Institut für Visualisierung und Interaktive Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Unsupervised Approach to Estimate the Optical Flow Using RAFT's Architecture

Vijay Kumar

Studiengang:	Computer Science
Prüfer/in:	Prof. Dr. Andrés Bruhn
Betreuer/in:	M. Sc. Azin Jahedi
Beginn am:	11. November 2020
Beendet am:	11. Mai 2021

Abstract

Optic flow estimation is a significant area of interest within the field of computer vision. Recently, end-to-end neural network-based approaches have received significant interest in the estimation of optic flow. One of the best methods in this field is RAFT, which outperforms previously published methods like PWC-Net. Although these approaches perform best on competitive benchmarks, they have their drawbacks. Most of the state-of-the-art techniques are supervised learning-based methods and require a large amount of annotated data. The creation of such data sets is an expensive tedious task. Due to this fact, unsupervised approaches that do not need annotated data for being trained to become more popular. One of the most successful unsupervised methods is ARFlow, which gains competitive results to the successful supervised approach of PWC-Net. The success of ARFlow comes from its unique pipeline that performs two forward passes of original and augmented image pairs and forces the consistency of the transformed estimate of the original image-pair with the computed flow of the augmented image-pair. ARFlow uses the architecture PWC-Net as baseline architecture. However, there have been no studies that incorporate RAFT's architecture as a baseline architecture in ARFlow.

This thesis aims to integrate the 2-view unsupervised approach of ARFlow with RAFT's architecture. The final developed approach is trained in an unsupervised way with the benefit of using augmentation as a regularization while it has the advantage of a more robust architecture introduced by RAFT. The final architecture is evaluated using recent optic flow benchmarks. The final results of the unsupervised RAFT model has good cross-data generalization and achieve 3.7 % better results on Sintel clean test dataset compared to ARFlow (PWC-Net as a base network).

Acknowledgements

I would like to express my deep gratitude to Professor Andrés Bruhn and Azin Jahedi. I have been extremely lucky to have a professor and supervisor who cared so much about my work and give me so many valuable comments on the research. I am particularly grateful for the assistance given by Azin Jahedi, who helped me a lot during the course of thesis and for her infinite patience and always encouraging way of helping, by correcting my numerous mistakes. Her suggestions were invaluable and she should know that her support and help was worth more than I can express on paper.

Many Thanks!

Contents

1 Introduction

1.1 Motivation

Optical flow estimation is the estimation of the motion of every pixel in a sequence of images. This problem is an essential aspect of computer vision with many applications, such as driver assistance [JGBG20], image segmentation, video editing [BTS+15], and action recognition. In general, the optical flow estimation problem is: given the image sequence, the displacement vector for each pixel must be determined that points to where thae pixels of the first image can be found in another image.

Research into optical flow estimation has a long history. For many years, the main focus of previous studies has been to minimize energy function to estimate the optical flow. The pioneer flow estimation method was introduced by Horn and Schunck [HS81]. Horn and Schunck introduced brightness constancy and spatial smoothness assumptions to estimate the flow. Many variations of these variational-based approaches have emerged, and these performed well. However, in recent years, deep learning-based optical flow methods with better accuracy and better computational speed have emerged and gradually replaced the variational-based approaches.

To date, several studies have incorporated deep learning models to estimate the optical flow. However, most of the architectures are trained and tested on the same data set. This has led to a decrease in model generalization. These models often learn what it is trained on. Lack of generalization of models can also be seen as over-fitting. One of the most significant challenges is that most of the architectures are supervised learning-based and require a large amount of labeled data. Getting these large labeled datasets for real-world videos requires an expensive data acquisition and a laborious tasks to find the ground truth flow vectors. Ground-truth motion labels in real-world videos are generally hard to annotate [JGW+17]. Therefore, many studies have begun to estimate the optical flow using unsupervised deep learning architectures that do not require ground truth.

One of the state-of-the-art optical flow estimation methods is Recurrent All Pairs Field Transforms(RAFT) [TD20]. This method is trained in a supervised way and gains the most accurate

results on recent benchmarks dataset like MPI-Sintel [BWSB12] and KITTI-2015 [MHG15]. RAFT exhibits good cross-data generalization, but it is a supervised learning-based architecture requiring many annotated flow data.

On the other hand, one of the most successful unsupervised methods in the field is ARflow [LZH+20] which gains competitive results to the successful supervised approach of PWC-Net [SYLK18]. The success of ARFlow comes from its particular pipeline that performs two forward passes of original and augmented image pairs and forces the consistency of the transformed estimate of the original image-pair with the computed flow of the augmented image-pair. This approach uses the architecture of PWC-Net as a base architecture. Even though ARFlow produces good results on benchmark datasets, it is specifically trained and finetuned for the specific dataset. Also, ARFlow architectures inherently embody the disadvantages of PWC-Net.

The goal of this thesis is to integrate the 2-view unsupervised approach of ARFlow with RAFT's architecture, which means that the final method is trained in an unsupervised manner with the benefit of using augmentation as a regularization while it has the advantage of a more powerful architecture introduced by RAFT. The developed approaches shall be evaluated using recent optical flow benchmarks.

The academic literature on optical flow has revealed the emergence of several themes. It is essential to know the development of researches in this field. The following section describes some of the related works, progress, and pitfalls in optical flow estimation.

1.2 Related Work

Traditionally, optical flow is expressed as a variational optimization problem to find pixel correspondences between two consecutive video frames [HS81]. With the modern advancement of deep convolutional neural networks (CNNs), deep learning-based methods have been adopted to learn optical flow estimation, where the deep networks are trained to estimate the optical flow.

Since most deep learning-based networks are trained using the ground truth flow, it requires a massive amount of annotated data. Acquiring ground-truth motion labels in real-world scenarios is generally hard and laborious to annotate. One alternative is to use synthetic datasets. Regrettably, most available optical flow datasets are from synthetic movies or synthetically generated. Unfortunately, there usually exists a large domain gap between the distribution of synthetic images and natural scenes. This resulted in the emergence of unsupervised models where deep networks are trained without using the ground truth; thereby, these methods can be

directly used on real-world videos without being explicitly annotated. In the following section, some work related to unsupervised learning is described.

Ren *et al.* [RYN+17] proposed the first unsupervised deep learning for optical flow estimation. They attempted to reconstruct every pixel in the first image by warping the second image by the estimated flow . The network used FlowNet [DFI+15] structure to output the pixel-level translation estimates. These are given to the bilinear net to generate warping feature maps. Finally, the photometric error between the warped feature map from the source image and the target image is taken as the loss. However, the performance of this method had a large gap compared to the supervised method because it tried to reconstruct every pixel in the target image.

Due to the occlusions, it may happen that correspondence for the occluded pixels could not be found. The photometric error must be discarded for those pixels. To solve this problem, Wang *et al.* [WYY+18] proposed occlusion aware unsupervised learning of optical flow. This network consists of two copies of FlowNetS [DFI+15], where the forward image sequences are given as input to the first copy of the network and reversed sequence images are given to the second copy to output forward and backward flow, respectively. The backward optical flow is used to generate occlusion maps, and forward flow is used to warp the second image towards the first. They excluded photometric loss in the occluded regions and introduced edge-aware [GMAB17] smoothness loss to guide the estimation at the occluded regions. The final evaluation results were better than the previously published unsupervised methods.

UnFlow [MHR18] is another end-to-end deep convolutional neural network-based model that outperformed many of the traditional flow methods and earlier unsupervised methods. UnFlow computed bi-directional optical flow in both forward and backward directions. The model consisted of two forward passes where input images are exchanged in the second forward pass to compute backward flow. UnFlow's loss function leverages bidirectional flow to explicitly reason about occlusion (Hur and Roth 2017) and uses the census transform to increase robustness on real images.

Ddflow [LKLX19] and Selflow [LLKX19] estimated optical flow with occlusions in an unsupervised manner using deep learning-based models. The training of these models is in two stages. In the first stage, a teacher model is trained to predict flow on the original data. Liu *et al.* created occlusion samples offline by randomly crop or masking out the image and used these artificial samples from the teacher model to update the student model. However, these methods are not end-end-end (because of frozen teacher and offline occlusion maps) and work only for partial occlusions.

1.3 Thesis Structure

The remainder of the thesis is organized as follows: Chapter 2 explains the background necessary for implementing the unsupervised RAFT. This chapter describes deep learning-based state-of-the-art architectures useful for optical flow estimation and the current implementations. Then, Chapter 3 illustrates the concepts and the workflow behind the unsupervised RAFT. The Chapter 4 explains the datasets used to train and evaluate the model. Afterward, the setup and implementation details are mentioned in Chapter 5. Then, Chapter 6 reports the results of the current work. At the end the Chapter 7 summarizes the overall thesis work.

2 Background

Optical flow is the motion field between two consecutive frames of an image sequence. An image is composed of many pixels, and when we move the camera around or capture images at series of time intervals, there is some motion change happening in those pixels. We want to figure out how that motions at the individual pixel level are happening. The problem of optic flow can be described as Figure 2.1. A image (I) intensity can be expressed as function of space $(x, y) \in \Omega$ and time $t \in \mathbb{R}_0^+$, where Ω represents image domain. In some cases we represent coordinates (x, y) as a vector $\mathbf{x} = (x, y) \in \Omega$. Imagine a pixel at (x, y) and time t in image I flows to $(x + u, y + v)$ in next image at time $t + 1$, where $(u, v) \in \mathbb{R}$. The optical flow vector is therefore $\mathbf{f(x)} = (u(\mathbf{x}), v(\mathbf{x}))^\top \in \mathbb{R}^2$. The optical flow problem is finding these motion vectors for every pixel coordinates **x** of the image, thereby understanding where the pixels are moving to from time t to t+1. In some cases flow **f** is denoted as U_{12} and throughout this thesis, both formulations will be used interchangeably.

Finding the flow field (u, v) for every pixel is a complex problem. Let us try to solve for the motion vectors. Let us assume pixel intensities of an object are constant between two frames. When the motion is small, this can be represented as,

$$I(x, y, t) = I(x + u, y + v, t + 1) \tag{2.1}$$

By expanding the terms by Taylor series, omitting the higher-order terms, and equating the equation 2.2, we can derive the following optical flow equation.

$$\frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \frac{\partial I}{\partial t} = 0 \tag{2.2}$$

We cannot directly solve the optical flow equation for u and v because there are two unknowns (u, v) in one equation. Therefore, in the past several decades, many researchers tried to provide other sets of equations of u, v to make it solvable. However, u and v might be large or small in real applications, spanning several to tens of pixels. Thus we can only get an approximation of the actual optical flow. For many years, variational methods have been developed in order to

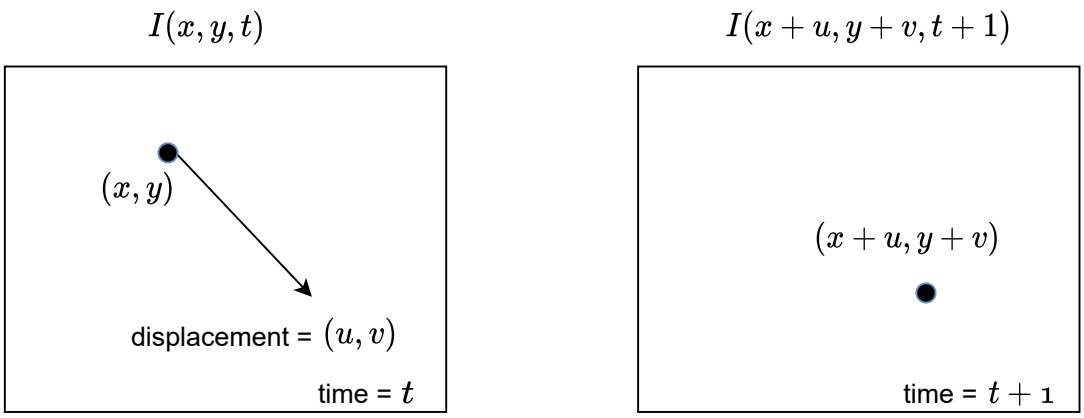

Figure 2.1: Explanation of a optical flow between two consecutive frames of an image sequence for a pixel coordinate located at (x, y).

solve optical flow problems. These methods adopted energy minimization approaches. However, optimizing a complex energy function is a tedious task and is computationally expensive.

Many researchers tried to estimate optical flow through deep learning methods in the recent deep learning era and have gained significant results over the variational methods. Before going into the current approach, it is necessary to describe state-of-the-art deep learning-based architectures to estimate the optical flow. The following section explains PWC-Net (base architecture of ARFlow), RAFT, and ARFlow architectures which are required to understand and implement the current work.

2.1 PWC-Net

PWC-Net is a compact and efficient end-to-end supervised model for optic flow estimation. It uses three basic techniques: pyramidal feature construction, warping the features, and cost volume construction. Figure 2.2 illustrates the pipeline of PWC-Net architecture. Given the two sequences of images, PWC-Net constructs l layer image feature pyramids at different scales. It estimates flow at a coarse level and warps the second image features towards the first image features. Using the first image features, warped image features, and estimated flow, a cost volume is constructed storing the matching cost for associating the pixels in the first image to corresponding pixels at the second image. Warping helps in reducing the search range for cost volume computation. Features of the first image, cost volume, and upsampled flow are given to the CNN-based optic flow estimator, which outputs the flow at the lth level. Finally, estimated

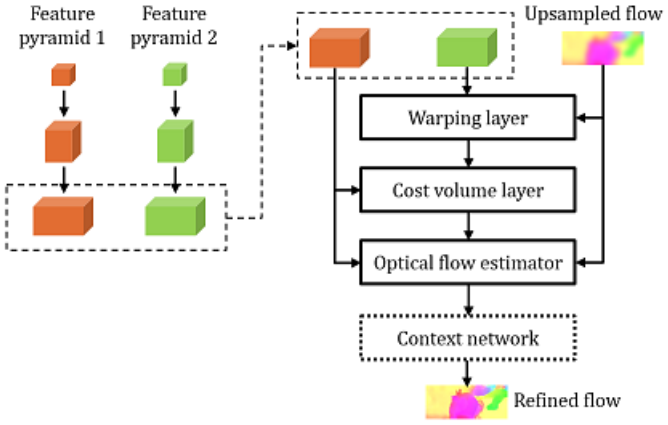

Figure 2.2: PWC-Net pipeline : Feature pyramid and refinement at one pyramid level by PWC-Net [SYLK18].

flow is processed by a feed-forward CNN-based context network which outputs the refined flow. A multi-scale training loss is used in the model. The training loss with learn-able parameter Θ of the network is defined as,

$$\mathcal{L}(\Theta) = \sum_{l=l_0}^{L} \alpha \sum_{\mathbf{x}} |\mathbf{f}_{\Theta}^{l}(\mathbf{x}) - \mathbf{f}_{GT}^{l}(\mathbf{x})|_2 + \gamma|\Theta|_2, \tag{2.3}$$

where $\mathbf{f}_{GT}^{l}$ denotes the ground truth flow re-scaled at lth level, $\mathbf{f}_{\Theta}^{l}$ denotes the flow predicted at lth pyramid level by the network and α represents weights for the each layer of the training loss. The network is trained on Sintel and KITTI-2015 benchmark datasets. A data specific fine-tuning also performed with training loss,

$$\mathcal{L}(\Theta) = \sum_{l=l_0}^{L} \alpha \sum_{\mathbf{x}} (|\mathbf{f}_{\Theta}^{l}(\mathbf{x}) - \mathbf{f}_{GT}^{l}(\mathbf{x})| + \epsilon)^q + \gamma|\Theta|_2, \tag{2.4}$$

where ϵ is a small constant and q penalizes the outliers. During fine-tuning q < 1 is used to give less penalty to outliers and $\ell 1$ distance is used between predicted and ground-truth flow at each scale.

PWC-Net consists of many important stages as illustrated in Figure 2.2. The main components of PWC-Net are explained below in subsections.

Feature Pyramid Extractor

Feature selection or extraction from the images is a common and important task in any machine learning task. Recently Convolutional Neural Networks (CNN) has become a trendy tool for

image processing and classification, which can automatically extract features better than other simple feature extraction techniques.

Given two Images I_1 and I_2, PWC-Net generates features at L-levels of the pyramid. Zeroth level $\mathbf{c}_t^0 = I_t$ is raw image. Convolutional filters are used to generate down-sampled features at the next upper level $\mathbf{c}_t^l$. PWC-Net uses six-level feature pyramids.

The idea behind feature pyramidal processing is to construct features at different scales. As small as the image size, the motion is much smaller. It is easier to estimate the optic flow at a smaller scale because a smaller search space is enough to find the correspondence. At a small scale, the network can get the global context in order to do the matching. PWC-Net builds an end-to-end CNN network to learn the feature for matching at different pyramid levels.

Warping Layer

Warping is a general technique often used in the pipeline of optical flow estimation. The idea behind backward warping is given the first image, second image, and corresponding flow vectors from every pixel in the first image to the second image; we want to warp the second image towards the first image and try to get something close to the first image. The motion between the warped image and the first image is much smaller now, and it is easier to estimate the optical flow with a small search range for finding correspondence. Backward warping technique with bilinear interpolation for a specific pixel is depicted in Figure 2.3,

For a pixel coordinate in the first image $I(x, y, t)$, we are looking for the appropriate pixel in the second image $I(x + u, y + v, t + 1)$ should be warped to the first image pixel coordinate location. Assuming the flow (u, v) are given for each pixel, we can find the corresponding pixel in the second image by adding the flows to the first image coordinates. Since the flow value will be at sub-pixel precision, after adding flows to the first image, interpolation is used to get the corresponding pixel value. PWC-Net uses the bilinear interpolation technique to find the value and copy that value to the first image pixel location.

PWC-Net uses backward warping with a bilinear interpolation technique in its pipeline. After constructing the features at L-levels of pyramid, at the l^{th} level, it warps the second image features towards the first using flow from the $l + 1^{th}$ level. Since the flow predicted at $l + 1^{th}$ level is at a different scale, upsampling $(\times 2)$ of flow is done to match to the l^{th} level:

$$\mathbf{c}_w^l(\mathbf{x}) = \mathbf{c}_2^l(\mathbf{x} + \mathrm{up}_2(\mathbf{f}^{l+1})(\mathbf{x})), \tag{2.5}$$

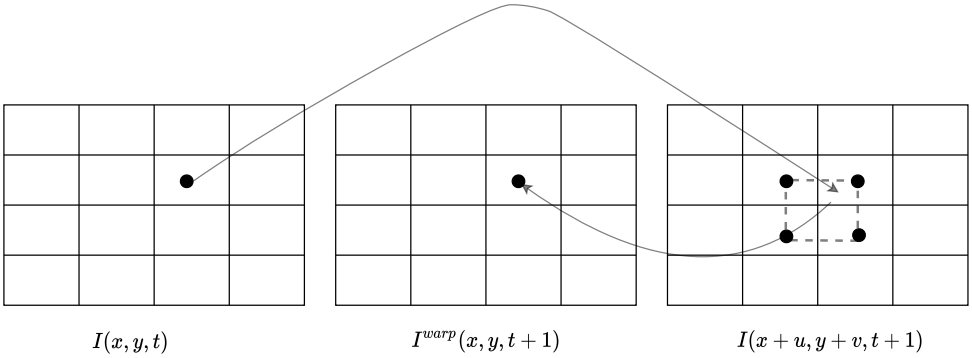

Figure 2.3: Illustration of basic backward warping technique with bilinear interpolation for a specific pixel.

where **x** denotes the pixel index. The second image feature at level l, warped image features at level l and flow at level $l+1$ is denoted as $\mathbf{c}_w^l$, $\mathbf{c}_w^l$ and $\mathbf{f}^{l+1}$ respectively. up_2 denotes upsampling by a factor of 2. For the top-level (original image size) $\text{up}_2(\mathbf{f}^{l+1})$ is set to 0.

Cost Volume Layer

Cost volume stores the matching cost for associating the pixel with its corresponding pixels in the following image. To find visual correspondence and solve computer vision tasks such as optical flow, stereo matching, Hosni *et al.* [HRB+12] proposes a generic framework by constructing cost volume and performing operations on it.

PWC-Net constructs cost volume by using the features that are extracted from the feature extraction stage. Matching cost is defined as the correlation between two features of the first and warped features of the second image:

$$\mathbf{cv}^l(\mathbf{x}_1, \mathbf{x}_2) = \frac{1}{N}(\mathbf{c}_1^l(\mathbf{x}_1))^\top \mathbf{c}_w^l(\mathbf{x}_2), \tag{2.6}$$

where T refers to transpose operation, $\mathbf{c}_1^l$ and $\mathbf{c}_w^l$ denotes the first image and warped image features respectively. N refers to the length of the column vector $\mathbf{c}_1^l$. Instead of computing full cost volume, since the motion is small between warped second image features and first image features at the top level of pyramids, only partial cost volume is computed with a range of d pixels. A small range d at the top level is sufficient to cover most of the pixels at the full resolution. A 3D Cost volume is constructed by matching every feature vector to d neighbor feature vectors of warped second image at each pyramid level. A small example of cost volume construction for a pixel is depicted in Figure 2.4. The dimension of cost volume is $d^2 * H^l * W^l$ where H and W are height and width of l_{th} pyramid level.

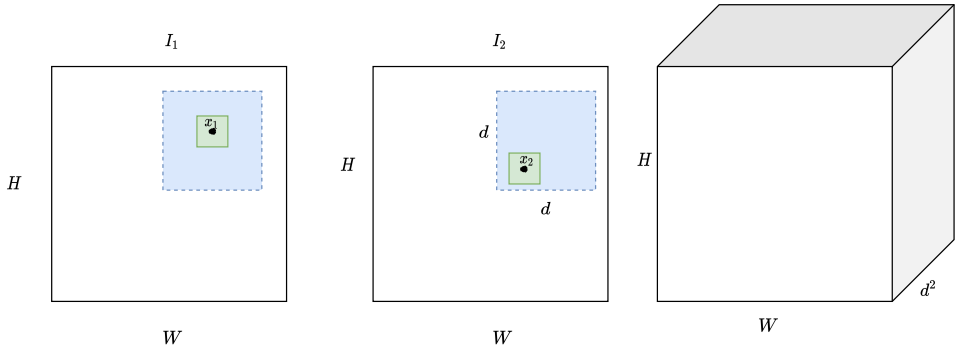

Figure 2.4: Cost volume construction for a single pixel in a $H * W$ image with neighborhood range d.

Optical Flow Estimator

Optic Flow estimator is a multi-layer CNN network. It takes cost volume, features of the first image, and the up-sampled optic flow as input and outputs the flow $\mathbf{f}^l$ at l^{th} pyramid layer. The weights of the CNN network are not shared between each pyramid level of the optical flow estimator. Flow estimator consists of a Leaky Relu activation that follows each convolutional layer except for the output layer, predicting the optical flow. The number of channels at each convolutional layer is kept fixed.

Context network

Context Network is for post processing the flow. Its inputs are estimated flow and features of the second last layer. It outputs a refined flow. Context network is again a CNN of 7 layers and each layer will have different dilation constant. Large dilation constant enlarges the receptive field of each output.

PWC-Net achieved good results on benchmarks like Sintel and KITTI-2015 datasets. PWC-Net showed that it could recover sharp motion boundaries despite large motion, storing shadows. However, flow prediction was inaccurate when small objects moved quickly. This is because of warping at a lower resolution; small objects vanishes when warped at smaller scale. Also, cross dataset evaluation results were not up to the mark compared to the state-of-the-art techniques signifying overfit to the dataset which it is trained on. The next section describes a recently published optical flow estimator. This model is robust and gains good cross data generalization results.

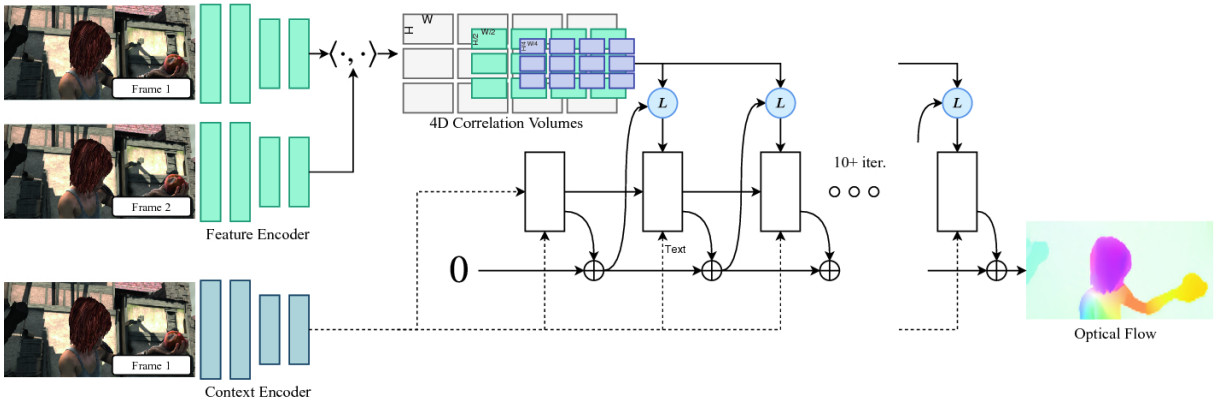

Figure 2.5: Overview of RAFT pipeline. Image source:[TD20].

2.2 Recurrent All-Pairs Field Transforms (RAFT)

RAFT is a state-of-the-art, deep learning network-based architecture to estimate the optical flow. It is trained in a supervised fashion; it requires ground truth optical flow to train the network. RAFT has strong cross-data generalization.

The pipeline of RAFT is showed in Figure 2.5. Given two image frames, RAFT extracts feature from the two frames using a feature encoder. Then RAFT computes visual similarities between all pairs of pixels by constructing a full correlation 4D volume using the image features. It has an iterative update operator where it estimates a sequence of flow $\mathbf{f}_1$ to $\mathbf{f}_n$ by performing lookup in the visual similarity matrix (4D volume), and it outputs the flow to refine the current estimate of the optical flow. All stages in RAFT are differentiable, and these components can be composed into an end-to-end trainable architecture.

The overall RAFT architecture can be distilled down to the following three stages.

Feature/Context Encoder

The feature encoder extracts features from each of the image sequences. The architecture is depicted in Figure 2.7. Feature encoder consists of convolutional layers fallowed by stacks of residual blocks and finally a convolution layer. Context encoder is similar to feature encoder, except this only extract features from the first image. We extract features using a context encoder on the first image, and later these features will be directly given to the update stage as input. Context encoder helps in spatial information to be better aggregated within the motion boundaries.

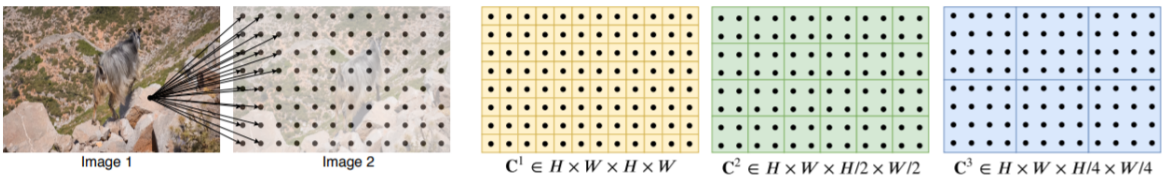

Figure 2.6: Correlation Volume: depicted as 2D slices of 4D Volume. Image source:[TD20].

Correlation Volume

A correlation matrix stores visual similarity between every pixel i in image 1 and every pixel j in image 2. Visual similarity is computed using dot product between feature vectors. The correlation volume C can be computed as:

$$\mathbf{C}(g_\theta(I_1), g_\theta(I_2)) \in \mathbb{R}^{H \times W \times H \times W}, \tag{2.7}$$

where I_1 and I_2 are input frame 1 and frame 2 and g_θ is the feature encoder. A single value in a 4D volume is computed as:

$$\mathbf{C}_{ijkl} = \sum_h g_\theta(I_1)_{ijh} \cdot g_\theta(I_2)_{klh}, \tag{2.8}$$

where i, j, k, l are indices of 4D correlation matrix. RAFT computes a correlation value for all pairs of feature vectors and generates 4D volume as depicted in Figure 2.6. Then, it applies 2×2 average pooling to last two dimensions of 4D volume to get correlation volume at different scales. After the correlation matrix is computed, it is fixed during the rest of the iterative update algorithm.

Iterative Update Operator

Iterative update operator estimates a sequence of flow $\mathbf{f}_1$ to $\mathbf{f}_n$. This iterative operator resembles the steps of a first-order optimization algorithm. Based on the optical flow estimate, the update operator looks up on 4D correlation volume and iteratively refines the flow. Given a current estimate of the optical flow, it adds the corresponding flow to each pixel of image 1. Then it defines the local grid around that point and performs a look up in the correlation matrix in those positions for each 4D channel. It concatenates these values for all the pixels resulting in correlation features. Using these correlation features, previously estimated flow and context features from image 1, update block updates the latent hidden state. These hidden states are then used to update the predicted flow. Update block is based on a gated activation unit.

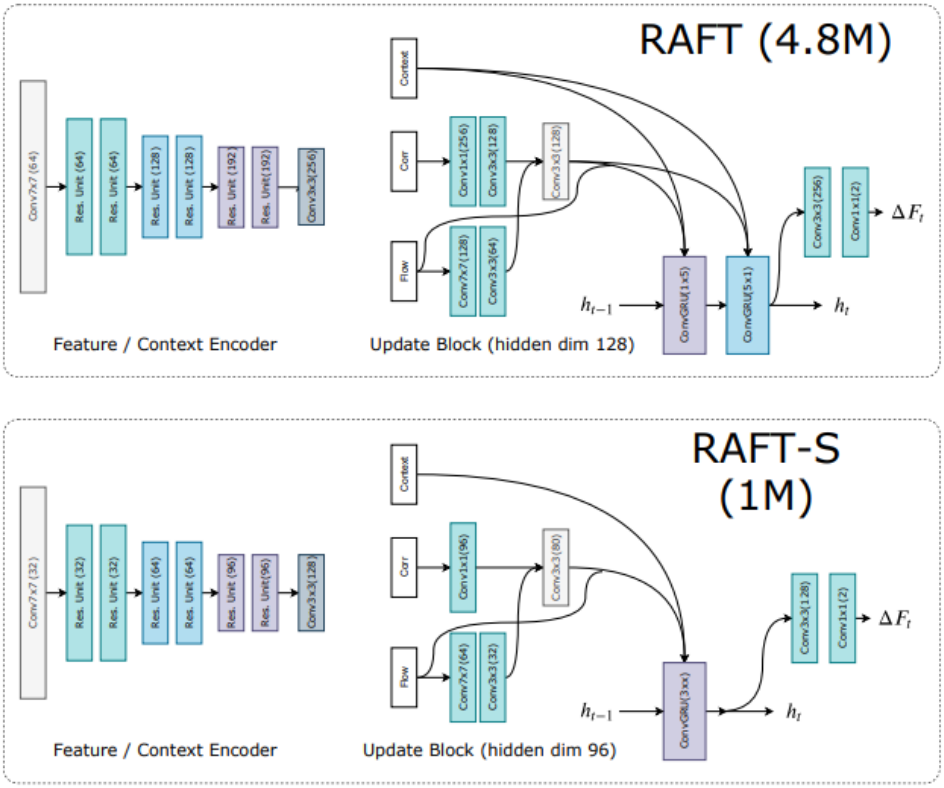

Figure 2.7: RAFT feature/context encoder and update block network architecture for large (RAFT) and small (RAFT-S) network. Image source: [TD20].

Loss function

The final loss function is $\ell 1$ distance between the ground truth flow and predicted sequence of flow :

$$\mathcal{L} = \sum_{i=1}^{N} \gamma^{N-i} \parallel \mathbf{f}_{gt} - \mathbf{f}_i \parallel_1, \tag{2.9}$$

where γ refers to exponentially increased weight, indicating more importance to output at longer iterations. γ is set to 0.8 in RAFT. There is also a miniature version of RAFT (RAFT-S), as shown in Figure 2.7 which has lower parameters than the original RAFT.

RAFT pre-trains the model on Flying Chairs and Flying Things datasets. Model results are tested on Sintel and KITTI-2015 data sets. RAFT shows good generalization across the data sets and performs better than PWC-NET (Section 2.1). RAFT is a state-of-the-art technique and achieves the lowest test errors on Sintel and KITTI-15. The main disadvantage of RAFT is that this model is supervised-based and requires a large amount of labeled data to train. Getting the labeled data is a tedious task in reality, and often it is a difficult process. One alternative is to use synthetic data

sets. Unfortunately, there usually exists a large domain gap between the distribution of synthetic images and natural scenes. So, there is a need for models which do not use ground truth for optical flow estimation. The following section describes a recent state-of-the-art unsupervised approach to estimate the optical flow.

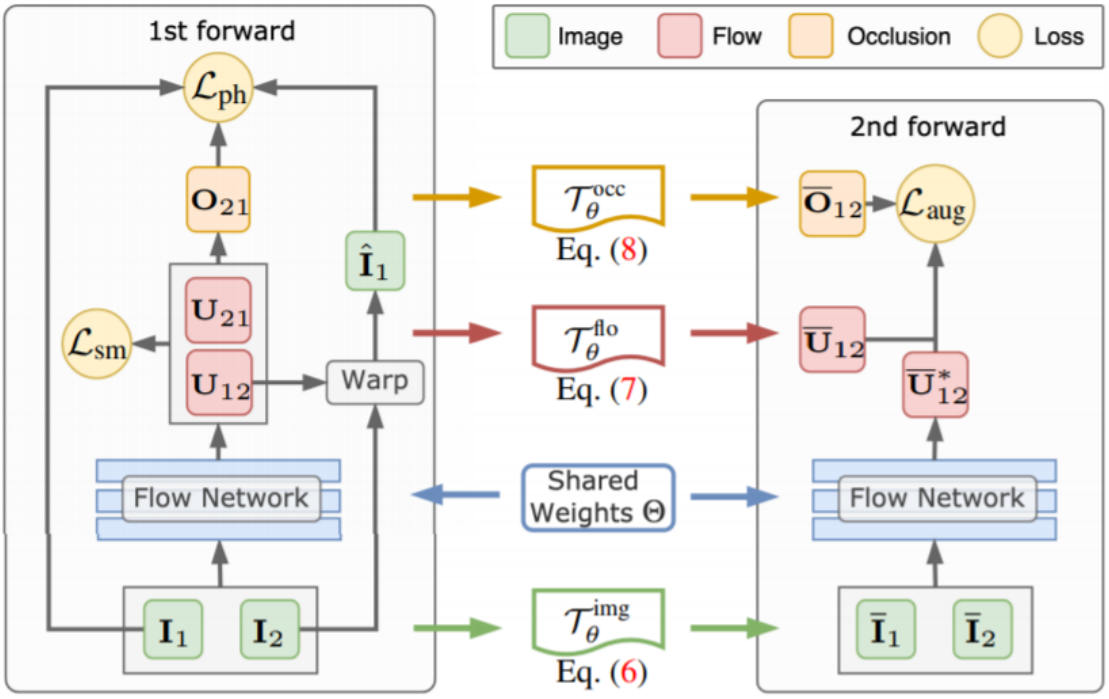

Figure 2.8: Schematic overview of ARFlow pipeline. Image source:[LZH+20].

2.3 ARFlow

ARFlow [LZH+20] is a recently published unsupervised based architecture to estimate the optical flow. For the supervision of a deep model, ARFlow tries to reconstruct the first image from the estimated flow and compare the original image and reconstructed image. The success of ARFlow comes from its unique pipeline that performs two forward passes of original and augmented image pairs and forces the consistency of the transformed estimate of the original image-pair with the computed flow of the augmented image-pair. In the second forward pass, it transforms the original image through augmentation, and also it transforms the corresponding predicted optical flow from first pass. The transformed predicted flow will be used for supervision of the second pass. ARFlow uses PWC-Net as a base network for flow prediction.

ARFlow pipeline is shown in Figure 2.8. In the first forward pass of the network, input images I_1 and I_2 are passed to PWC-Net, predicting the forward flow U_{12}. Using the second image and current estimate of the forward flow, it warps the second image towards the first image as described in Figure 2.3 . The warped image is denoted as $\hat{I}_1$ and formulated as:

$$\hat{I}_1(\mathbf{x}) = I_2(\mathbf{x} + U_{12}(\mathbf{x})), \tag{2.10}$$

where **x** denotes pixel coordinates. Bilinear sampling is used for continuous coordinates because flow is at sub-pixel precision. Since we are not using the ground truth in unsupervised learning, the original image and warped image are compared through photometric loss $\mathcal{L}_{ph}$ for training the network in the first forward pass. Photometric loss is defined as pixel-wise similarity between the first and warped second image:

$$\mathcal{L}_{ph} \sim \sum_{\mathbf{x}} \rho(\hat{I}(\Theta), I), \tag{2.11}$$

where ρ is pixel-wise similarity measure and **x** is the pixel index.

Photometric loss is violated at the pixels where pixels move out of the scene or when the pixels get occluded. So ARFlow excludes the photometric loss where there are occlusions. The occlusions are determined from forward-backward check, where backward flow U_{21} is computed by reversing the first and second image and passing through the flow network. Photometric loss is ambiguous for the texture-less region or where there are repetitive patterns. ARFlow includes smoothness loss $\mathcal{L}_{sm}$ in the pipeline to handle this situation. Smoothness Loss is derived from the literature and defined as:

$$\mathcal{L}_{sm} \sim \sum_{d \in x,y} \sum_{\mathbf{x}} \|\nabla_d U_{12}\|_1 e^{-\|\nabla_d \mathcal{I}\|}, \tag{2.12}$$

where U_{12} represents the forward flow, ∇ represents gradient operator. Smoothness loss constraints the flow predictions similar to that of its neighbours in x and y directions when there is no significant information available at a the region.

To make the model more robust, it does a second forward pass with augmented image pairs and forces the consistency of the transformed estimate of the original image pair with the computed flow of the augmented image pair. Input images are transformed into augmented images $\bar{I}_1$ and $\bar{I}_2$. Some of the transformations applied on input images change the pixel locations, so the corresponding estimated flow should also be transformed consistently to make flow to be reliable supervision signal. The predicted flow on transformed images is compared with the transformed predicted flow of the first forward pass through augmentation loss $\mathcal{L}_{aug}$. Augmentation loss is defined as:

$$\mathcal{L}_{aug} \sim \sum_{\mathbf{x}} (|S(\overline{U}_{12}(\mathbf{x})) - \overset{*}{\overline{U}}_{12}(\mathbf{x})| + \epsilon)^q, \tag{2.13}$$

where ϵ refers to a small constant and S refers to stop gradient. A small q is chosen in order to give less penalty to the outliers. ARFlow tries to minimize the total loss:

$$\mathcal{L}_{all} = \mathcal{L}_{ph}(U_{12}) + \lambda_1 \mathcal{L}_{sm}(U_{12}) + \lambda_2 \mathcal{L}_{aug}(S(\overline{U}_{12}), \overset{*}{\overline{U}}_{12})), \tag{2.14}$$

where λ_1 and λ_2 are regularization parameters. By setting a small value of λ_2, it prevents the dominance of augmentation distribution.

ARFlow uses PWC-Net as base architecture. ARFlow trained and fine-tuned individually on Sintel and KITTI-15 benchmarks and evaluated on these, respectively. ARFlow achieves significant state-of-the-art results on Sintel and KITTI-15 datasets. However, model has inherent disadvantages of PWC-Net. It does not provide results for cross dataset evaluations as in Section 2.2 (training on Chairs and Flying-Things and evaluating on Sintel). The overcome this problem, the next chapter tries to explain the current thesis work by modifying the RAFT network and using RAFT network as a base model in ARFlow.

3 Unsupervised RAFT

As described in the earlier Section 2.2, RAFT is one of the state-of-the-art optical flow estimation architecture which achieves good cross data generalization and outperforms all the published methods. However, RAFT is supervised learning-based and requires a large amount of annotated data. On the contrary, ARFlow is a state-of-the-art unsupervised method but lacks cross-data generalization and has inherent problems from its base network PWC-Net. The goal of the thesis is to convert RAFT into an unsupervised approach by incorporating the ARFlow network. This can be achieved by replacing the PWC-Net flow network in ARFlow architecture with RAFT.

3.1 Pipeline

The pipeline of unsupervised RAFT is shown in Figure 3.1. The pipeline is similar to ARFlow, consisting of two forward passes. In the first forward pass, two image sequences I_1 and I_2 are given to the RAFT network as input. RAFT network outputs the forward flow, denoted by U_{12}. Then the second image is warped (Figure 2.3) towards the first using the forward flow. The photometric loss $\mathcal{L}_{ph}$ as mentioned in Equation(3.9) was calculated using the warped second image $\hat{I}_1$ and the first image.

The photometric loss at occluded regions was discarded. The occlusion masks O_{21} are calculated using the standard forward-backward check. In particular, the photometric loss is invalid at texture-less regions or where there were repetitive patterns. To overcome this, smoothness regularization was incorporated using smoothness loss $\mathcal{L}_{sm}$ as mentioned in Equation(3.20).

A special second forward pass was used to generate more challenging samples, model the network on challenging scenarios and reduce overfitting. Input images were transformed using the augmentation techniques. Transformed images are denotes as $\bar{I}_1$ and $\bar{I}_2$ and transformation operator with parameter θ is denoted by $\mathcal{T}_\theta^{img}$. The transformed images were passed through the same RAFT in the second forward pass to predict the flow $\overset{*}{\overline{U}}_{12}$.

For supervision of second forward pass, the flow predicted from first forward pass U_{12} were transformed consistently using the corresponding transformation flow operator $\mathcal{T}_\theta^{flo}$. The transformed

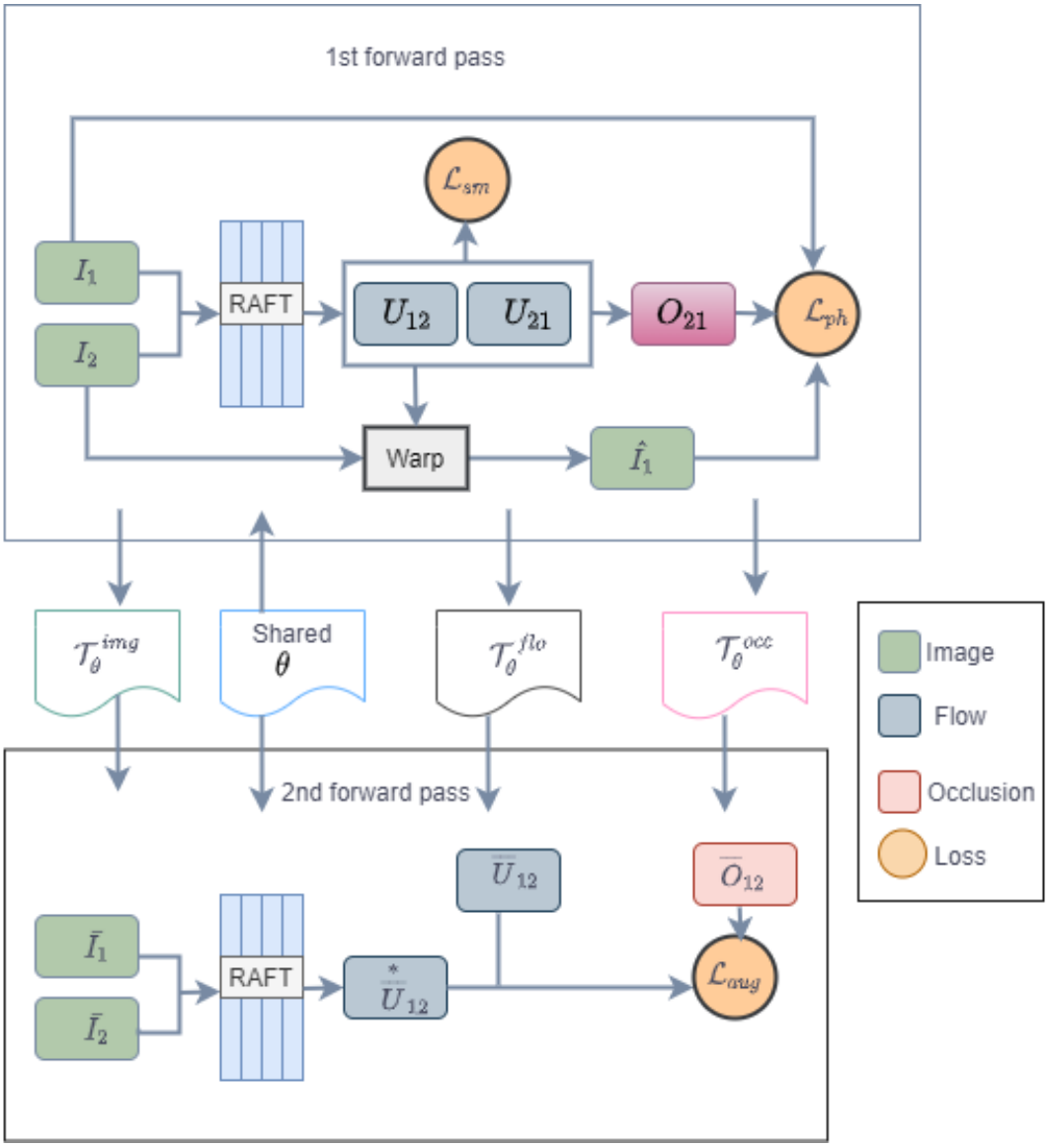

Figure 3.1: Unsupervised RAFT pipeline. Image adopted from [LZH+20].

flow $\overline{U}_{12}$ from the first forward pass and the predicted flow $\overline{U}^*_{12}$ from second forward pass were compared through the augmentation loss $\mathcal{L}_{aug}$ as mentioned in Equation(3.22). Augmentation loss was also excluded at the occluded regions. The occlusions at second forward pass were calculated by consistently transforming the occlusions from first forward pass. The transformed occlusions are denoted as $\overline{O}_{12}$.

Then, the network was trained with the objective of optimizing the weights of the network by minimizing the total loss defined as in Equation(3.8). The following sections describes

the unsupervised RAFT thesis work: the RAFT network for flow prediction, loss function for optimization of network parameters, and training with augmentation procedures.

3.2 RAFT Network

RAFT network architecture is used to make flow prediction. RAFT architecture is briefly described in Section 2.2. In both the first and second forward pass, the original image sequence and the augmented image sequences are passed to the feature encoder, which extracts the features from the images. Using these extracted features correlations volume is constructed. Then, with the help of the update operator, iteratively flow is updated by performing a look-up in correlation volume. In this section, the main components of the RAFT flow network are described.

3.2.1 Feature/Context Encoder

Feature encoder maps the input images to output feature maps. The architecture of feature encoder is depicted in Figure 3.2. It consists of series of convolutional layers and residual unit blocks. Feature network takes the RGB image as input and outputs the features at 1/8 resolution with 256 feature channels:

$$g_\theta : \mathbb{R}^{H \times W \times 3} \mapsto \mathbb{R}^{H/8 \times W/8 \times D}, \tag{3.1}$$

where feature mapper function g is parameterized by θ, H, W are input image sizes and D denotes number of feature channels.

The feature encoder consists of 6 residual blocks. A residual block is nothing but a building block of a ResNet [HZRS16]. A residual block is the activation of a layer is fast-forwarded to a deeper layer in the network. A single residual unit is depicted in Figure 3.3. Residual blocks help to solve vanishing or exploding gradients problems and also helps to ease the training of deep networks. In the feature encoder, instance normalization is used where it normalizes each batch independently.

A smaller version of the RAFT network is called RAFT-S. RAFT-S is a smaller version of RAFT network with few parameters. In feature encoders of RAFT-S, as depicted in right Figure 3.2, the number of feature channels are reduced to reduce the model size and residual units are replaced with bottleneck residual units.

Context encoder is very similar to feature encoder, and the only change is that this uses batch normalization in the encoder network. Batch normalization normalizes all images across the

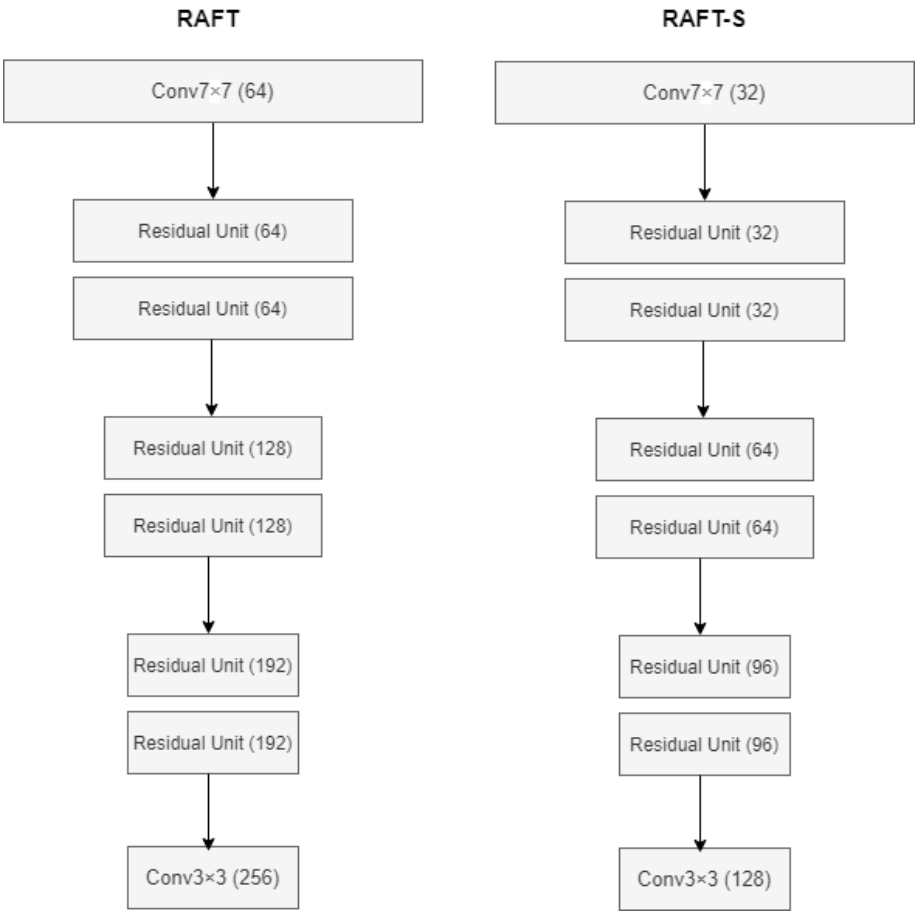

Figure 3.2: RAFT feature/context encoder consisting of convolutional and residual blocks. Image adopted from [TD20]. Left: Feature encoder of RAFT and Right:Feature encoder of RAFT-S (small version).

batch and spatial locations. Only the first input image is given to context encoder as input, where it extracts the features.

3.2.2 Correlation Volume Computation

Correlation volume stores the visual similarity between the pixels. A full correlation volume is constructed between all pairs. A full correlation volume is formed by taking the dot product between all pairs of feature vectors. We know that a feature encoder g_θ maps input image to the feature maps. Given the two input images I_1 and I_2, image feature maps are extracted using the feature encoder. By using these image feature maps $\mathbf{g}_\theta(I_1) \in \mathbb{R}^{H \times W \times D}$ and $\mathbf{g}_\theta(I_2) \in \mathbb{R}^{H \times W \times D}$, a full 4D correlation volume C is computed by taking the dot product between all pairs of

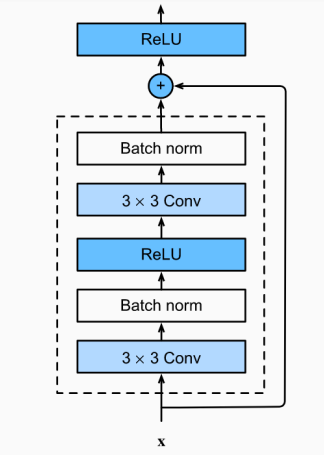

Figure 3.3: An example of single residual unit with batch normalization.

the feature vectors. 4D correlation volume is depicted in Figure 3.4 and a single entry in 4D correlation volume computed as:

$$\mathbf{C}_{ijkl} = \sum_{h} \mathbf{g}_\theta(I_1)_{ijh} \cdot \mathbf{g}_\theta(I_2)_{klh} \tag{3.2}$$

After construction of Correlation volume, A 4-layer correlation pyramid $\mathbf{C}^1, \mathbf{C}^2, \mathbf{C}^3, \mathbf{C}^4$ are constructed by pooling the last two dimensions of the original correlation volume with kernel sizes 1,2,4, and 8. One level of the down-sampling operation is depicted in Figure 3.5 . Correlation volume $\mathbf{C}^k$ has $H \times W \times H/2^k \times W/2^k$ dimensions. The original correlation volume is constructed at different scales so that volume gives information about both large and small displacements. The first two dimensions are maintained as is to recover the motions of small, fast-moving objects.

Once the full 4D correlation volume is computed, it is fixed during the rest of the iterative stages.

3.2.3 Lookup Operator

A lookup operator generates correlation feature maps by indexing from the correlation pyramid. Given a current estimate of the flow, we add the flow to each pixel $\mathbf{x}$ of the first image I_1 to its estimated location $\mathbf{x}'$ in the second image I_2. Now, a local grid is defined around the location $\mathbf{x}'$. The local grid around pixel $\mathbf{x}'$ with radius r is defined as:

$$\mathcal{N}(\mathbf{x}')_r = \{\mathbf{x}' + dx | dx \in \mathbb{Z}^2, ||dx||_1 \leq r\} \tag{3.3}$$

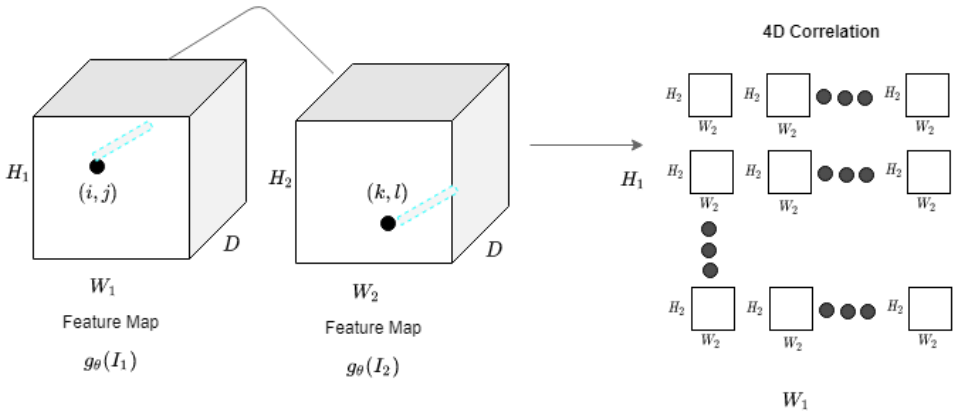

Figure 3.4: Illustration of 4D correlation volume computation from feature maps. (4D correlation volume is depicted as 2D slice)

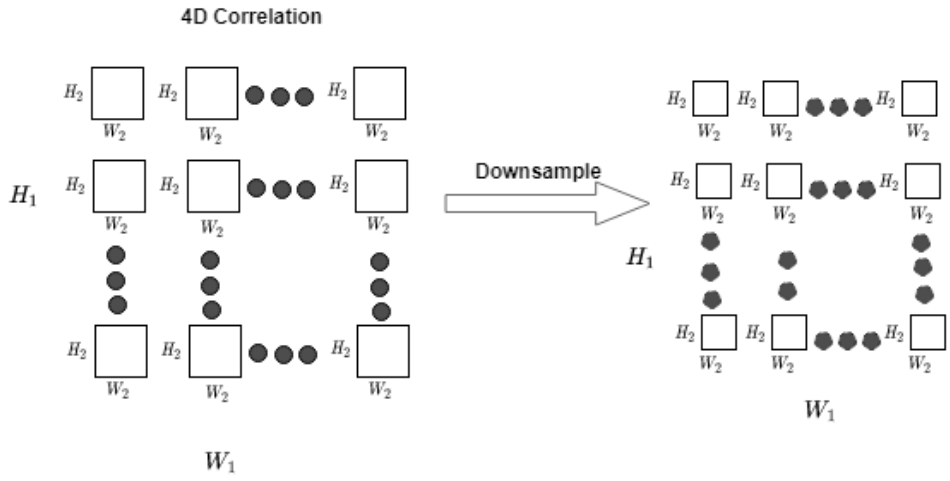

Figure 3.5: Correlation Pyramid construction by down-sampling the last two dimensions.

These neighborhood locations are used as an index to lookup in correlation volume as shown in Figure 3.6. Since $\mathcal{N}(\mathbf{x}')_r$ is a grid of real numbers, bilinear sampling is used if pixel lies in between the grids. These lookup operations are performed on all levels of the correlation pyramid, and the output 3d feature maps are concatenated as a single feature map.

3.2.4 Iterative Updates

A sequence of flows is estimated at each iteration of the update. An update operator takes the current estimated flow (initialized to 0 at the starting point), correlation features retrieved after lookup, and a hidden state as input. Update operator updates the flow and hidden state. These

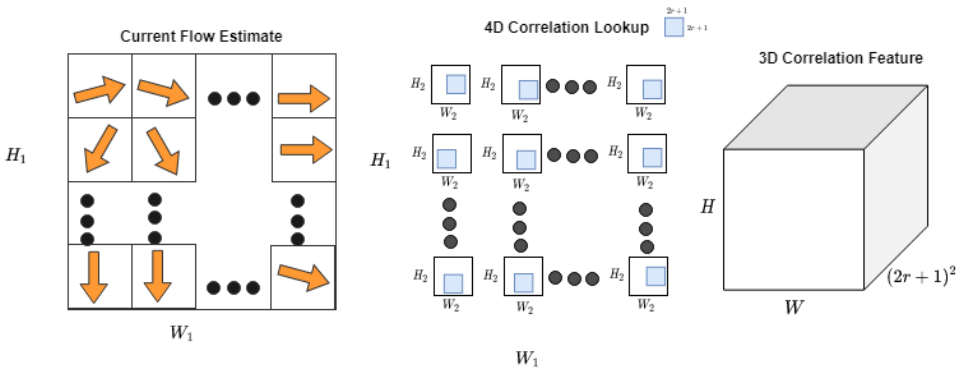

Figure 3.6: Correlation lookup on correlation volume using current estimates of the flow. Correlation lookup results in 3D correlation features.

update operator iterations are designed to mimic the steps of an optimization algorithm. The update operator is trained to perform updates so that the flow converges to a fixed point.

As mentioned in Section 3.2.3, an update operator performs a lookup in the correlation volume pyramids and constructs a correlation feature map given the current estimate of the flow. Convolutional layers then process these correlation features. Features from current estimates of the flow are also extracted using additional convolutional layers. Then, correlation features, flow features, and context features(features extracted from the first image using context encoder) are concatenated to form input features.

The main component of the update operator is a gated activation unit based on GRU cell. The fully connected layers are replaced with convolutions in GRU cell and the following operation are performed:

$$\mathbf{z}_t = \sigma(Conv_{3\times3}([\mathbf{h}_{t-1}, \mathbf{x}_t], \mathbf{W}_z)) \tag{3.4}$$

$$\mathbf{r}_t = \sigma(Conv_{3\times3}([\mathbf{h}_{t-1}, \mathbf{x}_t], \mathbf{W}_r)) \tag{3.5}$$

$$\tilde{\mathbf{h}_t} = tanh(Conv_{3\times3}([\mathbf{r}_t \bigodot \mathbf{h}_{t-1}, \mathbf{x}_t], \mathbf{W}_h])) \tag{3.6}$$

$$\mathbf{h}_t = (1 - \mathbf{z}_t) \bigodot \mathbf{h}_{t-1} + \mathbf{z}_t \bigodot$$

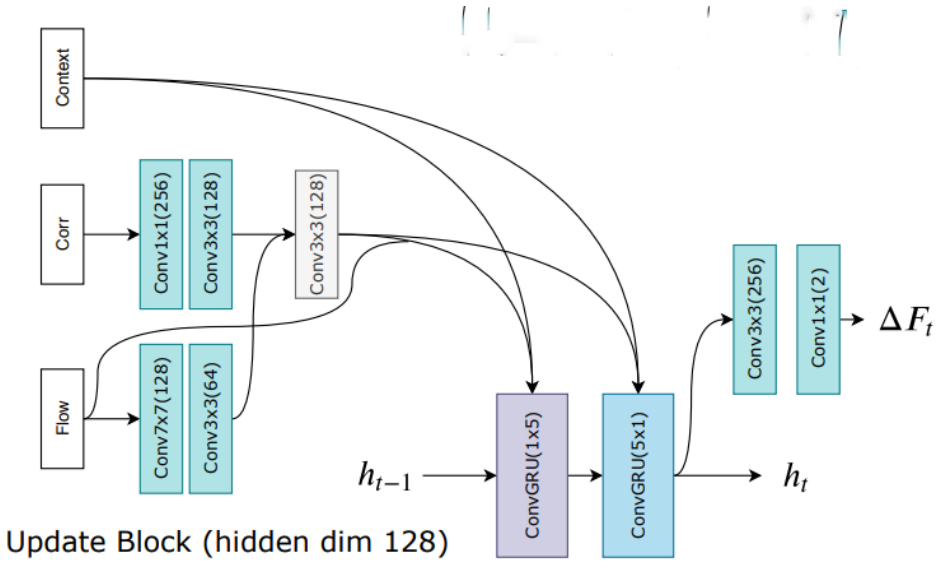

Figure 3.7: Network architecture of update block. Image source: [TD20].

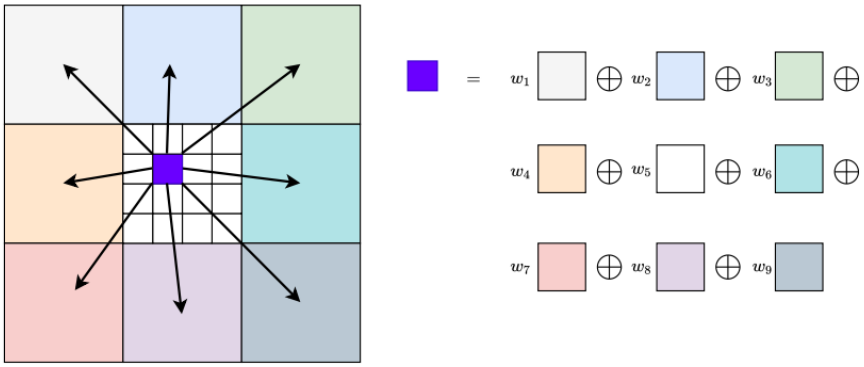

Figure 3.8: Illustration of convex upsampling module. Image source: [TD20].

where $\mathbf{x}_t$ is the input feature formed by concatenation of flow, correlation and context features and $\mathbf{h}_t$ is hidden state at iteration t. The architecture of update block is depicted in Figure 3.7. The hidden state, which is the output of GRU, is passed through two convolutional layers to predict the flow update. The output flow is at 1/8 resolution of input image. To use the final flow for training the network (unsupervised construction of loss+ optimization) and to evaluate the model, flow is upsampled.

In upsampling, each pixel of the high-resolution field is taken to be the convex combination of its 9 coarse resolution neighbors using weights predicted by the network as depicted in Figure 3.8. The following section describes the loss function used to train the unsupervised RAFT model.

3.3 Training Loss

The loss function is an essential factor in the training process of the model. The loss function decides how good the model prediction is in terms of predicting the expected outcome. In training the model, we define a loss function and then optimize the algorithm to minimize the loss function. By doing this, we converted our learning process into an optimization problem. Minimizing the loss results in the model parameters (weights and biases) to be tweaked to the correct value, which intern makes good predictions.

In unsupervised RAFT, a combination of photometric loss, smoothness loss, and augmentation loss is used. Individual losses from the sequence of predictions by the iterative operator of RAFT are weighted with exponentially increasing weights. We define the total loss as:

$$\mathcal{L}_{all} = \left(\sum_{i=1}^{N} \gamma^{N-i} \left(\mathcal{L}_{ph}(U_{12}^{i}) + \mathcal{L}_{sm}(U_{12}^{i}) \right) \right) + \lambda \, \mathcal{L}_{aug}(S(\overline{U}_{12}^{N}), \overset{*}{\overline{U}}_{12}^{N})), \tag{3.8}$$

where sequence of predictions from the first forward pass are denote by { U_{12}^{1}, U_{12}^{2} ... U_{12}^{N} }. γ and λ are the constants used for weighting each sequence of predictions and augmentation loss respectively. Each of the loss terms are explained in detail in the following sections.

3.3.1 Photometric Loss

Given the predicted flow, the photometric loss is computed as the difference between the first image and backward warped second image as formed in equation 2.11. The difference between the first and warped second image can be computed through different similarity measures. In unsupervised RAFT, the photometric loss was computed as:

$$\mathcal{L}_{ph} \sim w_{\ell 1} * (\sum_{p} \ell 1(\hat{I}_1(\Theta), I_1) + w_{\text{SSIM}} * (\sum_{p} \text{SSIM}(\hat{I}_1(\Theta), I_1) + w_{\text{ternary}} * (\sum_{p} \text{ternary}(\hat{I}_1(\Theta), I_1), \tag{3.9}$$

where $\ell 1$, SSIM, ternary corresponds to ℓ_1 distance measure, structural similarity measure and ternary measure respectively. w_{l1}, w_{SSIM} and w_{ternary} are the weights corresponding to those similarity measures. $\hat{I}_1$ and I_1 represents backward warped second image (reconstructed image from the weights Θ of the model) and the first image respectively.

Structural Similarity Measure

Structural similarity (SSIM) compares local patterns of pixel intensities that have been normalized for luminance and contrast. SSIM measurement was used in computing distance measure between the images in part of the photometric loss of unsupervised RAFT.

Structure similarity between two image signal x and y can be calculated as follows:

$$SSIM(x,y) \sim \frac{(2\mu_x\mu_y + C_1) * (2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}, \tag{3.10}$$

where C_1 and C_2 are small constants included to avoid instability, when intensities are close to zero. μ_x and μ_y are the mean intensities of two images, σ_x, σ_y are the standard deviations of the two image signals, which are calculated as:

$$\mu_x = \frac{1}{N}\sum_{i=1}^{N} x_i, \tag{3.11}$$

$$\sigma_x = \left(\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_x)^2)\right)^{\frac{1}{2}}, \tag{3.12}$$

and

$$\sigma_{xy} = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_x)(y_i - \mu_y) \tag{3.13}$$

Ternary Loss Measure

The ternary census transform is a non-parametric matching cost, which relies only on the ordering of pixel intensities. Ternary loss is invariant to illumination changes in a real-world scenario.

The ternary census transform is defined as,

$$CT(x,y) = \bigotimes_{i=-n'}^{n'} \bigotimes_{j=-m'}^{m'} t\left(I(x,y), I(x+i, y+j)\right), \tag{3.14}$$

where $\otimes$ is the concatenation operator and CT is the code image after transformation, t is the ternary operator defined as:

$$t(x,y,\delta) = \begin{cases} 00, & \text{if } x > y + \delta \\ 01, & \text{if } y - \delta \leq x \leq y + \delta \\ 11, & \text{if } x < y - \delta \end{cases}, \tag{3.15}$$

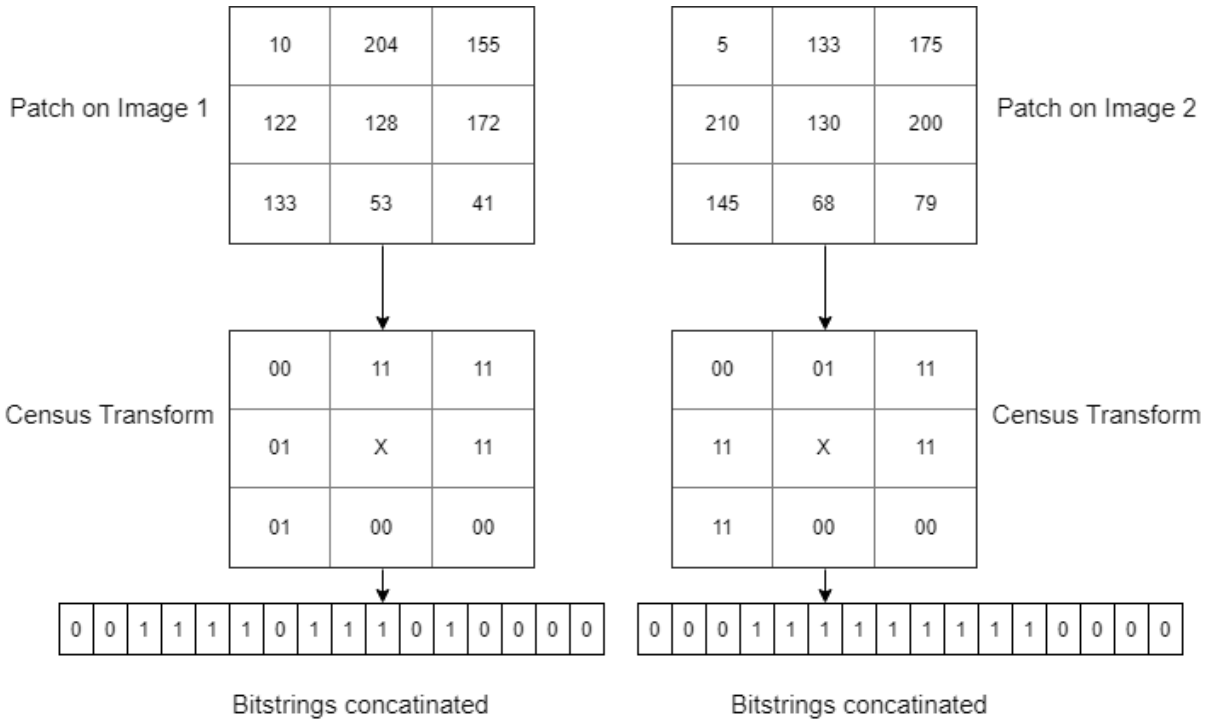

Figure 3.9: Example of ternary census transform using 3×3 patch and $\delta = 8$.

where δ is threshold for equality.

The first image and the warped second image are transformed to bit strings using ternary census transform, and hamming distance is calculated between the transformed bit strings. Final ternary loss is obtained through the mean of the normalized Hamming distance measure. Figure 3.9 shows an example of ternary census transform.

Distance Measure ($\ell 1$)

The most superficial and most widely used similarity metric is $\ell 1$ distance, computed by the sum of the absolute difference of first image and backward warped second image pixels.

$$\ell 1(\hat{I}(\mathbf{x}), I(\mathbf{x})) = \|\hat{I}(\mathbf{x}) - I(\mathbf{x})\|_1 \tag{3.16}$$

But only considering the photometric loss for all the pixels is not enough. Photometric similarities are not valid at the occluded locations. The following section describes the occlusions and how to detect the occluded pixels.

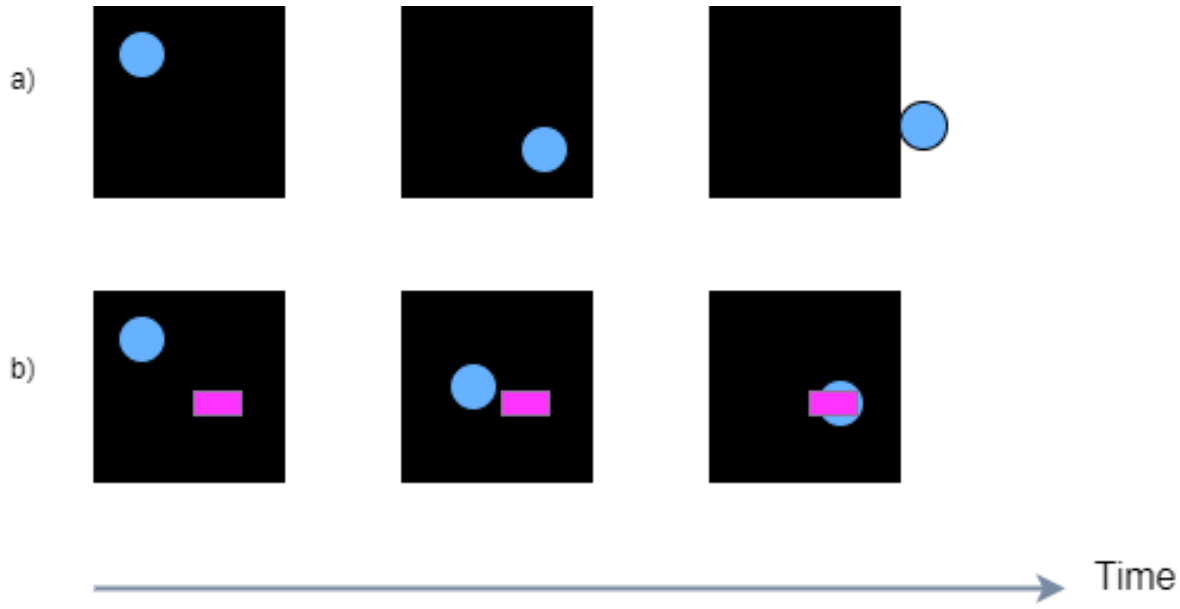

Figure 3.10: Occlusion scenarios. a) pixels moving out of the image boundary over the time b) overlapping pixels over a image sequence.

3.3.2 Occlusions

Occlusions are common when we consider image sequences. When pixels are moved out or pixels overlap in an image sequence, considered to be occluded pixels. These occlusion scenarios are depicted in Figure 3.10.

Photometric loss (2.11) is violated in the occluded regions because this loss is based on the observation that a pixel in the first frame should be similar to the pixel in the second frame to which the flow maps it. We will not be able to find correspondence for occluded pixels. Photometric loss is ignored at the occluded regions to avoid incorrect learning deformations that fill in the occluded pixels. The Photometric loss is modified as:

$$\mathcal{L}_{ph} = \sum_{\mathbf{x}} (1 - O_{21}(\mathbf{x})) * \rho(\hat{I}(\mathbf{x}), I(\mathbf{x})), \tag{3.17}$$

where $O_{21}(\mathbf{x})$ represents if pixel $\mathbf{x}$ is occluded. More precisely,

$$O_{21}(\mathbf{x}) = \begin{cases} 1, & \text{if } \mathbf{x} \text{ is occluded in } I_2 \\ 0, & \text{otherwise} \end{cases}, \tag{3.18}$$

Photometric loss was excluded in all distance measures described above.

Finding Occlusions

Occlusions were found through standard forward-backward consistency assumption [SBK10]. This assumption is, the forward flow should be the inverse of the backward flow at the corresponding pixel in the second frame for non-occluded pixels. The pixels are set to be occluded whenever the mismatch between these forward and inverse backward flows is too large.

Occlusions were set to 1 for pixels satisfying the following equation:

$$\|U_{12}(\mathbf{x}) + U_{21}(\mathbf{x} + U_{12})\|_2^2 > \alpha_1 \left(\|U_{12}(\mathbf{x})\|_2^2 + \|U_{21}(\mathbf{x} + U_{12}(\mathbf{x}))\|_2^2 \right) + \alpha_2, \tag{3.19}$$

where U_{12} and U_{21} are predicted forward and backward flow from the RAFT, respectively. Backward flow is calculated by reversing the image sequence and giving these swapped images to RAFT. α_1 and $\alpha_2$are small constant and are set to 0.01 and 0.5 respectively.

During initial iterations of the network, optical flow predictions are totally inaccurate. When the optical flow predictions are totally inaccurate, the occlusion map by bidirectional reasoning will be all zeros, and thus the photometric loss becomes invalid. To overcome this in unsupervised RAFT implementation, for some initial iterations, occlusions were estimated from the backward flow as calculated in UnOS [WWY+19]. Finding occlusion from the backward flow can avoid this problem and generate a more accurate occlusion when the flow is inaccurate.

3.3.3 Smoothness Loss

Photometric loss is ambiguous at the texture-less regions or where there are repetitive patterns, for example in case of the aperture problem. A smoothness loss aims to regularize the output flow by minimizing the difference in flow between neighboring pixels in x and y directions:

$$\mathcal{L}_{sm} \sim \sum_{d \in x,y} \sum_{\mathbf{x}} \|\nabla_d U_{12}\|_1 e^{-\|\nabla_d \mathcal{I}\|}, \tag{3.20}$$

When there is no significant gradient exists, smoothness loss tries to constrain the optical flow prediction similar to the neighbors in x and y directions. The smoothness loss is edge-aware. That is, whenever a significant gradient exists it tries not to smooth the flow across the edge. As we can see in Equation (3.20), if there is an edge in the considered neighborhood, the gradient value is large, and when we raise it exponentially in the negative direction, we get a small value close to 0, thereby not smoothening across the edges. On the other hand, when no significant gradient value exists, it tries to fetch flow information similar to that of its neighbors.

ARFlow achieves a significant result through its special pipeline, which uses augmentation as regularization in its second pass. The same augmentation techniques are incorporated in our unsupervised RAFT. The following section describes augmentations and augmentation loss used in the second pass of the network.

3.3.4 Augmentation

In deep learning, training a machine learning model refers to tuning its parameters such that it can map a particular input to some output. During training, the optimization goal is to find a sweet spot where the model's loss is low, which happens when model parameters are tuned correctly. State-of-the-art deep learning models typically have parameters in the orders of millions. For example, VGGNet for identifying the image category has around 100 million parameters, and GNMT, which is used for language translation, has around 380 million parameters. Naturally, if a model has many parameters, then the model should be trained with a sizeable proportional amount of data. Having a large dataset is crucial for the performance of deep learning models. A model with large parameters trained on a smaller data set results in overfitting and does not generalize on the new unseen sample.

Data augmentation is used to improve the performance of the model, thereby preventing overfitting. Minor changes are made to existing input images such as flips, translations, and rotations to generate more challenging data. The deep network would think these as distinct images and try to fit the model parameters to all the samples, including augmented samples.

In unsupervised optical flow estimation, most of the previous works suffer from unreliable supervision for challenging scenes. However, ARFlow [LZH+20] comes up with an effective data augmentation techniques along with reliable supervision for augmented samples. The same techniques are incorporated in the unsupervised RAFT in order to increase the model performance.

Augmentation as Regularization

The unsupervised RAFT trains the network on original data and makes use of augmented samples as a regularization. To be precise, along with first forward pass where original images are given as input to the RAFT model, a second forward pass is incorporated where transformed augmented images are used as input. In the first forward pass, image synthesis loss is used as supervision where we tried to reconstruct the first image from the second image and predicted flow. In the second forward pass, the optical flow predicted in the first forward pass is transformed

consistently and used as a supervision signal. In the second forward pass, the basic assumption is that augmentation generates more challenging samples in which view synthesis loss (unsupervised loss) is unreliable. However, the consistent transformation of original predictions from the first forward pass can serve as reliable supervision for the second pass.

Let us assume the augmentation is parameterized by a random vector θ is defined as,

$$\mathcal{T}_{\theta}^{img} : I_t \mapsto \bar{I}_t, \tag{3.21}$$

where I_t and $\bar{I}_t$ is original image and augmented image respectively. The augmented images $\{\bar{I}_1, \bar{I}_2\}$ are sampled from transformation $\mathcal{T}_{\theta}^{img}$ as in equation 3.21.

3.3.5 Augmentation Loss

General Charbonnier function is used in augmentation loss. Also, loss in occluded regions is omitted.

$$\mathcal{L}_{aug} \sim \sum_{\mathbf{x}} (|S(\overline{U}_{12}(\mathbf{x})) - \overset{*}{\overline{U}}_{12}(\mathbf{x})| + \epsilon)^q, \tag{3.22}$$

where $\overline{U}_{12}$ refers to transformed predicted flow from the first forward pass and $\overset{*}{\overline{U}}_{12}$ refers to flow predicted during the second pass. However, the transformation of predicted flow from the first forward pass should be consistent with transformations applied on original images. So this transformation can be depicted as $\mathcal{T}_{\theta}^{flow} : U_{12} \mapsto \overline{U}_{12}$. In Equation (3.22), a small constant q and ϵ are used to give less penalty to outliers. To stop the gradients of augmentation loss propagating to the transformed original flow $\overline{U}_{12}$ stop-gradient is used. This stop gradient is denoted by S(.) in the equation.

In unsupervised RAFT for optical flow estimation following transformations are used.

Spatial Transformation

Transformations that only change the spatial properties of an image are categorized as spatial transformations. These transformations result in a change in the locations of pixels. Spatial transformations include zoom, flip, random crop, affine transformations, or more complicated transformations such as thin-plate-spline.

A general spatial transformation of a image $\mathcal{T}_{\theta}^{img} : I_t \mapsto \bar{I}_t$ can be formulated as:

$$\bar{I}_t(\mathbf{x}) = I_t(\tau_\theta(\mathbf{x})), \tag{3.23}$$

where τ_θ represents transformation of pixel coordinates. This can be implemented by a warping process similar as in equation 2.10. Spatial transforms change the location of pixels, therefore the corresponding flow has to be transformed consistently. The spatial transformations are applied to original input images and also on predicted flow from the first forward pass so that the transformed flow can be used for supervision of the second forward pass. The transformation of the flow can be formulated as:

$$\begin{cases} \widetilde{U}_{12}(\mathbf{x}) = \tau_\theta(\mathbf{x} + U_{12}(\mathbf{x})) - \tau_\theta(\mathbf{x}) \\ \overline{U}_{12}(\mathbf{x}) = \widetilde{U}_{12}({\tau_\theta}^{-1}(\mathbf{x})), \end{cases} \tag{3.24}$$

the warping is applied on intermediate flow since there is a change in the location of pixels and to write it to the original pixel coordinates, an inverse coordinate transformation operation is performed.

In addition, the change in pixel locations may bring new occlusions. Since we have to exclude the augmentation loss in the occluded regions, these new occlusions should also be calculated once the input undergoes spatial transform. The appearing new occlusions can be found through standard forward-backward check during the second pass, but flow predictions on transformed samples are noisy. Therefore transformed occlusion maps are inferred from original predictions. The occlusion transformation $\mathcal{T}_\theta^{occ} : O_{12} \mapsto \bar{O}_{12}$ consists of two parts. The first part is old occlusions in the new transformed view. These are calculated by the same warping process $\mathcal{T}_\theta^{img}$, but instead of bilinear interpolation, nearest-neighbor interpolation is used. The second part of occlusions is newly appeared occlusions $\overline{O}_{12}^{new}$because of the change in locations of the pixel. These new occlusion maps are estimated from the flow $\bar{U}_{12}$ by checking the boundary.

$$\overline{O}_{12}^{new}(\mathbf{x}) = \Big(\mathbf{x} + \overline{U}_{12}(\mathbf{x})\Big) \notin \Omega, \tag{3.25}$$

where Ω represents image domain boundary. The final occlusion maps are the union set of old occlusions and the new occlusions. An example of spatial transform and corresponding flow transform are shown in Figure 3.11.

Appearance Transformation

Appearance Transformations only change the appearance of images. For example, by changing the brightness or by adding random color jitter, blur, or noise, one could generate different samples

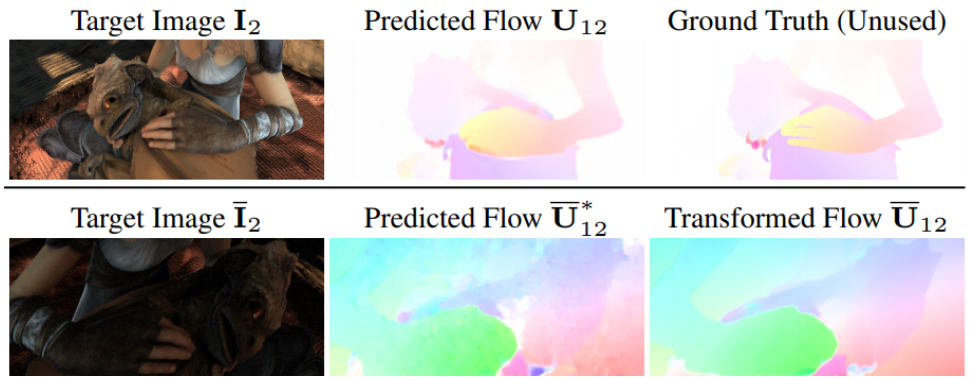

Figure 3.11: Example of spatial transformation (zoom) and appearance transformation (brightness change) generates large displacement and low brightness [TD20].

from the original image. These transformations do not change the locations of pixels. Hence, no transformation is required for the corresponding optical flow. Appearance transformations do not create new occlusions. Only the input images are transformed, and occlusions predicted flow from the first forward pass used in the second forward pass. An example of appearance transform (brightness change) is shown in Figure 3.11.

Occlusion Transformation

Prior to ARFlow, recent works proposed to learn flow in arbitrary occluded regions. This process consists of training teacher model, offline creating occlusion samples, and distilling to a student model. These approaches were not end-to-end. ARFlow incorporates these occlusion as a occlusion transformations in the network in complete end-to-end training process. Similar transformation methods are used in unsupervised RAFT. Occlusion transformation is created in two steps. The first one is to crop the image randomly. The second step is to randomly mask out some pixels in the original images with Gaussian noise, which results in new occlusions. An example of occlusion transformation is shown in Figure 3.12.

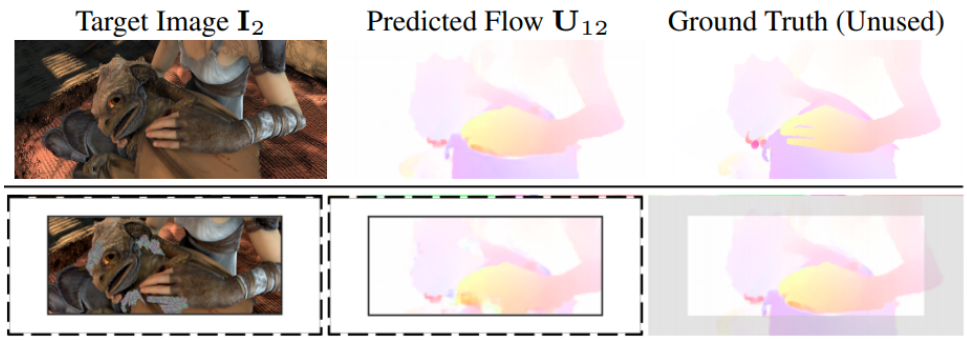

Figure 3.12: Example of occlusion transformation [TD20].

4 Datasets

Training any deep network based models require large datasets. In unsupervised RAFT implementation four different datasets were used to train. The ground truth is not used in any of the stages of training. However for evaluation of the models ground truth flows were used. The following sections describes the datasets used to train and evaluate the unsupervised RAFT.

4.1 Flying Chairs

Flying Chairs [DFI+15] are synthetic dataset. This dataset is provided for research purpose. Flying Chair dataset are created in order to provide large annotated optical flow training samples. This dataset is generated by applying affine transformations to images collected from Flickr and a publicly available set of renderings of 3D chair models. Dosovitskiy et.al [DFI+15] retrieved images from image hosting service Flikr from the categories 'city', 'landscape' and 'mountains' with resolution of 1024×768. They retrieved 321, 129 and 514 images from each of the above categories respectively, making total of 964 images. Then, the images were cropped into 4 quadrants and used the each image crops as with resolution 512×384 as background. To foreground chair images were added publicly available 3D CAD models of chairs. To generate motion, they randomly sample 2D affine transformation parameters for the background and the chairs. The chairs transformations are relative to the background transformation, which can be interpreted as both the camera and the objects moving. With the help of these transformation parameters, the second image and the ground truth optical flow were generated. Examples are shown in Figure 4.1. Using this procedure, total of 22,872 image pairs and corresponding ground truth flows were generated. This dataset is used in many optical flow estimation researches, in order to train/pre-train the deep learning based networks.

Figure 4.1: Example of Flying Chairs dataset. First three column represents generated image pairs and corresponding flow. Last three column are augmented image pairs and flow. Ground truth flow is colour coded. Image source: [DFI+15].

4.2 Flying Things 3D

Flying Things 3D [NEP+16] dataset consists of general everyday objects flying along randomized 3D trajectories. This dataset consists of 25000 stereo frame along with the corresponding ground truth. The main idea for creating this dataset is to ease training of large deep convolutional networks, which should benefit from the large variety. The generation of dataset rely on randomness and a large pool of rendering assets to generate large amount of samples without repetition or saturation. The base of each scene ground plane with texture. Background objects were generated by using random shapes from cuboids and deformed cylinders. In total 200 background static objects are generated. Each object is textured and randomly augmented with random scaling, rotations. To populate the scene, 3D models from Stanford's ShapeNet6 [SCH15] database used. From these, 32872 training sets and 3055 test sets are generated. Example of the dataset is shown in Figure Figure 4.2.

4.3 KITTI-2015

KITTI [MG15] dataset is consists of dynamic real world scenes and ground truth is established through semi-automatic process. In this work Menze and Geiger take advantage of their autonomous driving platform any way to develop challenging real-world benchmarks. Tasks of interest are stereo, optical flow, visual odometry, 3d object detection and 3d tracking. For this purpose they equipped a standard station wagon with two high resolution color and grayscale video cameras. Using a baseline of 54 centimeters accurate ground truth information is provided by a Valentine laser scanner and a GPS localization system with integrated inertial measurement unit and RTK corrections. KITTI data sets are captured by travel around in a mid-sized city rural

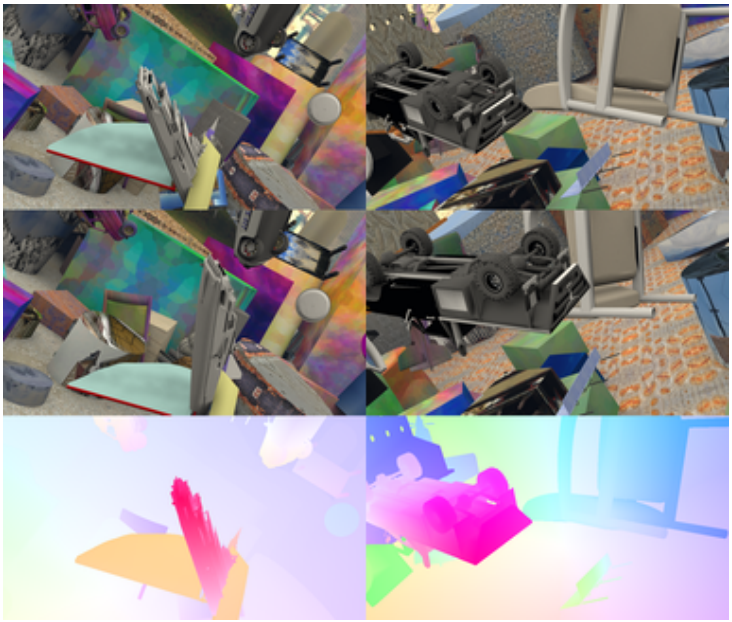

Figure 4.2: Example of Flying Things dataset. First two rows column represents generated image pairs and third row represents corresponding flow.Ground truth flow is colour coded. Image adopted from [NEP+16].

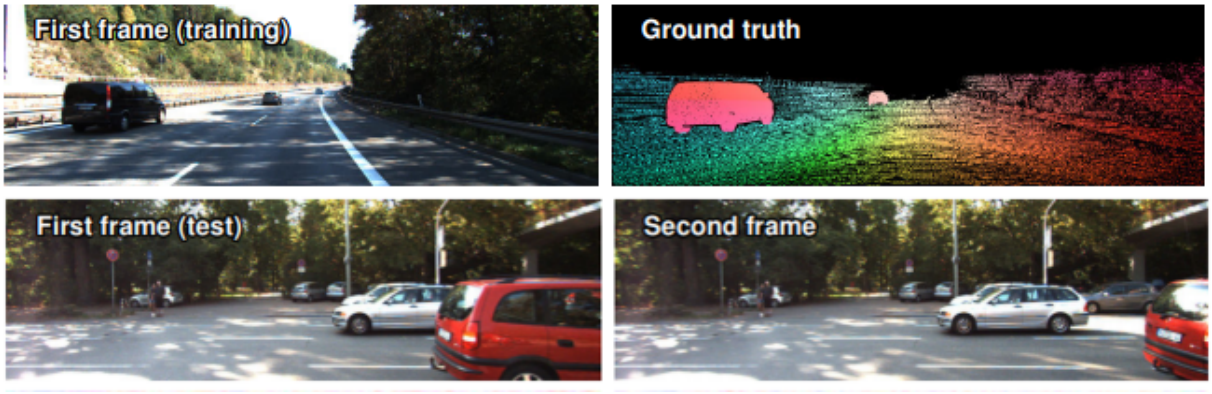

Figure 4.3: Kitti dataset: sample from KITTI 2015 dataset. Image source: [MG15].

areas and on highways, up to 15 cars and 30 pedestrians are visible per image. Besides providing all data in RAW format they extract benchmarks for each task. For each of the benchmarks they also provide an evaluation metric and an online evaluation website.

KITTI 2015 optical flow benchmark comprises of 200 training and 200 test image pairs at a resolution of half a mega pixel. The benchmark includes difficulties such as nonlinear version and reflecting surfaces consisting of large feature displacements, different materials as well as a variety of lighting conditions.

4.4 MPI-Sintel

MPI-Sintel [BWSB12] is a optical flow dataset derived from open source 3D animated short film "Sintel". This data set has long sequences, specular reflections, large motions, motion blur, defocus blur, and atmospheric effects which is not covered in earlier datasets. Example of MPI-Sintel dataset is in Figure 4.4. In ordet to generate samples, D.J. Butler et al. rendered the same Sintel movie scenes with different render settings there by gradually increasing complexity. This allowed model to evaluate the predicted flow in different conditions to where and how they failed. MPI-Sintel has two variations of datasets. These two variations are called Sintel Clean and Sintel Final.

MPI-Sintel Clean

Clean pass adds complexity by introducing illumination of various kinds. Surfaces exhibit smooth shading, self shadowing, darkening in cavities, and darkening where an object is close to a surface. Additionally this pass includes more complex illumination and reflectance properties including specular reflections, inter-reflections, and mirroring effects.

MPI-Sintel Final

This pass is similar to the final artistic rendering included in the released film. Beyond the Clean pass it adds atmospheric effects, depth of field blur, motion blur, color correction, and possibly other artistic embellishments to lighting and overall appearance.

Several recent deep networks train and evaluate on Sintel dataset and use Sintel as benchmark dataset in optical flow estimation. Also, this thesis mainly focuses on Sintel clean and Sintel final dataset for training and evaluations.

The following sections describe standard dataset-specific error measures used for the evaluation of the models.

4.5 Error Measures

For checking the performance of the published models, standard error measures for datasets are defined. The most common error measures are listed below.

Figure 4.4: MPI-Sintel dataset: some examples of ground truth flow fields and its corresponding images. Image source: [BWSB12].

End Point Error

End Point Error (EPE) is an error measure and used for evaluating the models. EPE is calculated by comparing an estimated optical flow vector ($U_{12}(\mathbf{x})$) with a ground-truth optical flow vector ($U_{GT}(\mathbf{x})$) for each of the pixels. Generally, EPE is defined as an average of the Euclidean distance between estimated optical flow and ground-truth flow:

$$L_{epe} = \frac{1}{N} \sum_{\mathbf{x}} \|U_{12}(\mathbf{x}) - U_{GT}(\mathbf{x})\|_2, \tag{4.1}$$

where N denotes total number of pixels.

EPE measure is used for Flying Chairs, Flying Things and MPI-Sintel datasets.

Fl Error Measure

KITTI dataset uses Fl error measure for evaluating the models. KITTI evaluation server computes the percentage of bad pixels averaged over all ground truth pixels of all 200 test images. KITTI evaluation, considers a pixel to be correctly estimated if the disparity or flow end-point error is $<$3px or $<$5%.

The next chapter describes the setup and implementation details of the unsupervised RAFT network.

5 Implementation and Setup

5.1 Resources

Unsupervised RAFT model is implemented, trained, and evaluated on a specific set of hardware and frameworks. To train the network, substantial amount of memory and an excellent graphic processor is required. The model is implemented on specific framework with the help of the software. To train the deep learning model, a specific values of hyper parameters are used. This section includes a short summary of the tools and frameworks, hyperparameters used in the implementation of the model.

5.1.1 Pytorch

Pytorch is an open-source framework for modeling neural networks. Pytorch is released under the Modified BSD license. It is used for applications like computer vision and natural language processing and developed by Facebook'sAI Research lab (FAIR). It is known for providing two of the most high-level features: tensor computations with strong GPU acceleration support and building deep neural networks on tape-based autograd systems ([PGM+19]). It has allowed deep learning scientists, machine learning developers, and neural network researchers to develop deep learning models. In this implementation, Pytorch version 1.7.0+cu110 is used.

5.1.2 Hardware

An extremely powerful graphics processor-based system is required to process a large amount of data and speed up the processing and optimization time. The unsupervised RAFT model is trained and evaluated on a nvidia GPU machine. The specification of the machine is as listed in Table 5.1.

Hardware Specification	
GPU name	GeForce RTX 3090
Memory size	24 GB
CUDA computing units	10496
Boost clock	1.70 GHz
CPU-Architecture	x86_64
CPU op-mode(s)	32-bit, 64-bit
CPU(s)	20

Table 5.1: Hardware Specifications.

5.1.3 Software

Model is trained using the Compute Unified Device Architecture (CUDA) compilation tools by Nvidia. CUDA build 11.1 version is used in work. Pytorch provides set up and runs CUDA operations on Nvidia graphics. It keeps track of the currently selected GPU and all CUDA tensors and performs tensor operations.

The unsupervised RAFT complete code is implemented using Python programming language. The code uses already available implementations of RAFT (https://github.com/princeton-vl/RAFT) and ARFlow (https://github.com/lliuz/ARFlow/) code from the GitHub code repository.

5.2 Implementation Details

5.2.1 Feature/Context Encoder

The implementation details of feature encoders of RAFT is shown in Figure 5.1 and Figure 5.2. The main difference between small (RAFT-S) and full feature encoder is that the original RAFT feature encoder consists of a residual block (5.1), whereas small feature encoder consists of bottleneck blocks. Also, as the name suggests small feature encoder has fewer parameters and fewer feature channels.

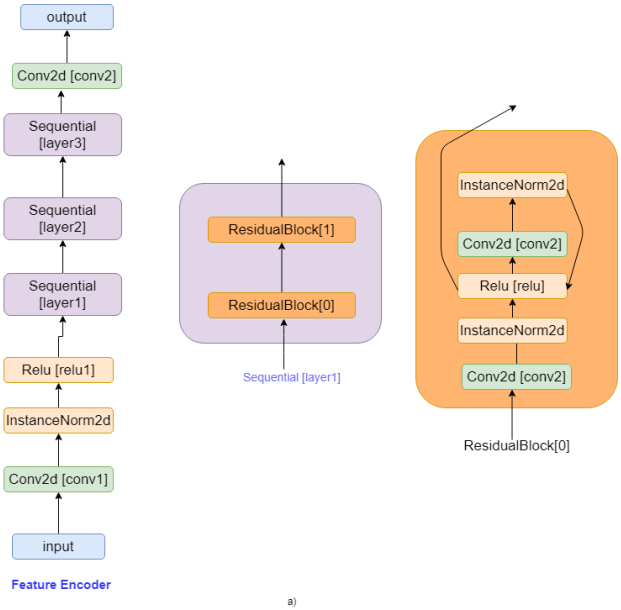

Figure 5.1: Implementation details of RAFT feature encoder using Pytorch tensor operations.

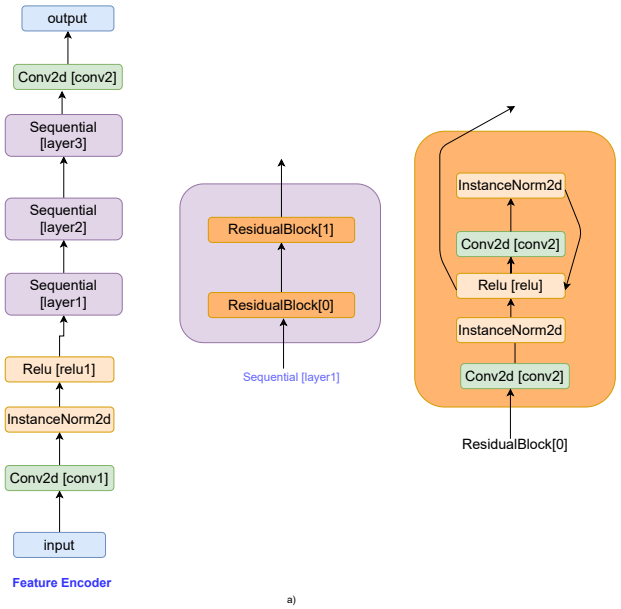

Figure 5.2: Implementation details of RAFT-s (small) feature encoder using Pytorch tensor operations.

5.2.2 Correlation lookup

In unsupervised RAFT, a full all-pairs correlation volume is constructed by taking the dot product between two extracted feature maps. Then, to extract the correlation feature, a look-up on correlation volume is performed with the following parameters.

- Lookup radius: The lookup radius specifies the dimensions of the grid used in the lookup operation. In unsupervised RAFT radius of 4 used, where as in small network implementation radius of 3 is used.
- Correlation range: A full 4D all pairs correlation volume is constructed in the current work.
- Correlation lookup sampling: Bilinear sampling is used when the loop up lies between the grid pixels.
- Correlation volume pooling: To capture large and small motions, the last two dimensions of correlation volume are pooled by applying a 2D average-pooling operation with the stride of 2. Averaging pooling was applied three times to create a correlation pyramid of 4 levels.

5.2.3 Update Block

Update block iteratively refines the flow. The initial flow is set to 0. RAFT update block setup is depicted in Figure 5.3. The correlation features are then processed by 2 convolutional layers and current flow estimate is processed by 2 convolutional layers to generate flow features. Finally, context features are directly injected as input and finally concatenated as one motion feature. These operations are depicted in BasicBlockEncoder (Figure 5.3). The concatenated feature given input to GRU block. The hidden state, which is output of GRU is passed through two convolutional layers to predict the flow update.

5.2.4 Augmentations and Transformations

Augmentation Transformations are implemented using torchvision and skimage python packages. Transforms are common image transformations. Each transformation can be chained together in a pipeline using the torchvision package. The scikit-image and torchvision library provide modules for some transformations, such as resizing, rescaling, downscale, zoom, and many other transformations. Torchvision also helps in stacking individual transforms.

Given input image sequences from the dataset, for each image, series of transformations are applied before passing to the RAFT model for flow predictions. The series of transformations

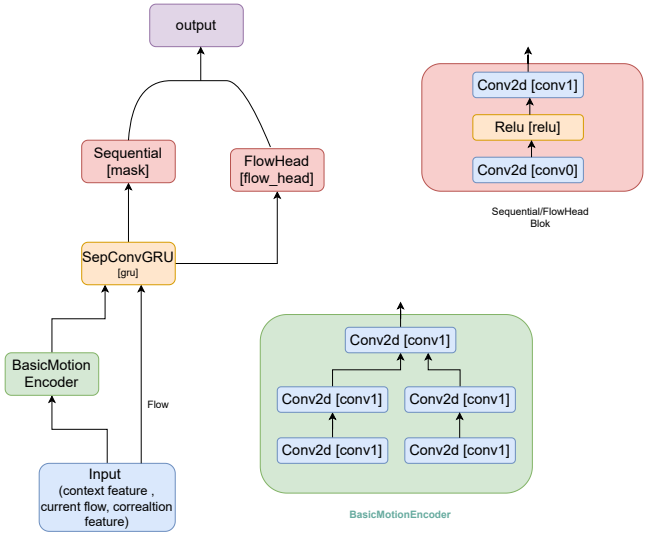

Figure 5.3: Implementation detail of update block using Pytorch.

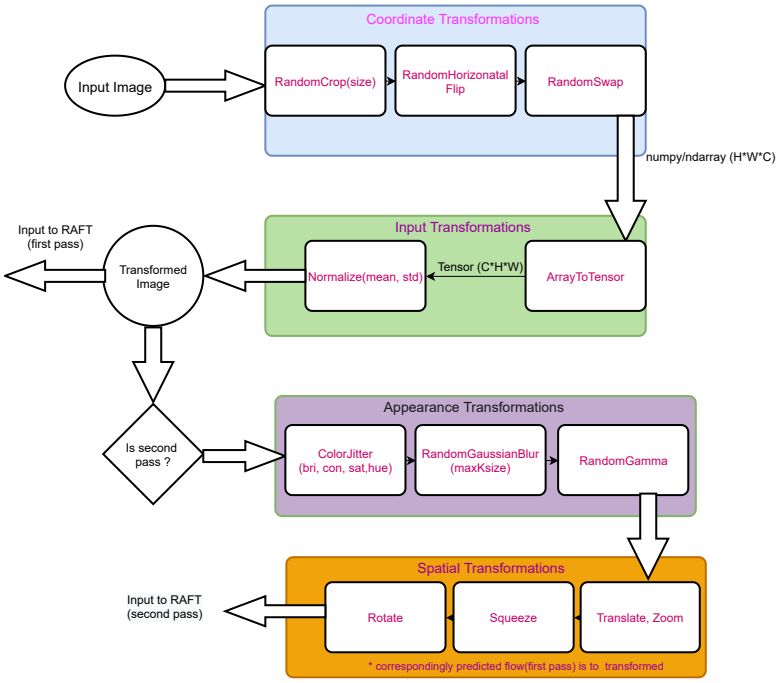

Figure 5.4: Flowchart of Transformations applied to input image.

applied is depicted in flow chart Figure 5.4. Each of these transformations is explained in the following subsections.

Coordinate Transformations

Input image goes through series of coordinate transformations like crop, horizontal flip, and swap as shown in Figure 5.4. All of these transformations are stacked using the pytorch torchvision package. The coordinate transforms outputs the NumPy ndarray (image) of size (H x W x C),

- $RandomCrop(size)$: crops the given image at a random location to have a region of the given size. The size parameter depends on the dataset. In current implementation, the dataset and the crop size is mentioned in the Table 5.2.
- $RandomHorizontalFlip$: Randomly horizontally flips the given image with a probability of 0.5.
- $RandomSwap$: Randomly swaps the given images with a probability of 0.5.

Dataset	Crop Size $(H \times W)$
Flying Chairs	368×496
Flying Things	400×720
MPI-Sintel	384×832
KITTI 2015	288×960

Table 5.2: Dataset and Crop Size $(H \times W)$.

Input Transformations

The output image after coordinate transformation is then passed to input transformation modules. Input transformations includes the following operations.

- $ArrayToTensor$: Converts a numpy array (H x W x C) to a torch Tensor of shape (C x H x W). This is helpful in normalizing the input data through tensor operations.
- $Nomalize(mean, std)$: Normalize a tensor image with mean and standard deviation for each of the channel. Given mean and standard deviation of each channel, this transform will normalize each channel of the input:
  $output[channel] = (input[channel] - mean[channel])/std[channel]$.
  In implementation, mean [0,0,0] and standard deviation [255,255,255] are used for 3 channels of the image.

The output of input transformations are passed to the RAFT model to predict the flow in the first forward pass. In the second pass (augmentation as regularization), the input image transformed further to create challenging scenes.

Appearance Transformations

Appearance transformations changes the the appearance of the image. The following set of appearance transformations are used.

- $ColorJitter(bri, con, sat, hue)$: Randomly change the brightness, contrast, saturation and hue of an image. The parameters are described in Table 5.3.
- $RandomGaussianBlur(maxKsize)$: Creates Gaussian blur. Blurs regions chosen uniformly from $[0, maxKsize]$ radius. In implementation $maxKsize = 3$ is used.
- $RandomGamma$: Transform by applying randomly Gamma function.

Parameter	Explanation	Value
bri	How much to jitter brightness and factor chosen uniformly from $[max(0, 1 - bri), 1 + bri]$	0.5
con	How much to jitter contrast and factor chosen uniformly from $[max(0, 1 - con), 1 + con]$	0.5
sat	How much to jitter saturation and factor chosen uniformly from $[max(0, 1 - sat), 1 + sat]$	0.5
hue	How much to jitter hue and factor chosen uniformly from $[-hue, hue]$	0.0

Table 5.3: ColorJitter transformation explanation and the values use for parameters.

Spatial Transformations

The transformation that changes the locations of the pixel is referred to as spatial transformations. If the input image undergoes spatial transformation, the corresponding predicted flow from the first forward pass should be transformed consistently. The following set of spatial transformations are used.

- Translate
- Zoom

- Squeeze
- Rotate

5.2.5 Implemented models

Two models are trained in unsupervised RAFT implementation. The first model trained is an unsupervised RAFT small (unsup-raft-sm) model consisting of RAFT-S as a base network for flow predictions. This model consists of fewer trainable parameters and uses less memory while training. The second model implemented is the unsupervised RAFT (unsup-raft) model, which uses the original full RAFT network for the prediction of the flow. Both models are trained on Flying Chairs and Flying Things dataset and evaluated on Sinteel and KITTI 15 dataset. Later the model is fine-tuned on the Sintel dataset. The unsup-raft-sm model contains 2.3 Million parameters, and the unsup-raft model contains 5.25 Million parameters.

5.2.6 Hyperparameters

Hyperparameters are parameters whose value is used to control the learning process. They are often used in processes to help estimate model parameters. They can often be set using heuristics. Some of the hyperparameter values are set in the previous sections. In this section, details of the essential hyperparameters used in the unsupervised RAFT implementation.

The batch size and number of epochs are important hyperparameters in training the model. The batch size and number of epochs trained are mentioned in the Table 5.4 and 5.5 to train unsup-raft-sm and unsup-raft model respectively.

The batch size is chosen such way that, the model memory fits in the hardware mentioned in the earlier section.

dataset	batch size	epochs
Flying Chairs	6	400000
Flying Things	4	400000
MPI-Sintel	4	400000

Table 5.4: Hyperparameters: batch size and epochs for training unsup-raft-sm model.

dataset	batch size	epochs
Flying Chairs	4	800000
Flying Things	2	800000
MPI-Sintel	2	800000

Table 5.5: Hyperparameters: batch size and epochs for training unsup-raft model.

Training Loss Parameters

Photometric loss is constructed as in Equation (3.9). In the warm-up stage (first 100k iterations), a sum of SSIM and l1 loss is used with weights of $w_{l1} = 0.15$ and $w_{\text{SSIM}} = 0.85$. Then, the l1 loss with census transform is used in the photometric los. ($w_{l1} = 0$ and $w_{\text{SSIM}} = 0$ and $w_{\text{ternary}} = 0$).

As defined in Equation (3.8), a sum of the photometric loss, smoothness loss, and augmentation loss are used for the training loss. $\gamma = 0.8$ and $= 0.01$ used in the current implementation.

Optimizer and Learning Rate Scheduler

Optimizers are algorithms or methods used to change the attributes of the neural network, such as weights and biases (parameters) and learning rate in order to reduce the losses.

Adaptive optimizers like Adaptive Moment Estimation (Adam) [**?**] has become a default choice for training neural network optimizations. In current implementation AdamW optimizer is used as optimizer in the unsupervised RAFT implementation. AdamW is an improved version of Adam where the weight decay is performed only after controlling the parameter-wise step size. AdamW yields better training loss, and that the models generalize much better.

When training deep neural networks, it is often useful to reduce learning rate as the training progresses. This can be done by using pre-defined learning rate schedules or adaptive learning rate methods. Learning rate schedules seek to adjust the learning rate during training by reducing the learning rate according to a pre-defined schedule.

OneCycleLR from Pytorch is used as a learning rate scheduler. OneCycleLR sets the learning rate of each parameter group according to the one-cycle learning rate policy. The 1cycle policy anneals the learning rate from an initial learning rate to some maximum learning rate and then from that maximum learning rate to some minimum learning rate much lower than the initial learning rate. This policy was initially described in the paper [ST19].

Pytorch AdamW optimizer method is of $AdamW(params, lr, weight\_decay, eps, ...)$ form. Important parameters of the method and the value used in the implementation is explained in the Table 5.6.

parameter	explanation	value
params	parameters to optimize	model params
lr	learning rate	refer (5.9)
weight_decay	weight decay coefficient	refer (5.7)
eps	small constant added to the denominator to improve numerical stability	1e-8

Table 5.6: AdamW optimizer parameter explanation and values used in implementation

dataset	weight_decay
Flying Chairs	0.0001
Flying Things	0.0001
MPI-Sintel	0.00001

Table 5.7: Weight decay parameter based on datasets.

Pytorch OnceCycleLR scheduler method is of -
$OneCycleLR(optimizer, max\_lr, total\_steps, pct\_start, cycle\_momentum, anneal\_strategy, ...)$
-form. Important parameters of the method and the value used in the method is explained in the Table 5.8.

It is essential to see how exemplary the current implementation is and compare the current model to other published methods. The next chapter explains the results and compares the results to other published methods.

parameter	explanation	value
optimizer	wrapped optimzer	AdamW
max_lr	upper learning rate boundaries in the cycle for each parameter group	refer (5.9)
total_steps	the total number of steps in the cycle	$epochs+100$ (5.4)
pct_start	small constant added to the denominator to improve numerical stability	1e-8
cycle_momentum	If True, momentum is cycled inversely to learning rate	false
pct_start	the percentage of the cycle (in number of steps) spent increasing the learning rate	0.5
anneal_strategy	specifies the annealing strategy	linear

Table 5.8: OnceCycleLR scheduler parameter explanation and values used in implementation.

dataset	learning rate
Flying Chairs	0.0004
Flying Things	0.000125
MPI-Sintel	0.000125

Table 5.9: Hyperparameter: learning rate (lr) initilaization.

6 Results

Unsupervised RAFT is trained on Flying Chairs and Flying Things dataset. It is then evaluated on Sintel and KITTI-2015 dataset. To compare the current model with ARFlow model and other recent models, our model is also fine-tuned on Sintel dataset. Predictions of our model on handpicked examples are visually depicted in Figure 6.1. The evaluation result is listed in Table 6.1.

The top half of the table shows some supervised learning-based models and their performance on Sintel and KITTI-2015 datasets. The bottom half of the table shows evaluations of unsupervised models. As can be seen from the table below, the current unsupervised RAFT (ours) model outperforms the ARFlow model. Closer inspection of the table shows that just by training with Flying Chairs (C) and Flying Things(T) dataset model achieves good results on Sintel and KITTI-2015. This signifies a good cross-data generalization of our model.

To compare the model with other published unsupervised models, the model is fine-tuned and tested on the Sintel train and test datasets. The model achieves 2.8% and 3.7 % better (EPE) performance than ARflow, on Sintel clean train and Sintel clean test dataset. The results were almost similar evaluting on the Sintel final train and test datasets. Also, the results were close to ARFlow on the Sintel final dataset.

Although the current model achieves good generalization results, it does not perform better than ARFlow or UFlow [JSB+20] models. This may because of the way each model is trained. The current model tries to train on Flying Chairs and Flying Things, and when fine-tuning on Sintel, it still loads the Flying Things dataset along with Sintel data and tries to fit the generalized distribution (A combination of Chairs, Things, and Sintel). ARFlow and UFlow models only load the Sintel dataset and try to fit the specific Sintel distribution.

If we compare our model with other supervised-based published methods, there is still a performance gap between unsupervised and supervised methods. However, the results of the current method were close to the PWC-Net (supervised) performance if we compare the two models by training on Chairs and Things datasets. On the other hand, it is unreasonable to compare unsupervised methods with supervised methods.

	Training data	Method	Sintel(train)		Sintel(test)		KITTI-2015
			clean	final	clean	final	train (Fl-all)
Supervised	C+T	RAFT [TD20]	1.43	2.71	-	-	17.4
	C+T+S/K	RAFT [TD20]	**(0.77)**	**(1.20)**	**2.08**	**3.41**	**(1.5)**
	C+T	PWC-Net [SYLK18]	2.55	3.93	-	-	33.7
	S/K	PWC-Net [SYLK18]	(2.02)	(2.08)	4.39	5.89	-
	C+T	FlowNet2 [IMS$^+$17]	2.02	3.54	-	-	30.0
Unsupervised	C+T	ours (small)	3.6	4.32	-	-	31.7
	C+T	ours	2.93	3.92	-	-	25.6
	C+T+S	ours	(2.71)	(3.76)	**4.61**	6.96	24.4
	S	ARFlow [LZH$^+$20]	(2.79)	(3.73)	4.78	**5.89**	-
	S	UnFlow-CSS [MHR18]	4.3	5.95	9.38	10.22	-
	S	OccAwareFlow [WYY$^+$18]	4.03	5.95	7.95	9.15	-
	S/K	UFlow [JSB$^+$20]	**(2.50)**	**(3.39)**	5.29	6.50	**(11.13)**

Table 6.1: Results on Sintel and KITTI-2015 datasets and comparison with state of the art. C, T, S denotes Flying Chairs, Flying Things and Sintel datasets. '-' indicates the results are not reported. () signifies the finetuning on the specific dataset it trained.S/K denotes the method which use Sintel data for finetuning on Sintel and KITTI data on finetuning on KITTI respectively. All the result were calculated using EPE measure except for KITTI-2015.

Overall, these results indicate that the current unsupervised RAFT model gains good cross-data generalization and can perform better than ARflow in cases of Sintel clean.

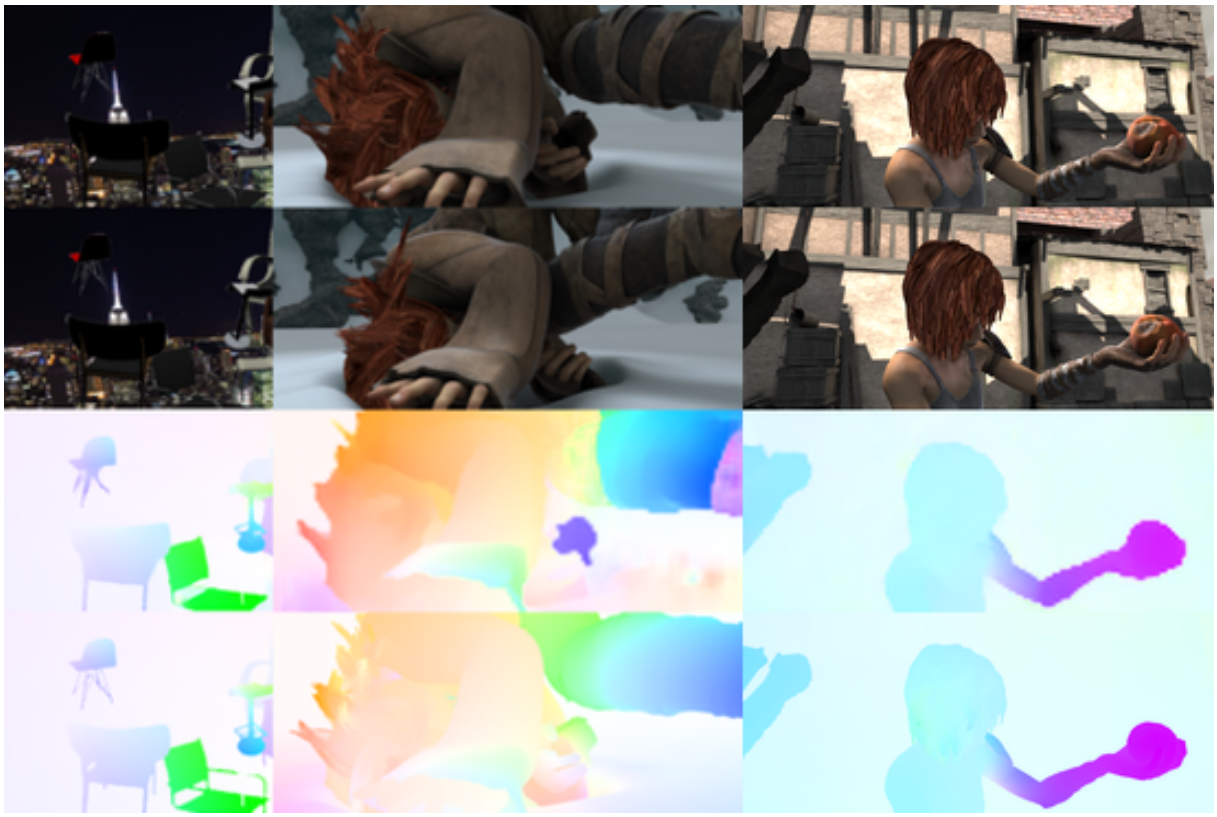

Figure 6.1: Visual results of our model on Sintel and Flying Chairs datasets. First two rows are sequences randomly picked from Flying Chairs and Sintel dataset. The third and last rows represent the flow predicted from our model and the corresponding ground truth.

7 Summary

The central thesis goal was to incorporate RAFT's efficient architecture in ARFlow and test the performance on recent optical flow benchmarks like Sintel and Kitti15. This thesis summarizes the ARFlow, PWC-Net, and RAFT, which are essential for understanding and achieving the thesis's desired goal. RAFT is a supervised model and achieves good results on benchmark datasets but requires a large amount of labeled data. Therefore, the unsupervised model, which does not require ground truth, is getting popular.

ARFlow is one of the recent unsupervised architectures to estimate the optical flow. PWC-Net was used as the base model in ARFlow. PWC-Net model performs poorly on cross data validation, signifying the overfit to the distribution the model is trained. Even though ARFlow gains good performance with its special two forward passes, the model suffers from some of the drawbacks of PWC-Net.

In the initial Chapters of this thesis, the critical architecture necessary for the current implementations was explained. Later in the chapters, the current architectural design and the implementation details are described. The current model is trained using the Flying Chairs and Flying Things dataset. Then, to fairly compare with the other published models, the current model was finetuned on the Sintel dataset, and the results were reported on Sintel and KITTI.

By incorporating RAFT as the base model for ARFlow, the results are 3.7 percent better than the ARFlow with PWC-Net on testing on Sintel clean dataset. The current model achieves good cross dataset generalization.

Bibliography

[BTS+15] N. Bonneel, J. Tompkin, K. Sunkavalli, D. Sun, S. Paris, H. Pfister. Blind video temporal consistency. *ACM Transactions on Graphics (TOG)*, 34(6):1–9, 2015. (Cited on page 9)

[BWSB12] D. J. Butler, J. Wulff, G. B. Stanley, M. J. Black. A naturalistic open source movie for optical flow evaluation. In *Proc. European Conference on Computer Vision (ECCV)*, pp. 611–625. Springer, 2012. (Cited on pages 10, 48 and 49)

[DFI+15] A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. Van Der Smagt, D. Cremers, T. Brox. Flownet: Learning optical flow with convolutional networks. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2758–2766. 2015. (Cited on pages 11, 45 and 46)

[GMAB17] C. Godard, O. Mac Aodha, G. J. Brostow. Unsupervised monocular depth estimation with left-right consistency. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 270–279. 2017. (Cited on page 11)

[HRB+12] A. Hosni, C. Rhemann, M. Bleyer, C. Rother, M. Gelautz. Fast cost-volume filtering for visual correspondence and beyond. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(2):504–511, 2012. (Cited on page 17)

[HS81] B. K. Horn, B. G. Schunck. Determining optical flow. *Artificial intelligence*, 17(1-3):185–203, 1981. (Cited on pages 9 and 10)

[HZRS16] K. He, X. Zhang, S. Ren, J. Sun. Deep residual learning for image recognition. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778. 2016. (Cited on page 29)

[IMS+17] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, T. Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2462–2470. 2017. (Cited on page 64)

[JGBG20] J. Janai, F. Güney, A. Behl, A. Geiger. Computer vision for autonomous vehicles: Problems, datasets and state of the art. *Foundations and Trends® in Computer Graphics and Vision*, 12(1–3):1–308, 2020. (Cited on page 9)

[JGW+17] J. Janai, F. Guney, J. Wulff, M. J. Black, A. Geiger. Slow flow: Exploiting high-speed cameras for accurate and diverse optical flow reference data. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3597–3607. 2017. (Cited on page 9)

[JSB+20] R. Jonschkowski, A. Stone, J. T. Barron, A. Gordon, K. Konolige, A. Angelova. What matters in unsupervised optical flow. *arXiv preprint arXiv:2006.04902*, 1(2):3, 2020. (Cited on pages 63 and 64)

[LKLX19] P. Liu, I. King, M. R. Lyu, J. Xu. Ddflow: Learning optical flow with unlabeled data distillation. In *Proc. AAAI Conference on Artificial Intelligence (AAAI)*, volume 33, pp. 8770–8777. 2019. (Cited on page 11)

[LLKX19] P. Liu, M. Lyu, I. King, J. Xu. Selflow: Self-supervised learning of optical flow. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4571–4580. 2019. (Cited on page 11)

[LZH+20] L. Liu, J. Zhang, R. He, Y. Liu, Y. Wang, Y. Tai, D. Luo, C. Wang, J. Li, F. Huang. Learning by analogy: Reliable supervision from transformations for unsupervised optical flow estimation. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6489–6498. 2020. (Cited on pages 10, 23, 28, 40 and 64)

[MG15] M. Menze, A. Geiger. Object scene flow for autonomous vehicles. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3061–3070. 2015. (Cited on pages 46 and 47)

[MHG15] M. Menze, C. Heipke, A. Geiger. Joint 3d estimation of vehicles and scene flow. *ISPRS annals of the photogrammetry, remote sensing and spatial information sciences*, 2:427, 2015. (Cited on page 10)

[MHR18] S. Meister, J. Hur, S. Roth. Unflow: Unsupervised learning of optical flow with a bidirectional census loss. In *Proc. AAAI Conference on Artificial Intelligence (AAAI)*, volume 32. 2018. (Cited on pages 11 and 64)

[NEP+16] N.Mayer, E.Ilg, P.Häusser, P.Fischer, D.Cremers, A.Dosovitskiy, T.Brox. A Large Dataset to Train Convolutional Networks for Disparity, Optical Flow, and Scene

Flow Estimation. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016. URL http://lmb.informatik.uni-freiburg.de/Publications/2016/MIFDB16. ArXiv:1512.02134. (Cited on pages 46 and 47)

[PGM+19] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, pp. 8026–8037. 2019. (Cited on page 51)

[RYN+17] Z. Ren, J. Yan, B. Ni, B. Liu, X. Yang, H. Zha. Unsupervised deep learning for optical flow estimation. In *Proc. AAAI Conference on Artificial Intelligence (AAAI)*, volume 31. 2017. (Cited on page 11)

[SBK10] N. Sundaram, T. Brox, K. Keutzer. Dense point trajectories by GPU-accelerated large displacement optical flow. In *Poc. European Conference on Computer Vision (ECCV)*, Lecture Notes in Computer Science. Springer, 2010. URL http://lmb.informatik.uni-freiburg.de/Publications/2010/Bro10e. (Cited on page 39)

[SCH15] M. Savva, A. X. Chang, P. Hanrahan. Semantically-enriched 3D models for common-sense knowledge. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 24–31. 2015. (Cited on page 46)

[ST19] L. N. Smith, N. Topin. Super-convergence: Very fast training of neural networks using large learning rates. In *Proc. Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*, volume 11006, p. 1100612. International Society for Optics and Photonics, 2019. (Cited on page 59)

[SYLK18] D. Sun, X. Yang, M.-Y. Liu, J. Kautz. Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume. In *Proc. Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8934–8943. 2018. (Cited on pages 10, 15 and 64)

[TD20] Z. Teed, J. Deng. Raft: Recurrent all-pairs field transforms for optical flow. In *Proc. European Conference on Computer Vision (ECCV)*, pp. 402–419. Springer, 2020. (Cited on pages 9, 19, 20, 21, 30, 34, 43, 44 and 64)

[WWY+19] Y. Wang, P. Wang, Z. Yang, C. Luo, Y. Yang, W. Xu. Unos: Unified unsupervised optical-flow and stereo-depth estimation by watching videos. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8071–8081. 2019. (Cited on page 39)

[WYY+18] Y. Wang, Y. Yang, Z. Yang, L. Zhao, P. Wang, W. Xu. Occlusion aware unsupervised learning of optical flow. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4884–4893. 2018. (Cited on pages 11 and 64)

All links were last followed on May 11, 2021.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Stuttgart, 11.05.2021, 

Ort, Datum, Unterschrift