EECS 504: Computer Vision Final Project
DCFlow: Optical Flow via Cost Volume Processing

Hsuan-Cheng Chen
University of Michigan
hsuanc@umich.edu

Po-Kang Chen
University of Michigan
pkchen@umich.edu

Yen-Ning Tai
University of Michigan
tyenning@umich.edu

Abstract

We present an optical flow estimation algorithm described in [26]. The project can be divided into two parts. The first part describes how to generate feature embedding via a convolutional neural network. The second part is how the 4-dimensional cost volume can be efficiently constructed and stored using the embedding. We conduct experiments on Sintel benchmark to evaluate the cost volume with a winner-take-all matching algorithm. We also show that adding a normalization layer to the CNN model outperforms the original model.

1. Introduction

Optical flow is the pattern of apparent motion of objects, surfaces, and edges in a visual scene caused by the relative motion between an observer and a scene. It is also one of the key building blocks of many computer vision systems. Despite the large amount of literature discussing the topic, some of the main challenges remain unsolved. For example, occlusions, motion discontinuities and large displacements.

Two of the classic tasks in vision stereo disparities [23] and optical flow [12] can be cast as dense correspondence matching. The common way for doing so are making use of a cost volume, typically a 4D tensor of match costs between all pixels in a 2D image and their potential matches in a 2D search window. For stereo problem, the dimension of the cost volume can be reduced to 3D volumetric representations because the search window reduces to a epipolar line [10, 13]. In contrast, the cost volume for optical flow is four-dimensional and its explicit construction because for optical flow problem, there is no epipolar property. However, four-dimensional cost volume method was considered infeasible due to the lack of computational resource. For this reason, in the previous work optical flow methods commonly rely on nearest neighbor search [19] and coarse-to-fine analysis [21].

The computational requirements of four-dimensional cost volume approach appeared to render it impractical, due both to the construction of the cost volume and the optimization over it. In 2017, Accurate Optical Flow via Direct Cost Volume Processing [26] proposed a state of the art method to show that an optical flow algorithm can combine the convenience and accuracy of cost-volume processing with speed. Our group decided to implement this paper as the final project because we are motivated by the method proposed in it. To be more specific, their work is based on learning an embedding into a higher dimensional feature space, such that matching scores between patches can be computed by inner products in this space. This method can transform the RGB channel into high dimension to increase the distinction of each pixel. By doing so, the matching problem can be solved more accurately. They also show that the full four-dimensional cost volume can be constructed in a fraction of a second due to its regularity. After the improvement of matching problem, they further adapt semi-global matching [10] to the four-dimensional setting to enhance the robustness of matching. For final step post processing, it is performed by fitting homographies to image regions and using these to regularize the flow field but we do not implement this.

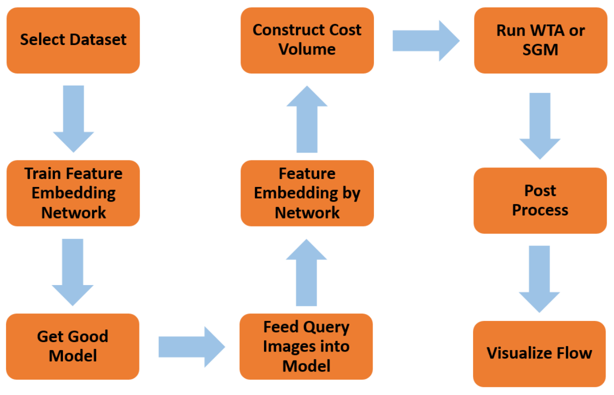

Figure 1. Pipeline diagram. This figure summarizes the procedure of our work.

To summarize our work in the paper, we make three main contributions:

- We evaluate the performance of normalized last layer in the neural network.
- We confirm that winner-take-all (WTA) can yield good results with the cost volume.
- We reproduce this paper in Python which has not been done before.

This paper is organized as follows. In Section 2, we review related work on large displacement optical flow We then present our method in Section 3 and the experiment in Section 4. Finally, Section 5 presents the conclusion. Source code is available here.

2. Related Works

Various approaches have tackled optical flow estimation since the work of Horn and Schunck [12]. This classic optical flow methods describes the consecutive pixel motion with respect to time caused by the relative movement. Two prevailed methods are respectively sparse and dense optical flow. The representing method of sparse optical flow is the well-known Lucas-Kanade[17] method. The intensities of each pixel, which could be described as $I(x, y, t) = I(x + dx, y + dy, t + dt)$, are assumed to be consistent. By taking the Taylor Series Approximation, the equation can be used to solve the x and y direction of pixel velocity, which could describe the pixel motion. Patch matching is one of the related topic of our work. The most common pixel-based matching costs include squared intensity differences (SD) [1][9][18][24] and absolute intensity differences (AD). In the video processing community, these matching criteria are referred to as the mean-squared error (MSE) and mean absolute difference (MAD) measures; the term displaced frame difference is also often used [25]. Other traditional matching costs include normalized cross-correlation [1][22], which behaves similar to sum-of-squared-differences (SSD). Another problem when dealing with the matching is the size of the search space. Several approach such as the nearest neighbor search was implemented in [19][2]. There are several recent works indicate that the complete cost volume is feasible and the structure supports the optimization method[3]. However, due to the high cost of the searching procedure, the cost volume construction is then implemented by us. Inspired by Full Flow [3] and [26], we implemented the 4D cost volume which could perform better as mentioned in [26]. Based on the regularity of the 4D cost volume, the semi-global matching [4][10] is then implemented. The Convolutional Neural Network trained with backpropagation[16] has frequently been used to perform large-scale image classification[15]. For patch-based feature extraction ,[7]extracts feature representations from CNNs. [27] trained a CNN with a Siamese architecture to predict the similarity of the corresponding patches. The implementation of the per-pixel predictions, which is same as the optical flow, is shown in several works such as [5][6][8]. One of solution to the problem is to utilize the sliding window method. It works well in various scenario but results in high computational cost. There has been work on machine learning techniques to optical flow before. [32] study statistics of optical flow and learn regularizers using Gaussian mixture. Principal components of a training set of flow fields were calculated in [4]. Other method concentrates on obtaining the occlusion probabilities by training classifiers[27].

3. Methods

3.1. Feature Embedding

To gain the information from patches, our goal is to embed those patches into a compact and informative feature space. In this way, it could not only distribute different features but also make the space being computed efficiently. This would benefit the 4D cost volume construction in the proceeding procedure. Upon these advantages, the fully-convolutional network is constructed. The network contains 4 convolutional layers. Each layer follows a pointwise rectifier linear unit(ReLU) to avoid negative values. All the size of the filters are 3×3. We do padding = 1 to remain the dimension of the features. The last layer of the network is constructed to be d filters and functions as to produced a unit-length feature vector $\mathbf{f} \in R^d$ such that $\|\mathbf{f}\|_2 = 1$. A feature embedding with unit length enables efficient distance computation and was shown in the next section.

A convolutional neural network $f : R^{9\times 9} \rightarrow \mathrm{R}^d$ was trained to embed the image patches into feature space. The θ is set as the parameter or the weight of the model. Some of the hyper parameters are as following. The learning rate of the training is set as $\alpha = 10^{-4}$ and the weight decay is set as $\times 10^{-4}$. The batch size is further set to be 16, and the optimizer is chosen to be Adam[14] because for our task it shows better performance than standard stochastic gradient descent. The following is the triplet loss we used to embed the patches:

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{|\mathcal{D}|} \sum_{i=1}^{|\mathcal{D}|} \Big[1 + \| f(\mathbf{x}_i^a; \boldsymbol{\theta}) - f(\mathbf{x}_i^p; \boldsymbol{\theta}) \|^2 - \| f(\mathbf{x}_i^a; \boldsymbol{\theta}) - f(\mathbf{x}_i^n; \boldsymbol{\theta}) \|^2 \Big] \tag{1}$$

As in Figure2, each of the image pairs, a random patch from the first image is then set a anchor $\mathbf{x}^a$. By using the ground-truth flow, the corresponding patch in the second image is then set as a positive patch $\mathbf{x}^p$. To gain the corresponding negative examples $\mathbf{x}^n$, three randomly selected

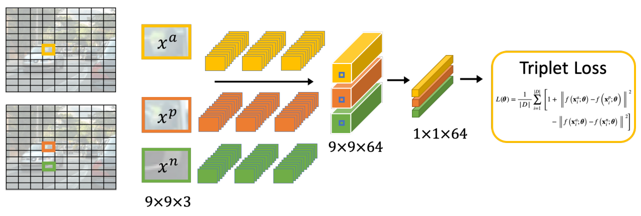
x^a
x^p
x^n
9×9×3
9×9×64
1×1×64
Triplet Loss
$L(\theta)=\frac{1}{|D|}\sum_{i=1}^{|D|}\left[1+\left\|f(\mathbf{x}_i^a;\theta)-f(\mathbf{x}_i^p;\theta)\right\|^2 - \left\|f(\mathbf{x}_i^a;\theta)-f(\mathbf{x}_i^n;\theta)\right\|^2\right]$

Figure 2. This figure shows the model of the feature embedding procedure for that $\mathbf{x}^a$ stands for the anchor patches, $\mathbf{x}^p$ stands for the positive patches, and $\mathbf{x}^n$ stands for the negative patches. The size of the feature would be $9 \times 9 \times 64$ after feature embedding. The center pixel feature vector of the patch is then set as the input of the triplet loss.

patches at the second image at the distance between 1 to 5 pixels are set to be $\mathbf{x}^n$. The distance from the anchor to the positive input is then minimized by this loss function, and the distance from the baseline (anchor) input to the negative input is maximized by the penalty setting.

3.2. Cost Volume Construction

To test the effectiveness of the feature embedding, we compute a feature embedding for each pixel within an image in a single forward pass through the trained convolutional neural network from the previous section. That is, the original $R^{H\times W\times 3}$ image data is encoded into $R^{H\times W\times d}$ feature space, where d is the dimensionality of feature vector, equal to 64 as in [26]. Then we normalize the length of each feature embedding $f \in R^{64}$, such that $\|f\|_2 = 1$. A feature embedding with unit length enables efficient distance computation, where $C(f_1, f_2)$ is the matching cost between f_1 and f_2, or say $pixel_1$ and $pixel_2$.

$$C(f_1, f_2) = \frac{1}{2}\|f_1 - f_2\|_2 = 1 - dot(f_1, f_2) \tag{2}$$

Here we take advantage of the connection between Euclidean Distance and the inner dot product. Also, Being able to use vector products has several benefits, including parallel computing and the existing optimizations within various programming languages.

With the efficient way to compute the matching cost between each feature embedding, we populate the 4-dimensional cost volume with the matching costs. Let the input be two images $I^1, I^2 \in R^{H\times W\times 3}$, and disparities or flow field be $V \in R^{H\times W\times 2}$ between I_1 and I_2. By assuming that $V_p \in |R|^2$, where

$$|R| = \{-r\_max, ..., -1, 0, 1, 2, ..., r\_max - 1, r\_max\} \tag{3}$$

and r_max is the preset maximal displacement. We can construct the optical flow cost volume as:

$$C\_vol(p, \mathbf{v}) = 1 - dot(F^1_p, F^2_{p+\mathbf{v}}) \tag{4}$$

Table 1. Running Time

Methods	Loops	NumPy	PyTorch
Time	$> 2\,hours$	$\sim 35\ min$	$4\ min$

where $C\_vol(p, \mathbf{v})$ is the cost between pixel $p \in I^1$ and pixel $p + \mathbf{v} \in I^2$. Note that $\mathbf{v}$ represents the displacement - V_p - of pixel p on the 2D plane. The whole algorithm for populating the cost volume is in [Algorithm 1].

Algorithm 1 Populate the cost volume

```
Data: $F^1, F^2 \in R^{H\times W\times d}$
Result: $C\_vol \in R^{H\times W\times |R|^2}$
1: for $i = 1 : H$ do
2:   for $j = 1 : W$ do
3:     for $v_1 = -r\_max : r\_max$ do
4:       for $v_2 = -r\_max : r\_max$ do
5:         $p = (i, j)$
6:         $\mathbf{v} = (v_1, v_2)$
7:         $C\_vol(p, \mathbf{v}) = 1 - dot(F^1_p, F^2_{p+\mathbf{v}})$
8:       end for
9:     end for
10:  end for
11: end for
```

Algorithm 2 Vectorized Algorithm

```
Data: $F^1, F^2 \in R^{H\times W\times d}$
Result: $C\_vol \in R^{H\times W\times |R|^2}$
1: for $i = 1 : H$ do
2:   for $j = 1 : W$ do
3:     $p = (i, j)$
4:     // $\mathbf{V}$ includes all the disparities
5:     $C\_vol(p, \mathbf{V}) = 1 - dot(F^1_p, F^2_{p+\mathbf{V}})$
6:     // $C\_vol(p, \mathbf{V}) \in R^{1\times 1\times |R|^2}$
7:     // $F^1_p \in R^d, F^2_{p+\mathbf{V}} \in R^{d\times |R|^2}$
8:   end for
9: end for
```

Without parallelization, the time complexity of the above algorithm is $O(H \times W \times R^2 \times d)$, with computing each entry in the cost volume requires $O(d)$ time, which is equal to the length of feature embedding. The maximal displacement assumption r_max and the dimensionality d of the feature embedding affects the computation cost with quadratic time and linear time respectively.

In practice, having the time complexity as high as $O(H \times W \times R^2 \times d)$ makes the algorithm useless. As shown in the table, cost volume construction with nested loop structure takes more than 2 hours to match a pair of images with size $(480, 640)$. Fortunately, it is possible to vectorize the algorithm as in [Algorithm 2], where $F^1_p \in R^d$

and $F^2_{p+\mathbf{V}} \in R^{d\times|R|^2}$. Therefore, $|R|^2$ cost volume entries are populated during each iteration via the matrix manipulations. This adaption significantly reduces the required running time for cost volume construction.

Considering the fact that 35 minutes still makes the algorithm infeasible, we rewrite the algorithm with PyTorch. By assigning feature embeddings to Torch Tensor on a CUDA device, we utilize the power of GPU to accelerate the cost volume construction process. In Google Colab, the PyTorch implementation of the algorithm is about 8 times faster than the NumPy implementation, reducing the running time from half an hour to less than 4 minutes.

Also, since the cost volume is 4-dimensional, a very large array to store is required to store it. During the experiments, we notice that the default data type for NumPy arrays, which is float, is too big that the process can easily run out all of the memories. For example, in Google Colab environment, the session would crash as the provided 25GB memory is exhausted. Therefore, we follow the suggestions in [26] to rescale the $C\_vol(p, \mathbf{V}_p)$ to 8 bit integer range and pick 8 or 16 bit unsigned integer as the data type for cost volume arrays.

3.3. Cost Volume Processing

After populating the cost volume, we have to perform image matching to determine the best 2-dimensional flow for each pixel. [26] claims that winner-take-all (WTA) philosophy already achieves good results without further processing. There is also another Flow-SGM algorithm which is based on the semi-global-matching algorithm.

3.3.1 Winner-Take-All

The WTA method is simple and straightforward yet can generate sufficiently good results as claimed in [26]. The fundamental idea behind WTA is to pick the disparity with the least value in the cost volume.

$$\mathbf{V}_p = argmin_{\mathbf{v}\in|R|^2}(C\_vol(p, \mathbf{v})) \tag{5}$$

3.3.2 Semi Global Matching

In addition to the WTA method, [26] also adopted the optimization-based semi-global matching (SGM) algorithm in [11] to further process the 4-dimensional cost volume. The discrete energy field for the optical flow field V is defined as:

$$E(V) = \sum_p (C(p, \mathbf{V}_p) + \sum_{q\in N(p)} P_1 \times [|\mathbf{V}_p - \mathbf{V}_q| = 1] + \sum_{q\in N(p)} P_2 \times [|\mathbf{V}_p - \mathbf{V}_q| > 1]) \tag{6}$$

, where $[\cdot]$ denotes the Iverson bracket. The first term is the sum of all pixel matching costs for the flow field V. The second term adds a constant penalty P_1 for all pixels q in the neighborhood $N(p)$ of p, for which the disparity changes a little bit (i.e. 1 pixel). For all larger disparity changes, the third term adds a larger penalty P_2, which is defined as:

$$P_2^{p,q} = \begin{cases} P_2/Q & ,\text{if } \ |I^1_p - I^1_q| \geq T \\ P_2 & , \ otherwise \end{cases}$$
$$where\, P_2,\ Q,\ T\ \in constants \tag{7}$$

Equation 7 is a modification from [11] to further support edge-aware smoothing of the cost volume. The energy of optical flow field V can be optimized with dynamic programming like techniques.

Although [26] presented breath-taking results with the Flow-SGM algorithm, we are unable to run experiments with our SGM implementation. The running time for the algorithm is much longer than we would like to see even with dynamic programming optimization. The complexity analysis shows that the running time for SGM is $O(M \times N \times Mu)$, where Mu denotes the time complexity for some matrix operations on $arrays \in R^{|R|^2\times|R|^2}$. We did not find a way to vectorize the algorithm, so the previous trick using PyTorch and CUDA device is not applicable here. Currently, we believe the only feasible solution is to implement the algorithm directly on GPU, using OpenCL as in [26] or CUDA programming language, which is summarized in future works.

4. Experiments

In this section, we evaluate the presented approach on the MPI Sintel dataset. MPI Sintel is a challenging dataset with large displacement, motion blur, and non-rigid motion [6]. First, we show ten results from our best trained model and give the accuracy of each test data. The accuracy is defined by the root-mean-square deviation. We calculate the error between our result and the ground truth. Next, we evaluate the performance of each training model. We expect to see the relationship between the effect of feature embedding and the matching accuracy.

4.1. Feature Embedding with Different Models

We use Kitti Optical Flow Evaluation 2015 dataset to train models. Prepossessing data is the first step for training. We cropped the 400 images into 600k patches including anchor, positive and negative patches to form 200k set of triplet (see section 3.1 for more details). The optimizer is Adam and the learning rate is 10^{-4}. We train our model less than 100 epochs and evaluate the model performance by predicted flow. Table 2 shows the result of our work including normalized features or unnormalized features in the network.

Num(o/n)	T = 0.5	T = 0.7	T = 1	T = 1.5	T = 2
200(o)	0.507	0.845	0.920	0.936	0.954
200(n)	0.508	0.847	0.928	0.937	0.955
2000(o)	0.506	0.843	0.918	0.935	0.953
2000(n)	0.526	0.862	0.924	0.940	0.957
10000(o)	0.501	0.839	0.915	0.934	0.953
10000(n)	0.523	0.857	0.920	0.939	0.958

Table 2. The table compares the accuracy of original and normed feature with different threshold number. It further indicates the effectiveness of different numbers of the patches used to train the model. T stands for threshold in Meter, ”o” stands for original, and ”n” stands for normalized feature vectors.

4.2. Flow Estimation Results

In this subsection, we report the flow field estimated with our best model. This model is trained on 200k patches for 30 epochs. Ten pictures are selected from Sintel dataset. The motion of data 1 to 6 are relatively small compared to the motion of data 7 to 10. The receptive field of the CNN is a 9 x 9 window. We use a feature dimensionality of d = 64 and construct the cost volume as described in Section 3.1. See Table 3 for the accuracy and See Fig 3 for visualization.

Num	T = 0.5	T = 0.7	T = 1	T = 1.5	T = 2
1	0.532	0.862	0.919	0.938	0.959
2	0.729	0.831	0.901	0.945	0.965
3	0.654	0.835	0.893	0.949	0.971
4	0.723	0.916	0.947	0.967	0.974
5	0.495	0.766	0.847	0.875	0.892
6	0.502	0.677	0.752	0.803	0.831
7	0.496	0.593	0.634	0.643	0.646
8	0.737	0.796	0.827	0.877	0.889
9	0.687	0.708	0.745	0.782	0.800
10	0.191	0.287	0.363	0.422	0.453

Table 3. The table compares the accuracy of our best model and ground truth flow with different threshold number. T stands for threshold in Meter. The performance of large motion are worse than relatively small motion

4.3. Discussion

First, we will focus on the accuracy calculated with $T \geq 1$ because the flow field ground truth is not integer, which means there may be some inevitable errors when computing the accuracy. In the first 5 rows from Table 3, our model successfully estimates the flow field for about 90% of pixels within a range of 1 pixel. About 96% of pixels can be found within a search radius $= 2$. Next, we observe that for images with large disparities, the flow estimation can be unstable, as data 9 and 10 shown in Fig. 3. A possible explanation is that since we implement the matching algorithm based on WTA philosophy, the global structure consistency is not well exploited, comparing to the global optimization methods or SGM.

4.3.1 How to Get Feature Embedding with Unit-Length

To compute cost volume with vector dot product, we have to normalize the feature embedding to unit length as described in section 3.2. In [26], the feature vectors are normalized outside the feature embedding prediction model. In contrast, we designed a convolution neural network in which the last layer normalize the prediction, which is the feature embedding with a 9×9 receptive field. The reason for the network modification is that the feature embedding is predicted through a network trained with triplet loss which computes the distance between feature vectors, which may not have unit length, in the original work. We notice that treating feature embedding with additional normalization may distort the distance between vectors in the feature space. On the other hand, if we add another normalization layer in the convolutional neural network, then the triplet loss is based on vectors with unit length. Hence, the feature embedding used to populate cost volume can represent patch difference that is well handled when training the feature embedding prediction model. The comparison between two feature embedding prediction models can be found in Table 2. Under all circumstances, the network directly predicting unit-long feature vectors consistently outperforms the one that requires further normalization afterwards.

4.3.2 Post Processing

In the experiments, we only apply the basic median filtering on the estimated flow field. This already produces good results. However, in practice, there are a lot of tricks to further refine the final results. People combine different techniques to achieve state-of-the-art performance. For example, forward-backward consistency checking helps remove low confidence predictions, generating an occlusion map as part of the results. The occluded areas can be compensated with proper interpolation or inpainting techniques. Another popular technique aims to separate large displacement from small disparity to increase the flow estimation precision within each iteration, such as coarse-to-fine method and Epic-Flow framework [20].

5. Conclusion

We have presented the flow estimation algorithm described in [26]. We have described how to train and analyze a CNN model with a 9×9 receptive field to predict pixel

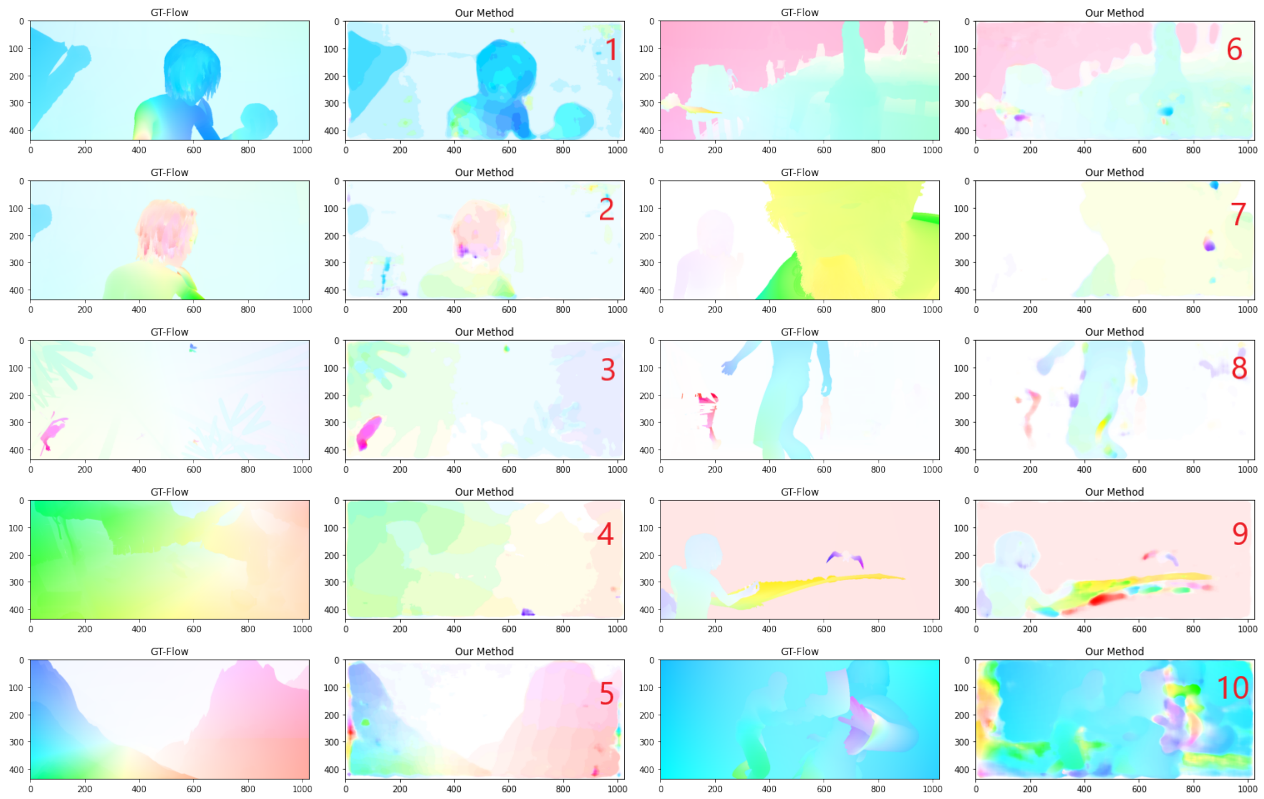

Figure 3. Examples of optical flow prediction on the Sintel dataset. Data 1 to 6 are small motion while Data 7 to 10 are large motion.

feature embedding. Also, we have showed that adding normalization to the last layer of the network can increase the performance. Next, we have discussed how to efficiently construct and store a 4-dimensional cost volume. A WTA philosophy based matching algorithm is also implemented to estimate flow fields and help evaluate the cost volume.

One of the main reasons that we chose to implement this paper is that there is no Python implementation published before. As most recent machine learning or computer vision works are done in Python, we believe this project can provide a more fair and convenient opportunity to compare DCFlow and other flow estimation algorithms.

For future works, we plan to conduct forward-backward consistency checking and apply interpolation and inpainting techniques to the occluded areas. Also, we can reimplement the algorithm directly on a GPU while keeping a Python interface. So that the project can remain easy to use and other time-consuming methods can be executed within acceptable running time, such as SGM and EpicFlow.

References

[1] P. Anandan. A computational framework and an algorithm for the measurement of visual motion. *International Journal of Computer Vision*, 2:283–310, 2004.

[2] Christian Bailer, Bertram Taetz, and Didier Stricker. Flow fields: Dense correspondence fields for highly accurate large displacement optical flow estimation, 2015.

[3] Qifeng Chen and Vladlen Koltun. Full flow: Optical flow estimation by global optimization over regular grids. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4706–4714, 2016.

[4] Amnon Drory, Carsten Haubold, Shai Avidan, and Fred A. Hamprecht. Semi-global matching: A principled derivation in terms of message passing. In *GCPR*, 2014.

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

[6] Clément Farabet, Camille Couprie, Laurent Najman, and Yann LeCun. Learning hierarchical features for scene labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35:1915–1929, 2013.

[7] Philipp Fischer, Alexey Dosovitskiy, and Thomas Brox. Descriptor matching with convolutional neural networks: a comparison to sift. *ArXiv*, abs/1405.5769, 2014.

[8] Yaroslav Ganin and Victor S. Lempitsky. N^4-fields: Neural network nearest neighbor fields for image transforms. In *ACCV*, 2014.

[9] Marsha Jo Hannah. Computer matching of areas in stereo images. 1974.

[10] Heiko Hirschmüller. Stereo processing by semi-global matching and mutual information. 2007.

[11] H. Hirschmuller. Stereo processing by semiglobal matching and mutual information, 2008. IEEE Transactions on Pattern Analysis and Machine Intelligence.

[12] Berthold K. P. Horn and Brian G. Schunck. Determining optical flow. *Artif. Intell.*, 17:185–203, 1980.

[13] Alex Kendall, Hayk Martirosyan, Saumitro Dasgupta, and Peter Henry. End-to-end learning of geometry and context for deep stereo regression. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 66–75, 2017.

[14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[16] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

[17] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *IJCAI*, 1981.

[18] Larry H. Matthies, Takeo Kanade, and Richard Szeliski. Kalman filter-based algorithms for estimating depth from image sequences. *International Journal of Computer Vision*, 3:209–238, 1989.

[19] Moritz Menze, Christian Heipke, and Andreas Geiger. Discrete optimization for optical flow. In *GCPR*, 2015.

[20] Jerome Revaud, Philippe Weinzaepfel, Zaid Harchaoui, and Cordelia Schmid. Epicflow: Edge-preserving interpolation of correspondences for optical flow, 2015. IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[21] Jérôme Revaud, Philippe Weinzaepfel, Zaïd Harchaoui, and Cordelia Schmid. Deepmatching: Hierarchical deformable dense matching. *International Journal of Computer Vision*, 120:300–323, 2016.

[22] Thomas W. Ryan, Robert T. Gray, and Bobby R. Hunt. Prediction of correlation errors in stereo-pair images. 1980.

[23] Daniel Scharstein and Richard Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, 47:7–42, 2001.

[24] Eero P. Simoncelli, Edward H. Adelson, and David J. Heeger. Probability distributions of optical flow. *Proceedings. 1991 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 310–315, 1991.

[25] A. Murat Tekalp. Digital video processing. 1995.

[26] J. Xu, R. Ranftl, and V. Koltun. Accurate optical flow via direct cost volume processing, 2017. IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[27] Jure Zbontar and Yann LeCun. Computing the stereo matching cost with a convolutional neural network. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1592–1599, 2015.