69]	Y	7.10	10.95	7.95	11.8	16.79	-
		DSTFlow(Sintel) [69]	Y	6.16	10.41	7.38	11.28	23.69	-
		DSTFlow(C+K) [69]	Y	7.51	-	8.29	-	22.93	-
		DSTFlow(C+S) [69]	Y	6.47	10.84	6.81	11.27	25.98	-
		UnFlow-C-Cityscapes [59]	N	-	-	8.23	-	10.78	0.85
		UnFlow-C [59]	N	-	-	8.64	-	8.80	0.88
		UnFlow-CS [59]	N	-	-	7.92	-	8.14	0.65
		UnFlow-CSS [59]	N	-	-	7.91	10.22	8.10	0.65
		GradNet-SO (Second Order)	**N**	6.65	10.30	7.81	11.25	16.28	1.14
		LikeNet (VGG)	**N**	5.7	10.02	6.49	10.69	14.66	0.78
		LikeNet (Tracking)	**N**	5.94	10.30	7.22	11.12	25.69	0.81
		QLikeNet (VGG)	**N**	**5.49**	**8.87**	6.99	10.24	17.78	**0.64**
		QLikeNet (Tracking)	**N**	**5.61**	**9.02**	7.00	10.38	17.34	**0.64**

Table 5.2: Average End-point Error (in pixels) of classic and DNN-based methods. QLikeNet performs better or in par with other unsupervised methods although it is not finetuned on any of the evaluation datasets and its capacity is considerably smaller that all other DNN-based methods.

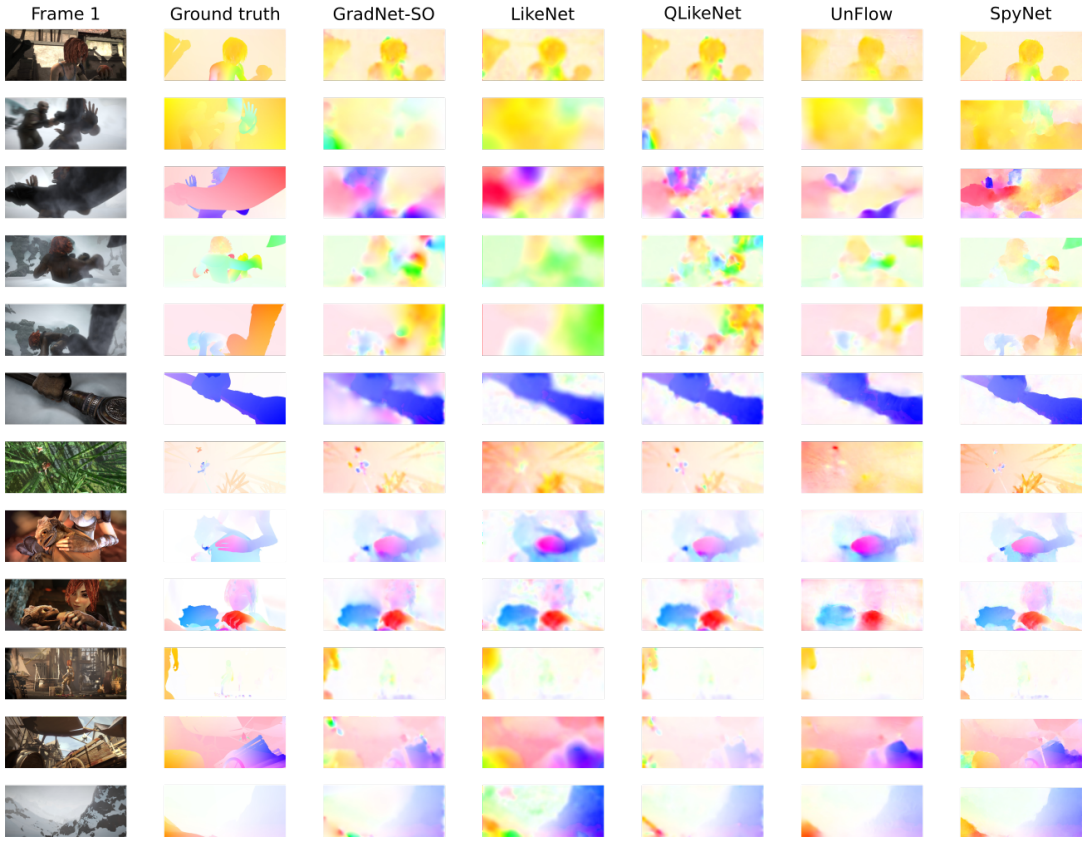

Figure 5.5: Comparison with DNN-based methods. The visualized motion fields calculated by UnFlow [59], SpyNet [67], GradNet Second Order (GradNet-SO), LikeNet, QLikeNet on MPISintel dataset [20].

in figure 5.5 to compare with DNN-based methods. Comparing with classic methods, the quality of the motion field estimated by LikeNet and QLikeNet look similar to the ones estimated by EpicFlow. Although, the quantitative results in table 5.1 suggests that EpicFlow performs better. Comparing LikeNet with QLikeNet, the motion field estimated by QLikeNet seems noisier than LikeNet. The reason is that QLikeNet is not trained in a Siamese way and there is not smoothness term in its loss function. Comparing with DNN-based methods, figures 5.5, it can be seen that there is still a gap between unsupervised methods and a supervised method. Although, UnFlow, which models occlusion, still performs better than GradNet, LikeNet, and QLikeNet.

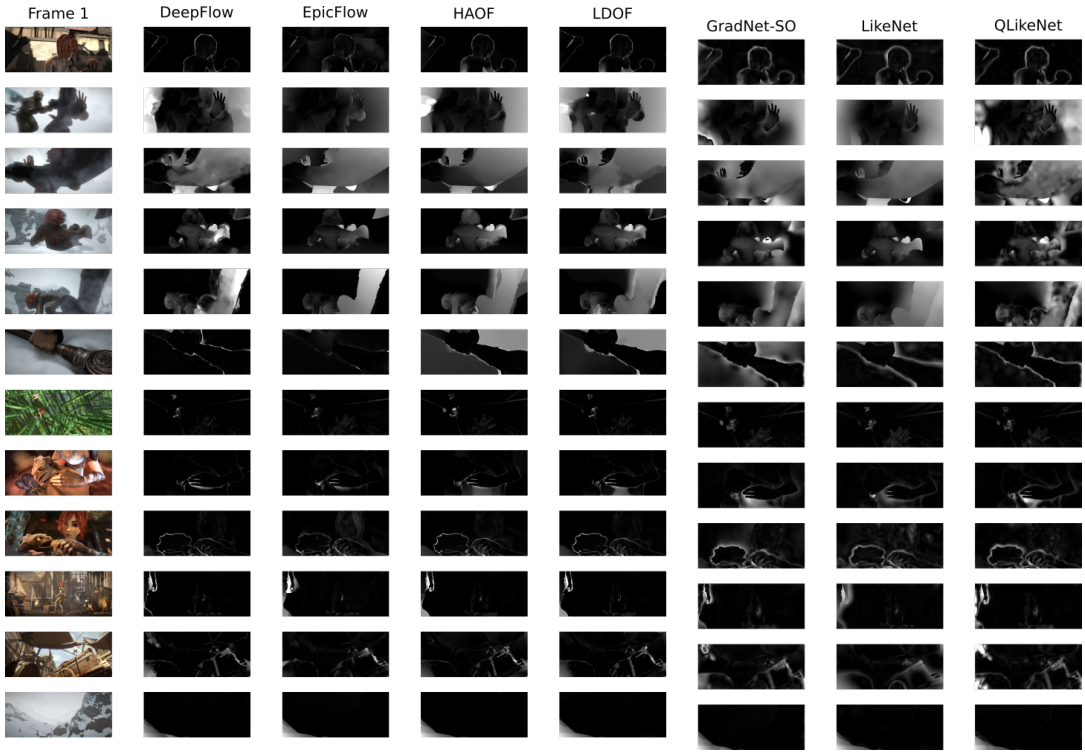

Figure 5.6: Comparison with classic methods. The AEE maps calclualted for several samples from several methods on MPISintel the final-training split. Each sample is normalized by a factor of $\frac{255}{max(d)}$, where $d = \sqrt{u_{gt}^2 + v_{gt}^2}$. Values more than 255 are rounded to 255.

The error maps are also visualized, respectively, in figures 5.6 to compare with classic methods and in figure 5.7 to compare with DNN-based methods. Figure 5.6 shows that all methods have difficulty in estimation at motion discontinuities, occluded area, low-texture area, and where the intensity variation is significant. Figure 5.7 shows that although UnFlow takes occlusion into consideration, it is yet not able to estimate for occluded area. The situation for the supervised method, SpyNet [67], is fairly better although the supervised method also fails when there are sever occlusion and intensity variation, e.g. 5^{th} sample. SpyNet also has problem for estimation around motion discontinuities. It also has problem in case of low-texture area, e.g. 2^{nd} sample.

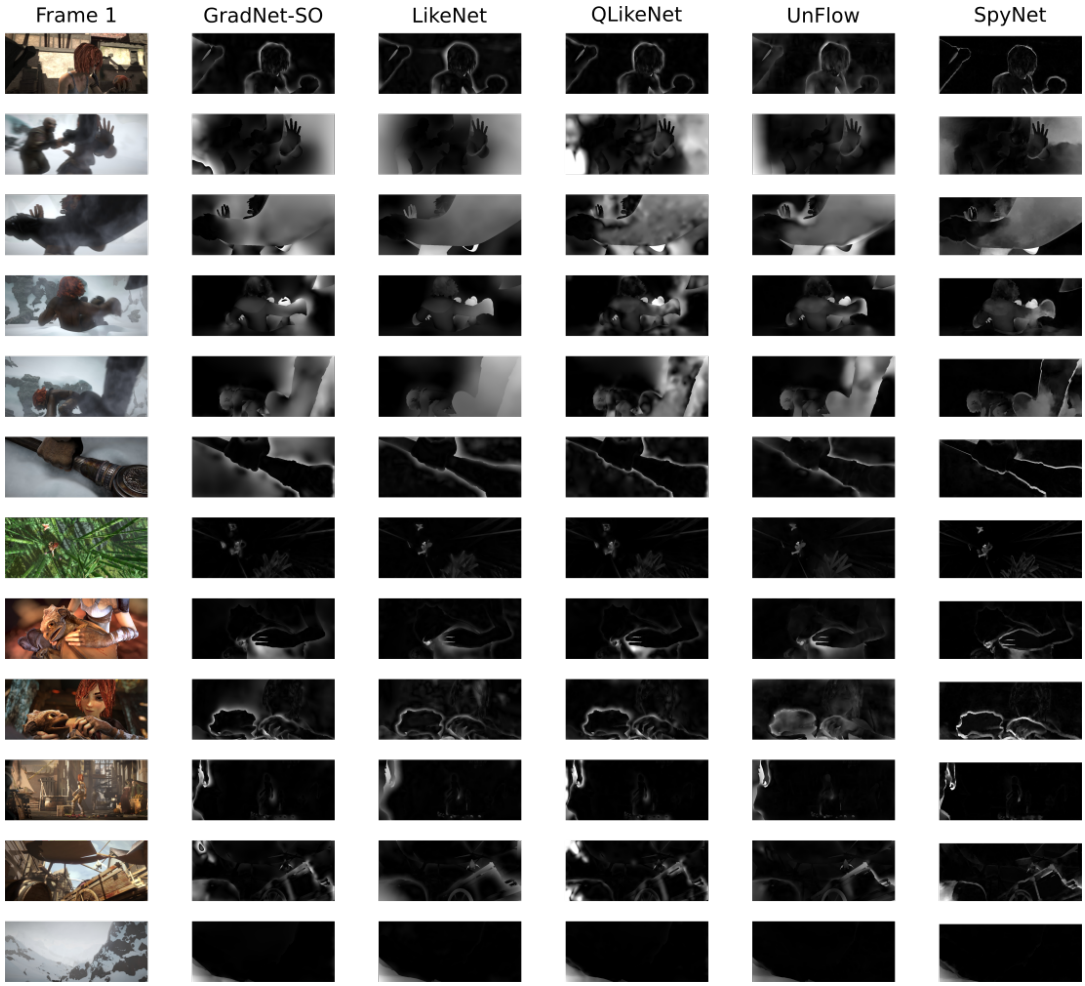

Figure 5.7: Comparison with DNN-based methods. The AEE maps calclualted for several samples from several methods on MPISintel the final-training split. Each sample is normalized by a factor of $\frac{255}{max(d)}$, where $d = \sqrt{u_{gt}^2 + v_{gt}^2}$. Values more than 255 are rounded to 255.

5.4 Computational Complexity

The runtime of QLikeNet is $161ms$ compared to LikeNet which is $19208ms$, GradNet which is $156ms$, and DSTFlow [69] which is $30ms$. DSTFlow has the same architecture as FlowNetS [25]. To measure the runtime, both algorithms are implemented using Lasagne in Theano and the exploited GPU is GeForce GTX 1080. Table 5.3 contains the runtime breakdown of GradNet.

In terms of learned parameters, QLikeNet has a comparably low number of learned

	QLikeNet (CNN)	Smoothening	Warping	**Sum**
scale 1	0.0018	0.0001	0.0007	0.0025
scale 2	0.0017	0.0001	0.0008	0.0026
scale 3	0.0021	0.0001	0.0010	0.0032
scale 4	0.0030	0.0002	0.0014	0.0046
scale 5	0.0044	0.0004	0.0023	0.0070
scale 6	0.0076	0.0008	0.0039	0.0123
scale 7	0.0140	0.0016	0.0075	0.0231
scale 8	0.0282	0.0031	0.0142	0.0454
scale 9	0.0540	0.0068	0.0000	0.608
Sum	0.1167	0.0132	0.0318	**0.1618**

Table 5.3: The runtime breakdown of GradNet in multiscale scheme in seconds. Scale 1 refers to the lowest resolution, scale 9 refers to the main resolution.

Method	Number of learned parameters
FlowNetS	32,070,472
FlowNetC	32,561,032
SpyNet	1,200,250
GradNet	3,666,562
LikeNet	697,028
QLikeNet	**3,716,550**

Table 5.4: Compared to other DNN-based methods, QLikeNet is comparably small in terms of learned parameters. DSTFlow [69] follows the FlowNetS architecture.

parameters in comparison with other supervised and unsupervised motion estimation methods. Table 5.4 contains a comparison with other CNN architectures trained for motion estimation.

5.5 Conclusion

In this chapter, we have shown how to squeeze the computationally heavy LikeNet in a lighter and quicker CNN, QLikeNet without a significant drop in the accuracy. We realize this by training QLikeNet using the supervision received from one branch of LikeNet. A LikeNet that is trained using a feature constancy constraint in an unsupervised manner and without the need for any handcrafted regularization or other constraints. QLikeNet is trained on the real UCF101 dataset. We show that it per-

forms better than the other state-of-the-art unsupervised methods that do not use bi-directional constraints, and that it can generalize very well to unknown datasets without the need for finetuning. QLikeNet is significantly faster than LikeNet, yet performing comparably.

Chapter 6

Conclusions

In this thesis, we have explored different unsupervised techniques for the problem of dense motion estimation. We started with a fully convolutional hourglass architecture which we trained in an unsupervised way, without the need for ground truth motion field, for motion estimation, GradNet. The loss function we minimized for training is based on the classic loss function proposed by Horn and Schunk [38] which penalizes the deviation from the intensity constancy assumption by minimizing the motion compensated intensity error. The loss is based on a first-order Taylor expansion which makes the backpropagation of error, computationally feasible. We also explore the case where a second-order Taylor expansion is used. We also show that if a specific polynomial is fitted to a specific set of pixels, the result is similar to the second order Taylor expansion. This also explains the connection to the widely used interpolation-based spatial transformer method [42]. Although GradNet is trained on a real dataset, UCF101, the evaluations on both synthetic and real datasets show that GradNet generalizes well to unknown datasets. The experiments show that a network trained using a higher-order expansion and applied in a multiscale scheme, on average performs slightly better than a network that is trained using a transformer layer and applied in single resolution. The terms in the loss function using which GradNet is trained do not model occlusion and non-rigid deformations, or provide robustness

towards intensity variations, although more sophisticated loss functions and features (instead of only intensity features) will be used in the future.

Although GradNet performs well, yet there was a fairly large gap between the supervised and unsupervised DNN-based methods. We propose a second method that performs much better than GradNet and works based on the similarities between the features extracted from the reference frame and the uniformly warped version of the target frame, LikeNet. Ahead of all other unsupervised methods, LikeNet focuses on the features calculated by the first layer of the VGG object recognition DNN rather than intensity features. Also, unlike other DNN-based methods, LikeNet solves motion estimation as a classification problem. LikeNet is embedded in the classic multiscale scheme and a Conditional Random Field (CRF) implemented as an RNN was used in the lowest scale of the multiscale scheme to improve the estimated motion field by preventing the propagation of error across the scales. Although LikeNet has the lowest number of learned parameters among supervised and unsupervised methods, it performs better than other uni-directional unsupervised DNN-based motion estimation techniques with more computational complexity. The flexible architecture of LikeNet allows for a trade-off between the required memory and computational load. Although LikeNet is fully parallelizable, the full parallelization is not feasible due to limited processing resources in available GPUs. This makes LikeNet run slower than other methods.

In Chapter 5, we propose to distill LikeNet in a much faster CNN, QLikeNet without loosing much of the accuracy. The architecture is inspired from GradNet, however slightly slower, and similarly embedded in a classic multiscale scheme.

6.1 Future Work

Most of the recent unsupervised motion estimation methods focus on modeling the occluded area which removing during the training is claimed to improve the performance during the test. Estimation of the motion field for the occluded area can be promising as no effort has been put in this direction.

So far, the loss functions we used were mainly focusing on feature constancy constraint. In the future, we will try to work on learning the estimation of optical flow under non-rigid transformations and provide more robustness against the intensity variations. Also, different augmentations such as scaling, rotation, translation, and color jittering will be tried to improve the results even further. So far, we have used intensity and VGG features although, another direction would be to use features that are more robust towards variations from the reference frame to the target frame for training.

The motion estimation method proposed in this thesis receive a pair of frames as input and calculate a motion field. Designing a framework that receives as input a sequence of frames and calculates the motion field can be promising. The input sequence provides the network with the motion history in previous frames which can improve the estimated motion at the current frame.

GANs have shown to have promising performance in different applications [32, 68, 3, 85, 65, 22, 24, 50, 29]. Given that the occluded area are problematic in motion estimation as the correspinding points are missing from one of the reference or target frames, using GANs to generate the motion field for the missing area can be a promissing direction.

DNNs trained for motion estimation are able to build low to high-level features in their intermediate layers. These representations encode important motion/object-related information and can be used as extra input data to ease other recognition/estimation tasks such as human action recognition and pose estimation.

Appendix **A**

Arriving at the higher-order Taylor Expansion of Motion Compensated Intensity by Fitting a Polynomial

As mentioned in Chapter 3, the motion compensated intensity, $I(x+\alpha, y+\beta, t+1)$, can be approximated using the interpolation of a set of nodes (pixels). By choosing a specific polynomial and calculating its parameters by fitting a specific set of nodes, the second-order Taylor expansion, in Chapter 3, can be derived. The derivatives in the derived Taylor expansion have to be calculated in a specific way. This appendix describes how the second-order Taylor expansion can be derived from an interpolation of a specific set of nodes, in section 3. From the following general polynomial formula,

$$\sum_{i}\sum_{j}\sum_{k} C_{ijk}\alpha^{i}\beta^{j}\gamma^{k} \tag{A.1}$$

where α, β, and γ are variables and $C_{i,j,k}$ are constants, specific terms are drawn to

approximate $I_{\alpha,\beta,\gamma}(x, y, t)$,

$$\begin{aligned} I_{\alpha,\beta,\gamma}(x, y, t) = C_{000} + C_{100}\alpha + C_{010}\beta + C_{001}\gamma + \\ \frac{1}{2!}(C_{110}\alpha\beta + C_{101}\alpha\gamma + C_{011}\beta\gamma \\ + C_{200}\alpha^2 + C_{020}\beta^2) \end{aligned} \tag{A.2}$$

The idea is to fit a polynomial of degree 2 with respect to x and y and degree 1 with respect to t given 9 points a, b, c, d, e, f from the reference frame and g, h, i from the target frame, Fig. A.1.

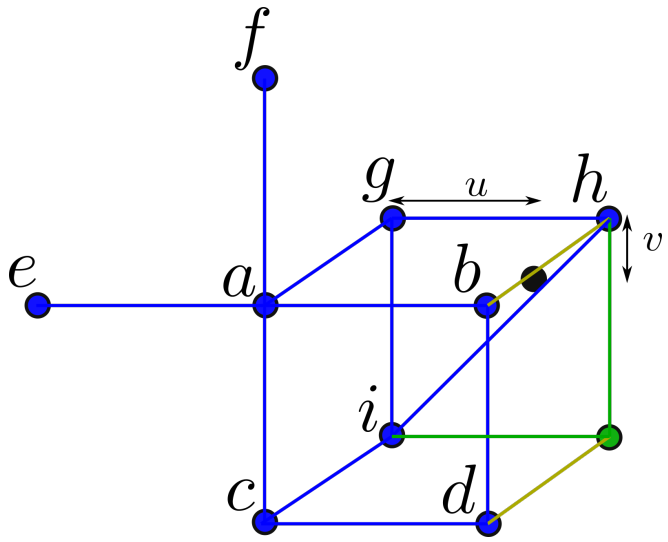

Figure A.1: The nodes participating in the interpolation.

In Fig. A.1, for node a, the variables (α, β, γ) are $(0, 0, 0)$ and for node h, the variables are (1,0,1). The other nodes follow the same coordination system.

A.1 Fitting Nodes in $t = 0$ Plane

By putting the nodes, a, b, c, d, e, and f in the Eq. A.2,

$$a = C_{000} \tag{A.3}$$

$$b = a + C_{100} + C_{200} \tag{A.4}$$

$$c = a + C_{010} + C_{020} \tag{A.5}$$

$$e = a - C_{010} + C_{200} \tag{A.6}$$

$$f = a - C_{010} + C_{020} \tag{A.7}$$

$$d = a + C_{100} + C_{010} + C_{200} + C_{020} + C_{110} \tag{A.8}$$

The addition of the left sides of Equations. A.7 and A.5 has to be equal to the addition of the right sides,

$$f + c = 2a + 2C_{020} \implies C_{020} = \frac{1}{2} I_{yy} | I_{yy} \triangleq (c - a) - (a - f) : \tag{A.9}$$

By adding both sides of Equations. A.6 and A.4,

$$b + e = 2a + 2C_{200} \implies C_{200} = \frac{1}{2} I_{xx} | I_{xx} \triangleq (b - a) - (a - e) : \tag{A.10}$$

By substituting C_{200} value obtained in Equation. A.10 into Equation. A.4,

$$b = a + C_{100} + \frac{b - 2a + e}{2} \implies C_{100} = I_x | I_x \triangleq \frac{b - e}{2} \tag{A.11}$$

And by substituting C_{020} value obtained in Equation. A.9 into Equation. A.5,

$$c = a + C_{010} + \frac{c - 2a + f}{2} \implies C_{010} = I_y | I_y \triangleq \frac{c - f}{2} \tag{A.12}$$

By substituting C_{100}, C_{010}, C_{200}, and C_{020}, values into Equation. A.8,

$$C_{110} = I_{xy} | I_{xy} \triangleq (d - c) - (b - a) \tag{A.13}$$

A.2 Fitting Nodes in $t = 1$ Plane

$$g = a + C_{001} \implies C_{001} = I_t | I_t \triangleq g - a \tag{A.14}$$

Having C_{001} calculated, the C_{101} and C_{011} can be calculated from the following Equation,

$$h = a + I_x + C_{001} + C_{101} + I_{xx}, \tag{A.15}$$

which leads to,

$$C_{101} = I_{xt} | I_{xt} \triangleq (h - g) - (b - a) \tag{A.16}$$

and,

$$i = a + I_y + C_{001} + C_{011} + I_{yy} \tag{A.17}$$

which leads to,

$$C_{011} = I_{yt}|I_{yt} \triangleq (i - g) - (c - a) \tag{A.18}$$

Finally, the calculated constants of Eq. A.2 would turn Eq. A.2 into:

$$I(x, y, t) = a + \alpha I_x + \beta I_y + I_t + \frac{1}{2}(\alpha^2 I_{xx} + \beta^2 I_{yy} + \alpha\beta I_{xy} + \alpha I_{xt} + \beta I_{yt}) \tag{A.19}$$

Bibliography

[1] A. Ahmadi, I. Marras, and I. Patras. Likenet: A siamese motion estimation network trained in an unsupervised way. In *BMVC*, 2018. 3

[2] A. Ahmadi and I. Patras. Unsupervised convolutional neural networks for motion estimation. In *Image Processing (ICIP), 2016 IEEE International Conference on*, pages 1629–1633. IEEE, 2016. 3, 5, 27, 69, 80

[3] Y. Almalioglu, M. R. U. Saputra, P. P. de Gusmao, A. Markham, and N. Trigoni. Ganvo: Unsupervised deep monocular visual odometry and depth estimation with generative adversarial networks. *arXiv preprint arXiv:1809.05786*, 2018. 88

[4] L. Alvarez, R. Deriche, T. Papadopoulo, and J. Sánchez. Symmetrical dense optical flow estimation with occlusions detection. *International Journal of Computer Vision*, 75(3):371–385, 2007. 30

[5] K. E. Atkinson and W. Han. *Elementary numerical analysis*. Wiley New York et al., 1985. 38

[6] M. Aubry, D. Maturana, A. A. Efros, B. C. Russell, and J. Sivic. Seeing 3d chairs: exemplar part-based 2d-3d alignment using a large dataset of cad models. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3762–3769, 2014. 19, 21, 22

[7] J.-F. Aujol, G. Gilboa, T. Chan, and S. Osher. Structure-texture image decomposition—modeling, algorithms, and parameter selection. *International journal of computer vision*, 67(1):111–136, 2006. 15

[8] C. Bailer, B. Taetz, and D. Stricker. Flow fields: Dense correspondence fields for highly accurate large displacement optical flow estimation. In *Proceedings of the*

IEEE International Conference on Computer Vision, pages 4015–4023, 2015. 3, 51, 69, 80

[9] S. Baker, D. Scharstein, J. Lewis, S. Roth, M. J. Black, and R. Szeliski. A database and evaluation methodology for optical flow. *International Journal of Computer Vision*, 92(1):1–31, 2011. 14, 24, 49

[10] F. Behar-Cohen, G. Baillet, T. de Ayguavives, P. O. Garcia, J. Krutmann, P. Peña-García, C. Reme, and J. S. Wolffsohn. Ultraviolet damage to the eye revisited: eye-sun protection factor (e-spf®), a new ultraviolet protection label for eyewear. *Clinical ophthalmology (Auckland, NZ)*, 8:87, 2014. 72

[11] L. Bertinetto, J. Valmadre, J. F. Henriques, A. Vedaldi, and P. H. Torr. Fully-convolutional siamese networks for object tracking. In *European conference on computer vision*, pages 850–865. Springer, 2016. 79

[12] M. J. Black. Explaining optical flow events with parameterized spatio-temporal models. In *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, volume 1, pages 326–332. IEEE, 1999. 2

[13] M. J. Black and P. Anandan. The robust estimation of multiple motions: Parametric and piecewise-smooth flow fields. *Computer vision and image understanding*, 63(1):75–104, 1996. 14, 25

[14] M. J. Black, Y. Yacoob, A. D. Jepson, and D. J. Fleet. Learning parameterized models of image motion. In *Proceedings of IEEE computer society conference on Computer vision and pattern recognition*, pages 561–567. IEEE, 1997. 17

[15] J. Braux-Zin, R. Dupont, and A. Bartoli. A general dense image matching framework combining direct and feature-based costs. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 185–192, 2013. 17

[16] T. Brox, A. Bruhn, N. Papenberg, and J. Weickert. High accuracy optical flow estimation based on a theory for warping. In *Computer Vision-ECCV 2004*, pages 25–36. Springer, 2004. x, xi, 2, 3, 4, 14, 15, 16, 25, 52, 53, 79

[17] T. Brox and J. Malik. Large displacement optical flow: descriptor matching in variational motion estimation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 33(3):500–513, 2011. x, xi, 2, 3, 4, 16, 51, 52, 53, 69, 79, 80

[18] A. Bruhn, J. Weickert, and C. Schnörr. Lucas/kanade meets horn/schunck: Combining local and global optic flow methods. *International journal of computer vision*, 61(3):211–231, 2005. 15

[19] P. Burt and E. Adelson. The laplacian pyramid as a compact image code. *IEEE Transactions on communications*, 31(4):532–540, 1983. 14

[20] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black. A naturalistic open source movie for optical flow evaluation. In *Computer Vision–ECCV 2012*, pages 611–625. Springer, 2012. x, xi, xii, 48, 50, 53, 79, 81

[21] Q. Chen and V. Koltun. Full flow: Optical flow estimation by global optimization over regular grids. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4706–4714, 2016. 25

[22] Y. Choi, M. Choi, M. Kim, J.-W. Ha, S. Kim, and J. Choo. Stargan: Unified generative adversarial networks for multi-domain image-to-image translation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8789–8797, 2018. 88

[23] E. L. Denton, S. Chintala, R. Fergus, et al. Deep generative image models using a laplacian pyramid of adversarial networks. In *Advances in neural information processing systems*, pages 1486–1494, 2015. 15

[24] H. Dong, P. Neekhara, C. Wu, and Y. Guo. Unsupervised image-to-image translation with generative adversarial networks. *arXiv preprint arXiv:1701.02676*, 2017. 88

[25] A. Dosovitskiy, P. Fischer, E. Ilg, P. Höusser, C. Hazırbaş, V. Golkov, P. v. d. Smagt, D. Cremers, and T. Brox. Flownet: Learning optical flow with convolutional networks. In *IEEE International Conference on Computer Vision (ICCV)*, Dec 2015. viii, ix, 2, 5, 19, 20, 21, 27, 28, 32, 51, 55, 69, 71, 76, 80, 83

[26] Q. Duan, E. Angelini, S. Homma, and A. Laine. Tracking endocardium using optical flow along iso-value curve. In *Engineering in Medicine and Biology Society, 2006. EMBS'06. 28th Annual International Conference of the IEEE*, pages 707–710. IEEE, 2006. 2

[27] F. Dufaux and J. Konrad. Efficient, robust, and fast global motion estimation for video coding. *Image Processing, IEEE Transactions on*, 9(3):497–501, 2000. 2

[28] P. Fischer, A. Dosovitskiy, E. Ilg, P. Häusser, C. Hazırbaş, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox. Flownet: Learning optical flow with convolutional networks. *arXiv preprint arXiv:1504.06852*, 2015. 18, 26, 61

[29] J. Gauthier. Conditional generative adversarial nets for convolutional face generation. *Class Project for Stanford CS231N: Convolutional Neural Networks for Visual Recognition, Winter semester*, 2014(5):2, 2014. 88

[30] F. C. Glazer. Hierarchical motion detection. 1987. 14

[31] P. Golland and A. M. Bruckstein. Motion from color. *Computer Vision and Image Understanding*, 68(3):346–362, 1997. 15

[32] K. Gwn Lore, K. Reddy, M. Giering, and E. A. Bernal. Generative adversarial networks for depth map estimation from rgb video. In *Proceedings of the IEEE*

Conference on Computer Vision and Pattern Recognition Workshops, pages 1177–1185, 2018. 88

[33] D. Hafner, O. Demetz, and J. Weickert. Why is the census transform good for robust optic flow computation? In *International Conference on Scale Space and Variational Methods in Computer Vision*, pages 210–221. Springer, 2013. 29

[34] R. Hamming. *Numerical methods for scientists and engineers*. Courier Corporation, 2012. 38

[35] N. Hata, A. Nabavi, W. M. Wells III, S. K. Warfield, R. Kikinis, P. M. Black, and F. A. Jolesz. Three-dimensional optical flow method for measurement of volumetric brain deformation from intraoperative mr images. *Journal of Computer Assisted Tomography*, 24(4):531–538, 2000. 2

[36] J. Hays and A. A. Efros. Im2gps: estimating geographic information from a single image. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008. 19

[37] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. 74

[38] B. K. Horn and B. G. Schunck. Determining optical flow. In *1981 Technical symposium east*, pages 319–331. International Society for Optics and Photonics, 1981. x, 3, 4, 11, 13, 14, 17, 25, 35, 36, 38, 51, 53, 86

[39] A. Hosni, C. Rhemann, M. Bleyer, C. Rother, and M. Gelautz. Fast cost-volume filtering for visual correspondence and beyond. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(2):504–511, 2013. 25

[40] P. J. Huber. *Robust statistics*. Springer, 2011. 14

[41] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. *arXiv preprint arXiv:1612.01925*, 2016. viii, 2, 5, 18, 22, 23, 26, 28, 51, 69, 80

[42] M. Jaderberg, K. Simonyan, A. Zisserman, et al. Spatial transformer networks. In *Advances in Neural Information Processing Systems*, pages 2017–2025, 2015. x, 26, 27, 31, 35, 37, 43, 46, 79, 86

[43] M. Jain, H. Jégou, and P. Bouthemy. Better exploiting motion for better action recognition. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 2555–2562. IEEE, 2013. 2

[44] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 2146–2153. IEEE, 2009. 62

[45] J. Y. Jason, A. W. Harley, and K. G. Derpanis. Back to basics: Unsupervised learning of optical flow via brightness constancy and motion smoothness. In *European Conference on Computer Vision*, pages 3–10. Springer, 2016. 7, 8, 27, 28, 29

[46] Y.-G. Jiang, Q. Dai, X. Xue, W. Liu, and C.-W. Ngo. Trajectory-based modeling of human actions with motion reference points. In *Computer Vision–ECCV 2012*, pages 425–438. Springer, 2012. 2

[47] S. L. Keeling. Medical image registration and interpolation by optical flow with maximal rigidity. In *Mathematical Models for Registration and Applications to Medical imaging*, pages 27–61. Springer, 2006. 2

[48] R. Kennedy and C. J. Taylor. Optical flow with geometric occlusion estimation and fusion of multiple frames. In *International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition*, pages 364–377. Springer, 2015. 17

[49] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2014. 47

[50] I. Korshunova, W. Shi, J. Dambre, and L. Theis. Fast face-swap using convolutional neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3677–3685, 2017. 88

[51] P. Krähenbühl and V. Koltun. Efficient inference in fully connected crfs with gaussian edge potentials. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 109–117. Curran Associates, Inc., 2011. 65

[52] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1106–1114, 2012. 4, 18

[53] W. Li. Mpeg-4 video verification model version 18.0. *ISO/IEC JTC1/SC29/WG11, N3908*, 2001. 2

[54] C. Liu, J. Yuen, A. Torralba, J. Sivic, and W. T. Freeman. Sift flow: Dense correspondence across different scenes. In *European conference on computer vision*, pages 28–42. Springer, 2008. 16

[55] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. *CVPR (to appear)*, Nov. 2015. 2

[56] D. G. Lowe. Object recognition from local scale-invariant features. In *iccv*, page 1150. Ieee, 1999. 16

[57] B. D. Lucas, T. Kanade, et al. An iterative image registration technique with an application to stereo vision. 1981. 11, 13, 14

[58] N. Mayer, E. Ilg, P. Hausser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4040–4048, 2016. 22, 26

[59] S. Meister, J. Hur, and S. Roth. Unflow: Unsupervised learning of optical flow with a bidirectional census loss. In *AAAI*, 2018. ix, xii, xiii, 3, 5, 28, 29, 30, 68, 69, 80, 81

[60] E. Mémin and P. Pérez. Dense estimation and object-based segmentation of the optical flow with robust techniques. *IEEE Transactions on Image Processing*, 7(5):703–719, 1998. 14

[61] M. Menze, C. Heipke, and A. Geiger. Joint 3d estimation of vehicles and scene flow. In *ISPRS Workshop on Image Sequence Analysis (ISA)*, 2015. 49

[62] Y. Mileva, A. Bruhn, and J. Weickert. Illumination-robust variational optical flow with photometric invariants. In *Joint Pattern Recognition Symposium*, pages 152–162. Springer, 2007. 15

[63] M. A. Mohamed, H. A. Rashwan, B. Mertsching, M. A. García, and D. Puig. Illumination-robust optical flow using a local directional pattern. *IEEE Transactions on Circuits and Systems for Video Technology*, 24(9):1499–1508, 2014. 16

[64] J.-M. Odobez and P. Bouthemy. Robust multiresolution estimation of parametric motion models. *Journal of visual communication and image representation*, 6(4):348–365, 1995. 14

[65] S. Palsson, E. Agustsson, R. Timofte, and L. Van Gool. Generative adversarial style transfer networks for face aging. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 2084–2092, 2018. 88

[66] N. Papenberg, A. Bruhn, T. Brox, S. Didas, and J. Weickert. Highly accurate optic flow computation with theoretically justified warping. *International Journal of Computer Vision*, 67(2):141–158, 2006. 16

[67] A. Ranjan and M. J. Black. Optical flow estimation using a spatial pyramid network. *arXiv preprint arXiv:1611.00850*, 2016. ix, xii, 3, 8, 22, 23, 25, 36, 51, 56, 61, 69, 72, 80, 81, 82

[68] A. Ranjan, V. Jampani, K. Kim, D. Sun, J. Wulff, and M. J. Black. Adversarial collaboration: Joint unsupervised learning of depth, camera motion, optical flow and motion segmentation. *arXiv preprint arXiv:1805.09806*, 2018. 88

[69] Z. Ren, J. Yan, B. Ni, B. Liu, X. Yang, and H. Zha. Unsupervised deep learning for optical flow estimation. In *AAAI*, pages 1495–1501, 2017. xiii, xiv, 5, 8, 27, 35, 51, 55, 56, 61, 66, 68, 69, 71, 72, 76, 80, 83, 84

[70] J. Revaud, P. Weinzaepfel, Z. Harchaoui, and C. Schmid. Epicflow: Edge-preserving interpolation of correspondences for optical flow. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1164–1172, 2015. x, xi, 2, 4, 16, 51, 53, 69, 79, 80

[71] J. Revaud, P. Weinzaepfel, Z. Harchaoui, and C. Schmid. Deepmatching: Hierarchical deformable dense matching. *International Journal of Computer Vision*, 120(3):300–323, 2016. 17

[72] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015. x, 48

[73] D. Rosenbaum, D. Zoran, and Y. Weiss. Learning the local statistics of optical flow. In *Advances in Neural Information Processing Systems*, pages 2373–2381, 2013. 17

[74] L. I. Rudin, S. Osher, and E. Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: nonlinear phenomena*, 60(1-4):259–268, 1992. 15

[75] K. Simonyan and A. Zisserman. Two-stream convolutional networks for action recognition in videos. In *Advances in neural information processing systems*, pages 568–576, 2014. 2, 4

[76] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015. 4, 63

[77] K. Soomro, A. Roshan Zamir, and M. Shah. UCF101: A dataset of 101 human actions classes from videos in the wild. In *CRCV-TR-12-01*, 2012. 44, 47, 61

[78] F. Stein. Efficient computation of optical flow using the census transform. In *Joint Pattern Recognition Symposium*, pages 79–86. Springer, 2004. 29

[79] M. Stoll, S. Volz, and A. Bruhn. Adaptive integration of feature matches into variational optical flow methods. In *Asian Conference on Computer Vision*, pages 1–14. Springer, 2012. 17

[80] D. Sun, S. Roth, and M. J. Black. A quantitative analysis of current practices in optical flow estimation and the principles behind them. *International Journal of Computer Vision*, 106(2):115–137, 2014. 23, 25, 46

[81] D. Sun, S. Roth, J. Lewis, and M. J. Black. Learning optical flow. In *European Conference on Computer Vision*, pages 83–97. Springer, 2008. 17

[82] D. Sun, X. Yang, M.-Y. Liu, and J. Kautz. Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8934–8943, 2018. ix, 3, 8, 18, 24, 25, 26, 36

[83] N. Sundaram, T. Brox, and K. Keutzer. Dense point trajectories by gpu-accelerated large displacement optical flow. In *European conference on computer vision*, pages 438–451. Springer, 2010. 28, 51, 69, 80

[84] I. Sutskever, J. Martens, G. E. Dahl, and G. E. Hinton. On the importance of initialization and momentum in deep learning. *ICML (3)*, 28:1139–1147, 2013. 67

[85] R. K. Thakur and S. Mukherjee. A conditional adversarial network for scene flow estimation. *arXiv preprint arXiv:1904.11163*, 2019. 88

[86] W. Trobin, T. Pock, D. Cremers, and H. Bischof. An unbiased second-order prior for high-accuracy motion estimation. In *Joint Pattern Recognition Symposium*, pages 396–405. Springer, 2008. 29

[87] J. van de Weijer and T. Gevers. Robust optical flow from photometric invariants. In *2004 International Conference on Image Processing, 2004. ICIP'04.*, volume 3, pages 1835–1838. IEEE, 2004. 15

[88] C. Vogel, S. Roth, and K. Schindler. An evaluation of data costs for optical flow. In *German Conference on Pattern Recognition*, pages 343–353. Springer, 2013. 29

[89] H. Wang and C. Schmid. Action recognition with improved trajectories. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 3551–3558. IEEE, 2013. 2

[90] Y. Wang, Y. Yang, Z. Yang, L. Zhao, and W. Xu. Occlusion aware unsupervised learning of optical flow. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4884–4893, 2018. ix, 3, 5, 28, 29, 30, 31, 32, 33, 35

[91] J. Weber and J. Malik. Robust computation of optical flow in a multi-scale differential framework. *International Journal of Computer Vision*, 14(1):67–81, 1995. 24

[92] A. Wedel, T. Pock, C. Zach, H. Bischof, and D. Cremers. An improved algorithm for tv-l 1 optical flow. In *Statistical and geometrical approaches to visual motion analysis*, pages 23–45. Springer, 2009. 15

[93] P. Weinzaepfel, J. Revaud, Z. Harchaoui, and C. Schmid. Deepflow: Large displacement optical flow with deep matching. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 1385–1392. IEEE, 2013. x, xi, 3, 4, 16, 19, 53, 79

[94] P. Weinzaepfel, J. Revaud, Z. Harchaoui, and C. Schmid. Learning to detect motion boundaries. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2578–2586, 2015. 3

[95] Z. Wu, Y.-G. Jiang, X. Wang, H. Ye, X. Xue, and J. Wang. Fusing multi-stream deep networks for video classification. *arXiv preprint arXiv:1509.06086*, 2015. 2

[96] J. Wulff and M. J. Black. Efficient sparse-to-dense optical flow estimation using a learned basis and layers. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 120–130, 2015. 51, 69, 80

[97] J. Xu, R. Ranftl, and V. Koltun. Accurate optical flow via direct cost volume processing. *arXiv preprint arXiv:1704.07325*, 2017. 25

[98] R. Zabih and J. Woodfill. Non-parametric local transforms for computing visual correspondence. In *European conference on computer vision*, pages 151–158. Springer, 1994. 29

[99] C. Zhang, Z. Li, R. Cai, H. Chao, and Y. Rui. As-rigid-as-possible stereo under second order smoothness priors. In *European Conference on Computer Vision*, pages 112–126. Springer, 2014. 29

[100] S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. H. Torr. Conditional random fields as recurrent neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1529–1537, 2015. 6, 8, 60, 61, 65, 66

[101] H. Zimmer, A. Bruhn, and J. Weickert. Optic flow in harmony. *International Journal of Computer Vision*, 93(3):368–388, 2011. 15

[102] H. Zimmer, A. Bruhn, J. Weickert, L. Valgaerts, A. Salgado, B. Rosenhahn, and H.-P. Seidel. Complementary optic flow. In *International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition*, pages 207–220. Springer, 2009. 16