Learning Image Relations with Contrast Association Networks

Yao Lu
ANU, Data61, ACRV
yaolubrain@gmail.com

Zhirong Yang
NTNU
zhirong.yang@ntnu.no

Juho Kannala
Aalto University
juho.kannala@aalto.fi

Samuel Kaski
Aalto University
samuel.kaski@aalto.fi

Abstract—Inferring the relations between two images is an important class of tasks in computer vision. Examples of such tasks include computing optical flow and stereo disparity. We treat the relation inference tasks as a machine learning problem and tackle it with neural networks. A key to the problem is learning a representation of relations. We propose a new neural network module, contrast association unit (CAU), which explicitly models the relations between two sets of input variables. Due to the non-negativity of the weights in CAU, we adopt a multiplicative update algorithm for learning these weights. Experiments show that neural networks with CAUs are more effective in learning five fundamental image transformations than conventional neural networks.

I. INTRODUCTION

Neural networks, especially convolutional neural network (CNN) [26], have been successfully applied in many computer vision tasks such as object recognition [6], [25]. A key to this success is that the neural networks allow appearance representation which is invariant of several image transformations such as small translation.

Another important class of tasks in computer vision is the inference of relations between two images. Two images can be related by object motion, camera motion or environmental factors such as lighting change. For these problems, instead of aiming for invariance of appearance changes, we want to detect and estimate appearance changes. For example, given two consecutive video frames, we want to infer the movement of each pixel from one image to the other (optical flow). For another example, given two images taken by a camera on a moving robot, we want to infer the ego-motion of the robot (visual odometry).

The traditional approach to perform the relation inference tasks is knowledge-based. By acquiring knowledge of a task and making reasonable assumptions for simplification, one designs an algorithm to perform the inference. A classic example is the Horn-Schunck algorithm for computing optical flow [13]. However, in situations where we lack sufficient knowledge or the assumptions fail, the knowledge-based approach may not perform well. For example, the Horn-Schunck algorithms assumes brightness constancy of the moving pixels. This assumption can be violated by many factors such as occlusion, shading and noise.

An alternative approach to solve the relation inference problem is learning-based [29]. From a collection of training data, we aim to learn a function

$$\mathbf{z} = F(\mathbf{x}, \mathbf{y}) \tag{1}$$

such that given two images $\mathbf{x}$ and $\mathbf{y}$ as inputs, the function will output their relation variables $\mathbf{z}$. The relation variables can be rotation angle, motion field, affine transformation parameters, etc. Compared to the knowledge-based approach, the learning-based approach does not require sufficient knowledge or assumptions but a large amount of training images with ground-truth relation variables. When it is difficult to obtain the ground-truth for real world images, one can often resort to synthetic images rendered by graphics engines. An example is the Sintel dataset for learning optical flow [4]. Recently, the learning-based approach has been adopted to compute optical flow [7], [17], [34], [40], stereo disparity [28], camera motion [41] and visual odometry [23]. Some of the results are competitive to the knowledge-based methods. Note that relation learning can also serve as supervision for learning appearance representation, as demonstrated in learning ego-motion [2], [18] and robot actions [32].

Additionally, there are methods combining knowledge-based and learning-based approaches [3], [37], [43], [44]. In these methods, a neural network is trained to match two image patches and a knowledge-based post-processing is applied on the matching results to output the relation variables. In this paper, we focus on the pure learning-based approach, that is, neural networks are trained in an end-to-end manner, given raw images as inputs and ground-truth relation variables as targets. Although a relation learning model can also be trained unsupervisedly [22], [30], [42], we restrict our discussion mainly to supervised learning.

While both object recognition and relation inference can be treated as supervised learning tasks, the two tasks have much difference in nature. Object recognition aims for invariance of several image transformations (e.g. translation, rotation and scaling) but relation inference aims for equivariance of these transformations. For example, the conventional CNN with a pooling operation is known to be invariant of small translation. This property is suitable for object recognition but not for motion detection, whose goal is to estimate the translation. This difference should be kept in mind when designing a relation learning model.

In this paper, we propose a new relation unit, *contrast association unit* (CAU). We show that CAUs are suitable
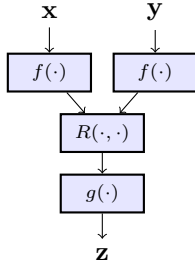
Fig. 1: Neural network architecture for relation learning

for relation learning tasks with analysis and experiments. We adopt a multiplicative update algorithm for learning the non-negative weights in CAUs. The multiplicative update algorithm is compatible with gradient descent algorithms for unconstrained weights. The whole neural network can be trained in an end-to-end manner.

Next, in Section II, we outline the general neural network architecture for relation learning. In Section III, we introduce our proposed relation unit CAU. In Section IV, we present the multiplicative update algorithm for learning the non-negative weights in CAUs. In Section V, we discuss models related to CAU. In Section VI, we present the experiments on the five relation learning tasks.

II. Architecture

As illustrated in Figure 1, the general neural network architecture of many relation learning models [2], [7], [30] can be described as

$$
\begin{aligned}
\mathbf{a} &= f(\mathbf{x}), \quad \mathbf{h} = R(\mathbf{a}, \mathbf{b}), \\
\mathbf{b} &= f(\mathbf{y}), \quad \mathbf{z} = g(\mathbf{h}),
\end{aligned} \tag{2}
$$

where $\mathbf{x}$ and $\mathbf{y}$ are the inputs (images), $\mathbf{z}$ are the targets (relation variables), $f(\cdot)$ is the feature extraction units, $R(\cdot, \cdot)$ is the relation units and $g(\cdot)$ is the decoding units. $f(\cdot)$ and $g(\cdot)$ can be parametric functions such as neural networks. When the relation of two images is on the pixel-level (e.g. affine transformation), $f(\cdot)$ can be the identity function such that the relation units can directly apply on the image pixels. When the relation units $R(\cdot, \cdot)$ can directly output relation variables instead of a hidden representation, $g(\cdot)$ can also be the identity function. For example, in [30], both $f(\cdot)$ and $g(\cdot)$ are the identity function. For another example, in the simple version of FlowNet [7], $f(\cdot)$ is the identity function and $g(\cdot)$ is a CNN while in the complex version, both $f(\cdot)$ and $g(\cdot)$ are CNNs. The main focus of this paper is the relation units $R(\cdot, \cdot)$. A suitable relational representation can reduce the sample complexity and model complexity of learning $g(\cdot)$. We present two common types of relation units in below.

A. Concatenation Units

Concatenation units are defined as

$$
\mathbf{h} = [\mathbf{a} \ \ \mathbf{b}]. \tag{3}
$$

For this simple representation, $g(\cdot)$ is solely responsible for learning the relations between $\mathbf{a}$ and $\mathbf{b}$. Concatenation units have been used in learning optical flow [7] and ego-motion [2].

B. Bilinear Units

Bilinear units have been previously proposed and developed [12], [31], [39]. They are defined as, for the k-th unit,

$$
h_k = \sum_{ij} W_{ijk} a_i b_j = \mathbf{a}^T \mathbf{W}_k \mathbf{b}, \tag{4}
$$

where $\mathbf{W}_k$ is the parameters to be learned. A bilinear unit models the pair-wise multiplicative intersection between two sets of input variables. It is equivalent to inner product when $\mathbf{W}_k$ is the identity matrix and equivalent to outer product since (4) can be written as $h_k = \text{vec}(\mathbf{W}_k)^T \text{vec}(\mathbf{ab}^T)$, where $\text{vec}(\cdot)$ denotes vectorization. Bilinear units have been used in learning image transformations [30], [35].

III. Contrast Association Units

We propose a new relation unit, *contrast association unit* (CAU), which associates two sets of input variables. CAUs are defined as, for the k-th unit,

$$
h_k = \frac{1}{2} \sum_{ij} W_{ijk} (a_i - b_j)^2, \tag{5}
$$

where $W_{ijk} \geq 0$. Each CAU can be interpreted as a weighted sum of mismatches between $\mathbf{a}$ and $\mathbf{b}$. The non-negative constraint on the weight matrices is indispensable since the mismatches should be non-negative to be accumulated. Otherwise, positive mismatches and negative mismatches would cancel each other.

Compared to concatenation and bilinear units, CAUs have two advantages: (1) as the name suggests, CAUs model the contrast between two sets of variables such that $R(\mathbf{a} + c, \mathbf{b} + c) = R(\mathbf{a}, \mathbf{b})$, where c is a scalar applied on each element of a vector. This property is desirable for relation inference. For example, when $\mathbf{a}$ and $\mathbf{b}$ are the raw pixels, their relations should not be affected by their absolute pixel intensity level. (2) CAUs represent relations more explicitly. Each CAU stands for the matching error of a certain relation encoded by the weight matrix. An example is given in Section III-B. This interpretablity also inspires the use of competition among CAUs as described below.

A. Competition

We apply a competition mechanism among CAUs. The competition encourages the unit standing for the minimum matching error to pop out. As a result, the relation representation can be more easily read-out by the decoding units. A classic competition mechanism is the winner-take-all (WTA), defined as

$$
h_k' = \begin{cases} 1, & \text{if } h_k = \min(\mathbf{h}), \\ 0, & \text{otherwise.} \end{cases} \tag{6}
$$

WTA is of conceptual interest, which is demonstrated in Section III-B. However, WTA is not differentiable and discards too much information. In practice, we can use softmin competition, defined as

$$h'_k = \frac{e^{-h_k}}{\sum_i e^{-h_i}}. \tag{7}$$

In all our experiments, adding the softmin competition significantly improves the results of neural networks with CAUs.

B. Example

To understand how neural networks with CAUs represent relations, let us consider a simple example of translation detection. Let

$$\mathbf{a} = (c_1, c_2, c_3, c_4, c_5), \tag{8}$$
$$\mathbf{b}_1 = (c_2, c_3, c_4, c_5, c_6), \tag{9}$$
$$\mathbf{b}_2 = (c_1, c_2, c_3, c_4, c_5), \tag{10}$$
$$\mathbf{b}_3 = (c_0, c_1, c_2, c_3, c_4), \tag{11}$$

where $\{c_i\}$ are arbitrary numbers. Let $z \in \{-1, 0, 1\}$ be the translation variable and

$$\mathbf{b} = \begin{cases} \mathbf{b}_1, & \text{if } z = -1, \\ \mathbf{b}_2, & \text{if } z = 0, \\ \mathbf{b}_3, & \text{if } z = 1. \end{cases} \tag{12}$$

Denote by $\mathbf{D}(\mathbf{a}, \mathbf{b})$ the matrix of pair-wise squared differences of the elements in $\mathbf{a}$ and $\mathbf{b}$. The element of index (i, j) in $\mathbf{D}(\mathbf{a}, \mathbf{b})$ is $(a_i - b_j)^2$. Then we have

$$\underbrace{\begin{bmatrix} * & * & * & * & * \\ 0 & * & * & * & * \\ * & 0 & * & * & * \\ * & * & 0 & * & * \\ * & * & * & 0 & * \end{bmatrix}}_{\mathbf{D}(\mathbf{a},\mathbf{b}_1)}, \underbrace{\begin{bmatrix} 0 & * & * & * & * \\ * & 0 & * & * & * \\ * & * & 0 & * & * \\ * & * & * & 0 & * \\ * & * & * & * & 0 \end{bmatrix}}_{\mathbf{D}(\mathbf{a},\mathbf{b}_2)}, \underbrace{\begin{bmatrix} * & 0 & * & * & * \\ * & * & 0 & * & * \\ * & * & * & 0 & * \\ * & * & * & * & 0 \\ * & * & * & * & * \end{bmatrix}}_{\mathbf{D}(\mathbf{a},\mathbf{b}_3)} \tag{13}$$

where $*$ denotes the element whose value we do not care about. We construct three CAUs (h_1, h_2, h_3) with the following weight matrices,

$$\underbrace{\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}}_{\mathbf{W}_1}, \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}}_{\mathbf{W}_2}, \underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}}_{\mathbf{W}_3} \tag{14}$$

respectively. If $\mathbf{b} = \mathbf{b}_i$, then $h_i = 0$ and $h_{j \neq i} > 0$ except for some special cases. With WTA competition, $h'_i = 1$ and $h'_{j \neq i} = 0$. The translation variable z can be inferred with simple decoding units $g(\mathbf{h}') = (-1, 0, 1) \cdot \mathbf{h}' = z$.

C. Low-rank Approximation

If $\mathbf{W}_k$ is large, we can approximate it in the following way. For rank-one approximation, let $\mathbf{W}_k = \mathbf{u}_k \mathbf{v}_k^T$, where $\mathbf{u}_k$ and $\mathbf{v}_k$ are a row of non-negative matrices $\mathbf{U}$ and $\mathbf{V}$, respectively.

Then the right side of (5) can be written in the matrix form as

$$\mathbf{h}^* = \frac{1}{2} \Big[(\mathbf{V}\mathbf{1}) \circ \mathbf{U}(\mathbf{a})^2 + (\mathbf{U}\mathbf{1}) \circ \mathbf{V}(\mathbf{b})^2 \Big] - (\mathbf{U}\mathbf{a}) \circ (\mathbf{V}\mathbf{b}), \tag{15}$$

where $\mathbf{1}$ is a vector of ones, $\circ$ is the element-wise multiplication and $(\cdot)^2$ is the element-wise square. The derivation can be found in Appendix. To obtain CAUs of higher ranks, we can apply sum-pooling over $\mathbf{h}^*$. That is, divide $\mathbf{h}^*$ into non-overlapping groups of equal size and sum the units in each group.

IV. Learning

To learn the non-negative weights in CAUs ($\mathbf{W}_k$ for full rank or $\mathbf{U}$ and $\mathbf{V}$ for rank-one), the conventional gradient descent based algorithms are not suitable because the non-negativity of weights cannot be maintained after each update. Simply projecting weights onto the space of the nonnegative matrices after each update performs poorly in our experiments, where the learning converges at an extremely low speed or even diverges in many cases.

To address the above problem, we adopt a multiplicative update algorithm for the non-negative weight matrices in a neural network, which was originally used for non-negative matrix factorization [27]. For a non-negative matrix $\mathbf{W}$ in a neural network and loss function E, we decompose the gradient into two positive parts $\frac{\partial E}{\partial \mathbf{W}} = \nabla^+ - \nabla^-$, where the two positive matrices ∇^+ and ∇^- can be computed by

$$\nabla^+ = \frac{1}{2} \left(\text{abs} \left(\frac{\partial E}{\partial \mathbf{W}} \right) + \frac{\partial E}{\partial \mathbf{W}} \right) + \epsilon, \tag{16}$$

$$\nabla^- = \frac{1}{2} \left(\text{abs} \left(\frac{\partial E}{\partial \mathbf{W}} \right) - \frac{\partial E}{\partial \mathbf{W}} \right) + \epsilon, \tag{17}$$

where $\text{abs}(\cdot)$ is the element-wise absolute value and ϵ is a small positive scalar applied to each element of a matrix. The multiplicative update algorithm is defined as

$$\mathbf{W} \leftarrow \mathbf{W} \circ \left(\frac{\nabla^-}{\nabla^+} \right)^\eta, \tag{18}$$

where η is the learning rate hyperparameter and the division and the exponentiation are both element-wise. If $\mathbf{W}$ is initialized to be positive, the updated matrix will remain positive since all factors on the right hand side are positive.

In practice, the multiplicative update algorithm can be used in a stochastic (or mini-batch) manner. Such use has been demonstrated in non-negative matrix factorization [36]. Note that the multiplicative update still requires the gradients calculated by back-propagation. The gradient calculation for CAU can be found in Appendix.

V. Related Work

A classic model related to the proposed CAU is the energy model for motion detection [1] and stereo disparity [8]. Each unit of the energy model computes the sum of squares of two Gabor filter outputs. No learning is involved in the energy model. There are models which compute the sum of squares

of learnable filter outputs such as adaptive-subspace self-organized maps (ASSOM) [21] and independent subspace analysis (ISA) [16]. Similar to CAU, ASSOM also has a competition mechanism (WTA). However, the goal of both ASSOM and ISA is to learn appearance features which are invariant of image transformations, as different from our goal. There is a line of research on relation learning based on Boltzmann machines [15], [30], [38]. While Boltzmann machines allow a probabilistic formulation of relation inference, the training of Boltzmann machines is much more expensive compared to their non-probabilistic counterpart. Non-negative weights have appeared in sum-product networks [33], multi-layer perceptrons [5] and natural image statistics models [9], [14]. Our derivation of the low-rank approximation of CAUs follows from the low-rank approximation of bilinear units [19], [30].

VI. EXPERIMENTS

We consider five fundamental image transformations for the relation learning tasks. For image $\mathbf{x}$, its transformed image $\mathbf{y}$ is synthetically generated with ground-truth transformation parameters (or relation variables) $\mathbf{z}$. For each task, neural networks are trained in a supervised manner, given $\mathbf{x}$ and $\mathbf{y}$ as inputs and $\mathbf{z}$ as targets. We describe the details below.

A. Tasks

The five image transformations are: translation, rotation, scaling, affine transformation and projective transformation. They are called geometric transformations and can be unified as follows [10]. An image can be transformed (or warped) by changing its coordinates. For each point of an image with homogeneous coordinates $\mathbf{p} = (p_1, p_2, 1)$, the transformed point is $\mathbf{p}' = \mathbf{H}\mathbf{p}$ with homography matrix

$$\mathbf{H} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}. \tag{19}$$

The type of the transformation depends on the parametrization of $\mathbf{H}$. Note that translation, rotation and scaling are special cases of affine transformation and affine transformation is a special case of projective transformation. We list the parametrization in each task and the range of the transformation parameters below.

- Translation. $\mathbf{z} \in [-5, 5]^2$ and $\mathbf{H} =$
$$\begin{bmatrix} 1 & 0 & z_1 \\ 0 & 1 & z_2 \\ 0 & 0 & 1 \end{bmatrix}.$$
Images are translated by z_1 pixels horizontally and z_2 pixels vertically.

- Rotation. $\mathbf{z} = z \in [-45, 45]$, $\theta = \text{radian}(z)$ and $\mathbf{H} =$
$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Images are rotated by z degrees.

- Scaling. $\mathbf{z} \in [0.5, 2]^2$ and $\mathbf{H} =$
$$\begin{bmatrix} z_1 & 0 & 0 \\ 0 & z_2 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$
Images are scaled by z_1 horizontally and z_2 vertically.

- Affine. $\mathbf{z} \in [-0.5, 0.5]^4$ and $\mathbf{H} =$
$$\begin{bmatrix} 1 + z_1 & z_2 & 0 \\ z_3 & 1 + z_4 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$
To simplify the problem, we discard translation such that h_{13} and h_{23} are zero.

- Projective. $\mathbf{z} \in [-0.5, 0.5]^4 \times [-0.01, 0.01]^2$ and $\mathbf{H} =$
$$\begin{bmatrix} 1 + z_1 & z_2 & 0 \\ z_3 & 1 + z_4 & 0 \\ z_5 & z_6 & 1 \end{bmatrix}.$$
z_5 and z_6 are much more sensitive than the other variables. Hence, their range is set to be smaller. Larger range will cause extremely distorted images. To simplify the problem, we discard translation such that h_{13} and h_{23} are zero.

B. Data

We generate training image patches from the gray-scaled CIFAR-10 dataset[1] [24]. For each task and for each image in CIFAR-10, we apply an image transformation with $\mathbf{z}$ randomly sampled from uniform distributions to obtain an image pair. Then we crop an image patch of size 11×11 at the center of each image of the image pair. Repeat the process 10 times. With this procedure, we obtain a training set of size 500,000 and a testing set of size 100,000 for the relation learning tasks.

C. Models

We test three neural network models, each of which uses a different type of relation units: concatenation, bilinear and CAU. For all the models, $f(\cdot)$ is the identity function and $g(\cdot)$ is a multi-layer perceptron. We call the three neural network models, concatenation network (CTN), bilinear network (BLN) and *contrast association network* (CAN), respectively. For BLN and CAN, we use low-rank approximation of the bilinear units and CAUs, as described in Section III-C. In BLN, we apply l_2 normalization, which empirically performs better than softmax (or softmin), on the outputs of bilinear units. We also experimented with the non-negative constraint on the bilinear units but found it performs poorly and therefore discarded it. For fairness of the comparison, all three models are set to have essentially the same size in each task. To test the generality, all models are set to have the same size in different
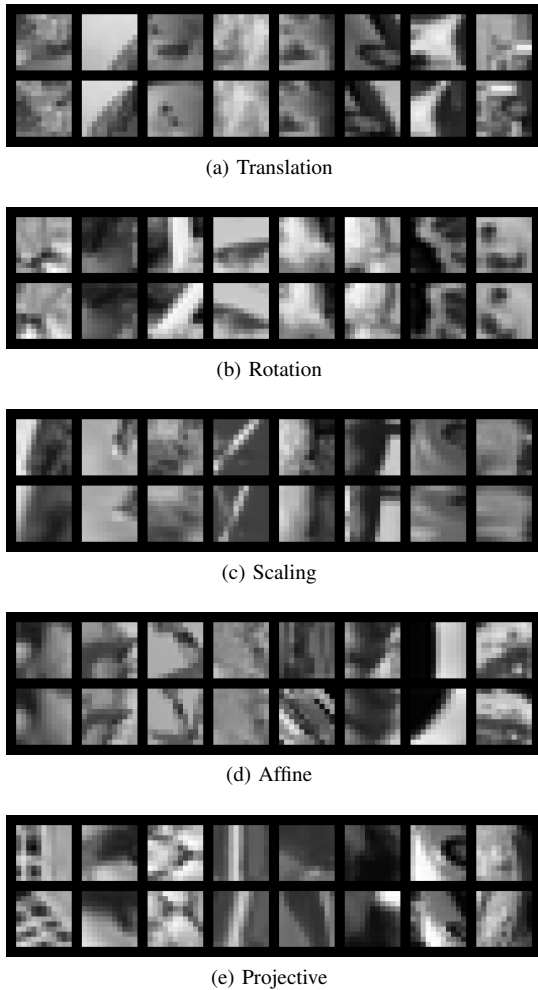
---

[1] https://www.cs.toronto.edu/~kriz/cifar.html

(a) Translation

(b) Rotation

(c) Scaling

(d) Affine

(e) Projective

Fig. 2: Sample image patches. For (a)-(e), the first row contains the original image patches $\mathbf{x}$ and the second row contains the corresponding transformed image patches $\mathbf{y}$.

tasks, except that the last layer depends on the dimensionality of the targets $\mathbf{z}$ in the task. The models are specified in Table I.

D. Settings

All image patches are of size 11×11 and are reshaped to vectors of size 121. The numerical range of each element of the vectors is normalized from $[0, 255]$ to $[-0.5, 0.5]$ by dividing 255 and then subtracting 0.5. This normalization significantly accelerates the training.

We use the mean-squared-error (MSE) as the loss function.

All the parameters of all models are initialized randomly according to uniform distributions. We use the multiplicative update algorithm (described in Section IV) for non-negative weights ($\mathbf{U}$ and $\mathbf{V}$) and Adam [20] for unconstrained weights.

For all models in all tasks, we use the same training hyper-parameter setting. The initial learning rates are $\eta = 0.005$ (multiplicative update) and $\alpha = 0.005$ (Adam). Both are multiplied by 0.95 for every 500 mini-batch updates. The

TABLE I: Model specification. The size of each layer is on the right, if applicable. Concat denotes concatenation. Linear denotes fully connected layer. PReLU denotes the parametric ReLU activation function [11]. * denotes rank-one approximation. The sum-pooling pools every 4 elements of a vector. dim($\mathbf{z}$) denotes the dimensionality of $\mathbf{z}$.

CTN	BLN	CAN
Concat	Bilinear*, 1200	CAU*, 1200
Linear, 1200	Sum-Pooling, 300	Sum-Pooling, 300
PReLU	l_2 Norm	Softmin
Linear, 300	Linear, 100	Linear, 100
PReLU	PReLU	PReLU
Linear, 100	Linear, 100	Linear, 100
PReLU	PReLU	PReLU
Linear, 100	Linear, dim($\mathbf{z}$)	Linear, dim($\mathbf{z}$)
PReLU		
Linear, dim($\mathbf{z}$)		

size of each mini-batch is 100. There are 200,000 mini-batch updates in total. $\epsilon = 10^{-20}$ in the multiplicative update.

We use MATLAB[2] for generating the data and Torch[3] for neural network training and testing.

E. Results

We use two measures of error to evaluate our results.

- Parameter error. It is defined as the MSE between the ground-truth transformation parameters $\mathbf{z}$ and the inferred parameters $\widehat{\mathbf{z}}$, that is, $\|\mathbf{z} - \widehat{\mathbf{z}}\|_2$. An advantage of parameter error is interpretability. For example, in inferring the rotation between two images, it is desirable that the inference error is measured in degrees. A disadvantage is that it is difficult to compare the inference error across tasks since different tasks have different range of transformation parameters.

- Transformation error. Define four points with homogeneous coordinates

$$\mathbf{p}_1 = (0, 0, 1), \ \mathbf{p}_2 = (1, 0, 1),$$
$$\mathbf{p}_3 = (1, 1, 1), \ \mathbf{p}_4 = (0, 1, 1).$$

Let $\mathbf{p}'_i = \mathbf{H}\mathbf{p}_i$ be the transformed point with the ground-truth homography $\mathbf{H}$ and let $\widehat{\mathbf{p}}'_i = \widehat{\mathbf{H}}\mathbf{p}_i$ be the transformed point with the inferred homography $\widehat{\mathbf{H}}$. The transformation error is defined as

$$\frac{\sum_{i=1}^4 \|\mathbf{p}'_i - \widehat{\mathbf{p}}'_i\|_2}{\sum_{j=1}^4 \|\mathbf{p}'_j\|_2}. \tag{20}$$

It is scale-invariant in the sense that the error is unchanged if we multiply $\mathbf{H}$ and $\widehat{\mathbf{H}}$ by a non-zero constant. Therefore it is more suitable to compare this inference error across different tasks.

The results are listed in Tables II and III, with errors averaged over the testing sets. We can see that CAN achieves the lowest errors in every task. From Table III, we can see that the transformation error of CAN goes up when the complexity of the tasks is increased.

TABLE II: Mean parameter error on testing sets.

Task	CTN	BLN	CAN
Translation	0.773	1.893	**0.049**
Rotation	9.854	5.925	**3.518**
Scaling	0.018	0.025	**0.017**
Affine	0.014	0.020	**0.010**
Projective	**0.030**	0.032	**0.030**

TABLE III: Mean transformation error on testing sets.

Task	CTN	BLN	CAN
Translation	0.198	0.322	**0.014**
Rotation	0.025	0.018	**0.014**
Scaling	0.056	0.065	**0.055**
Affine	0.084	0.100	**0.072**
Projective	0.160	0.166	**0.158**

VII. DICUSSION

In our present work, the experiments are limited to small image patches and to simple image transformations. It is not clear how well CAUs perform on whole images and more complex tasks. Future work includes extending CAUs to handle whole images and more complex tasks such as three-dimensional transformations.

APPENDIX

Rank-One Approximation of CAU

Since $\mathbf{W}_k = \mathbf{u}_k \mathbf{v}_k^T$, we have

$$h_k = \frac{1}{2} \sum_{ij} W_{ijk} (a_i - b_j)^2 \tag{21}$$

$$= \frac{1}{2} \sum_{ij} W_{ijk} (a_i^2 + b_j^2) - \sum_{ij} W_{ijk} a_i b_j \tag{22}$$

$$= \frac{1}{2} \left[\mathbf{1}^T \mathbf{W}_k^T (\mathbf{a})^2 + \mathbf{1}^T \mathbf{W}_k (\mathbf{b})^2 \right] - \mathbf{a}^T \mathbf{W}_k \mathbf{b} \tag{23}$$

$$= \frac{1}{2} \left[\mathbf{1}^T \mathbf{v}_k \mathbf{u}_k^T (\mathbf{a})^2 + \mathbf{1}^T \mathbf{u}_k \mathbf{v}_k^T (\mathbf{b})^2 \right] - \mathbf{a}^T \mathbf{u}_k \mathbf{v}_k^T \mathbf{b}. \tag{24}$$

In matrix form

$$\mathbf{h} = \frac{1}{2} \left[(\mathbf{V1}) \circ \mathbf{U} \mathbf{a}^2 + (\mathbf{U1}) \circ \mathbf{V} \mathbf{b}^2 \right] - (\mathbf{Ua}) \circ (\mathbf{Vb}). \tag{25}$$

Gradients of CAU

$$\frac{\partial E}{\partial \mathbf{a}} = \sum_k \frac{\partial E}{\partial h_k} \frac{\partial h_k}{\partial \mathbf{a}}, \quad \frac{\partial E}{\partial \mathbf{b}} = \sum_k \frac{\partial E}{\partial h_k} \frac{\partial h_k}{\partial \mathbf{b}}, \tag{26}$$

$$\frac{\partial h_k}{\partial \mathbf{a}} = (\mathbf{W}_k^T \mathbf{1}) \circ \mathbf{a} - \mathbf{W}_k \mathbf{b}, \tag{27}$$

$$\frac{\partial h_k}{\partial \mathbf{b}} = (\mathbf{W}_k \mathbf{1}) \circ \mathbf{b} - \mathbf{W}_k^T \mathbf{a}, \tag{28}$$

$$\frac{\partial E}{\partial \mathbf{W}_k} = \frac{\partial E}{\partial h_k} \frac{\partial h_k}{\partial \mathbf{W}_k}, \tag{29}$$

$$\frac{\partial h_k}{\partial \mathbf{W}_k} = \frac{1}{2} \left[(\mathbf{a}^2) \mathbf{1}^T + \mathbf{1} (\mathbf{b}^2)^T \right] - \mathbf{a} \mathbf{b}^T. \tag{30}$$

For $\mathbf{W}_k = \mathbf{u}_k \mathbf{v}_k^T$,

$$\frac{\partial E}{\partial \mathbf{u}_k} = \frac{\partial E}{\partial h_k} \frac{\partial h_k}{\partial \mathbf{u}_k}, \quad \frac{\partial E}{\partial \mathbf{v}_k} = \frac{\partial E}{\partial h_k} \frac{\partial h_k}{\partial \mathbf{v}_k}, \tag{31}$$

$$\frac{\partial h_k}{\partial \mathbf{u}_k} = \frac{1}{2} \left[(\mathbf{v}_k^T \mathbf{1}) \mathbf{a}^2 + (\mathbf{v}_k^T \mathbf{b}^2) \mathbf{1} \right] - (\mathbf{v}_k^T \mathbf{b}) \mathbf{a}, \tag{32}$$

$$\frac{\partial h_k}{\partial \mathbf{v}_k} = \frac{1}{2} \left[(\mathbf{u}_k^T \mathbf{1}) \mathbf{b}^2 + (\mathbf{u}_k^T \mathbf{a}^2) \mathbf{1} \right] - (\mathbf{u}_k^T \mathbf{a}) \mathbf{b}. \tag{33}$$

REFERENCES

[1] E. H. Adelson and J. R. Bergen. Spatiotemporal energy models for the perception of motion. *JOSA A*, 1985.
[2] P. Agrawal, J. Carreira, and J. Malik. Learning to see by moving. *ICCV*, 2015.
[3] C. Bailer, K. Varanasi, and D. Stricker. Cnn-based patch matching for optical flow with thresholded hinge loss. *arXiv*, 2016.
[4] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black. A naturalistic open source movie for optical flow evaluation. *ECCV*, 2012.
[5] J. Chorowski and J. M. Zurada. Learning understandable neural networks with nonnegative weight constraints. *IEEE TNNLS*, 2015.
[6] D. Ciresan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. *CVPR*, 2012.
[7] A. Dosovitskiy, P. Fischer, E. Ilg, P. Häusser, C. Hazırbaş, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox. Flownet: Learning optical flow with convolutional networks. *ICCV*, 2015.
[8] D. J. Fleet, H. Wagner, and D. J. Heeger. Neural encoding of binocular disparity: energy models, position shifts and phase shifts. *Vision Research*, 1996.
[9] M. U. Gutmann and A. Hyvärinen. A three-layer model of natural image statistics. *Journal of Physiology-Paris*, 2013.
[10] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. 2003.
[11] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *ICCV*, 2015.
[12] G. F. Hinton. A parallel computation that assigns canonical object-based frames of reference. *IJCAI*, 1981.
[13] B. K. Horn and B. G. Schunck. Determining optical flow. *Artificial Intelligence*, 1981.
[14] P. O. Hoyer and A. Hyvärinen. A multi-layer sparse coding network learns contour coding from natural images. *Vision research*, 2002.
[15] Y. Huang, W. Wang, and L. Wang. Conditional high-order boltzmann machine: A supervised learning model for relation learning. *ICCV*, 2015.
[16] A. Hyvärinen and P. Hoyer. Emergence of phase-and shift-invariant features by decomposition of natural images into independent feature subspaces. *Neural Computation*, 2000.
[17] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. *arXiv*, 2016.
[18] D. Jayaraman and K. Grauman. Learning image representations tied to ego-motion. *ICCV*, 2015.
[19] J.-H. Kim, K.-W. On, J. Kim, J.-W. Ha, and B.-T. Zhang. Hadamard product for low-rank bilinear pooling. *ICLR*, 2017.

[20] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *ICLR*, 2014.

[21] T. Kohonen. Emergence of invariant-feature detectors in the adaptive-subspace self-organizing map. *Biological Cybernetics*, 1996.

[22] K. Konda and R. Memisevic. Unsupervised learning of depth and motion. *arXiv*, 2013.

[23] K. R. Konda and R. Memisevic. Learning visual odometry with a convolutional network. *VISAPP*, 2015.

[24] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. 2009.

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *NIPS*, 2012.

[26] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

[27] D. D. Lee and H. S. Seung. Algorithms for non-negative matrix factorization. *NIPS*, 2001.

[28] W. Luo, A. G. Schwing, and R. Urtasun. Efficient deep learning for stereo matching. *CVPR*, 2016.

[29] R. Memisevic. Learning to relate images. *IEEE TPAMI*, 2013.

[30] R. Memisevic and G. E. Hinton. Learning to represent spatial transformations with factored higher-order boltzmann machines. *Neural Computation*, 2010.

[31] B. A. Olshausen, C. H. Anderson, and D. C. Van Essen. A neurobiological model of visual attention and invariant pattern recognition based on dynamic routing of information. *Journal of Neuroscience*, 1993.

[32] L. Pinto, D. Gandhi, Y. Han, Y.-L. Park, and A. Gupta. The curious robot: Learning visual representations via physical interactions. *ECCV*, 2016.

[33] H. Poon and P. Domingos. Sum-product networks: A new deep architecture. *NIPS*, 2011.

[34] A. Ranjan and M. J. Black. Optical flow estimation using a spatial pyramid network. *arXiv*, 2016.

[35] I. Rocco, R. Arandjelović, and J. Sivic. Convolutional neural network architecture for geometric matching. *CVPR*, 2017.

[36] R. Serizel, S. Essid, and G. Richard. Mini-batch stochastic approaches for accelerated multiplicative updates in nonnegative matrix factorisation with beta-divergence. *MLSP*, 2016.

[37] A. Shaked and L. Wolf. Improved stereo matching with constant highway networks and reflective confidence learning. *arXiv*, 2016.

[38] J. Susskind, G. Hinton, R. Memisevic, and M. Pollefeys. Modeling the joint density of two images under a variety of transformations. *CVPR*, 2011.

[39] J. B. Tenenbaum and W. T. Freeman. Separating style and content with bilinear models. *Neural Computation*, 2000.

[40] J. Thewlis, S. Zheng, P. H. Torr, and A. Vedaldi. Fully-trainable deep matching. *BMVC*, 2016.

[41] B. Ummenhofer, H. Zhou, J. Uhrig, N. Mayer, E. Ilg, A. Dosovitskiy, and T. Brox. Demon: Depth and motion network for learning monocular stereo. *arXiv*, 2016.

[42] J. J. Yu, A. W. Harley, and K. G. Derpanis. Back to basics: Unsupervised learning of optical flow via brightness constancy and motion smoothness. *arXiv*, 2016.

[43] S. Zagoruyko and N. Komodakis. Learning to compare image patches via convolutional neural networks. *CVPR*, 2015.

[44] J. Zbontar and Y. LeCun. Stereo matching by training a convolutional neural network to compare image patches. *JMLR*, 2016.