gvnn: Neural Network Library for Geometric Computer Vision

Ankur Handa[1](✉), Michael Bloesch[3], Viorica Pătrăucean[2], Simon Stent[2], John McCormac[1], and Andrew Davison[1]

[1] Dyson Robotics Laboratory, Department of Computing,
Imperial College London, London, UK
handa.ankur@gmail.com,
{brendon.mccormac13,ajd}@ic.ac.uk

[2] Department of Engineering, University of Cambridge, Cambridge, UK
{vp344,sais2}@cam.ac.uk

[3] Robotic Systems Lab, ETH Zurich, Zurich, Switzerland
bloeschm@ethz.ch

Abstract. We introduce **gvnn**, a neural network library in Torch aimed towards bridging the gap between classic geometric computer vision and deep learning. Inspired by the recent success of Spatial Transformer Networks, we propose several new layers which are often used as parametric transformations on the data in geometric computer vision. These layers can be inserted within a neural network much in the spirit of the original spatial transformers and allow backpropagation to enable end-to-end learning of a network involving any domain knowledge in geometric computer vision. This opens up applications in learning invariance to 3D geometric transformation for place recognition, end-to-end visual odometry, depth estimation and unsupervised learning through warping with a parametric transformation for image reconstruction error.

Keywords: Spatial transformer networks · Geometric vision · Unsupervised learning

1 Introduction

Spatial transformers [1] represent a class of differentiable layers that can be inserted in a standard convolutional neural network architecture to enable invariance to certain geometric transformations on the input data and warping for reconstruction error [2]. In this work, we build upon the 2D transformation layers originally proposed in the spatial transformer networks [1] and provide various novel extensions that perform geometric transformations which are often used in geometric computer vision. These layers have mostly no internal parameters that need learning but allow backpropagation and can be inserted in a neural network for any fixed differentiable geometric operation to be performed on the data. This opens up an exciting new path to blend ideas from geometric computer vision into deep learning architectural designs allowing the exploitation of problem-specific domain knowledge.

G. Hua and H. Jégou (Eds.): ECCV 2016 Workshops, Part III, LNCS 9915, pp. 67–82, 2016.
DOI: 10.1007/978-3-319-49409-8_9

Geometric computer vision has heavily relied on generative parametric models of inverse computer graphics to enable reasoning and understanding of real physical environments that provide rich observations in the form of images or video streams. These fundamentals and principles have been very well understood and form the backbone of large-scale point cloud reconstruction from multi-view image data, visual odometry, and image registration. In this work, we provide a comprehensive library that allows implementation of various image registration and reconstruction methods using these geometric transformation modules within the framework of convolutional neural networks. This means that certain elements in the classic geometric vision based methods that are hand-engineered can be replaced by a module that can be learnt end-to-end within a neural network. Our library is implemented in Torch [3] and builds upon the open source implementation of spatial transformer networks [4].

2 gvnn: Geometric Vision with Neural Networks

We introduce **gvnn**, a Torch package dedicated to performing transformations that are often used in geometric computer vision applications within a neural network. These transformations are implemented as fixed differentiable computational blocks that can be inserted within a convolutional neural network and are useful for manipulating the input data as per the domain knowledge in geometric computer vision. We expand on various novel transformation layers below that form the core part of the library built on top of the open source implementation [4] of spatial transformer networks.

Let us assume that $\mathcal{C}$ represents the cost function being optimised by the neural network. For a regression network it can take the following form *e.g.* $\mathcal{C} = \frac{1}{2}||\mathbf{y}_{pred} - \mathbf{y}_{gt}||^2$ where $\mathbf{y}_{pred}$ is a prediction vector produced by the network and $\mathbf{y}_{gt}$ is the corresponding ground truth vector. This allows us to propagate derivatives from the loss function back to the input to any layer in the network.

2.1 Global Transformations

We begin by extending the 2D transformations introduced in the original spatial transformer networks (STN) to their 3D counterparts. These transformations encode the global movement of the whole image *i.e.* the same transformation is applied to every pixel in the image or any 3D point in the world.

SO3 Layer. Rotations in our network are represented by the so(3) vector (or $\mathfrak{so}(3)$ skew symmetric matrix), which is compact 3×1 vector representation, $\mathbf{v} = (v_1, v_2, v_3)^T$, and is turned into a rotation matrix via the SO3 exponential map, *i.e.* $\mathsf{R}(\mathbf{v}) = \exp([\mathbf{v}]_\times)$. The backpropagation derivatives for $\mathbf{v}$ can be conveniently written as [5]

$$\frac{\partial \mathcal{C}}{\partial \mathbf{v}} = \frac{\partial \mathcal{C}}{\partial \mathsf{R}(\mathbf{v})} \cdot \frac{\partial \mathsf{R}(\mathbf{v})}{\partial \mathbf{v}} \tag{1}$$

where

$$\frac{\partial \mathsf{R}(\mathbf{v})}{\partial v_i} = \frac{v_i[\mathbf{v}]_\times + [\mathbf{v} \times (\mathsf{I} - \mathsf{R})e_i]_\times}{||\mathbf{v}||^2}\mathsf{R} \tag{2}$$

$[\ \ ]_\times$ turns a 3×1 vector to a skew-symmetric matrix and $\times$ is a cross product operation. I is the Identity matrix and e_i is the i^{th} column of the Identity matrix. We have also implemented different parameterisations *e.g.* quaternions and Euler-angles for rotations as additional layers. Below we show the code-snippet that performs backpropagation on this layer.

```
function RotationSO3:updateGradInput(_tranformParams, _gradParams)

  -- _transformParams are the input parameters i.e. so3 vector
  -- _gradParams is the derivative of the cost function
  -- with respect to the rotation matrix

  -- gradInput is the derivative of cost
  -- function with respect to so3 vector

  local tParams, gradParams
  tParams = _tranformParams
  gradParams = _gradParams:clone()

  local batchSize = tParams:size(1)
  self.gradInput:resizeAs(tParams)

  local rotDerv = torch.zeros(batchSize, 3, 3):typeAs(tParams)
  local gradInputRotationParams = self.gradInput:narrow(2,1,1)

  -- take the derivative with respect to v1
  rotDerv = dR_by_dvi(tParams,self.rotationOutput,1, self.threshold)
  local selectGradParams = gradParams:narrow(2,1,3):narrow(3,1,3)
  gradRotParams:copy(torch.cmul(rotDerv,selectGradParams):sum(2):sum(3))

  -- take the derivative with respect to v2
  rotDerv = dR_by_dvi(tParams,self.rotationOutput,2, self.threshold)
  gradRotParams = self.gradInput:narrow(2,2,1)
  gradRotParams:copy(torch.cmul(rotDerv,selectGradParams):sum(2):sum(3))

  -- take the derivative with respect to v3
  rotDerv = dR_by_dvi(tParams,self.rotationOutput,3, self.threshold)
  gradRotParams = self.gradInput:narrow(2,3,1)
  gradRotParams:copy(torch.cmul(rotDerv,selectGradParams):sum(2):sum(3))

  return self.gradInput

end
```

SE3 Layer. The SE3 layer adds translations on top of the SO3 layer where translations are represented by a 3×1 vector $\mathbf{t}$, and together they make up the 3×4 transformation, *i.e.* $\mathsf{T} = [\mathsf{R}|\mathbf{t}] \in$ SE3.

Sim3 Layer. Sim3 layer builds on top of the SE3 layer and has an extra scale factor s to allow for any scale changes associated with the transformations $\mathsf{T} = \begin{bmatrix} s\mathsf{R} & \mathbf{t} \\ \mathsf{0} & \mathsf{1} \end{bmatrix}$.

3D Grid Generator. The 3D grid generator is an extension of the 2D grid generator proposed in the original STN. It takes additionally a depth map as input, to map the image pixels to corresponding 3D points in the world and transforms these points with T coming from the SE3 layer. Note that we have used a regular grid in this layer, but it is possible to extend this to the general case where the grid locations can also be learnt.

Projection Layer. Projection layer maps the transformed 3D points, $\mathbf{p} = (u, v, w)^T$, onto 2D image plane using the focal lengths and the camera centre location. *i.e.*

$$\pi \begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} f_x \frac{u}{w} + p_x \\ f_y \frac{v}{w} + p_y \end{pmatrix} \tag{3}$$

where f_x and f_y represent the focal lengths of the camera along X and Y axes and p_x and p_y are the camera center locations. The backpropagation derivatives can be written as

$$\frac{\partial C}{\partial \mathbf{p}} = \frac{\partial C}{\partial \pi(\mathbf{p})} \cdot \frac{\partial \pi(\mathbf{p})}{\partial \mathbf{p}} \tag{4}$$

where

$$\frac{\partial \pi \begin{pmatrix} u \\ v \\ w \end{pmatrix}}{\partial \begin{pmatrix} u \\ v \\ w \end{pmatrix}} = \begin{pmatrix} f_x \frac{1}{w} & 0 & -f_x \frac{u}{w^2} \\ 0 & f_y \frac{1}{w} & -f_y \frac{v}{w^2} \end{pmatrix} \tag{5}$$

In fact, if focal lengths are also involved in the optimisation, it is straightforward to include them in the network for any geometric camera calibration style optimisations. Note that special care must be taken to ensure that w is not very small. Fortunately, in many geometric vision problems w corresponds to the z-coordinate of a 3D point and is measured in metres — when using Kinect or ASUS xtion cameras this happens to be always greater than 10 cm[1].

[1] We discovered that anything below than that the forward/backward gradient check fails.

2.2 Per-pixel Transformations

In many computer vision problems, particularly related to understanding dynamic scenes, it is often required to have per-pixel transformations to model the movements of the stimuli in the scene. In the following, we propose different layers for modelling per-pixel transformations for both RGB and RGB-D inputs.

RGB Based. In the context of RGB data, the classic optic flow problem is a case of per-pixel transformation to model the movement of pixels across time. We implement both the well-known minimal parameterisation in the form of translation as well as more recently studied over-parameterised formulations that encapsulate the knowledge of scene geometry into the flow movement.

Mimimal Parameterisation Optic Flow. In its minimal parameterisation, optic flow (t_x, t_y) models the movement of pixels in the 2D image plane *i.e.*

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x + t_x \\ y + t_y \end{pmatrix} \tag{6}$$

This is the most well-known and studied parameterisation of optic flow in the literature and needs only 2 parameters per-pixel. In general, an extra smoothness penalty is imposed to ensure that the gradient of the flow varies smoothly across a pixel neighbourhood. Patraucean *et al.* [2] implement exactly this to model the optic flow and use Huber penalty for smoothness. We include this as a part of our library together with recent extensions with over-parameterised formulations.

Over-Parameterised Optic Flow. Attempts to use the popular differential epipolar constraint [6] and the recent over-parameterised formulations of [7] and [8] have shown that if knowledge about the scene geometry and motion can be used, it can greatly improve the flow estimates per-pixel. For instance, if the pixel lies on a planar surface, the motion of the pixel can be modelled by an affine transformation. Although [8] use a 9-DoF per-pixel transformation that includes the knowledge about the homography, we describe the affine parameterisation used in [7].

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} a_0 & a_1 & a_2 \\ a_3 & a_4 & a_5 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{7}$$

It is interesting to note that popular 2-DoF translation optic flow describe earlier happens to be a special case of affine transformation.

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{8}$$

We provide implementations of 6-DoF affine transformation as well as SE(2) transformation per-pixel but extensions to 9-DoF paramterisation [8] are straightforward.

```
function AffineOpticFlow:updateGradInput(_PerPixelAffineParams, _gradGrid)

   local batchsize = _PerPixelAffineParams:size(1)

   self.gradInput:resizeAs(_PerPixelAffineParams):zero()

   -- batchGrid is the regular 2D grid: B H W 2
   -- batches: B, height: H, width: W, channels: 2

   local Lx_x = torch.cmul(_gradGrid:select(4,1), self.batchGrid:select(4,1))
   local Lx_y = torch.cmul(_gradGrid:select(4,1), self.batchGrid:select(4,2))

   local Ly_x = torch.cmul(_gradGrid:select(4,2), self.batchGrid:select(4,1))
   local Ly_y = torch.cmul(_gradGrid:select(4,2), self.batchGrid:select(4,2))

   self.gradInput:select(4,1):copy(Lx_x)
   self.gradInput:select(4,2):copy(Lx_y)
   self.gradInput:select(4,3):copy(_gradGrid:select(4,1))

   self.gradInput:select(4,4):copy(Ly_x)
   self.gradInput:select(4,5):copy(Ly_y)
   self.gradInput:select(4,6):copy(_gradGrid:select(4,2))

   return self.gradInput

end
```

Slanted Plane Depth Disparity. Similar ideas have been used in [9] to obtain disparity of a stereo pair. They exploit the fact that scenes can be decomposed into piecewise slanted planes and consequently the disparity of a pixel can be expressed by the plane equation. This results in a over-paramterised 3-DoF formulation of disparity.

$$d = ax + by + c \tag{9}$$

Again, this over-parameterisation greatly improves the results. Note that this formulation can be easily generalised and lifted to higher dimensions in the spirit of Total Generalised Variation (TGV) [10], but we have only implemented the 3-DoF formulation.

We would like to stress that these layers are particularly tailored towards warping images which could be used as a direct signal for feedback loop in image reconstruction error in unsupervised training [2,11].

RGB-D Based. Our layers can be easily adapted to RGB-D to enable 3D point cloud registration and alignment via per-pixel rigid transformations. Such transformations have been used extensively in the computer graphics community for some time and exploited by [12–14] for non-rigid alignment. We extend similar ideas and implement 3D transformations for each pixel containing a 3D vector $\mathbf{x}$, the 3D spatial coordinates coming from a depth-map. In principle, such alignment is general and not limited to just 3D spatial points *i.e.* any 3D feature per-pixel can be transformed. This is particularly useful when aligning feature maps as used in sketch and style transfer using deep learning [15].

Per-pixel Sim3 Transformation. We extend the global Sim3 transformation that models scale s, Rotation R, and translation $\mathbf{t}$ to a per-pixel Sim3 transformation *i.e.* $\mathsf{T}_i = \begin{bmatrix} s_i\mathsf{R}_i & \mathbf{t}_i \\ \mathsf{0} & 1 \end{bmatrix}$ where $\mathsf{R} \in \mathrm{SO3}$.

$$\begin{pmatrix} x_i' \\ y_i' \\ z_i' \end{pmatrix} = \mathsf{T}_i \begin{pmatrix} x_i \\ y_i \\ z_i \\ 1 \end{pmatrix} \tag{10}$$

This allows for the attention like mechanism of [1] in 3D, as specific voxel areas can be cropped and zoomed, and also modelling any 3D registrations that require scale.

Per-pixel 10 DoF Transformation. In many non-rigid alignments the rotation need not happen around the origin but around an anchor point $\mathbf{p}_i$ which is also jointly estimated. In this case, the transformation extends to 10 degrees of freedom [12].

$$\mathbf{x}_i' = s_i(\mathsf{R}_i(\mathbf{x}_i - \mathbf{p}_i) + \mathbf{p}_i) + \mathbf{t}_i \tag{11}$$

Additionally, smoothness constraints can be added to ensure that transformations are locally smooth in just the same way as Huber penalty is imposed for smoothing 2D optic flow.

2.3 M-Estimators

The standard least-squares loss function often employed in parameter fitting greatly affects the quality of the solution obtained at convergence. Built on the assumption that noise in the data follows Gaussian distribution, the least-squares function treats both the inliers and outliers in the data uniformly. This is undesirable because even one bad sample in the data can sway the optimisation to an unexpected convergence point. Therefore, outlier samples should be culled or down-weighted accordingly to maintain the optimisation and estimation process from getting influenced by them. Fortunately, in computer vision this has been long studied since the early 90s by Black *et al.* [18–20] who pioneered the use of robust cost functions, often termed M-estimators for estimating a statistically robust mean of the data. We adapted the standard $\mathcal{L}_2^2$ loss function with various popular M-estimators. The table below shows various M-estimators, $\rho(\mathrm{x})$ and their corresponding derivatives, $\psi(\mathrm{x})$.

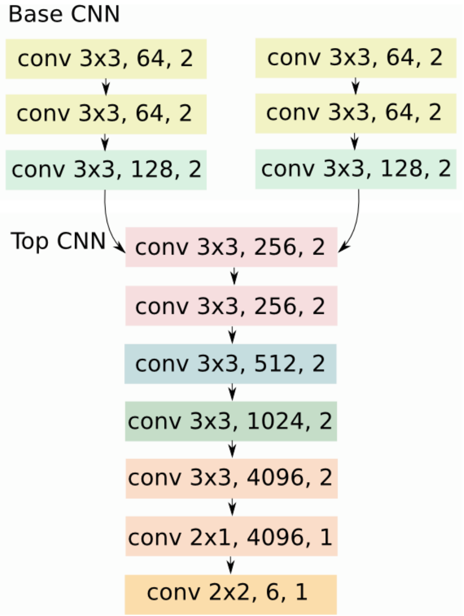


Fig. 1. Our Siamese network is inspired by the popular VGG-16 network [16] where 3×3 convolutions are used in most layers and works for 320×240 image resolution. Each convolution layer is followed by PReLU non-linearity [17]. We explicitly avoid any pooling and use a stride of 2 in every convolution layer for any downsampling.

M-estimator	ρ(x)	ψ(x)
Huber $\begin{cases}\text{if } \lvert x\rvert \le \epsilon, \\ \text{otherwise.}\end{cases}$	$\begin{cases}\frac{x^2}{2}, \\ \epsilon(\lvert x\rvert - \frac{\epsilon}{2})\end{cases}$	$\begin{cases}x, \\ \epsilon\frac{x}{\lvert x\rvert}\end{cases}$
Cauchy	$\frac{c^2}{2}\log(1+(\frac{x}{c})^2)$	$\frac{x}{1+(\frac{x}{c})^2}$
Geman-McClure	$\frac{x^2/2}{1+x^2}$	$\frac{x}{(1+x^2)^2}$
Tukey $\begin{cases}\text{if}\lvert x\rvert \le c \\ \text{otherwise.}\end{cases}$	$\begin{cases}\frac{c^2}{6}(1-(1-(\frac{x}{c})^2)^3) \\ \frac{c^2}{6}\end{cases}$	$\begin{cases}x(1-(\frac{x}{c})^2)^2, \\ 0\end{cases}$

The use of M-estimators has already started to trickle down in the deep learning community *e.g.* Patraucean *et al.* [2] use a Huber loss function in the smoothness term to regularise the optic flow. We believe our library will also continue to encourage people to use different loss functions that are more pertinent to the tasks where Gaussian noise assumptions fall apart.

3 Application: Training on RGB-D Visual Odometry

We perform early experiments on visual odometry for both SO3 as well as SE3 motion that involves depth based warping. We believe this is the first attempt towards end-to-end system for Visual Odometry with deep learning. Since we are aligning images *à la* dense image registration methods, this allows us to do sanity checks on different layers *e.g.* SE3 layer, 3D Grid Generator, and Projection layer all within the same network and optimisation scheme. Note that we could have also chosen to do minimisation on re-projection error of sparse keypoints as in classic Bundle Adjustment. However, this approach does not lend itself to generic iterative image alignment where each iteration provides a warped version of the reference image and can be fed back into the network for an end-to-end RNN based visual odometry system. Moreover, our approach is also naturally suited for unsupervised learning in the spirit of [2,11].

3.1 Network Architecture

Our architecture is composed of a siamese network that takes in a pair of consecutive frames, $\mathcal{I}_{ref}$ and $\mathcal{I}_{live}$, captured at time instances t and $t+1$ respectively, and returns a 6-DoF pose vector, δ_{pred} — where the first three elements correspond to rotation and the last three to translation — that transforms one image to the other. In case of pure rotation, the network predicts a 3×1 vector. It is assumed that the scene is mostly static and rigid, and the motion perceived in the image is induced only via the camera movement. However, instead of naïvely comparing the predicted 6-DoF vector, δ_{pred}, with the corresponding ground truth vector, δ_{gt}, we build upon the work of Patraucean *et al.* [2], to warp the images directly using our customised *3D Spatial Transformer* module, to compute the image alignment error as our cost function. This allows us to compare the transformations in the right space: naïve comparison of 6-DoF vectors would have involved a tunable parameter beforehand to weigh the translation and rotation errors appropriately to define the cost function since they are two different entities. Searching for the right weighting can quickly become tedious and may not generalise well. Although [21] are able to minimise a cost function by appropriately weighing the rotation and translation errors within optimal hand-eye coordination loop, this is not possible all the time. Discretising the poses as done in [22] may hamper the accuracy of pose estimation. On the other hand, computing pixel error via warping, as often done in classic dense image alignment methods [23,24], allows to compare the transformations in the space of pixel intensities without having to tune any external parameters. Moreover, dense alignment methods have an added advantage of accurately recovering the transformations by minimising sum of squared differences of pixel values at corresponding locations in the two images *i.e.*

$$\mathcal{C} = \frac{1}{2}\sum_{i=1}^{N}\Big(\mathcal{I}_{ref}(\mathbf{x}) - \mathcal{I}_{live}(\pi(\mathsf{T}_{lr}\hat{\mathsf{p}}(\mathbf{x})))\Big)^2$$

where $\mathbf{x}$ is a homogenised 2D pixel location in the reference image, $\hat{\mathsf{p}}(\mathbf{x})$ is the 4×1 corresponding homogenised 3D point obtained by projecting the ray from that given pixel location (x, y) into the 3D world via classic inverse camera projection and the depth, $\mathsf{d}(x, y)$, at that pixel location.

$$\mathbf{x} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}, \quad \hat{\mathsf{p}}(x, y) = \begin{pmatrix} \mathsf{K}^{-1}\mathbf{x} \cdot \mathsf{d}(x, y) \\ 1 \end{pmatrix} \tag{12}$$

$$\mathsf{K} = \begin{bmatrix} f_x & 0 & p_x \\ 0 & f_y & p_y \\ 0 & 0 & 1 \end{bmatrix}, \quad \pi \begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} f_x \frac{u}{w} + p_x \\ f_y \frac{v}{w} + p_y \end{pmatrix} \tag{13}$$

K is the camera calibration matrix, f_x and f_y denote the focal lengths of the camera (in pixels) while p_x, p_y are the coordinates of the camera center location. π is the projection function that maps a 3D point to a 2D plane and T_{lr} (or T_{pred}) is a 3×4 matrix that transforms a 3D point in the reference frame to the live frame. In this work, we bridge the gap between learning and geometry based methods with our *3D Spatial Transformer* module which explicitly defines these operations as layers that act as computational blocks with no learning parameters but allow backpropagation from the cost function to the input layers.

Figure 2 shows an example of our customised STN for 3D transformation. The siamese network predicts a 6×1 vector that is turned into a 3×4 transformation matrix T_{pred} via SE3 layer. This matrix transforms the points generated by the 3D grid generator that additionally takes depth image as input and turns it into 3D points via inverse camera projection with K^{-1} as in Eq. 1. These transformed points are then projected back into the 2D image plane via the Projection layer (*i.e.* the π function) and further used to bilinearly interpolate the warped image as in the original STN [1].

Our siamese network is inspired from the popular VGG-16 network [16] and uses 3×3 convolutions in all but the last two layers where 2×1 and 2×2 convolutions are used to compensate for the 320×240 resolution used as input as opposed to the 224×224 used in original VGG-16. Figure 1 shows our siamese network where two heads are fused early to ensure that the relevant spatial information is not lost by the depth of the network. We also avoid any pooling operations throughout the network, again to ensure that the spatial information is preserved. All convolutional layers, with the exception of the last three, are followed by a non-linearity. We found PReLUs [17] to work better both in terms of convergence speed and accuracy than ReLUs for our network and therefore used them for all the experiments. We also experimented with recently introduced ELUs [25] but did not find any significant difference in the end to PReLUs. Weights of all convolution layers are initialised with MSRA initialisation proposed in [17]. However, the last layer has the weights all initialised to zero. This is to ensure that the relative pose between the consecutive frames is initialised with Identity transformation, as commonly used in many dense image alignment methods.

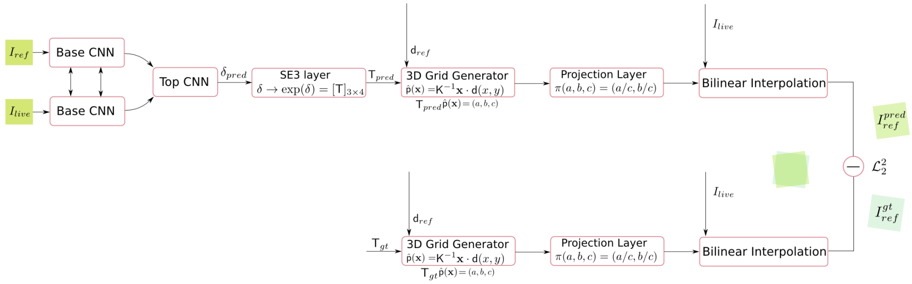

I_{ref}
Base CNN
I_{live}
Base CNN
Top CNN
δ_{pred}
SE3 layer
$\delta \rightarrow \exp(\delta) = [\mathsf{T}]_{3\times4}$
T_{pred}
d_{ref}
3D Grid Generator
$\tilde{\mathsf{p}}(\mathbf{x}) = \mathsf{K}^{-1}\mathbf{x} \cdot \mathsf{d}(x, y)$
$\mathsf{T}_{pred}\tilde{\mathsf{p}}(\mathbf{x}) = (a, b, c)$
Projection Layer
$\pi(a, b, c) = (a/c, b/c)$
I_{live}
Bilinear Interpolation
I_{ref}^{pred}
$\mathcal{L}_2^2$
I_{ref}^{gt}
d_{ref}
T_{gt}
3D Grid Generator
$\tilde{\mathsf{p}}(\mathbf{x}) = \mathsf{K}^{-1}\mathbf{x} \cdot \mathsf{d}(x, y)$
$\mathsf{T}_{gt}\tilde{\mathsf{p}}(\mathbf{x}) = (a, b, c)$
Projection Layer
$\pi(a, b, c) = (a/c, b/c)$
I_{live}
Bilinear Interpolation

Fig. 2. We train a siamese network to regress to the relative pose vector between the two consecutive frames, I_{ref} and I_{live}. This pose vector is turned into a 3×4 transformation matrix that transforms 3D points coming from the 3D grid generator and further projected into a 2D plane via projection layer which are used to generate a warped image. Additionally, the 3D grid generator needs an explicit depth-map as input to generate 3D points for any 3D warping.

While one could use the pixel difference between the predicted live image, using the transformation returned by the siamese network, and the live image as the cost function, we chose instead to take the pixel difference between the predicted live image with the predicted transformation and the predicted live image with the ground truth transformation. This is because if there is significant motion between the input frames, warping may possibly lead to missing pixels in the predicted image which will get unnecessary penalised if compared against the live image directly since there is no explicit way to block out the corresponding pixels in the live image. However, if the predicted images from the predicted and ground truth transformations are compared, at optimal predicted transformations both should have the same missing pixels which would allow implicitly blocking out those pixels. Moreover, any external artefact in the images in the form of motion blur, intensity changes, or image noise would affect the registration since the cost function is a pixel-wise comparison. On the other hand, our way of comparing the pixels ensures that at convergence, the cost function is as close to zero as possible and is able to handle missing pixels appropriately. Ultimately, we only need a way to compare the predicted and ground truth transformations in the pixel space. We show early results of training on SO3 (pure rotation) and SE3 motion (involving rotation and translation).

SO3 Motion: Pure Rotation. To experiment with pure rotation motion, we gathered IMU readings of a camera undergoing rapid hand-held motion: we used [26] to capture an outdoor dataset but dropped the translation readings. This is only to ensure that the transformation in the images correspond to the real hand-held motion observed in real world. We use the rotation matrices to synthetically generate new images in the dataset and feed the corresponding pair through the network. We perform early experiments that serve as sanity checks for different layers working together in a network. Figure 3 shows how our system is able to register the images over a given training episode. The first row shows a high residual in the image registration but as the network improves with the training, the residual gradually starts to decrease: last row shows that the network is capable of registering images involving very large motion. Note that the prediction images at the start of training have no missing pixels (since the network is initialised with Identity transformation) but gradually start moving towards the ground truth image.

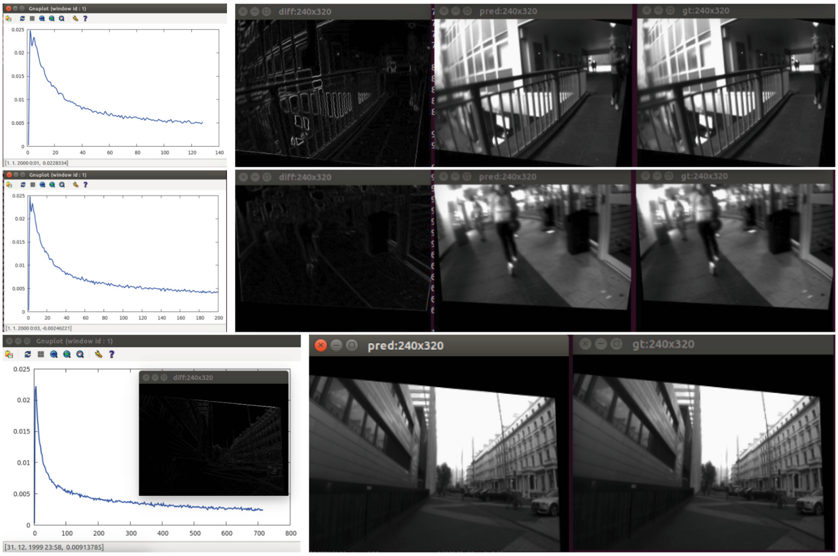
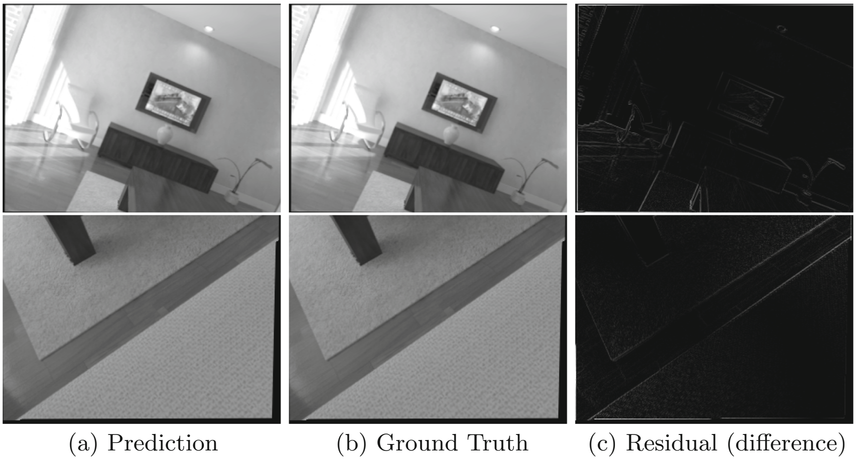


Fig. 3. Training results on pure rotation motion. The graphs show how the training error decreases as number of epochs increase. This serves as a sanity check for our network that includes many new layers that we propose in this library. The improvement in the training is qualitatively evident from the difference images: early stages in the optimisation show high residual in the registration but as more epochs are thrown to the optimisation, the residual error gracefully decreases.

SE3 Motion: Rotation and Translation. SE3 motion needs depth to enable registration of two images involving both rotation and translation. This is possible with our SE3 layer that additionally takes in depth-map as input and produces the interpolation coordinates to be further used by the bilinear interpolation layer. We use ICL-NUIM [27] and generate a long trajectory of 9.5K frames and use this as our training set. Figure 4 shows samples of generated frames in this new trajectory. Since we need per-pixel depth for this experiment we opted for synthetic dataset only for convenience. In future, we would like to test our approach on real world data.

Fig. 4. Sample frames from our new ICL-NUIM trajectory.

Similar to the pure rotation (SO3) motion, we show early results on SE3 motion involving rotation and translation. Figure 5 shows the network's ability to learn to align the predicted image with the ground truth image using depth that is given as an additional input to the 3D grid generator.



(a) Prediction (b) Ground Truth (c) Residual (difference)

Fig. 5. Sample results on the new trajectory generated with ICL-NUIM dataset. The SE3 layer allows warping image with 3D motion and this is evident in the registration error in the residual image. Note that the relative motion between consecutive frames is generally slow in the whole trajectory.

4 Future Work

We have only shown training on visual odometry as sanity checks of our layers and their ability to blend in with the standard convolution neural network.

In future, we would like to train both feed-forward as well as feedback connections based neural network on large training data. This data could either come from standard Structure from Motion [28], large scale synthetic datasets e.g. SceneNet [29] or large scale RGB or RGB-D videos for unsupervised learning.

5 Conclusions

We introduced a new library, **gvnn**, that allows implementation of various standard computer vision applications within a deep learning framework. In its current form, it allows end-to-end training for optic flow, disparity or depth estimation, visual odometry, small-scale bundle adjustment, super-resolution, place recognition with geometric invariance all with both supervised and unsupervised settings. In future, we plan to extend this library to include various different lens distortion models, camera projection models, IMU based transformation layers, sign distance functions, level-sets, and classic primal-dual methods [30] as RNN blocks to allow embedding higher order priors in the form of TGV [10]. We hope that our library will encourage researchers to use and contribute towards making this a comprehensive and complete resource for geometric computer vision with deep learning in the same way the popular **rnn** package [31] has fostered research in recurrent neural networks in the community. Upon publication, we will release the full source code and sample application examples at https://github.com/ankurhanda/gvnn.

Acknowledgements. AH and AD would like to thank Dyson Technology Ltd. for kindly funding this research work.

References

1. Jaderberg, M., Simonyan, K., Zisserman, A., Kavukcuoglu, K.: Spatial transformer networks. In: NIPS (2015)
2. Patraucean, V., Handa, A., Cipolla, R.: Spatio-temporal video autoencoder with differentiable memory. CoRR abs/1511.06309 (2015)
3. Collobert, R., Kavukcuoglu, K., Farabet, C.: Torch7: a matlab-like environment for machine learning. In: BigLearn, NIPS Workshop. Number EPFL-CONF-192376 (2011)
4. Moodstocks: Open Source Implementation of Spatial Transformer Networks (2015). https://github.com/qassemoquab/stnbhwd
5. Gallego, G., Yezzi, A.J.: A compact formula for the derivative of a 3-D rotation in exponential coordinates (2013)
6. Brooks, M.J., Chojnacki, W., Baumela, L.: Determining the egomotion of an uncalibrated camera from instantaneous optical flow. JOSA A (1997)
7. Nir, T., Bruckstein, A.M., Kimmel, R.: Over-parameterized variational optical flow. Int. J. Comput. Vis. (IJCV) **76**(2), 205–216 (2008)
8. Hornáček, M., Besse, F., Kautz, J., Fitzgibbon, A., Rother, C.: Highly overparameterized optical flow using patchmatch belief propagation. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (eds.) ECCV 2014. LNCS, vol. 8691, pp. 220–234. Springer, Heidelberg (2014). doi:10.1007/978-3-319-10578-9_15

9. Bleyer, M., Rhemann, C., Rother, C.: PatchMatch stereo – stereo matching with slanted support windows. In: Proceedings of the British Machine Vision Conference (BMVC) (2011)
10. Pock, T., Zebedin, L., Bischof, H.: TGV-fusion. In: Calude, C.S., Rozenberg, G., Salomaa, A. (eds.) Rainbow of Computer Science. LNCS, vol. 6570, pp. 245–258. Springer, Heidelberg (2011). doi:10.1007/978-3-642-19391-0_18
11. Garg, R., BG, V.K., Reid, I.D.: Unsupervised CNN for single view depth estimation: geometry to the rescue. CoRR abs/1603.04992 (2016)
12. Sumner, R.W., Schmid, J., Pauly, M.: Embedded deformation for shape manipulation. In: Proceedings of SIGGRAPH (2007)
13. Zollhöfer, M., Nießner, M., Izadi, S., Rehmann, C., Zach, C., Fisher, M., Wu, C., Fitzgibbon, A., Loop, C., Theobalt, C., et al.: Real-time non-rigid reconstruction using an RGB-D camera. ACM Trans. Graph. (TOG) (2014)
14. Newcombe, R.A., Fox, D., Seitz, S.M.: Dynamicfusion: reconstruction and tracking of non-rigid scenes in real-time. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2015)
15. Johnson, J., Alahi, A., Li, F.: Perceptual losses for real-time style transfer and super-resolution. CoRR abs/1603.08155 (2016)
16. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: Proceedings of the International Conference on Learning Representations (ICLR) (2015)
17. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: surpassing human-level performance on imagenet classification. In: Proceedings of the International Conference on Computer Vision (ICCV) (2015)
18. Black, M.J., Anandan, P.: A framework for the robust estimation of optical flow. In: Proceedings of the International Conference on Computer Vision (ICCV) (1993)
19. Black, M., Anandan, P.: Robust dynamic motion estimation over time. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (1991)
20. Black, M.J., Sapiro, G., Marimont, D.H., Heeger, D.: Robust anisotropic diffusion. IEEE Trans. Image Process. **7**, 421–432 (1998)
21. Strobl, K.H., Hirzinger, G.: Optimal hand-eye calibration. In: 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE (2006)
22. Agrawal, P., Carreira, J., Malik, J.: Learning to see by moving. In: Proceedings of the IEEE International Conference on Computer Vision (2015)
23. Lucas, B.D., Kanade, T.: An iterative image registration technique with an application to stereo vision. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI) (1981)
24. Drummond, T., Cipolla, R.: Visual tracking and control using lie algebras. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (1999)
25. Clevert, D.A., Unterthiner, T., Hochreiter, S.: Fast and accurate deep network learning by exponential linear units (elus). In: ICLR (2016)
26. Leutenegger, S., Lynen, S., Bosse, M., Siegwart, R., Furgale, P.: Keyframe-based visual-inertial odometry using nonlinear optimization. Int. J. Robot. Res. (2014)
27. Handa, A., Whelan, T., McDonald, J.B., Davison, A.J.: A benchmark for RGB-D visual odometry, 3D reconstruction and SLAM. In: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA) (2014)
28. Wu, C.: VisualSfM : A visual structure from motion system. http://ccwu.me/vsfm/

29. Handa, A., Pătrăucean, V., Badrinarayanan, V., Stent, S., Cipolla, R.: SceneNet: understanding real world indoor scenes with synthetic data. arXiv preprint (2015). arXiv:1511.07041
30. Chambolle, A., Pock, T.: A first-order primal-dual algorithm for convex problems with applications to imaging. J. Math. Imaging Vis. **40**(1), 120–145 (2011)
31. Léonard, N., Waghmare, S., Wang, Y., Kim, J.: RNN: recurrent library for torch. CoRR abs/1511.07889 (2015)