

CLKN: Cascaded Lucas-Kanade Networks for Image Alignment

Che-Han Chang Chun-Nan Chou Edward Y. Chang
HTC Research

{CheHan.Chang, Jason.CN.Chou, Edward.Chang}@htc.com

Abstract

This paper proposes a data-driven approach for image alignment. Our main contribution is a novel network architecture that combines the strengths of convolutional neural networks (CNNs) and the Lucas-Kanade algorithm. The main component of this architecture is a Lucas-Kanade layer that performs the inverse compositional algorithm on convolutional feature maps. To train our network, we develop a cascaded feature learning method that incorporates the coarse-to-fine strategy into the training process. This method learns a pyramid representation of convolutional features in a cascaded manner and yields a cascaded network that performs coarse-to-fine alignment on the feature pyramids. We apply our model to the task of homography estimation, and perform training and evaluation on a large labeled dataset generated from the MS-COCO dataset. Experimental results show that the proposed approach significantly outperforms the other methods.

1. Introduction

Image alignment, or estimating a parametric motion model between two images, is essential for tasks like panoramic image stitching [5], optical flow [6], simultaneous localization and mapping (SLAM) [11], visual odometry (VO) [12], and many others. A robust image alignment algorithm should cope with photometric variations and large motion variations while giving a sub-pixel accurate alignment. Most image alignment approaches can be classified into two categories [26]: *feature-based* methods and *pixel-based* methods.

Feature-based methods extract distinct features, match them, and then estimate the motion model from point correspondences. These methods are robust to large differences in scales, orientations, and lighting because feature descriptors such as SIFT [21] and HOG [9] are invariant to these variations. However, achieving a sub-pixel accurate alignment heavily relies on accurate localization and an even distribution of features, which is challenging in low-textured scenes. In contrast, *pixel-based* (or *direct*) methods, mostly

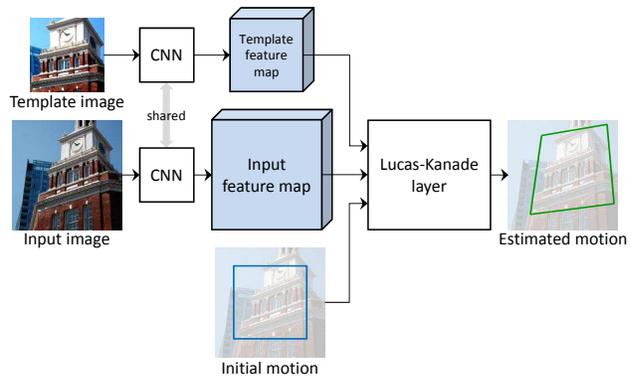


Figure 1. Our network takes a template image, an input image and an initial motion as inputs. The two CNNs with shared weights transform the two images into two multi-channel feature maps. Then, the Lucas-Kanade layer takes these two feature maps and an initial motion as inputs, performs the inverse compositional Lucas-Kanade algorithm [3] to obtain the estimated motion.

based on the Lucas-Kanade algorithm [22], estimate the motion model directly from raw pixel intensities. These methods often perform better on low-textured images since all the pixels are used to estimate a small number of parameters. Pixel-based methods have received great attention in SLAM [11] and VO [12] lately due to their effectiveness. Nonetheless, pixel-based methods are not robust to lighting changes and large motions.

Recently, several methods [1][2][8] were proposed to combine the Lucas-Kanade algorithm with feature descriptors. We refer to these methods as FBLK standing for the *feature-based Lucas-Kanade* methods. The central idea of FBLK is to perform image alignment on densely-sampled feature descriptors. FBLK combine the strengths of both feature-based and pixel-based methods, and are robust to both lighting variations and low-textured scenes. However, FBLK still suffer from two shortcomings. First, commonly used feature descriptors are hand-designed for finding sparse correspondences, which may be suboptimal on some scenes. Second, FBLK are prone to fail in the presence of large motions.

In this paper, we propose combining deep learning with the Lucas-Kanade algorithm to address the shortcomings of FBLK. The key idea is to acquire big data to train a convolutional neural network (CNN) that adds a *Lucas-Kanade layer*. The CNN learns data-driven features and the *Lucas-Kanade layer* performs the inverse compositional Lucas-Kanade algorithm [3]. Figure 1 depicts our network architecture. Our Lucas-Kanade layer is differentiable, which enables us to train our network by standard back-propagation algorithm. Our network combines the strengths of both the CNN and the Lucas-Kanade algorithm. The CNN provides the ability to learn features that are both useful for alignment and robust to photometric variations. The Lucas-Kanade layer offers the ability to achieve sub-pixel accuracy.

In training, we employ a hinge loss to effectively train our network. Furthermore, to address the issue of large motions, we propose a cascaded feature learning method, which learns a pyramid representation of convolutional features in a cascaded manner. The yielded model is a *cascaded Lucas-Kanade network* (CLKN) that performs coarse-to-fine alignment on the feature pyramids.

Our approach works for any parametric motions that the Lucas-Kanade algorithm supports. In this paper, we apply our model to the task of homography estimation, which enjoys important applications in both image processing and augmented reality [23], such as image stitching [27], video stabilization [19], and planar object tracking [4]. By generating random homography warps on the MS-COCO images [18], we constructed a large labeled dataset for homography estimation. We trained and evaluated our models on this dataset. Experimental results show that our method achieves sub-pixel accuracy and is robust to color variations and large motions.

Related Work

We review previous work relevant to the development of our approach. Szeliski [26] provided a comprehensive overview of feature-based methods and pixel-based methods. Baker *et al.* [3] proposed a unified framework for the Lucas-Kanade algorithm and its variants. The idea of adopting hand-crafted feature descriptors in direct image alignment are recently explored in 3D model tracking [8], facial image alignment [2], and template tracking [1]. Crivellaro and Lepetit [8] developed a derivative-based feature descriptor for direct alignment to address the issue of specularities and low textures. Antonakos *et al.* [2] proposed the employment of hand-crafted feature descriptors for the Lucas-Kanade algorithm and active appearance models (AAMs). They experimented with various feature descriptors and demonstrated that SIFT and HOG are the most effective. Alismail *et al.* [1] presented bit-planes, a binary descriptor for real-time tracking under drastic illumination changes.

Recently, there has been an increasing interest in leverag-

ing the power of CNNs to solve geometric computer vision problems. DeTone *et al.* [10] presented deep homography net for homography estimation, which is the most relevant to our work. They trained a VGG-style CNN to directly regress the homography between two images. However, their CNN model cannot achieve sub-pixel accuracy.

Contribution Summary

In summary, our work makes the following three contributions:

1. We propose a novel network architecture that performs the Lucas-Kanade algorithm on convolutional features.
2. Our cascaded feature learning method enables our network to perform coarse-to-fine alignment.
3. Experiments show that our approach outperforms the other methods significantly. Our method enjoys a wider range of convergence and achieves higher sub-pixel accuracy.

2. Model Architecture

Given an input image I and a template image T , our goal is to bring these two images into alignment by estimating the underlying parametric motion between I and T . The motion model between I and T is represented by a warping function $W(\mathbf{x}; \mathbf{p})$ parameterized by a vector \mathbf{p} . W takes a pixel $\mathbf{x} = [x, y]^T$ in the coordinate of template image and maps it to a sub-pixel location $\mathbf{x}' = [x', y']^T = W(\mathbf{x}; \mathbf{p})$ in the coordinate of input image. A homography has eight parameters $\mathbf{p} = [p_1, \dots, p_8]^T$ and can be parameterized as [3]

$$W(\mathbf{x}; \mathbf{p}) = \frac{1}{1 + p_7x + p_8y} \begin{bmatrix} (1 + p_1)x + p_2y + p_3 \\ p_4x + (1 + p_5)y + p_6 \end{bmatrix}. \quad (1)$$

Our model consists of two stages: The first stage contains two CNNs that extract multi-channel feature maps for both I and T . The second stage is a Lucas-Kanade layer, which performs the inverse compositional Lucas-Kanade algorithm on these two feature maps to estimate the motion parameters \mathbf{p} .

2.1. Convolutional Neural Networks

We extract multi-channel feature maps for both I and T by using two CNNs with shared weights. The CNN we employ here is fully convolutional [20] and hence can take input of arbitrary sizes. Each convolutional layer is followed by a Rectified Linear Units (ReLU) [24] and then a batch normalization [16]. In the convolutional layer, we use a set of 3×3 learnable filters. If all filters have a stride 1, then the output feature map is a full-resolution one. If a downsampled feature map with a factor of 2^k is required, we achieve this by setting the first k convolutional layers to have a stride 2. We denote the output feature maps of I and T as F_I and F_T , respectively.

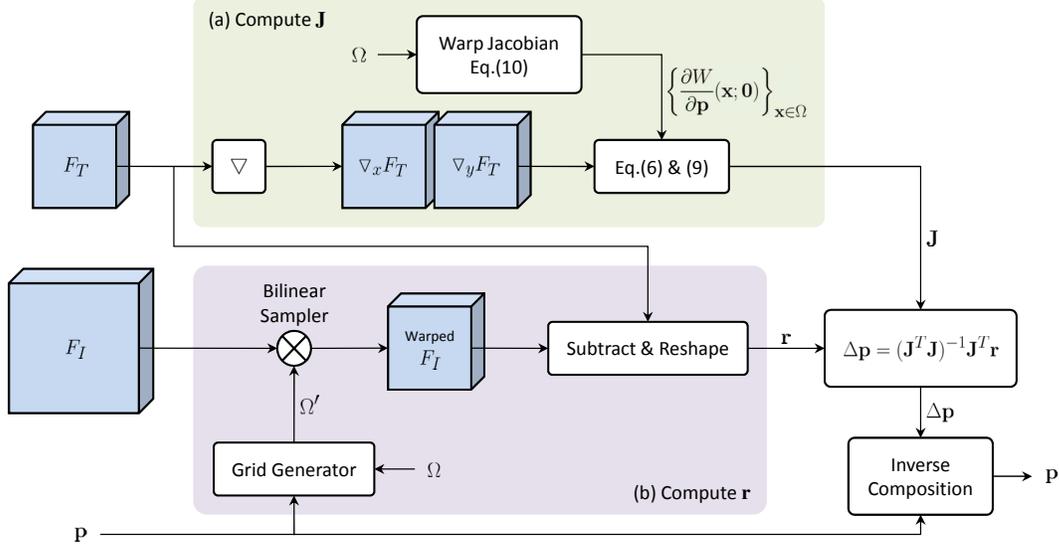


Figure 2. Full schematic diagram of our Lucas-Kanade layer which performs the inverse compositional Lucas-Kanade algorithm. (a) The Jacobian matrix \mathbf{J} is constructed from the warp Jacobian and the spatial gradient of the template feature map. (b) The residual vector \mathbf{r} is a vector reshaped from the difference between the template feature map and the warped input feature map.

2.2. Lucas-Kanade Layer

By taking the input feature map F_I , the template feature map F_T , and an initial motion as inputs, the Lucas-Kanade layer performs the Lucas-Kanade algorithm and outputs the estimated motion parameters \mathbf{p} . Figure 2 depicts our Lucas-Kanade layer. In the following, we briefly review the feature-based Lucas-Kanade algorithm and then describe the details of the Lucas-Kanade layer.

The feature-based Lucas-Kanade algorithm aims at finding the motion parameters \mathbf{p} that minimizes the following error function:

$$E(\mathbf{p}) = \frac{1}{2} \sum_{\mathbf{x} \in \Omega} \|F_T(\mathbf{x}) - F_I(W(\mathbf{x}; \mathbf{p}))\|^2. \quad (2)$$

Here, the regular grid $\Omega = \{\mathbf{x}_i\}_{i=1}^N = \{(x_i, y_i)\}_{i=1}^N$ is the set of pixel locations in the template image, and N is the number of template image pixels. $E(\mathbf{p})$ measures the sum of squared error between the template feature map $F_T(\mathbf{x})$ and the warped input feature map $F_I(W(\mathbf{x}; \mathbf{p}))$.

Minimizing $E(\mathbf{p})$ is a nonlinear optimization problem because the feature map $F_I(\mathbf{x})$ is highly non-linear in the pixel coordinates \mathbf{x} . To optimize $E(\mathbf{p})$, the Lucas-Kanade algorithm assumes that an initial motion is known and then iteratively solves for an incremental update $\Delta\mathbf{p}$. In particular, we optimize $E(\mathbf{p})$ by using the inverse compositional algorithm [3], which minimizes the following error function:

$$E(\Delta\mathbf{p}) = \frac{1}{2} \sum_{\mathbf{x} \in \Omega} \|F_T(W(\mathbf{x}; \Delta\mathbf{p})) - F_I(W(\mathbf{x}; \mathbf{p}))\|^2 \quad (3)$$

and then updates the motion parameters by inverse composition as

$$W(\mathbf{x}; \mathbf{p}) \leftarrow W(\mathbf{x}; \mathbf{p}) \circ W(\mathbf{x}; \Delta\mathbf{p})^{-1}. \quad (4)$$

The inverse compositional Lucas-Kanade algorithm optimizes $E(\Delta\mathbf{p})$ by using the Gauss-Newton method. $E(\Delta\mathbf{p})$ is first approximated by performing a first order Taylor expansion on $F_T(W(\mathbf{x}; \Delta\mathbf{p}))$ at $\Delta\mathbf{p} = \mathbf{0}$, and then it has the following closed-form solution [3]:

$$\Delta\mathbf{p} = \mathbf{H}^{-1} \sum_{\mathbf{x} \in \Omega} J(\mathbf{x})^T (F_I(W(\mathbf{x}; \mathbf{p})) - F_T(\mathbf{x})). \quad (5)$$

Here, $J(\mathbf{x})$ is the Jacobian matrix of $F_T(W(\mathbf{x}; \Delta\mathbf{p}))$ at $\Delta\mathbf{p} = \mathbf{0}$. $\mathbf{H} = \sum_{\mathbf{x} \in \Omega} J(\mathbf{x})^T J(\mathbf{x})$ is the Hessian matrix.

Equation 5 can be rewritten into a more compact form. To achieve this, we introduce two notations: the residual vector \mathbf{r} and the Jacobian matrix \mathbf{J} . They are defined as

$$\mathbf{J} = [J(\mathbf{x}_1)^T \cdots J(\mathbf{x}_N)^T]^T, \text{ and} \quad (6)$$

$$\mathbf{r} = \begin{bmatrix} F_I(W(\mathbf{x}_1; \mathbf{p})) - F_T(\mathbf{x}_1) \\ \vdots \\ F_I(W(\mathbf{x}_N; \mathbf{p})) - F_T(\mathbf{x}_N) \end{bmatrix}. \quad (7)$$

With \mathbf{J} and \mathbf{r} , the update formula in Equation 5 can be rewritten as

$$\Delta\mathbf{p} = (\mathbf{J}^T \mathbf{J})^{-1} \mathbf{J}^T \mathbf{r}. \quad (8)$$

Equation 8 represents the major computation of the Lucas-Kanade layer, which requires computing \mathbf{J} and \mathbf{r} , and then

combining them into $\Delta\mathbf{p}$. In the following, we explain the details of the Lucas-Kanade layer.

Compute \mathbf{J} . As shown in Equation 6, \mathbf{J} is constructed from a vertical concatenation of $\{J(\mathbf{x})\}_{\mathbf{x}\in\Omega}$. By the definition of $J(\mathbf{x})$ and the chain rules, we have

$$\begin{aligned} J(\mathbf{x}) &= \left. \frac{\partial}{\partial \mathbf{p}} F_T(W(\mathbf{x}; \mathbf{p})) \right|_{\mathbf{p}=\mathbf{0}} \\ &= \left. \frac{\partial}{\partial \mathbf{x}'} F_T(\mathbf{x}') \right|_{\mathbf{x}'=W(\mathbf{x}, \mathbf{0})=\mathbf{x}} \left. \frac{\partial}{\partial \mathbf{p}} W(\mathbf{x}; \mathbf{p}) \right|_{\mathbf{p}=\mathbf{0}} \\ &= \nabla F_T(\mathbf{x}) \cdot \frac{\partial W}{\partial \mathbf{p}}(\mathbf{x}; \mathbf{0}), \end{aligned} \quad (9)$$

which is a product of the spatial gradient $\nabla F_T(\mathbf{x})$ and the warp Jacobian $\frac{\partial W}{\partial \mathbf{p}}(\mathbf{x}; \mathbf{0})$. $\nabla F_T(\mathbf{x}) = [\nabla_x F_T(\mathbf{x}), \nabla_y F_T(\mathbf{x})]$ is a $C \times 2$ matrix, where C is the number of channels of F_T .

The warp Jacobian purely depends on the type of motion model and its parameterization. Consider a homography parameterized as Equation 1, then its corresponding warp Jacobian is written as [3]

$$\frac{\partial W}{\partial \mathbf{p}}(\mathbf{x}; \mathbf{0}) = \begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -x^2 & -xy \\ 0 & 0 & 0 & x & y & 1 & -xy & -y^2 \end{bmatrix}. \quad (10)$$

The resulting \mathbf{J} is a $CN \times 8$ matrix. Since \mathbf{J} is independent of \mathbf{p} , we compute \mathbf{J} once and reuse it in each iteration.

Compute \mathbf{r} . The major computation of \mathbf{r} in Equation 7 comes from the warped input feature map $F_I(W(\mathbf{x}; \mathbf{p}))$, which requires interpolating F_I at the sub-pixel location $W(\mathbf{x}; \mathbf{p})$. It can be implemented using the spatial transformer network [17]. The spatial transformer layer consists of a grid generator and a bilinear sampler. Here the grid generator acts as the warping function W , which is a homography in our case. It takes the motion parameters \mathbf{p} and the regular grid $\Omega = \{\mathbf{x}_i\}_{i=1}^N$ as inputs, and outputs the sampling grid $\Omega' = \{W(\mathbf{x}_i; \mathbf{p})\}_{i=1}^N$. Then, a bilinear sampler takes Ω' and F_I as inputs, and renders the warped input feature map $F_I(W(\mathbf{x}; \mathbf{p}))$.

Inverse Composition. Given the computed $\Delta\mathbf{p}$, we then perform the inverse composition in Equation 4 to update \mathbf{p} . First, a homography parameterized as Equation 1 can also be represented by a 3×3 homography matrix as

$$\begin{bmatrix} 1 + p_1 & p_2 & p_3 \\ p_4 & 1 + p_5 & p_6 \\ p_7 & p_8 & 1 \end{bmatrix} \quad (11)$$

We denote the corresponding homography matrices of \mathbf{p} and $\Delta\mathbf{p}$ as $H_{\mathbf{p}}$ and H_{Δ} , respectively. Then, the inverse composition of homography can be written as

$$H_{\mathbf{p}} \leftarrow H_{\mathbf{p}} H_{\Delta}^{-1}. \quad (12)$$

Finally, $H_{\mathbf{p}}$ is scaled such that $H_{\mathbf{p}}[3, 3] = 1$, and we obtain the updated \mathbf{p} .

Number of iterations. In general, the Lucas-Kanade algorithm requires running multiple iterations to find the true motion. The number of iterations required for convergence varies and often depends on the magnitude of motion between images. Images with large motion usually require a large number of iterations while those with subtle motion could converge in few steps. Therefore, it is more reasonable to set the number of iterations in an adaptive way than setting it to a fixed number. Our Lucas-Kanade layer acts as the same way as the Lucas-Kanade algorithm, which stops its iterative process when the change of the motion parameters is below a threshold, or when a maximum number of iteration is exceeded.

Matrix Inverse. Both Equations 8 and 12 require computing a matrix inverse, which is a differentiable operation. Since there is a need for the back-propagation algorithm to derive the gradient, we present the formula of matrix gradient in the following. Consider a square matrix \mathbf{A} , its inverse $\mathbf{W} = \mathbf{A}^{-1}$, and a loss function L , then $\frac{\partial L}{\partial \mathbf{A}}$ and $\frac{\partial L}{\partial \mathbf{W}}$ are related by [25]

$$\frac{\partial L}{\partial \mathbf{A}} = -\mathbf{A}^{-T} \frac{\partial L}{\partial \mathbf{W}} \mathbf{A}^{-T}. \quad (13)$$

In summary, our Lucas-Kanade layer first computes \mathbf{J} and \mathbf{r} , then combines them into $\Delta\mathbf{p}$ (Equation 8), and finally performs the inverse composition (Equation 12) to update \mathbf{p} .

3. Learning

In this section, we first describe the loss function used in training our network. We then describe our cascaded feature learning method, which incorporates the coarse-to-fine strategy into the learning process.

3.1. Loss Function

Training a Lucas-Kanade network is challenging because the training may require a dynamic number of iterations. To deal with such difficulty, we instead propose to train a one-step Lucas-Kanade network with a specially designed loss function. Consider a ground truth motion $\hat{\mathbf{p}}$ and a sequence $\mathbf{p}_{(1)}, \mathbf{p}_{(2)}, \dots, \mathbf{p}_{(t)}$ obtained from running multiple iterations (or steps) in the Lucas-Kanade layer. In order to arrive at the ground truth $\hat{\mathbf{p}}$, we want that each step could make progress in terms of the distance from $\hat{\mathbf{p}}$, *i.e.*,

$$d(\mathbf{p}_{(t+1)}, \hat{\mathbf{p}}) < d(\mathbf{p}_{(t)}, \hat{\mathbf{p}}), \quad (14)$$

where d is a distance function that measures the dissimilarity between two motion models. Let $\mathbf{e}_1, \dots, \mathbf{e}_4$ be the four corner positions of the template image, and we define $d(\mathbf{p}_1, \mathbf{p}_2)$ by the sum of squared distance of the warped cor-

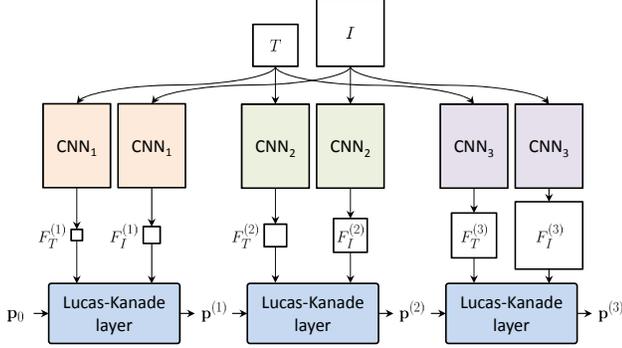


Figure 3. A schematic diagram of a 3-level CLKN. Please see text for details.

ners as

$$d(\mathbf{p}_1, \mathbf{p}_2) = \sum_{j=1}^4 \|W(\mathbf{e}_j; \mathbf{p}_1) - W(\mathbf{e}_j; \mathbf{p}_2)\|_2^2. \quad (15)$$

Based on Equation 14, we propose to train a one-step Lucas-Kanade network with the following 0-1 loss:

$$L_{01}(\mathbf{p}_0, \mathbf{p}, \hat{\mathbf{p}}) = \mathbb{1} [d(\mathbf{p}, \hat{\mathbf{p}}) > d(\mathbf{p}_0, \hat{\mathbf{p}}) - \delta]. \quad (16)$$

Here \mathbf{p}_0 , \mathbf{p} , and $\hat{\mathbf{p}}$ are the initial, estimated, and ground truth motion parameters, respectively. $\mathbb{1}[\cdot]$ is the indicator function, and $\delta \in \mathbb{R}^+$ is a margin hyper-parameter that controls the desired amount of improvement to achieve in one step.

Since the 0-1 loss is difficult to optimize, we approximate it by a hinge loss:

$$L(\mathbf{p}_0, \mathbf{p}, \hat{\mathbf{p}}) = \max(0, 1 + \delta + d(\mathbf{p}, \hat{\mathbf{p}}) - d(\mathbf{p}_0, \hat{\mathbf{p}})). \quad (17)$$

The hinge loss is a convex upperbound to the 0-1 loss. In addition, when δ is large enough, minimizing the hinge loss reduces to minimizing $d(\mathbf{p}, \hat{\mathbf{p}})$.

In the training stage, our network is enforced to run single iteration and is trained by the hinge loss. In the testing stage, our network acts as the same as the Lucas-Kanade algorithm, which runs multiple iterations until the stopping condition is met.

3.2. Cascaded Feature Learning

To address the issue of large motions, we propose a cascaded feature learning method, which incorporates the coarse-to-fine strategy into our learning process. In particular, we aim to represent an image by a feature pyramid where each feature map is obtained from a forward pass of a CNN associated to that level. Then, we perform the Lucas-Kanade algorithm sequentially from coarse to fine levels. Such process can be equivalently expressed by a *cascaded Lucas-Kanade network* (CLKN), which is a sequence of Lucas-Kanade networks progressively refining the estimate of motion parameters.

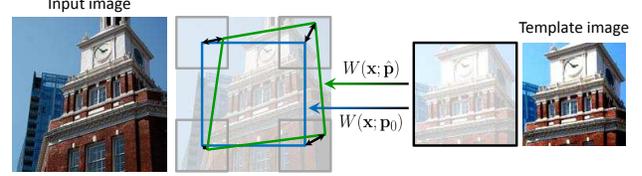


Figure 4. Data generation. The initial warp $W(\mathbf{x}; \mathbf{p}_0)$ maps the dark square to the blue square. The ground truth warp $W(\mathbf{x}; \hat{\mathbf{p}})$ maps the dark square to the green quadrilateral, which is obtained from perturbing the four corners of the blue square. The perturbation range for each corner is shown as a gray square. The template image T is rendered by $T(\mathbf{x}) \leftarrow I(W(\mathbf{x}; \hat{\mathbf{p}}))$.

Figure 3 shows a diagram of a 3-level CLKN. At each stage $k = 1 \dots K$ in the cascade, the CNN at level k extracts feature maps $F_I^{(k)}$ and $F_T^{(k)}$ from I and T , respectively. In particular, the CNN produces the output feature map with a downsampling factor of 2^{K-k} . Based on $F_I^{(k)}$, $F_T^{(k)}$ and $\mathbf{p}^{(k-1)}$, which is the previous Lucas-Kanade network’s motion estimate, the Lucas-Kanade layer compute a new motion estimate $\mathbf{p}^{(k)}$.

In the training stage, these K CNNs are learned sequentially one by one in a coarse-to-fine manner. More specifically, the CNN at the top pyramid level ($k = 1$) is trained on the original training set $\{(T, I, \mathbf{p}_0, \hat{\mathbf{p}})\}$. \mathbf{p}_0 is the parameters of initial motion, and $\hat{\mathbf{p}}$ is the parameters of ground truth motion. For the other levels $k = 2 \dots K$, the CNN at level k is trained on the training set $\{(T, I, \mathbf{p}^{(k-1)}, \hat{\mathbf{p}})\}$ where $\mathbf{p}^{(k-1)}$ is the result from stage $k - 1$ and plays the role of initial motion in stage k .

3.3. Dataset Generation

Training CNNs from scratch requires a large amount of labeled training data. In our case, we aim to construct a large labeled set where each sample is a quadruplet $(T, I, \mathbf{p}_0, \hat{\mathbf{p}})$. Our process of data generation is based on DeTone *et al.*’s work [10]. This enables us to have a direct comparison with their method. We illustrate in Figure 4 the process of data generation and describe its details below.

We generated a large number of labeled examples by applying random homographies to the MS-COCO images [18]. To generate a training sample from an image, we first downsample this image such that the shorter side is 240. Then we randomly crop a square window with size 192×192 and assign it to be the input image I . Next, we define the initial warp $W(\mathbf{x}; \mathbf{p}_0)$ to be a translation that maps the domain of the template image to a square window centering in I (the blue square in Figure 4). Then we randomly perturb the four corners of the window to construct a quadrilateral by using a uniform distribution within the range $[-\beta, \beta]$ for both x and y values. β is set to 32. Similarly, we define the ground truth warp $W(\mathbf{x}; \hat{\mathbf{p}})$ to be

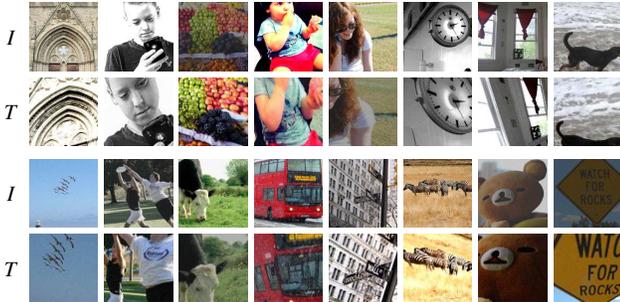


Figure 5. Representative examples in our training set, which is generated from the MS-COCO dataset.

the homography that maps the template boundary (the dark square in Figure 4) to the quadrilateral (the green square in Figure 4). Finally, T is generated by $I(W(\mathbf{x}; \hat{\mathbf{p}}))$, and its size is 128×128 . To avoid unrealistic shape distortions in the induced homography, the maximum angle of the quadrilateral is restricted to being less than $\frac{3}{4}\pi$ during the random perturbation process. The homography $W(\mathbf{x}; \hat{\mathbf{p}})$ can be interpreted as viewing a 2D natural image from a frontal view to a non-frontal view.

To create a more realistic and challenging dataset, we further synthesize photometric variations between I and T . We randomly pick one from I and T and manipulate its colors. Following [15], we enhance the brightness, contrast and saturation with an amount between 0.5 to 1.5 in a random order. Then, we add Gaussian noise with standard deviation 0.02 on both I and T . Figure 5 shows some representative examples of our training images.

We created our training, validation, and testing sets from the corresponding ones of the MS-COCO dataset, respectively. We used the whole training set of MS-COCO to construct our training set (82K images). For our validation/testing set, we used a subset of MS-COCO with 6.4K images.

4. Experiments

Evaluation metric. Based on [3] and [10], we use the *corner error* as our evaluation metric, which is defined as

$$e_c(\mathbf{p}, \hat{\mathbf{p}}) = \frac{1}{4} \sum_{j=1}^4 \|W(\mathbf{e}_j; \mathbf{p}) - W(\mathbf{e}_j; \hat{\mathbf{p}})\|_2. \quad (18)$$

The corner error measures the L^2 distance of the warped corners and then takes average over the four corners. Note that e_c is similar to but different from the distance function d in Equation 15. e_c measures an average L^2 distance in number of pixels, whereas d measures a *squared* distance (as a part of the hinge loss).

We compute the corner error for each test sample and then plot a cumulative error distribution of the test set to

Level	d	T-size	L	LR	α	ϵ
1	8	$4 \times 16 \times 16$	7	5×10^{-6}	4.0	3.5
2	4	$4 \times 32 \times 32$	7	1×10^{-5}	4.0	1.0
3	2	$4 \times 64 \times 64$	7	1×10^{-4}	1.0	0.1
4	1	$4 \times 128 \times 128$	3	1×10^{-4}	0.1	0.05

Table 1. The hyperparameter setting for CLKN. d : downsampling factor. T-size: the size of template feature map. L : the number of convolutional layers in the CNN. LR: learning rate. α : the required amount of decreased corner error for setting the margin hyperparameter δ . ϵ : the stopping threshold (pixels) in the Lucas-Kanade layer.

show the performance of each experimented method. In addition, we measure the alignment accuracy by the percentage of test samples successfully converged. Following the work of [3], we define that one test sample is successfully converged if its corner error is less than 1.0 pixel.

Implementation details. The number of stages in our CKLN is set to 4. The effect of this hyperparameter will be discussed in more detail at the end of this section. For the CNN at level $k = 1 \dots K$, all its convolutional layers have 64 filters except for the last layer, which has C filters to produce an output feature map with C channels. We set $C = 4$ for a good compromise between accuracy and efficiency. In addition, the first $K - k$ convolutional layers are set to have a stride 2 to produce a downsampled feature map with a factor of 2^{K-k} .

For the stopping criterion in the Lucas-Kanade layer, the change of motion parameters is measured by the corner error (Equation 18) at the original resolution. The maximum number of iterations for each Lucas-Kanade layer is set to 20, which is sufficient in our experiments.

For the hinge loss in Equation 17, the setting of the margin δ is driven by the corner error. If each corner decreases its L^2 error by α pixels, then such amount of improvement corresponds to $\delta = d(\mathbf{p}_0, \hat{\mathbf{p}}) - \max(0, \sqrt{d(\mathbf{p}_0, \hat{\mathbf{p}})} - 2\alpha)^2$.

We utilized the open source Torch framework [7] to implement the proposed method. We trained our network using an NVIDIA GeForce GTX TITAN X GPU. It took approximately 25 hours to train our network with four levels, each of which is trained for 50 epochs. Our network is trained by stochastic gradient descent. We preprocessed the images by the standard normalization (subtract mean and divide by standard deviation). The mini-batch size is 64. The network’s parameters are initialized by He’s method [14]. Table 1. lists the hyperparameter setting of our CLKN. The hyperparameters are determined from evaluating the average 0-1 loss (Equation 16) on the validation set. We used a shallower CNN (three layers) in the last level to reduce the memory usage and the training time. We found that such CNN is sufficient to bring roughly aligned images into sub-pixel alignment.

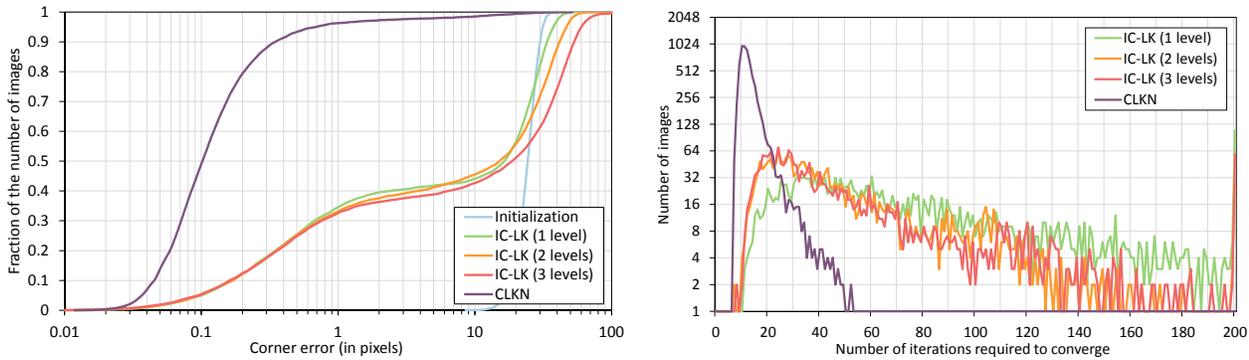


Figure 6. Comparison with our method and the IC-LK algorithm on the test set. *Left*: The cumulative error distributions. The X-axis is the corner error in log scale, and the Y-axis is the fraction of the number of test images. *Right*: The histograms of the number of iterations required to converge successfully (*i.e.* corner error is less than 1.0 pixel). The X-axis is the number of iterations used to converge. The Y-axis is the number of test images in log scale.

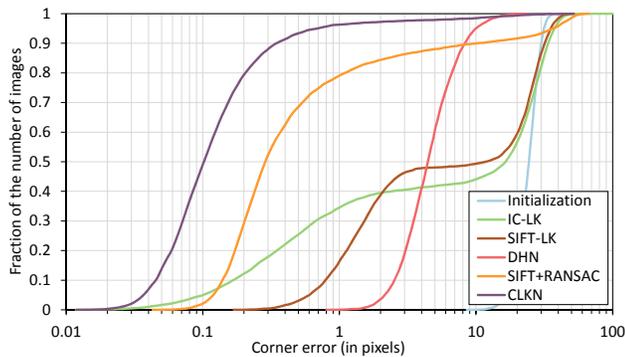


Figure 7. The cumulative error distributions of our method and four representative methods on the test set.

Comparison with the Lucas-Kanade algorithm. We first compare our method (CLKN) with the inverse compositional Lucas-Kanade (IC-LK) algorithm [3]. The coarse-to-fine strategy is adopted in IC-LK to deal with large motions. Three different numbers of pyramid levels are evaluated in this experiment. Figure 6 (left) shows the cumulative error distributions of CLKN and IC-LK. We also show the distribution of the corner errors of using the initial motion parameters \mathbf{p}_0 as the estimated ones. The results show that our method performs significantly better than IC-LK. We can see that adopting the coarse-to-fine strategy in IC-LK does not improve the range of convergence. The main reason is that the motion magnitude between the downsampled images is still too large, which makes IC-LK give a large but inaccurate motion update. Figure 6 (right) shows the histograms of the number of iterations, and we can see that our method requires much less iterations to converge.

Comparison with the other methods. We compare with the following three representative methods:

1. **SIFT-LK** [2]: A feature-based Lucas-Kanade method that performs the inverse compositional Lucas-Kanade algorithm on dense SIFT features. (We implemented their method.)
2. **SIFT+RANSAC** [13]: A feature-based method that extracts SIFT feature points, performs sparse feature matching, and then applies the direct linear transform (DLT) with RANSAC to estimate the homography. (We used the implementation in the OpenCV library.)
3. **DHN** [10]: A deep learning based method that trains a VGG-style CNN to directly regress the homography. (We implemented their method and trained their model on our training set.)

Figure 7 shows the comparison of our method with the aforementioned methods. The IC-LK (1 level), as a pixel-based method, is also included into the comparison. SIFT-LK enjoys a higher accuracy at threshold 3 than IC-LK due to its robustness to color variations. However, SIFT-LK suffers from a lower alignment accuracy compared to IC-LK. Both IC-LK and SIFT-LK degrade in the presence of large motions. SIFT+RANSAC is the second best in terms of alignment accuracy, while it often fails in processing low-textured images. Among these methods, DHN performs the most consistently across all test samples. However, it could only give a rough alignment. The reason may be that its network architecture is suboptimal to the task of homography estimation. Our method significantly outperforms all the competing methods. Our success lies in the pyramid representation of data-driven features, which effectively learn/extract features under different circumstances.

Visualization of feature pyramids. In Figure 8, we show examples of our learned pyramid representation. Different feature maps in the same level focus on different aspects

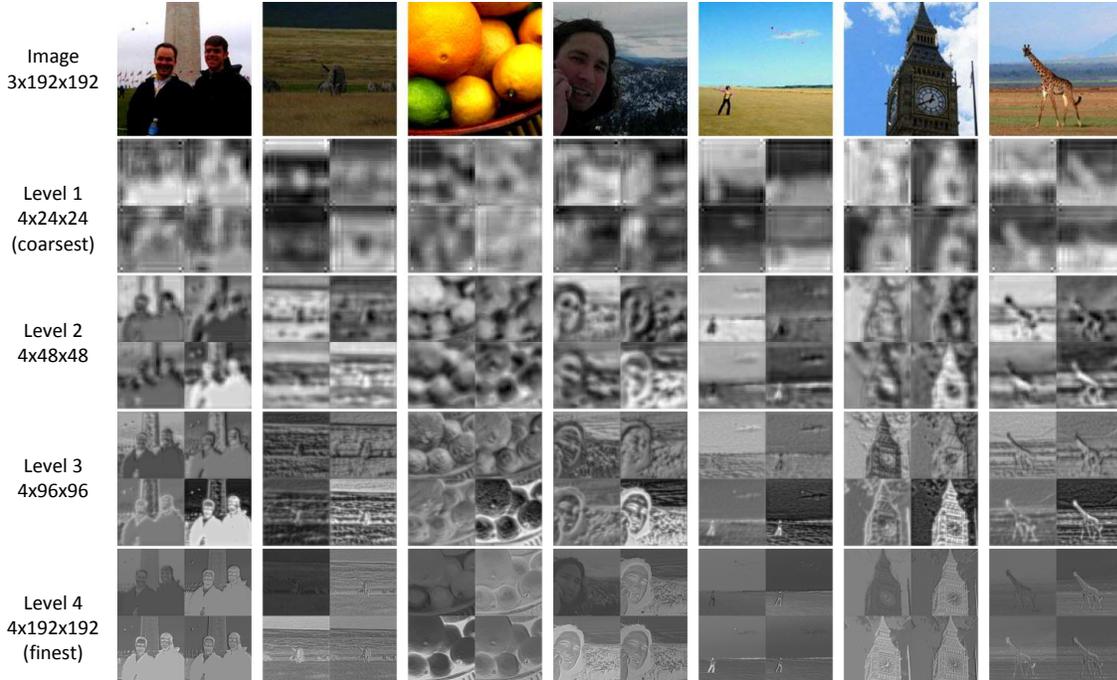


Figure 8. Representative examples of our 4-level convolutional feature pyramids, where each level is a 4-channel feature map.

# levels	# params	# iterations	Time (ms)
3	79.1K	15.6	265
4	98.0K	13.9	260
5	116.9K	19.6	304

Table 2. Performance analysis of CLKN with different number of pyramid levels.

of salient information and complement to each other. The CNNs at level 1 and 2 capture mid-level semantic information such as object parts, and render the feature maps in a spatially-smooth way. On the other hand, the CNNs at level 3 and 4 capture semantic edge information and also fine-scale structures.

Number of pyramid levels. In our method, the number of pyramid levels is an important hyperparameter. We evaluated different numbers of pyramid levels and show the results in Figure 9. Using a 3-level pyramid already outperforms all the competing methods. Using four levels further improves the alignment accuracy. Using more pyramid levels (*e.g.*, five) may degrade the accuracy. The reason is that as the feature map at stage one becomes smaller, the Lucas-Kanade layer has less information to perform alignment. Table 2 summarizes performance analysis. We set the number of pyramid levels to 4 since it achieves the best performance in terms of both accuracy and efficiency.

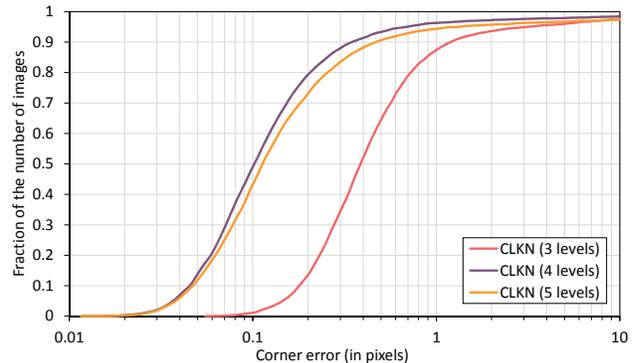


Figure 9. The cumulative error distributions of CLKN with different number of pyramid levels.

5. Conclusion

In this paper, we presented a novel network architecture that performs the Lucas-Kanade algorithm on convolutional feature maps. Adopting our cascaded feature learning method leads to a cascaded Lucas-Kanade network, which performs alignment in a coarse-to-fine manner. Experimental results show that our pyramid representation significantly improves the convergence range of the Lucas-Kanade algorithm. In future work we plan to apply our model to other tasks, such as non-rigid registration and multi-modal image matching.

References

- [1] H. Alismail, B. Browning, and S. Lucey. Robust tracking in low light and sudden illumination changes. In *3DV*, 2016.
- [2] E. Antonakos, J. Alabort-i Medina, G. Tzimiropoulos, and S. P. Zafeiriou. Feature-based Lucas-Kanade and active appearance models. *IEEE Transactions on Image Processing*, 2015.
- [3] S. Baker and I. Matthews. Lucas-Kanade 20 years on: A unifying framework. *IJCV*, 2004.
- [4] S. Benhimane and E. Malis. Real-time image-based tracking of planes using efficient second-order minimization. In *IROS*, 2004.
- [5] M. Brown and D. G. Lowe. Automatic panoramic image stitching using invariant features. *IJCV*, 2007.
- [6] A. Bruhn, J. Weickert, and C. Schnörr. Lucas/Kanade meets Horn/Schunck: Combining local and global optic flow methods. *IJCV*, 2005.
- [7] R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, 2011.
- [8] A. Crivellaro and V. Lepetit. Robust 3D tracking with descriptor fields. In *CVPR*, 2014.
- [9] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *CVPR*, 2005.
- [10] D. DeTone, T. Malisiewicz, and A. Rabinovich. Deep image homography estimation. *arXiv:1606.03798*, 2016.
- [11] J. Engel, T. Schöps, and D. Cremers. LSD-SLAM: Large-scale direct monocular SLAM. In *ECCV*, 2014.
- [12] C. Forster, M. Pizzoli, and D. Scaramuzza. SVO: Fast semi-direct monocular visual odometry. In *ICRA*, 2014.
- [13] R. Hartley and A. Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.
- [14] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, 2015.
- [15] A. G. Howard. Some improvements on deep convolutional neural network based image classification. *arXiv:1312.5402*, 2013.
- [16] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.
- [17] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu. Spatial transformer networks. In *NIPS*, 2015.
- [18] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *ECCV*, 2014.
- [19] S. Liu, L. Yuan, P. Tan, and J. Sun. Steadyflow: Spatially smooth optical flow for video stabilization. In *CVPR*, 2014.
- [20] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015.
- [21] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 2004.
- [22] B. D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *IJCAI*, 1981.
- [23] E. Marchand, H. Uchiyama, and F. Spindler. Pose estimation for augmented reality: a hands-on survey. *IEEE Transactions on Visualization and Computer Graphics*, 2016.
- [24] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.
- [25] K. B. Petersen and M. S. Pedersen. The matrix cookbook. *Technical University of Denmark*, 2008.
- [26] R. Szeliski. Image alignment and stitching: A tutorial. *Foundations and Trends® in Computer Graphics and Vision*, 2006.
- [27] J. Zaragoza, T.-J. Chin, M. S. Brown, and D. Suter. As-projective-as-possible image stitching with moving DLT. In *CVPR*, 2013.