Deep Learning for Optical Flow Estimation FlowNets & SPyNet

Jianping Fan
Department of Computer Science
UNC-Charlotte

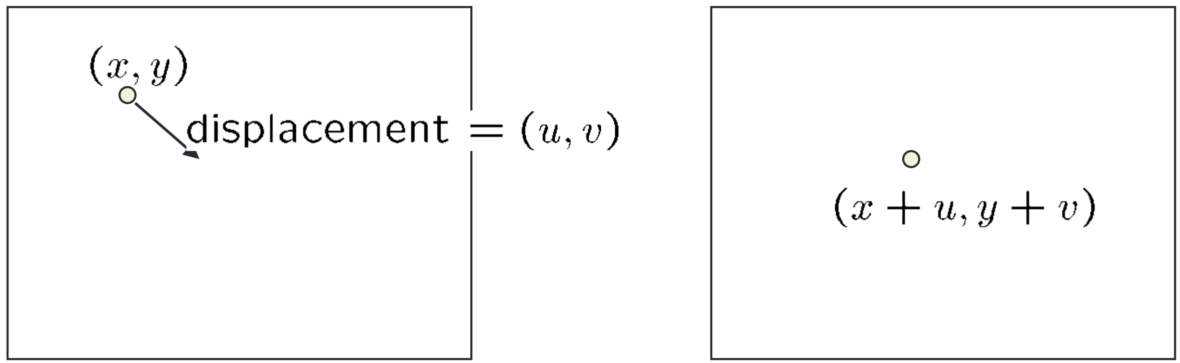
Course Website:
http://webpages.uncc.edu/jfan/itcs5152.html

FlowNet 1.0 & FlowNet 2.0

Convolutional neural networks (CNNs) have made great contributions to various computer vision tasks. Recently, CNNs have been successfully used in estimating optical flow. Compared with traditional methods, these methods achieved a large improvement in quality.

Both FlowNet1.0 and FlowNet2.0 are end-to-end architectures. FlowNet2.0 is stacked by FlowNetCorr and FlowNetS, and has much better results than both of FlowNetCorr and FlowNetS. FlowNetS simply stacks two sequentially adjacent images as input, while in FlowNetCorr, two images are convoluted separately, and are combined by a correlation layer. In a spatial pyramid network, the authors trained one deep network for each level independently to compute the flow update. Both the SPyNet and FlowNet2.0 estimate large motions in a coarse-to-fine manner. FlowNet2.0 has the best performance among these architectures, and SPyNet has the least model parameters.

FlowNet 1.0 & FlowNet 2.0

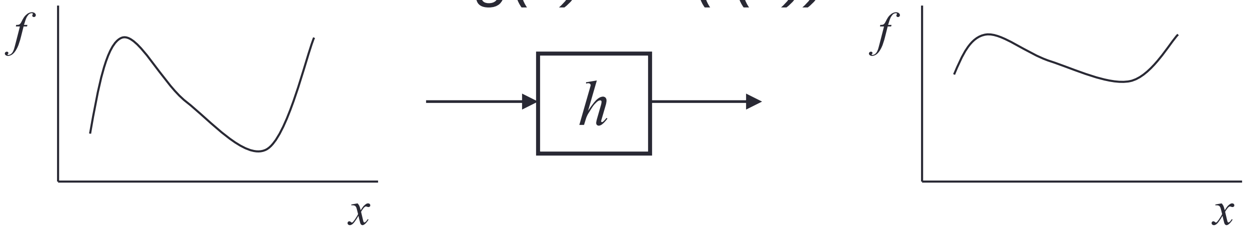


$I(x, y, t)$ $\qquad$ $I(x, y, t+1)$

$$
\begin{aligned}
0 &= I(x+u, y+v, t+1) - I(x, y, t) \\
&\approx I(x, y, t+1) + I_x u + I_y v - I(x, y, t) \\
&\approx [I(x, y, t+1) - I(x, y, t)] + I_x u + I_y v \\
&\approx I_t + I_x u + I_y v \\
&\approx I_t + \nabla I \cdot < u, v >
\end{aligned}
$$

Image Warping

- image filtering: change *range* of image
  - $g(x) = h(f(x))$



- image warping: change *domain* of image
  - $g(x) = f(h(x))$

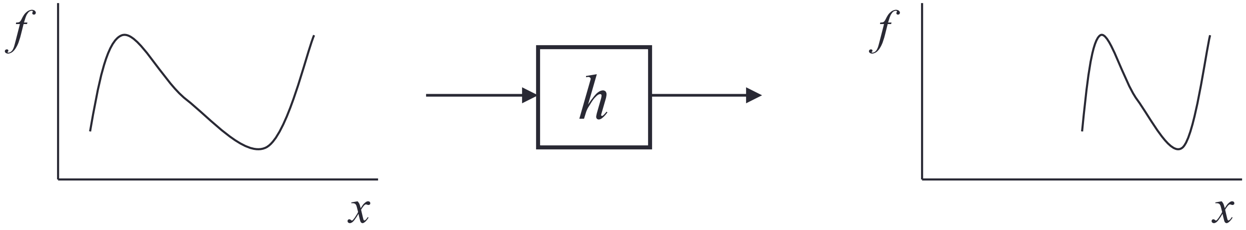
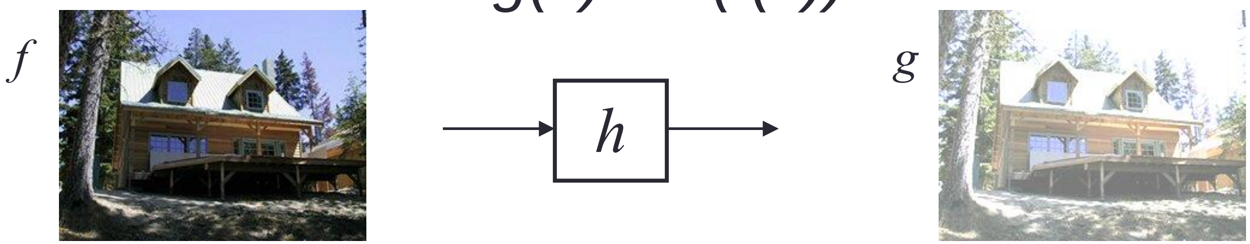
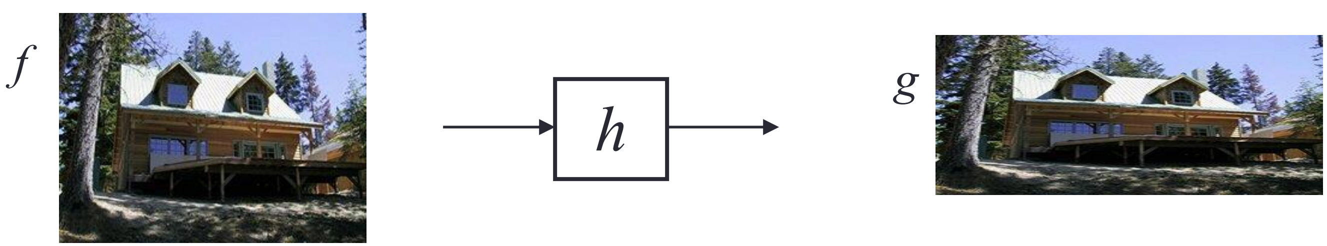


Image Warping

- image filtering: change *range* of image
  - $g(x) = h(f(x))$



- image warping: change *domain* of image
  - $g(x) = f(h(x))$



Parametric (global) warping

- Examples of parametric warps:

translation

rotation

aspect

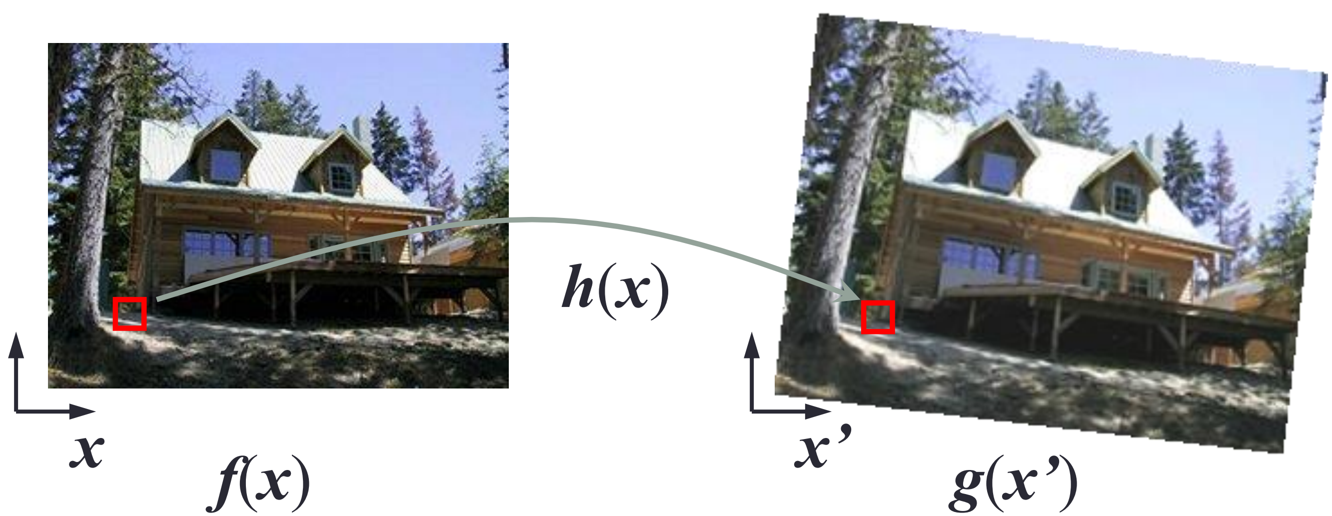
affine

perspective

cylindrical

Image Warping

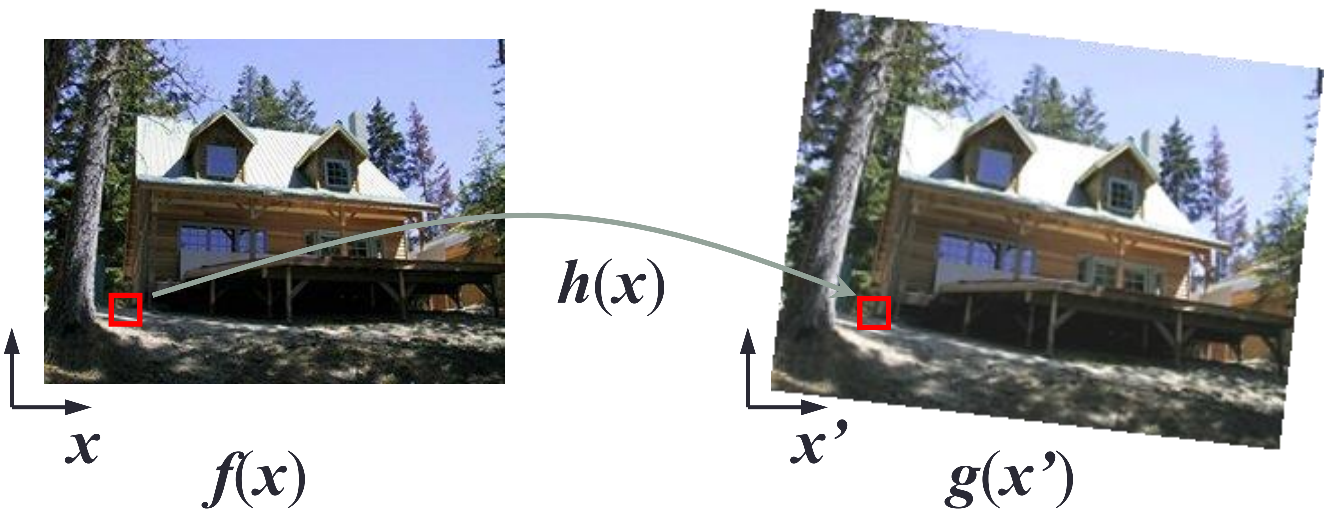
- Given a coordinate transform $\boldsymbol{x'} = \boldsymbol{h}(\boldsymbol{x})$ and a source image $\boldsymbol{f}(\boldsymbol{x})$, how do we compute a transformed image $\boldsymbol{g}(\boldsymbol{x'}) = \boldsymbol{f}(\boldsymbol{h}(\boldsymbol{x}))$?



Forward Warping

- Send each pixel ***f***(***x***) to its corresponding location ***x'*** = ***h***(***x***) in ***g***(***x'***)

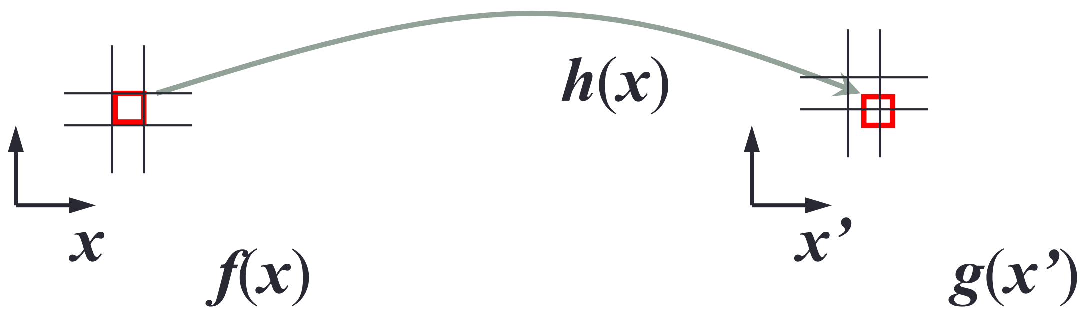
- What if pixel lands "between" two pixels?



Forward Warping

- Send each pixel $\boldsymbol{f}(\boldsymbol{x})$ to its corresponding location $\boldsymbol{x'} = \boldsymbol{h}(\boldsymbol{x})$ in $\boldsymbol{g}(\boldsymbol{x'})$

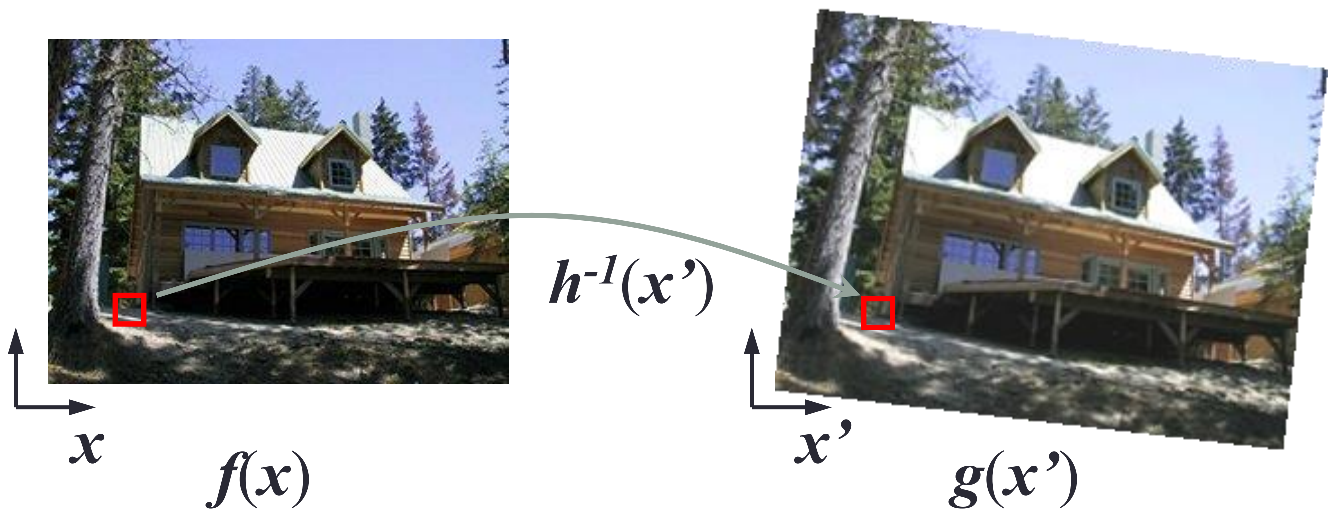
- What if pixel lands “between” two pixels?
- Answer: add “contribution” to several pixels, normalize later (*splatting*)



Inverse Warping

- Get each pixel $\boldsymbol{g}(\boldsymbol{x'})$ from its corresponding location $\boldsymbol{x} = \boldsymbol{h}^{-1}(\boldsymbol{x'})$ in $\boldsymbol{f}(\boldsymbol{x})$

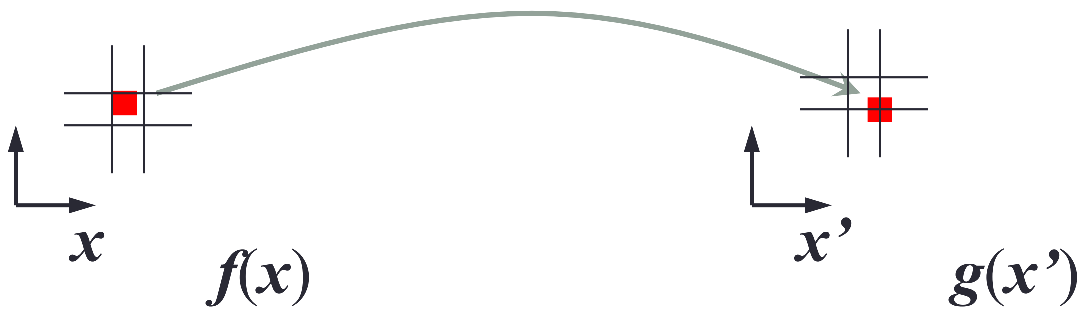
- What if pixel comes from “between” two pixels?



Inverse Warping

- Get each pixel $\boldsymbol{g}(\boldsymbol{x'})$ from its corresponding location $\boldsymbol{x} = \boldsymbol{h}^{-1}(\boldsymbol{x'})$ in $\boldsymbol{f}(\boldsymbol{x})$

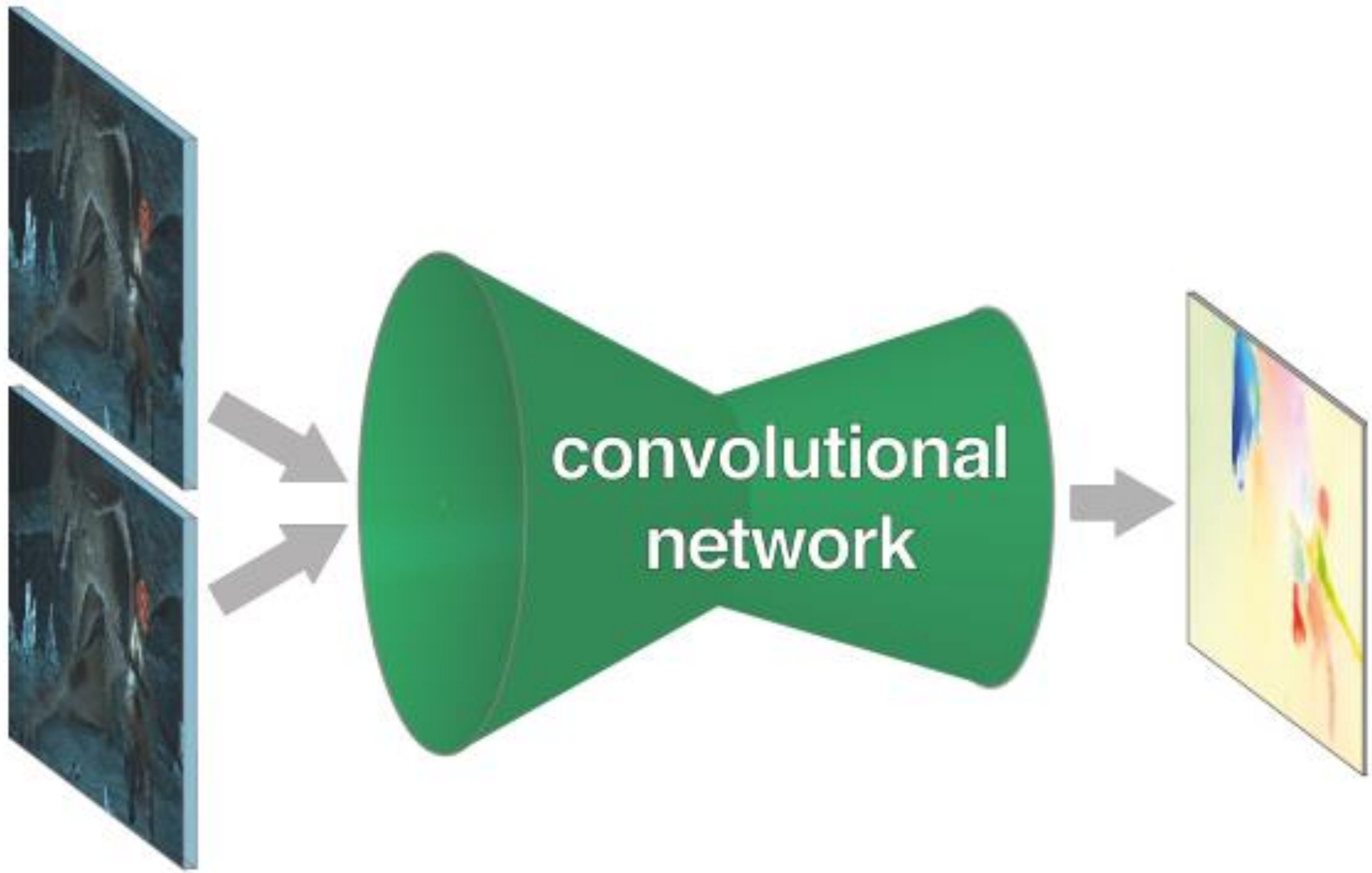
- What if pixel comes from “between” two pixels?
- Answer: *resample* color value from *interpolated* (*prefiltered*) source image



Interpolation

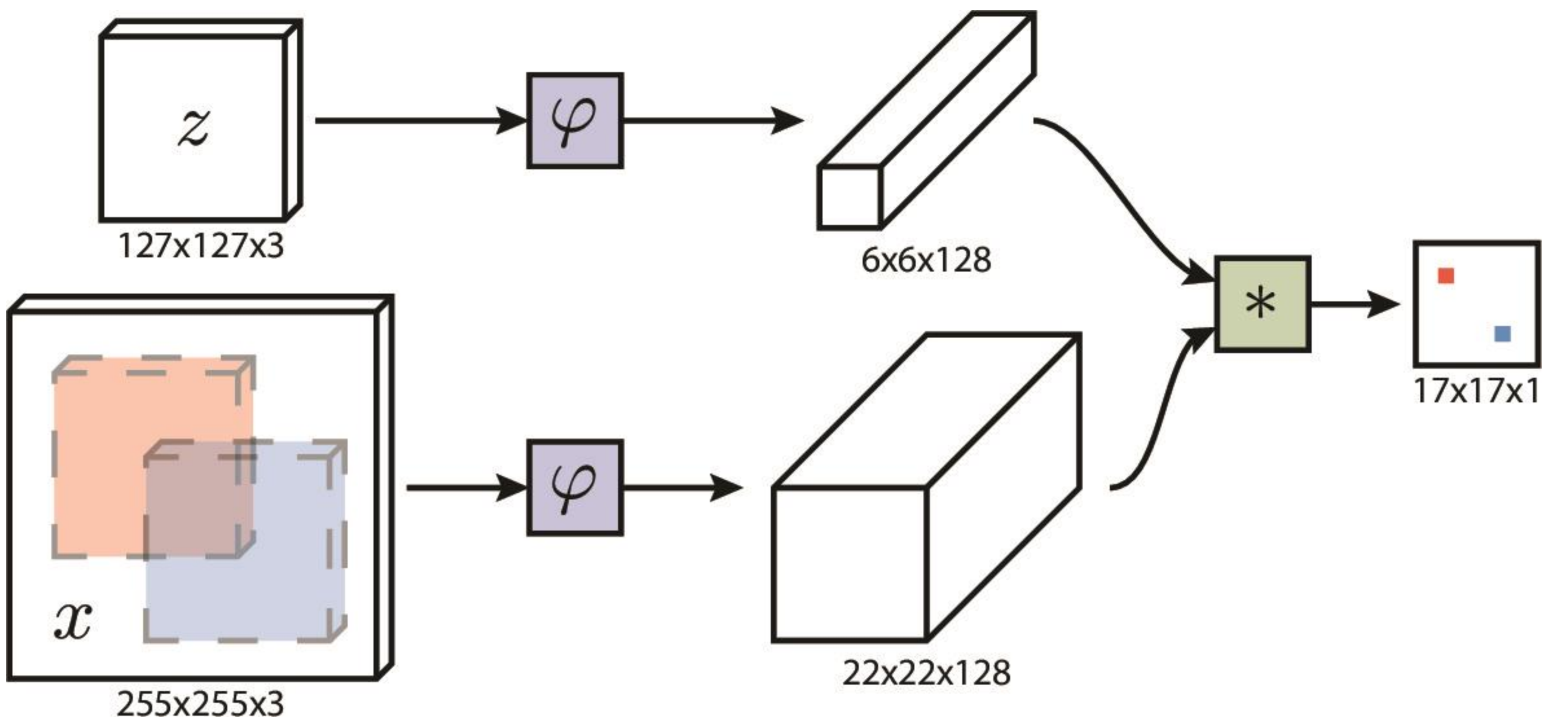
- Possible interpolation filters:
  - nearest neighbor
  - bilinear
  - bicubic (interpolating)
  - sinc / FIR
- Needed to prevent "jaggies" and "texture crawl" (see demo)

FlowNet 1.0 & FlowNet 2.0

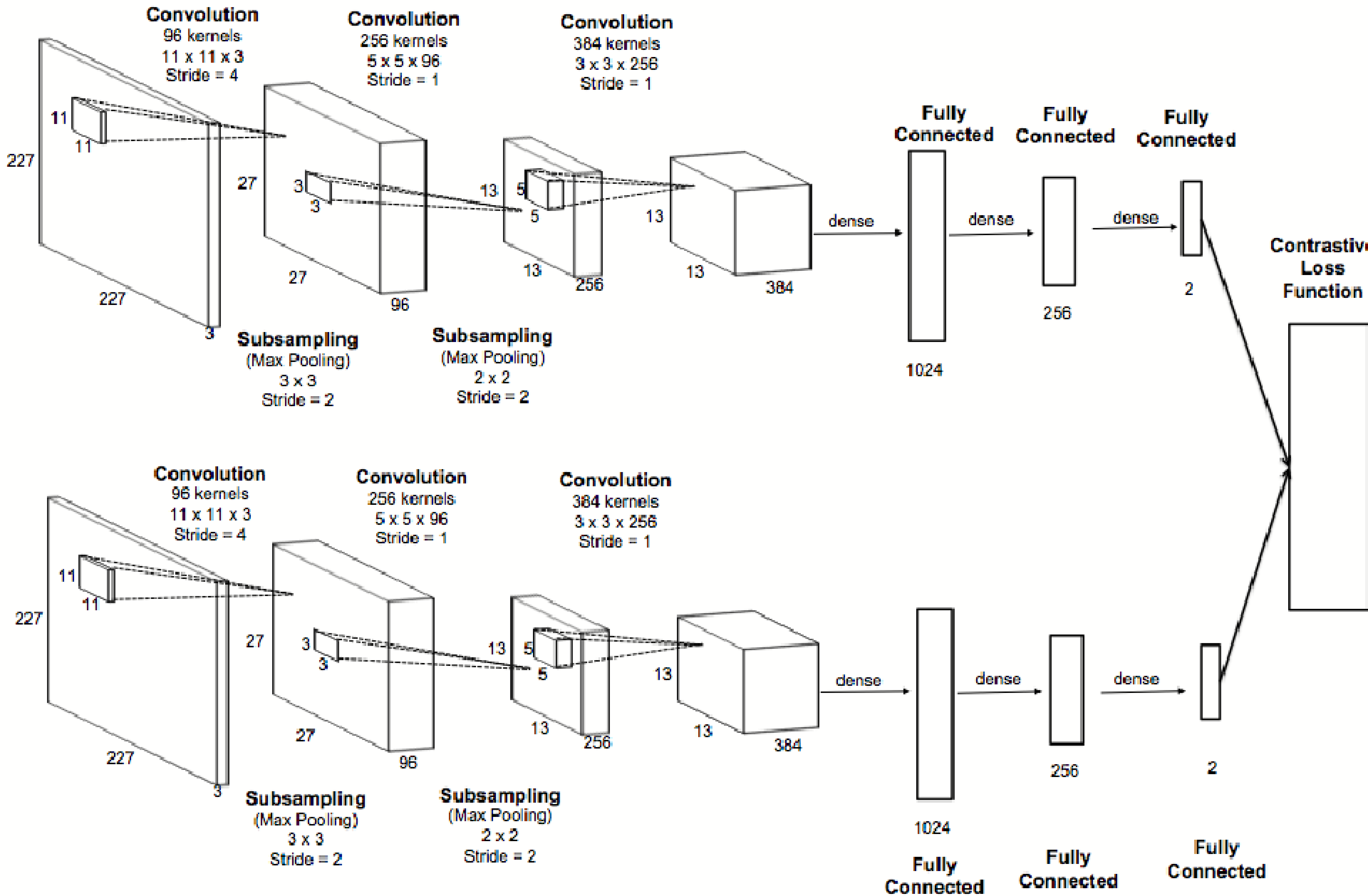


FlowNet 1.0 & FlowNet 2.0

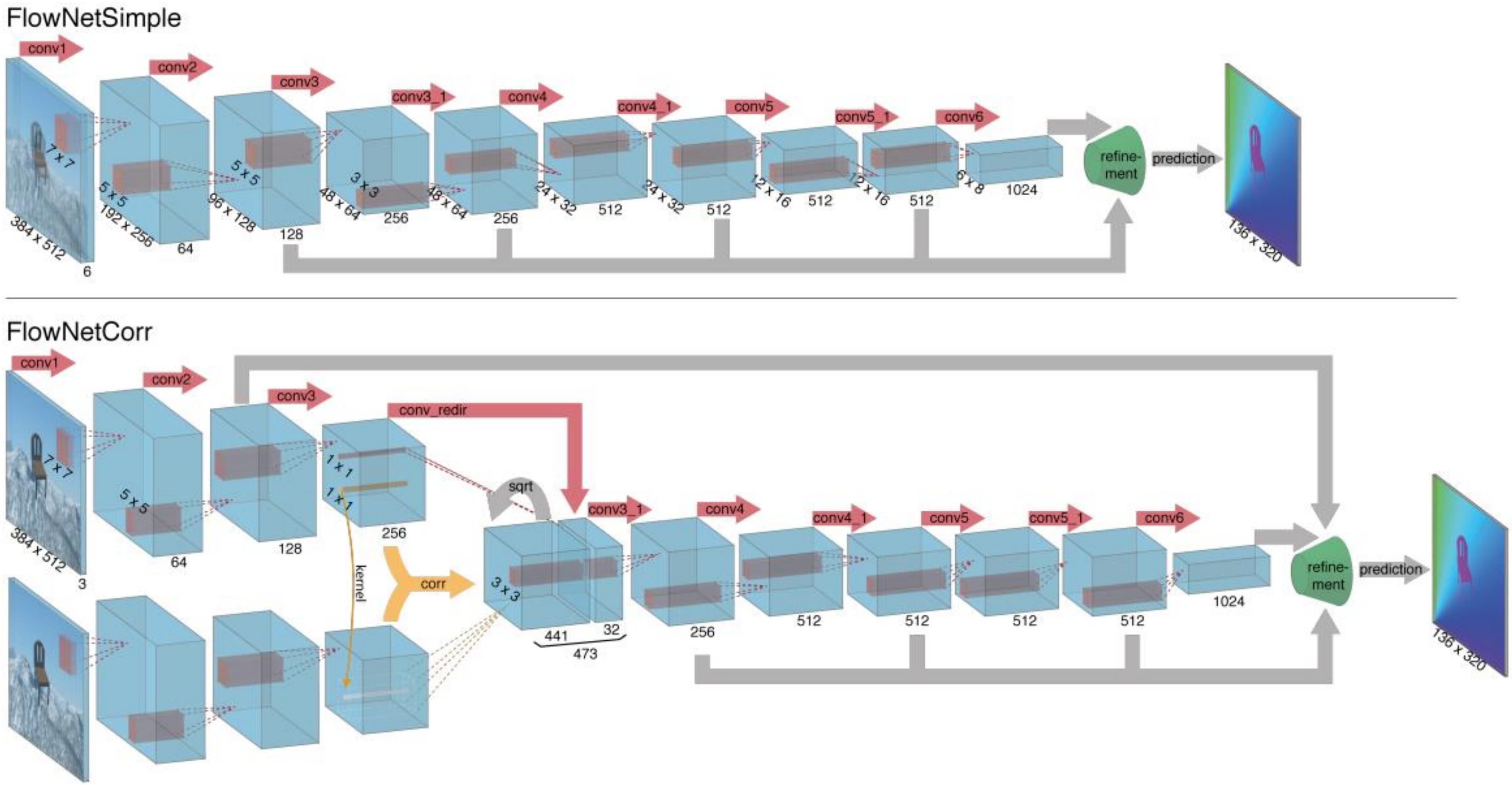
While optical flow estimation needs precise per-pixel localization, it also requires finding correspondences between two input images. This involves not only learning image feature representations, but also learning to match them at different locations in the two images. In this respect, optical flow estimation fundamentally differs from previous applications of CNNs.



Siamese Network in FlowNet 1.0 & FlowNet 2.0



FlowNet 1.0 & FlowNet 2.0



FlowNet 1.0 & FlowNet 2.0

In FlowNet1.0, two architectures are proposed:
(a) FlowNetSimple;
(b) FlowNetCorr.

Both of the two architectures are end-to-end learning approaches. In FlowNetSimple, two sequentially adjacent input images are simply stacked together and they feeded through the network.

Compared with FlowNetSimple, FlowNetCorr first produce representations of the two images separately, and then combines them together in the ‘correlation layer’, and learn the higher representation together. Both of the two architectures have refinements which are used for upsampling resolution.

FlowNet 1.0 & FlowNet 2.0

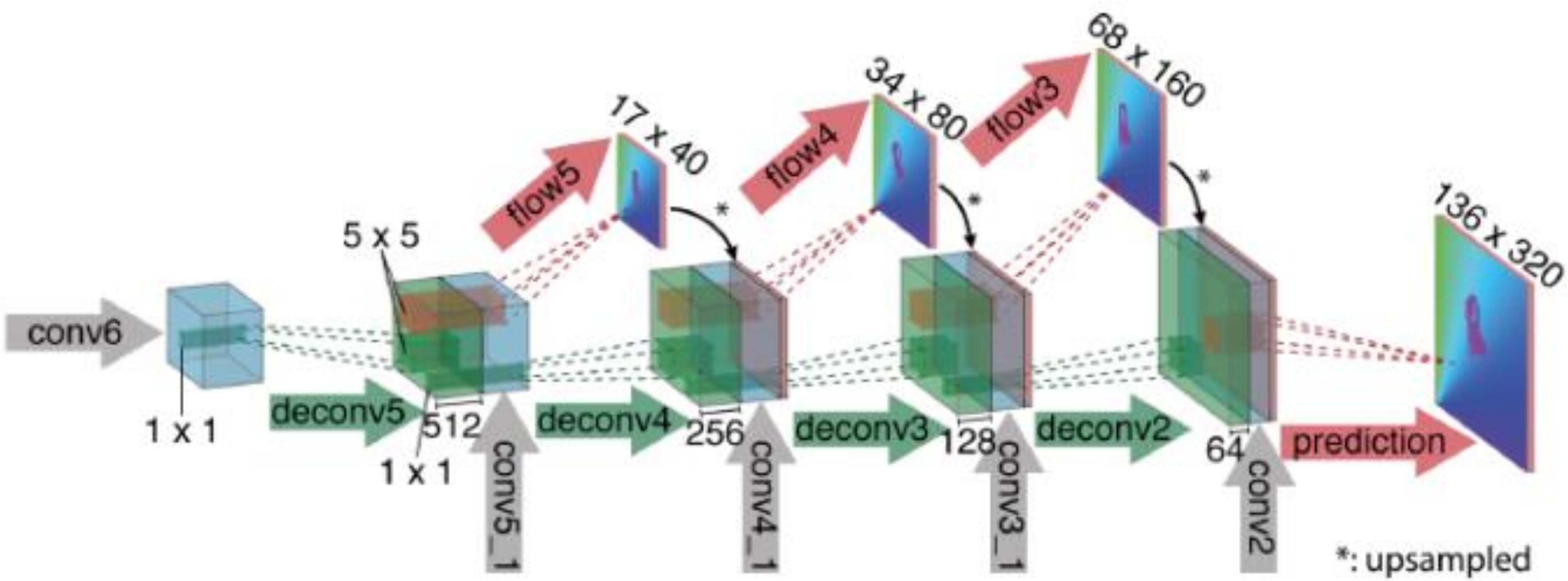
Correlation layer is used to perform multiplicative patch comparisons between two feature maps. More specifically, given two multi-channel feature maps f1, f2, with w, h, and c being their width, height and number of channels. The ‘correlation’ of two patches centered at x1 in the first map and x2 in the second map is then defined as:

$$c(\mathbf{x}_1, \mathbf{x}_2) = \sum_{\mathbf{o} \in [-k,k] \times [-k,k]} \langle \mathbf{f}_1(\mathbf{x}_1 + \mathbf{o}), \mathbf{f}_2(\mathbf{x}_2 + \mathbf{o}) \rangle$$

where x1 and x2 are the center of the first map and the second map respectively, and the square space patch of size K = 2k+1. For computation reasons, the maximum displacement is limited. To be specific, for each location x1, the range of x2 by computing correlations in a neighborhood of size D = 2d+1, and d is a given maximum displacement. The size of an output is ($w*h*D^2$). Afterwards, the feature map is concatenated, which is extracted from f1 using convolution layer, with the output.

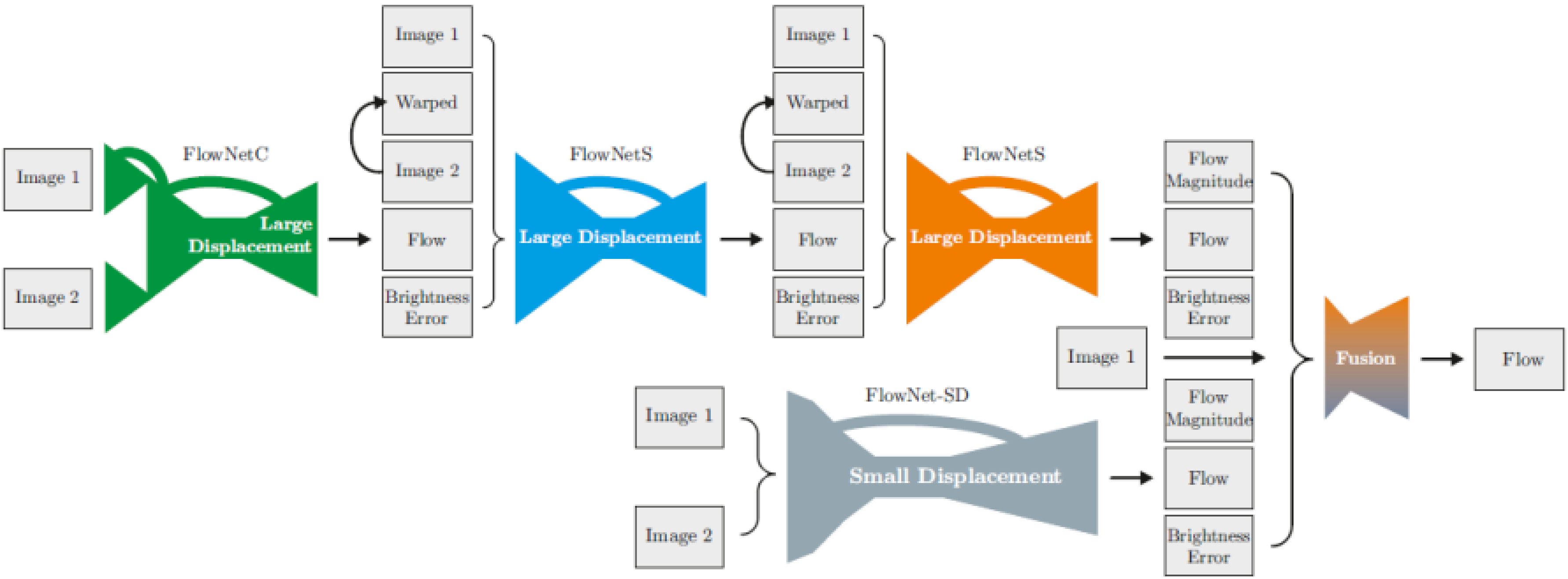
FlowNet 1.0 & FlowNet 2.0

After a series of convolution layers and pooling layers, resolution has been reduced. Thus, the coarse pooled representation is refined by ‘upconvolution’ layers, consisting of unpooling and upconvolution. After upconvolutioning the feature maps, the corresponding feature maps are concatenated and an upsampled coarse flow prediction.

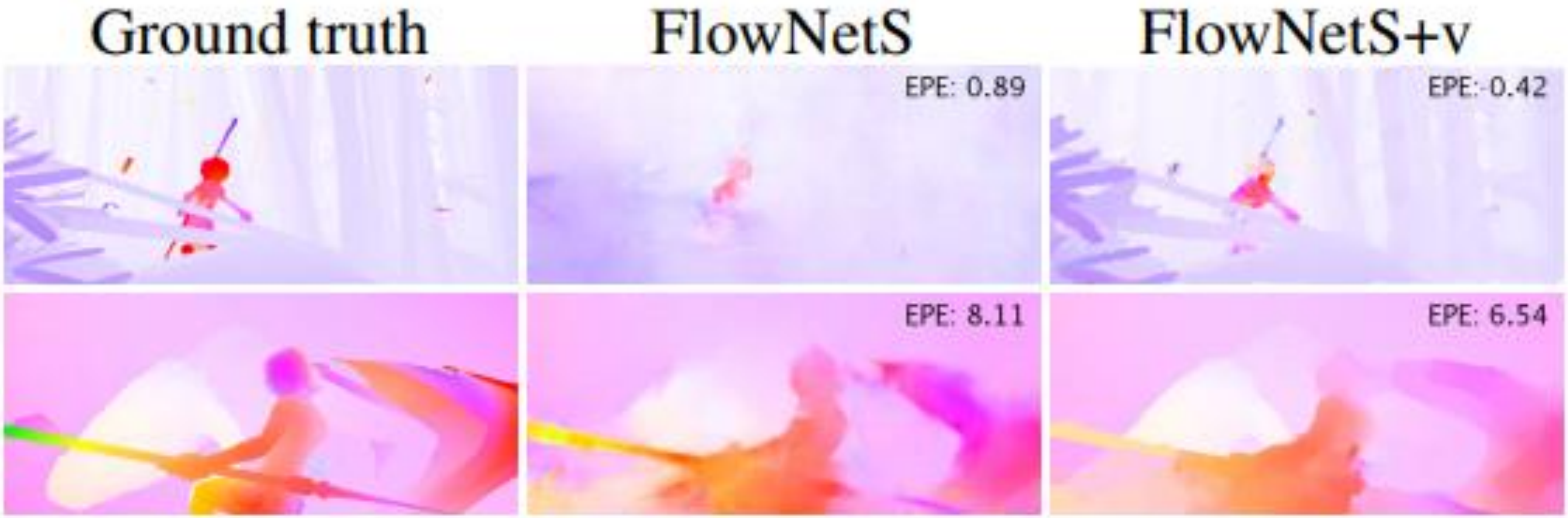


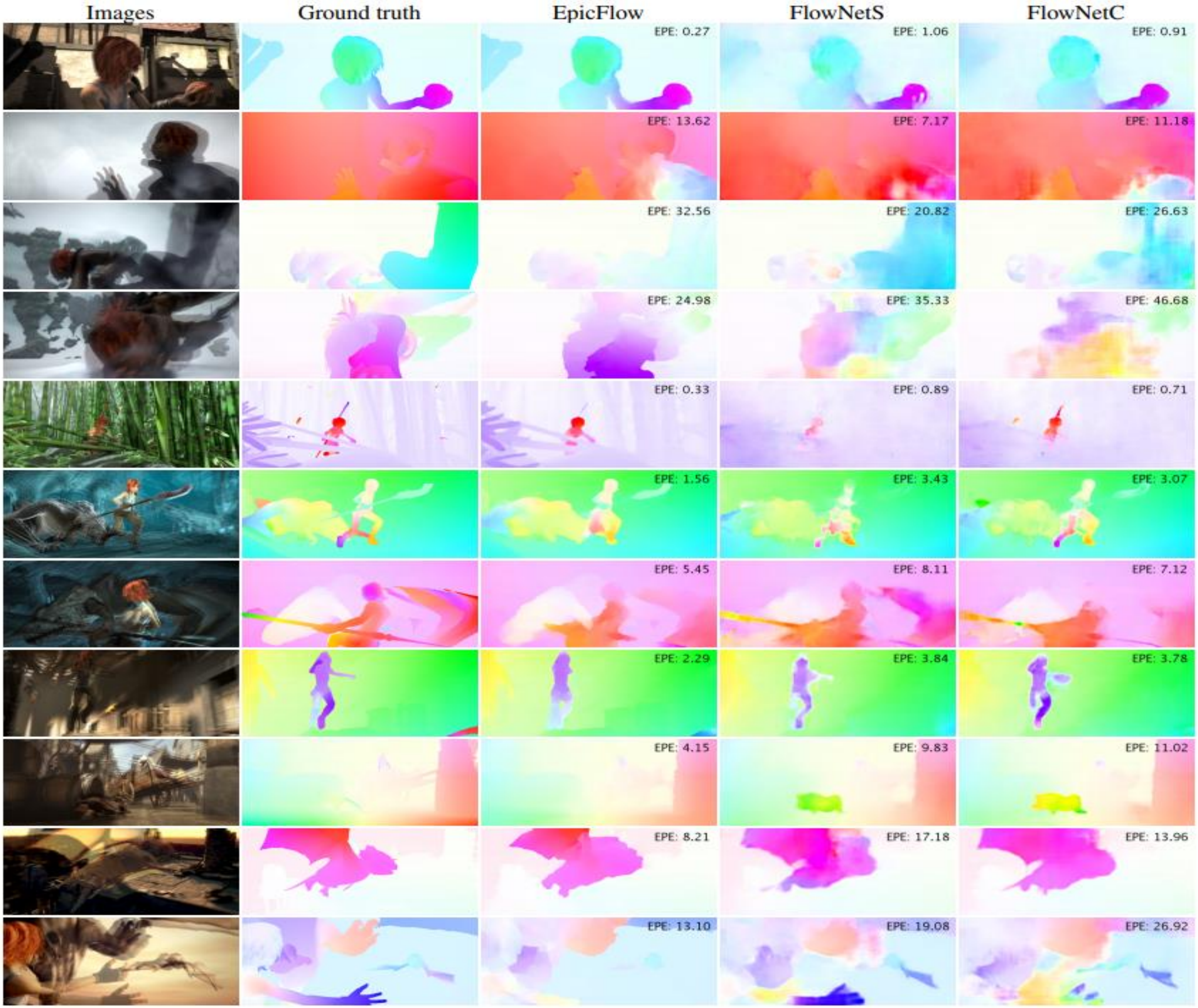
FlowNet 1.0 & FlowNet 2.0

To compute large displacement of optical flow, FlowNetS and FlowNetCorr are stacked.



FlowNet 1.0 & FlowNet 2.0




Images
Ground truth
EpicFlow
FlowNetS
FlowNetC
EPE: 0.27
EPE: 1.06
EPE: 0.91
EPE: 13.62
EPE: 7.17
EPE: 11.18
EPE: 32.56
EPE: 20.82
EPE: 26.63
EPE: 24.98
EPE: 35.33
EPE: 46.68
EPE: 0.33
EPE: 0.89
EPE: 0.71
EPE: 1.56
EPE: 3.43
EPE: 3.07
EPE: 5.45
EPE: 8.11
EPE: 7.12
EPE: 2.29
EPE: 3.84
EPE: 3.78
EPE: 4.15
EPE: 9.83
EPE: 11.02
EPE: 8.21
EPE: 17.18
EPE: 13.96
EPE: 13.10
EPE: 19.08
EPE: 26.92

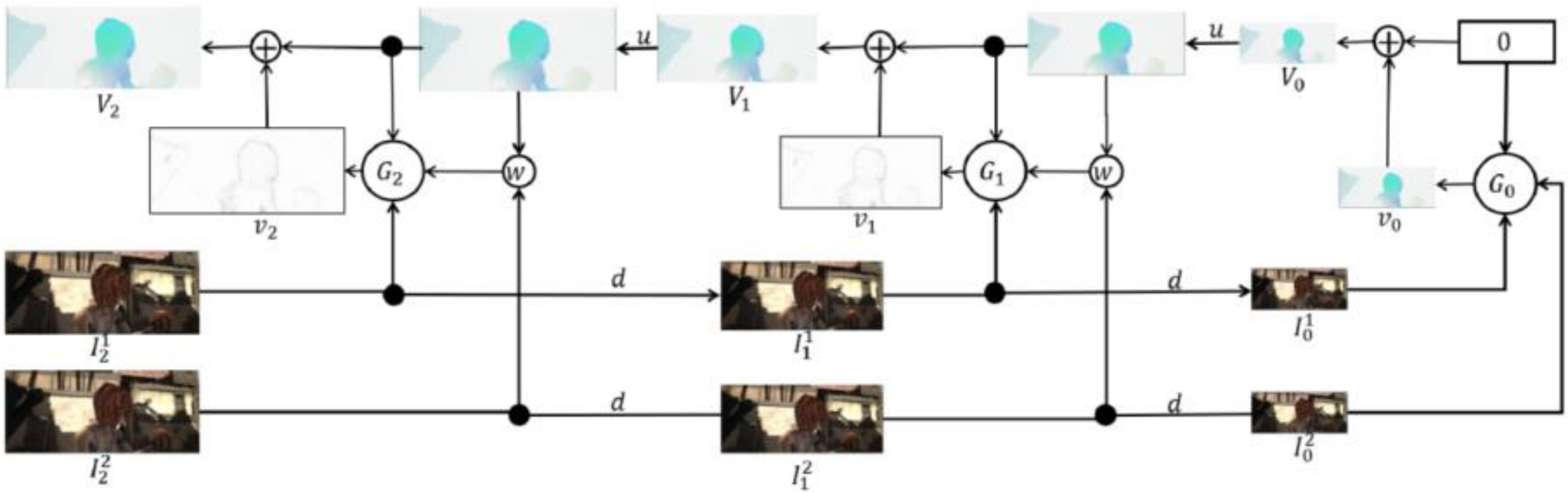
SPyNet

SPyNet combines a classic spatial-pyramid formulation with deep learning. Thus SPyNet is a coarse-to-fine approach.

At each level of the spatial pyramid, the authors train a deep neural network to estimate a flow instead of solely training one deep network. This method is beneficial to arbitrarily large motions, because each network has less work to do and the motion at each network become smaller.

Compared to FlowNet, SPyNet is much simpler and 96% smaller in terms of model parameters. Also, for some standard benchmarks, SPyNet is more accurate than FlowNet1.0.

Architecture of SPyNet



A 3-level pyramid network is shown:

d() is the downsampling function that decrease an m*n image I to m/2*n/2

u() is the resampling function that resample optical flow field

w(I,V) is used for warpping image I, according to optical flow field V

{G_0,…,G_K} is a set of trained convolutional neural network

v_k is residual flow computed by convnet Gk at the k-th pyramid level

SPyNet

$$v_k = G_k(I_k^1, w(I_k^2, u(V_{k-1})), u(V_{k-1}))$$

At the k-th pyramid level, residual flow v_k is computed by G_k using I_k1, the upsampled flow from the previous pyramid, and I_k2 which is warpped by upsample flow. Then, the flow V_k can be represented by

$$V_k = u(V_{k-1}) + v_k.$$

Convents {G_0,…G_k} are trained independently to compute the residual flow v_k. Also, the ground truth residual flows v^_k is obtained by subtracting downsampled ground truth flow V^_k and u(V_k-1). Authors train the networks by minimizing the average End Point Error(EPE) loss on the residual flow v_k

$$\hat{v}_k = \hat{V}_k - u(V_{k-1}).$$

SPyNet