University of Michigan
EECS 442: Foundations of Computer Vision
Fall 2021. Instructor: Andrew Owens.

Problem Set 10: Optical Flow and Epipolar Geometry

Posted: Wednesday, Dec 1, 2021 **Due:** Wednesday, Dec 8, 2021

Please convert your Colab notebook to a PDF file and submit the PDF file to Gradescope, *make sure to label your answers*. We have included the PDF conversion script at the end of the notebook. Nothing needs to be submitted to Canvas.

The starter code can be found at:
https://colab.research.google.com/drive/1IxO7g27c31rZqtwG8HuWvHU_3goho6ZY?usp=sharing

We recommend editing and running your code in Google Colab, although you are welcome to use your local machine instead.

Problem 10.1 *Optical flow*

Algorithm 1 PWC-Net(I_1, I_2)

1: Build 7-level feature pyramids c_1, c_2 from I_1, I_2.
2: Compute a cost volume using c_1^l, c_2^l and then estimate the optical flow $\mathbf{w}^l$ at the coarsest pyramid level $l = 6$.
3: **for** `level` l `= 5,4,3,2,` **do**
4: Warp the feature map c_2^l to obtain c_w^l, using upsampled coarse flow from $\mathbf{w}^{l+1}$.
5: Compute a cost volume using c_1^l and c_w^l, according to Equation (2).
6: Estimate the optical flow $\mathbf{w}^l$.
7: **end for**
8: Refine the final optical flow $\mathbf{w}^2$ using a context network.
9: **return** $\mathbf{w}^2$

In this problem, you will complete an implementation of PWC-Net [1], a neural network that estimates the optical flow estimation between a pair of images I_1 and I_2. Your solution will only require approximately *one line of code*! There are three key components in the PWC-Net, namely, **P**yramid, **W**arping and **C**ost-volume, which will be described in more detail below. Fig. 1 shows an overview of the network structure. You should refer to the `PWCNet` in the provided notebook for further implementation details.

Computing optical flow with PWC-Net. The way that PWC-Net computes optical flow closely resembles the "classic" multi-scale optical flow pipeline we discussed in class.

However, instead of matching images by directly comparing pixel intensities, it uses learned CNN features (Algorithm 1).

PWC-Net matches the images in a multi-level, coarse-to-fine manner. It extracts multi-level feature pyramids from both images using fully convolutional networks and then computes a *cost volume* at each level. This is a volume $C(x, y, \Delta u, \Delta v)$ that measures the similarity between the features at position (x, y) in I_1 and the feature $(x + \Delta u, y + \Delta v)$ in I_2, and is computed by taking the dot product between the features in the two feature maps. We then process the cost volume using multiple convolution layers to estimate the flow at that level.

We'll repeat this procedure at multiple levels. After estimating a flow $\mathbf{w}^{l+1} = (u_{l+1}, v_{l+1})$ at level $l + 1$, when we repeat our search in subsequent levels, we'll *recenter* our search on these new flow locations, i.e. we'll search for Δu and Δv only within a small window around $(x + u_{l+1}, v_{l+1})$. For simplicity and speed, we do this by *warping* the features in the second image, moving the matched features at position $(x + u_{l+1}, y + v_{l+1})$ to (x, y). This allows us to simply write a function that simply performs the match in a small window around each feature.

As in classic flow methods, we'll perform this iterative computation in conjunction with a *pyramid*. We'll compute optical flow at coarse levels of the pyramid first, and use our flow estimates to initialize finer levels of the pyramid. We'll use different layers of a CNN as our feature pyramid representation.

PWC-Net components. While we've given you a large system to implement, you'll only need to implement two pieces of it: setting up the flow with right scale for warping at each level, and computing the cost volume. For reference, we provide more information about the important components of PWC-Net below:

- **Pyramid:** PWC-Net uses a shared encoder to generate two L-level feature pyramids, one for input image I_1 as $\{c_1^l; l = 0, 1, \ldots L - 1\}$, one for input image I_2 as $\{c_2^l; l = 0, 1, \ldots L - 1\}$. The bottom (0-th) level of the pyramid is the input image, i.e., $c_t^0 = I_t$, where $t = 1, 2$. To generate feature representation at the l-th layer, c_t^l, we use layers of convolutional filters to downsample the features at the $(l - 1)$-th pyramid level, c_t^{l-1}, by a factor of 2. The extracted feature pyramids of two images are then used to estimate optical flow in a coarse to fine manner. In this problem set, we use feature pyramids with $L = 7$.

- **Warping:** The network estimates multi-scale flows at level $l = 2, 3, 4, 5, 6$, where the flow at level l has a downsampling rate of 2^l. At the l-th level, PWC-Net warps the features of the second image toward the first image using the $\times 2$ upsampled flow from the $(l + 1)$-th level:

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

  where $\mathbf{x}$ is the pixel index and the upsampled flow $\mathrm{up}_2\left(\mathbf{w}^{l+1}\right)$ is set to be zero at the top level. As the large pixel motions are compensated by the warping operation, it allows the pixel correspondence search between two images to be done efficiently using a cost-volume (see next part) with small search range.

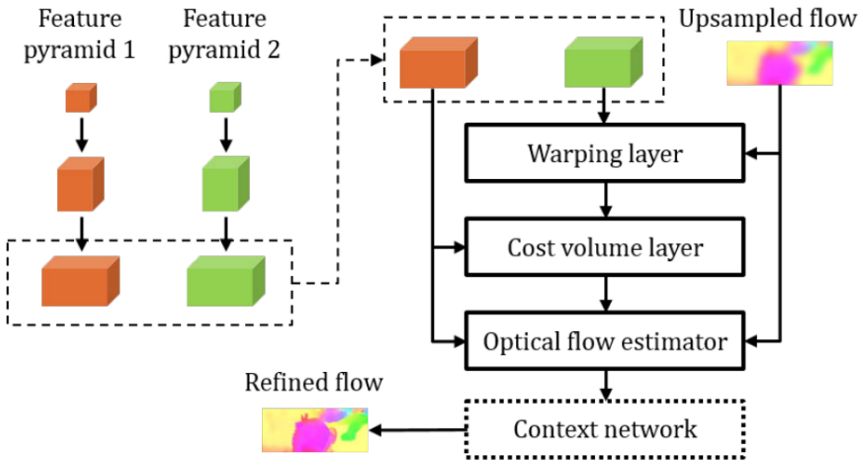

Figure 1: Feature pyramid and refinement at one pyramid level by PWC-Net. PWC-Net warps features of the second image using the upsampled flow, computes a cost volume, and process the cost volume using CNNs. The context network is only used in the last level of the pyramid.

- **Cost volume:** We use the features to construct a cost volume that stores the matching costs for associating a pixel with its corresponding pixels in the other image. We define the cost-volume as the correlation between features of the first image $\mathbf{c}_1^l$ and warped features of the second image $\mathbf{c}_w^l$.

  In this problem set, we computes the cost-volumes at pyramid level $l = 2, 3, 4, 5, 6$.

Implementing PWC-Net. Your task is to complete the implementation of PWC-Net. Most of the code has been implemented for you, you only need to write one line of code for part (b). Optionally, you can also apply your optical flow algorithm to *motion magnification* [2].

(a) **(0 points)** The warping layer at each pyramid level warps the feature of the second image towards the first image using the coarse flow estimated from the previous level. Fill in the scaling factors for `self.warp(...)` appearing inside the `forward` function of the `PWCNet`.

Hint: The optical flow magnitude in pixel is 2 times smaller when the resolution is downsampled by 2.

(b) **(3 points)** Complete the implementation of cost-volume layer. The cost-volume **cv** returned by the cost-volume layer is a 4D tensor of shape $[B, (2 \times \text{MaxD} + 1)^2, H, W]$, where MaxD is the maximum search radius. Its calculation is illustrated in Fig. 2. $\mathbf{c}_1^l$ is zero-padded on all four borders by MaxD. $\mathbf{c}_w^l$ is sliding across all possible locations within the padded $\mathbf{c}_1^l$, in the same manner as a usual convolution filter. At each sliding location, a similarity map of size $[B, 1, H, W]$ is computed as:

$$\mathbf{cv}^l(\mathbf{x}) = \frac{1}{N}\langle \tilde{\mathbf{c}}_1^l(\mathbf{x}), \mathbf{c}_w^l(\mathbf{x}) \rangle, \tag{2}$$

where $\tilde{\mathbf{c}}_1^l$ is the patch of the feature map $\mathbf{c}_1^l$ that is covered by $\mathbf{c}_w^l$, N is the channel numbers, and $\langle \dots \rangle$ is the inner product. After implementing the cost-volume layer, run the verification code provided to check your implementation.

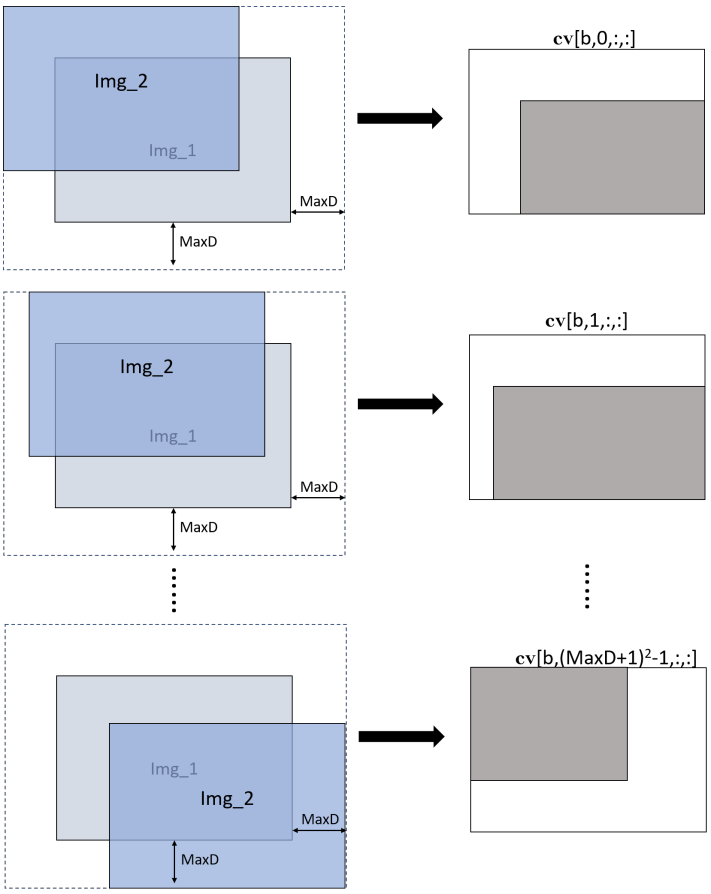

Figure 2: Illustration of the cost-volume computation.

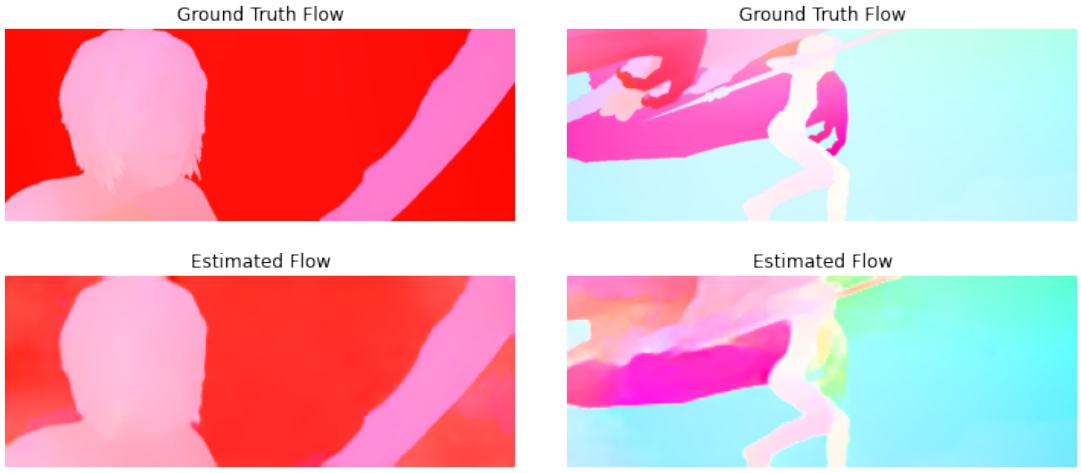

Figure 3: Visualization of estimated optical flow, color-coded using the visualization method described in [3].

(c) **(0 points)** Train the network on single image pair to verify your network is working properly. The final training end-point error should be around 0.04.

(d) **(0 points)** Since training the network on the full dataset takes long time, we have provided the trained model for you. Load the trained model and run the algorithm on the test images supplied with the code. Report the test end point error (EPE) and

visualize the result. Fig. 3 shows the estimated optical flow image. You EPE on the test dataset should be 4.06.

(e) **(0 points)** One application of the optical flow is motion magnification. Run the provided code block and visualize the saved 'movie.gif' file. Note that the motion between video frames are magnified.

Problem 10.2 *Epipolar Geometry*

In this problem, you will plot *epipolar lines*. Given an input pixel in a *reference view*, we will plot the line containing all of its possible correspondences in a second view.

The reference view will be located at the origin, with an identity rotation matrix:

$$P_1 = K[I \mid 0], \tag{3}$$

while the second view's projection matrix is given by:

$$P_2 = K[R \mid t]. \tag{4}$$

Both cameras have the same intrinsics matrix K. For a review of epipolar geometry, please see this book chapter from Hartley and Zisserman.

(a) **(3 points)** Fill in the `coordinate_transform` function. First, obtain the ray $\mathbf{v} = K^{-1}p$ that connects the reference camera's center of projection to pixel p on the image plane (using homogeneous coordinates for p). We will walk along the ray at various distances $d_1, d_2, ...d_N$. For each one, create a 3D point $\mathbf{X}_i = d_i\mathbf{v}$. Project it into the second view. The provided code will then draw a dot at this position. After processing all N depth values, you should see a line. The correspondence to p should lie somewhere on this line.

(b) **(0 points)** Fill in the `compute_fundamental_matrix` function. We will estimate the fundamental matrix $F \in \mathbb{R}^{3\times3}$ from camera pose. Note that it is also possible to directly estimate F from correspondences, without the full camera pose. First compute the epipole:

$$\mathbf{e} = KR^T\mathbf{t}. \tag{5}$$

We will then compute F between another image frame with respect to the reference frame.

$$F = K^{-T}RK^T[e]_\times, \tag{6}$$

where $[e]_\times$ is the cross product of the epipole.

(c) **(0 point)** Fill in the `visualize_epipolar_line` function. For any pixel $\mathbf{p}$ in the reference frame, we can compute the line $\mathbf{u} = F\mathbf{p}$. The pixel in the other image that corresponds to $\mathbf{p}$ falls along $\mathbf{u}$. The vector $\mathbf{u} = [a, b, c]^T$ represents a line in the form $ax + by + c = 0$.

References

[1] Deqing Sun, Xiaodong Yang, Ming-Yu Liu, and Jan Kautz. Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8934–8943, 2018.

[2] Ce Liu, Antonio Torralba, William T Freeman, Frédo Durand, and Edward H Adelson. Motion magnification. *ACM transactions on graphics (TOG)*, 24(3):519–526, 2005.

[3] Simon Baker, Daniel Scharstein, JP Lewis, Stefan Roth, Michael J Black, and Richard Szeliski. A database and evaluation methodology for optical flow. *International journal of computer vision*, 92(1):1–31, 2011.