DEEPV2D: VIDEO TO DEPTH WITH DIFFERENTIABLE STRUCTURE FROM MOTION

Anonymous authors
Paper under double-blind review

ABSTRACT

We propose DeepV2D, an end-to-end deep learning architecture for predicting depth from video. DeepV2D combines the representation ability of neural networks with the geometric principles governing image formation. We compose a collection of classical geometric algorithms, which are converted into trainable modules and combined into an end-to-end differentiable architecture. DeepV2D interleaves two stages: motion estimation and depth estimation. During inference, motion and depth estimation are alternated and converge to accurate depth.

1 INTRODUCTION

In video to depth, the task is to estimate depth from a video sequence. The problem has traditionally been approached using Structure from Motion (SfM), which takes a collection of images as input, and jointly optimizes over 3D structure and camera motion (Schonberger & Frahm, 2016b). The resulting camera parameter estimates can be used as input to Multi-View Stereo in order to build a more complete 3D representation such as surface meshes and depth maps (Furukawa et al., 2015; Furukawa & Ponce, 2010).

In parallel, deep learning has been highly successful in a number of 3D reconstruction tasks. In particular, given ground truth depth, a network can learn to predict depth from a single image (Eigen et al., 2014; Eigen & Fergus, 2015; Laina et al., 2016), stereo images (Kendall et al., 2017; Mayer et al., 2016), or collections of frames (Zhou et al., 2018; Kar et al., 2017; Tang & Tan, 2018; Yao et al., 2018). One advantage of deep networks is that they can use single-image cues such as texture gradients and shading as shown by their strong performance on depth estimation from a single image (Eigen et al., 2014; Eigen & Fergus, 2015; Laina et al., 2016). Furthermore, differentiable network modules can be composed so that entire pipelines (i.e. feature extraction, feature matching, regularization) can be learned directly from training data. On the other hand, as recent work has shown, it is often hard to train generic network layers to directly utilize multiview geometry (e.g. using interframe correspondence to recover depth), and it is often advantageous to embed knowledge of multiview geometry through specially designed layers or losses (Ummenhofer et al., 2017; Kendall & Cipolla, 2017; Zhou et al., 2017; Vijayanarasimhan et al., 2017; Zhou et al., 2018).

In this work, we continue the direction set forth by recent works (Ummenhofer et al., 2017; Kendall et al., 2017; Tang & Tan, 2018; Zhou et al., 2018; Kar et al., 2017; Wang et al., 2018) that combine the representation ability of neural networks with the geometric principles underlying image formation. We propose DeepV2D, a composition of classical geometrical algorithms which we turn into differentiable network modules and combine into an end-to-end trainable architecture. DeepV2D in...
terleaves two stages: camera motion estimation and depth estimation (Figure 1). The motion module takes depth as input, and outputs an incremental update to camera motion. The depth module takes camera motion as input, and performs stereo reconstruction to predict depth. At test time, DeepV2D acts as block coordinate descent, alternating between updating depth and camera motion.

To estimate camera motion we introduce Flow-SE3, a new motion estimation architecture, which outputs an incremental update to camera motion. Flow-SE3 takes depth as input, and estimates dense 2D correspondence between pairs of frames. We unroll a single iteration of Perspective-n-Point (PnP) (Lepetit et al., 2009; Li et al., 2012) performing Gauss-Newton update over SE3 perturbations to minimize geometric reprojection error. The new estimate of camera motion can then be fed back into Flow-SE3, which re-estimates correspondence for a finer grain pose update.

Our Depth Module builds upon prior work (Kendall et al., 2017; Yao et al., 2018) and formulates multiview-stereo (MVS) reconstruction as a single feed-forward network. Like classical MVS, we leverage geometry to build a cost volume over video frames, but use trainable network for both feature extraction and matching.

Our work shares similarities with prior works (Ummenhofer et al., 2017; Kendall et al., 2017; Tang & Tan, 2018; Zhou et al., 2018; Kar et al., 2017; Wang et al., 2018) that also combine deep learning and multiview geometry, but is novel and unique in that it essentially “differentializes” a classical SfM pipeline that alternates between stereopsis, dense 2D feature matching, and PnP. As a comparison, DeMon (Ummenhofer et al., 2017) and DeepTAM (Zhou et al., 2018) differentialize stereopsis and feature matching, but not PnP because they use a generic network to predict camera motion.

Another comparison is with BA-Net (Tang & Tan, 2018), whose classical analogue is performing bundle adjustment from scratch to optimize feature alignment over camera motion and the coefficients of a limited set of depth maps (depth basis). In other words, BA-Net performs one joint nonlinear optimization over all variables, whereas we decompose the joint optimization into more tractable subproblems and do block coordinate descent. Our decomposition is more expressive in terms of reconstruction since we can optimize directly over per-pixel depth and are not constrained by a depth basis, which can potentially limit the accuracy of the final depth.

In our experiments, we demonstrate the effectiveness of DeepV2D across a variety of datasets and tasks, and outperform strong methods such as DeepTAM (Zhou et al., 2018), DeMoN (Ummenhofer et al., 2017), BA-Net (Tang & Tan, 2018), and MVSNet (Yao et al., 2018). As we show, alternating depth and motion estimation quickly converges to good solutions. On all datasets we outperform all existing single-view and multi-view approaches. We also show superior cross-dataset generalizability, and can outperform existing methods even when training on entirely different datasets.

2 Related Work

Structure from Motion: Beginning with early systems designed for small image collections (Longuet-Higgins, 1981; Mohr et al., 1995), Structure from Motion (SfM) has improved dramatically in regards to robustness, accuracy, and scalability. Advances have come from improved features (Lowe, 2004; Han et al., 2015), optimization techniques (Snavely, 2009), and more scalable data structures and representations (Schonberger & Frahm, 2016a; Gherardi et al., 2010), culminating in a number of robust systems capable of large-scale reconstruction task (Schonberger & Frahm, 2016a; Snavely, 2011; Wu et al., 2011). Ranftl et al., (2016) showed that SfM could be extended to reconstruct scenes containing many dynamically moving objects. However, SfM is limited by the accuracy and availability of correspondence. In low texture regions, occlusions, or lighting changes SfM can produce noisy or missing reconstructions.

Simultaneous Localization and Mapping (SLAM) jointly estimates camera motion and 3D structure from a video sequence (Engel et al., 2014; Mur-Artal et al., 2015; Mur-Artal & Tardós, 2017; Newcombe et al., 2011; Engel et al., 2018). LSD-SLAM (Engel et al., 2014) is unique in that it relies on a featureless approach to 3D reconstruction, directly estimating depth maps and camera pose by minimizing photometric error. Our Motion Network behaves similarly to the tracking component in LSD-SLAM, but we use a network which predicts misalignment directly instead of using intensity gradients. We end up with an easier optimization problem characteristic of indirect methods (Mur-Artal et al., 2015), while retaining the flexibility of direct methods in modeling edges and smooth intensity changes (Engel et al., 2018).
**Geometry and Deep Learning:** Geometric principles has motivated the design of many deep learning architectures. In video to depth, we need to solve two subproblems: depth estimation and motion estimation.

**Depth:** End-to-end networks can be trained to predict accurate depth from a rectified pair of stereo images [Han et al., 2015; Mayer et al., 2016; Kendall et al., 2017; Chang & Chen, 2018; Kendall et al., 2017] and Chang & Chen (2018) design network architectures specifically for stereo matching. First, they apply a 2D convolutional network to extract learned features, then build a cost volume over the learned features. They then apply 3-D convolutions to the cost volume to perform feature matching and regularization. A similar idea has been extended to estimate 3D structure from multiple views [Kar et al., 2017; Yao et al., 2018]. In particular, MVSNet (Yao et al., 2018) estimates depth from multiple images. However, these works require known camera poses as input, while our method estimates depth from a video where the motion of the camera is unknown and estimated during inference.

**Motion:** Several works have used deep networks to predict camera pose. [Kendall et al., 2015] focus on the problem of camera localization, while other work [Zhou et al., 2017; Vijayanarasimhan et al., 2017; Wang et al., 2017] propose methods which estimate camera motion between a pairs of frames in a video. Networks for motion estimation have typically relied on generic network components whereas we formulate motion estimation as a least-squares optimization problem. Whereas prior work has focused on estimating relative motion between pairs of frames, we can jointly update the pose of a variable number of frames.

**Depth and Motion:** Geometric information has served as a self-supervisory signal for many recent works (Vijayanarasimhan et al., 2017; Zhou et al., 2017; Wang et al., 2018; Yin & Shi, 2018; Yang et al., 2018; Godard et al., 2017; Mahjourian et al., 2018). In particular, Zhou et al. (2017) and Vijayanarasimhan et al. (2017) trained a single-image depth network and a pose network while supervising on photometric consistency. However, while these works use geometric principles for training, they do not use multiple frames to predict depth at inference.

DeMoN (Ummenhofer et al., 2017) and DeepTAM (Zhou et al., 2018) where among the first works to combine motion estimation and multi-view reconstruction into a trainable pipeline. DeMoN (Ummenhofer et al., 2017) operates on two frames and estimates depth and motion in separate network branches, while DeepTAM (Zhou et al., 2018) can be used on variable number of frames. Like our work and other classical SLAM frameworks (Engel et al., 2014; Newcombe et al., 2011), DeepTAM separates depth and motion estimation, however we maintain end-to-end differentiability between our modules. A major innovation of DeepTAM was to formulate camera motion estimation in the form of incremental updates. In each iteration, DeepTAM renders the keyframe from a synthetic viewpoint, and predicts the residual motion from the rendered viewpoint and the target frame.

Estimating depth and camera motion can be naturally modeled as a non-linear least squares problem, which has motivated several works to include an differentiable optimization layer within network architectures (Tang & Tan, 2018; Wang et al., 2018; Clark et al., 2018; Bloesch et al., 2018). We follow this line of work, and propose the Flow-SE3 module which introduces a direct mapping from 2D correspondence to a 6-dof camera motion update. Our Flow-SE3 module is different from prior works such as DeMon (Ummenhofer et al., 2017) and DeepTAM (Zhou et al., 2018) which do not impose geometric constraints on camera motion and use generic layers. BA-Net (Tang & Tan, 2018) and LS-Net (Clark et al., 2018) include optimization layers, but instead optimize over photometric error (either pixel alignment (Clark et al., 2018) or feature alignment (Tang & Tan, 2018)). Our Flow-SE3 module still imposes geometric constraints on camera motion like BA-Net (Tang & Tan, 2018), but we show that in minimizing geometric reprojection error (difference of pixel locations), we end up with a well-behaved optimization problem, well-suited for end-to-end training.

An important difference between our approach and BA-Net is that BA-Net performs one joint optimization problem by formulating Bundle-Adjustment as a differentiable network layer, whereas we separate motion and depth estimation. With this separation, we avoid the need for a depth basis. Our final reconstructed depth is the product of a cost volume, which can adapt the reconstruction as camera motion updates improve, while the output of BA-Net is restricted by the initial quality of the depth basis produced by a single-image network.
3 APPROACH

DeepV2D predicts depth from a calibrated video sequence. We take a video as input and output dense depth. We consider two subproblems: depth estimation and motion estimation. Both subproblems are formulated as trainable neural network modules, which we refer to as the Depth Module and the Motion Module. Our depth module takes camera motion as input and outputs an updated depth prediction. Our motion module takes depth as input, and outputs a motion correction term. In the forward pass, we alternate between the depth and motion modules as we show in Figure 1.

Notation and Camera Geometry: As a preliminary, we define some of the operations used within the depth and motion modules. We define \( \pi \) to be the camera projection operator which maps a 3D point \( X = (X, Y, Z, 1)^T \) to image coordinates \( x = (u, v) \). Likewise, \( \pi^{-1} \) is defined to be the backprojection operator, which maps a pixel \( x \) and depth \( z \) to a 3D point. Using the pinhole camera model with intrinsics \((f_x, f_y, c_x, c_y)\) we have

\[
\pi(X) = (f_x \frac{X}{Z} + c_x, f_y \frac{Y}{Z} + c_y), \quad \pi^{-1}(x, z) = \left(z \frac{u - c_x}{f_x}, z \frac{v - c_y}{f_y}, z, 1\right)^T
\] (1)

The camera pose is represented using rigid body transform \( G \in SE(3) \). To find the image coordinates of point \( X \) in camera \( i \), we chain the projection and transformation: \((u, v)^T = \pi(G_i X)\), where \( G_i \) is the pose of camera \( i \).

Now, given two cameras \( G_i \) and \( G_j \). If we know the depth of a point \( x^i = (u^i, v^i) \) in camera \( i \), we can find its reprojected coordinates in camera \( j \):

\[
\left(\frac{u^j}{v^j}\right) = \pi(G_j G_i^{-1} \pi^{-1}(x, z)) = \pi(G_{ij} \pi^{-1}(x, z))
\] (2)

using the notation \( G_{ij} = G_j G_i^{-1} \) for the relative pose between cameras \( i \) and \( j \).

3.1 DEPTH MODULE

The depth module takes a collection of frames, \( I = \{I_1, I_2, ..., I_N\} \), along with their respective pose estimates, \( G = \{G_1, G_2, ..., G_N\} \), and predicts a dense depth map \( D^* \) for the keyframe (Figure 2). The depth module works by building a cost volume over learned features. Information is aggregated over multiple viewpoints by applying a global pooling layer which pools across viewpoints.

The depth module can be viewed as the composition of 3 building blocks: 2D feature extractor, cost volume backprojection, and 3D stereo matching.

2D Feature Extraction: The Depth Module begins by extracting learned features from the input images. The 2D encoder consists of 2 stacked hourglass modules [Newell et al., 2016] which maps
each image to a dense feature map \( I_i \rightarrow F_i \). More information regarding network architectures is provided in the appendix.

**Cost Volume Backprojection:** Take \( I_1 \) to be the keyframe, a cost volume is constructed for each of the remaining N-1 frames. The cost volume for frame \( j \), \( C_j \), is constructed by backprojecting 2D features into the coordinate system defined by the keyframe image. To build the cost volume, we enumerate over a range of depths \( z_1, z_2, ..., z_D \) which is chosen to span the ranges observed in the dataset (0.2m - 10m for indoor scenes). For every depth \( z_k \), we use Equation (2) to find the reprojected coordinates on frame \( j \), and then use differentiable bilinear sampling of the feature map \( F_j \).

More formally, given a pixel \( x = (u, v) \in \mathbb{N}^2 \) in frame \( I_1 \) and depth \( z_k \):

\[
C_{uvw}^j = F_j (\pi(G_j G_1^{-1} x^{-1}(x, z_k))) \in \mathbb{R}^{H \times W \times D \times C}
\]

where \( F(\cdot) \) is the differentiable bilinear sampling operator \cite{Jaderberg2015}. Since the bilinear sampling is differentiable, \( C_j \) is differentiable w.r.t all inputs, including the camera pose.

Applying this operation to each frame, gives us a set of N-1 cost volumes each with dimension \( H \times W \times D \times C \). As a final step, we concatenate each cost volume with the keyframe image features increasing the dimension to \( H \times W \times D \times 2C \). By concatenating features, we give the network the necessary information to perform feature matching between the keyframe/image pairs without decimating the feature dimension.

**3D Matching Network:** The set of N-1 cost volumes are first processed by a series of 3D convolutional layers to perform stereo matching. We then perform view pooling by averaging over the N volumes to aggregate information across frames. View pooling leaves us with a single volume of dimension \( H \times W \times D \times C \). The aggregated volume is then processed by a series of 3D hourglass modules, each outputs an intermediate depth.

Each 3D hourglass module predicts an intermediate depth estimate. We produce an intermediate depth representation by first applying a 1x1x1 convolution to a produce \( H \times W \times D \times V \). We then apply the softmax operator over the depth dimension, so that for each pixel, we get a probability distribution over depths. We map the probability volume into a single depth estimate using the differentiable argmax function \cite{Kendall2017} which computes the expected depth.

### 3.2 Motion Module

The objective of the motion module is to update the camera motion estimates given depth as input. Given the input poses, \( G = \{G_1, G_2, ..., G_N\} \), the motion module outputs a set of local perturbations \( \xi = \{\xi_1, \xi_2, ..., \xi_N\} \), \( \xi_i \in \mathbb{R}^3 \) used to update the poses. For each pose, \( G_i \), the updated pose is found by applying the local perturbation using the exponential map: \( G'_i = e^{t \xi_i} G_i \). The updates are found by setting up a least squares optimization problem which is solved using a differentiable in-network optimization layer.

**Initialization:** We use a generic network architecture to predict the initial pose estimates similar to prior work \cite{Zhou2017}. We choose one frame to be the keyframe. The poses are initialized by setting the keyframe pose to be the identity matrix, and then predicting the relative motion between the keyframe and each of the other frames in the video.

**Feature Extraction:** Our motion module operates over learned features. The feature extractor maps every frame to a dense feature map, \( I_i \rightarrow F_i \). The weights of the feature extractor are shared across all frames. Network architecture details are provided in the appendix.

**Error Term:** Take two frames, \( (I_i, I_j) \), with respective poses \( (G_i, G_j) \) and feature maps \( (F_i, F_j) \). Given depth \( Z_i \) we can use Equation (2) we can warp \( F_j \) onto camera \( i \) to generate the warped feature map \( \tilde{F}_j \). If the relative pose \( G_{ij} = G_j G_i^{-1} \) is correct, then the feature maps \( F_i \) and \( \tilde{F}_j \) should align. However, if the relative pose is noisy, then there will be misalignment between the feature images which should be corrected by the pose update.

We concatenate \( F_i \) and \( \tilde{F}_j \), and send the concatenated feature map through an hourglass network to predict the dense residual flow between the feature maps, which we denote \( R \), and corresponding confidence map \( W \). Using the residual flow, we define the following error term:
The previous section showed how two frames \( W \) can be used to define the optimization objective. Pose increments \( \xi \) are found by performing a single differentiable Gauss-Newton optimization step.

\[
e^{ij}_{k}(\xi_i, \xi_j) = r_k - \pi((e^i_{k} G_j)(e^i_{k} G_i)^{-1} X^i_k) - \pi(G_{ij}X^i_k)), \quad X^i_k = \pi^{-1}(x_k, z_k)
\]  

where \( r_k \) is the residual flow at pixel \( x_k \) predicted by the network, and \( z_k \) is the predicted depth. The weighting map \( W \) is mapped to \((0,1)\) using the sigmoid activation, and is used to determine how the individual error terms are weighted in the final objective.

**Optimization Objective:** The previous section showed how two frames \((i, j)\) can be used to define a collection of error terms \( e^{ij}_{k}(\xi_i, \xi_j) \) for each pixel \( x_k \) in image \( I_i \). The final optimization objective is a weighted combination of error terms:

\[
E(\xi) = \sum_{(i,j) \in C} \sum_{k} e^{ij}_{k}(\xi_i, \xi_j)^T diag(w_k) e^{ij}_{k}(\xi_i, \xi_j), \quad diag(w_k) = \begin{pmatrix} w^n_k & 0 \\ 0 & w^c_k \end{pmatrix}
\]  

This leaves us with the question of which frames pairs \((i, j) \in C\) to use when defining the optimization objective. In this paper, we consider two different approaches which we refer to as \textit{Global} pose optimization and \textit{Keyframe} pose optimization.

**Global Pose Optimization:** Our global pose optimization uses all pairs of frames \( C = (i, j), i \neq j \) to define the objective function (Equation 5) and the pose increment \( \xi \) is solved for jointly over all poses. Therefore, given \( N \) frames, dense pose optimization uses \( N \times N - 1 \) frame pairs. Since every pair of frames is compared, this means that the global pose optimization requires the predicted depth maps for all frames as input. Although each pair \((i, j)\) only gives us information about the relative pose \( G_{ij}\), considering all pairs allows us to converge to a globally consistent pose graph.

**Keyframe Pose Optimization:** Our keyframe pose optimization selects a given frame to be the keyframe (i.e select \( I_1 \) as the keyframe), and only computes the error terms between the keyframe and each of the other frames: \( C = (1, j) \) for \( j = 2, ..., N \).

Fixing the pose of the keyframe, we can remove \( \xi_1 \) from the optimization objective. This means that each error \( e^{ij}_{k}(0, \xi_j) \) term is only a function of a single pose increment \( \xi_j \). Therefore, we can solve for each of the \( N - 1 \) pose increments independently. Additionally, since \( i = 1 \) for all pairs \((i, j) \in C\), we only need the depth of the keyframe as input when using keyframe pose optimization.

**LS-Optimization Layer:** Using the optimization objective in Equation 5 we solve for the pose increments \( \xi \) by applying a single Gauss-Newton update. We backpropogate through the Gauss-Newton update so that the weights of the motion module (both feature extractor and flow network) can be trained on the final objective function. In the appendix, we provide additional information for how the update is derived and the expression for the Jacobian of Equation 4.

### 3.3 Full System

During inference, we alternate the depth and motion modules for a selected number of iterations. The motion module uses depth to predict camera pose. As the depth estimates converge, the camera
poses become more accurate. Likewise, as camera poses converge, the depth module can estimate more accurate depth.

**Initialization:** We try two different strategies for initialization in our experiments: (1) self initialization initializes DeepV2D with a constant depth map and (2) single image initialization uses the output of a single-image depth network for initialization. Both methods give good performance.

### 3.4 Supervision

**Depth Supervision:** We supervise on the L1 distance between the ground truth and predicted depth. We additionally apply a small L1 smoothness penalty to the predicted depth map. Given predicted depth $Z$ and ground truth depth $Z^*$, the depth loss is defined as:

$$L_{\text{depth}}(Z) = \sum_{x_i} |Z(x_i) - Z^*(x_i)| + w_s \sum_{x_i} |\partial_x Z(x_i)| + |\partial_y Z(x_i)|$$  \hspace{1cm} (6)

**Motion Supervision:** We supervise pose using the geometric reprojection error. Given predicted pose $G$ and ground truth pose $G^*$, the pose loss is defined

$$L_{\text{motion}}(G) = \sum_{x_i} ||\pi(G^{-1}(x_i, Z(x_i))) - \pi(G^*^{-1}(x_i, Z(x_i)))||_\delta$$  \hspace{1cm} (7)

where $|| \cdot ||_\delta$ is the robust Huber loss; we set $\delta = 1$.

**Total Loss:** The total loss is taken as a weighted combination of the depth and motion loss terms:

$$\mathcal{L} = L_{\text{depth}} + \lambda L_{\text{motion}},$$

where we set $\lambda = 1.0$ in our experiments.

### 4 Experiments

We test DeepV2D across a wide range of benchmarks to provide a thorough comparison to other methods. While the primary focus of these experiments is to compare to other works which estimate depth from multiple frames, often single-view networks still outperform multiview depth estimation. To put our results in proper context, we include both multiview and state-of-the-art single-image comparisons. Since it is not possible to recover the absolute scale of the scene through SfM, we report all results (both ours and all other approaches) using scale matched depth [Tang & Tan 2018].

Our primary experiments are on NYU, ScanNet, SUN3D, and KITTI, and we report strong results across all datasets. We show visualization of our predicted depth maps in Figure 4. The figure shows that DeepV2D can recover accurate and sharp depth even when applied to unseen datasets. One aspect of particular interest is cross-dataset generalizability. Our results show that DeepV2D generalizes very well—we achieve the highest accuracy on ScanNet and SUN3D even without training on either dataset.
Figure 5: Impact of the number of iterations (left) and frames (right) on sc-inv validation accuracy. (left) shows that DeepV2D quickly converges within a small number of iterations. In (right) we see that accuracy consistently improves as more views are added. DeepV2D can be applied to variable numbers of views for a variable number of iterations without retraining.

4.1 Depth Experiments

We evaluate depth on NYU (Silberman et al., 2012), ScanNet (Dai et al., 2017), SUN3D (Xiao et al., 2013), and KITTI (Geiger et al., 2013). On all datasets, DeepV2D is given a video clip with unknown camera poses and alternates depth and pose updates and is evaluated after 8 iterations.

NYU: NYU depth (Silberman et al., 2012) is a dataset composed of videos taken in indoor settings including offices, bedrooms, and libraries. We experiment on NYU using the standard train/test split (Eigen et al., 2014) and report results in Table 1 using scaled depth (Zhou et al., 2017; Tang & Tan, 2018). We evaluate two different initialization methods of our approach. Self-init uses a constant depth map for initialization, while fcrn-init uses the output of a FCRN (Laina et al., 2016)—a single-view network for initialization. Using a single-image depth network for initialization gives a slight improvement in performance.

<table>
<thead>
<tr>
<th>Method</th>
<th>Abs Rel</th>
<th>Sc Inv</th>
<th>RMSE</th>
<th>log10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ours (self-init) - Keyframe</td>
<td>0.937</td>
<td>0.985</td>
<td>0.995</td>
<td>0.075</td>
</tr>
<tr>
<td>Ours (fcrn-init) - Keyframe</td>
<td>0.952</td>
<td>0.989</td>
<td>0.996</td>
<td>0.065</td>
</tr>
<tr>
<td>Ours (fcrn-init) - Global</td>
<td>0.951</td>
<td>0.989</td>
<td>0.997</td>
<td>0.065</td>
</tr>
</tbody>
</table>

Table 1: Results on the NYU dataset. Our approach outperforms existing single-view and multi-view depth estimation methods. Ours (self-init) uses a constant depth map for initialization while ours(fcrn-init) uses a single-image depth network for initialization.

We compare to state-of-the-art single-image depth networks DORN (Fu et al., 2018) and DenseDepth (Alhashim & Wonka, 2018) which are built on top of a pretrained ResNet (DORN) or DenseNet-201 (DenseDepth). The results show that we can do much better than single-view depth by using multiple views. We also include classical multi-view approaches such as COLMAP (Schonberger & Frahm, 2016a) and DiUSMC (Ha et al., 2016) which estimate poses with bundle adjustment, followed by dense stereo matching. While COLMAP uses SIFT features, DiUSMC is built on local-feature tracking and is designed for small baseline videos.

Table 1 also includes results using multi-view deep learning approaches. MVSNet (Yao et al., 2018) is trained to estimate depth from multiple viewpoints. Unlike our approach which estimates camera pose during inference, MVSNet requires ground truth poses as input. We train MVSNet on NYU and use poses estimated from OpenMVG (Moulin et al., 2017) during inference. Finally, we also evaluate DeMoN (Ummenhofer et al., 2017) on NYU. DeMoN is not originally trained on NYU, but instead trained on a combination of 5 other datasets. We also try a version of DeMoN which we retrain on NYU using the code provided by the authors (denoted †).
ScanNet experiments evaluating depth and pose accuracy and cross-dataset generalization. Our approach trained on NYU (ours nyu) outperforms BA-Net despite BA-Net being trained on ScanNet data; training on ScanNet (ours scannet) gives even better performance.

Figure 5 shows the impact of the number of iterations and views on the scale-invariant (sc-inv) validation set accuracy. Figure 5 (left) shows that DeepV2D requires very few iterations to converge, suggesting that block coordinate descent is effective for estimate depth from small video clips. In Figure 5 (right) we test accuracy as a function of the number of input frames used. Although DeepV2D is trained using a fixed number (4) frames as input, accuracy continues to improve as more frames are added.

**ScanNet:** ScanNet is a large indoor dataset consisting of 1513 RGB-D videos in distinct scenes. We use the train/test split proposed by Tang & Tan (2018) and evaluate depth and pose accuracy in Table 2. While our primary focus is on depth, DeepV2D accurately predicts camera motion.

We use ScanNet to test cross-dataset generalization and report results from two versions of our approach: ours (nyu) is our method trained only on nyu, ours (scannet) is our method trained on ScanNet. As expected, when we train on the ScanNet training set we do better than if we train only on NYU. But the performance of our NYU model is still good and outperforms BA-Net on all metrics. The design of our approach is motivated by generalizability. Our network only needs to learn feature matching and correspondence; this experiment indicates that by learning these low level tasks, we can generalize well to new data.

The results on ScanNet support our use of block coordinate descent to optimize camera motion and depth. BANet is the best example of a deep learning approach which performs a joint optimization of motion and depth. Our approach removes the need for a depth basis and ultimately achieves better performance. A key reason is that DeepV2D estimates depth as a direct product of stereo matching, whereas other approaches such as BANet, DeMoN, and DeepTAM rely on single-image backbones.

**SUN3D:** SUN3D (Xiao et al., 2013) is another indoor scenes dataset which we use for comparison with DeepTAM. DeepTAM only evaluates their depth module in isolation using the poses provided by dataset, while our approach is designed to estimate poses during inference. We provide results from our SUN3D experiments in Table 3.

<table>
<thead>
<tr>
<th>SUN3D</th>
<th>Training Data</th>
<th>L1-Inv ↓</th>
<th>L1-Rel ↓</th>
<th>Sc-Inv ↓</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGM</td>
<td>-</td>
<td>0.197</td>
<td>0.412</td>
<td>0.340</td>
</tr>
<tr>
<td>DTAM</td>
<td>S11+RGBD+MVS+SUN3D</td>
<td>0.210</td>
<td>0.423</td>
<td>0.374</td>
</tr>
<tr>
<td>DeMoN</td>
<td>MVS+SUNCG+SUN3D</td>
<td>0.054</td>
<td>0.101</td>
<td>0.128</td>
</tr>
<tr>
<td>Ours</td>
<td>NYU</td>
<td>0.056</td>
<td>0.106</td>
<td>0.134</td>
</tr>
<tr>
<td>Ours</td>
<td>NYU + ScanNet</td>
<td>0.041</td>
<td>0.077</td>
<td>0.104</td>
</tr>
</tbody>
</table>

Table 3: Results on SUN3D dataset and comparison to DeepTAM. DeepTAM only evaluates depth in isolation and uses the poses from the dataset during inference, while our approach jointly estimates camera poses during inference. We outperform DeepTAM and DeMoN on SUN3D even when we do not use SUN3D data for training.

We cannot train using the same data as DeepTAM since DeepTAM is trained using a combination of SUN3D, SUNCG, and MVS, and, at this time, neither MVS nor SUNCG are publicly available. Instead we train on alternate data and test on SUN3D. We test two different versions of our model; one where we train only on NYU, and another where we train on a combination of NYU and ScanNet data. Our NYU model performs similar to DeepTAM; When we combine with ScanNet data, we outperform DeepTAM even though DeepTAM is trained on SUN3D and is evaluated with ground truth pose as input.

**KITTI:** The KITTI dataset (Geiger et al., 2013) is captured from a moving vehicle and has been widely used to evaluate depth estimation and odometry. We follow the Eigen train/test split (Eigen et al., 2014), and report results in Table 4. We compare to the state-of-the-art single-view methods
and also multiview approaches such as BA-Net (Tang & Tan, 2018). We outperform all existing methods on the KITTI dataset. Notably, we outperform BA-Net on relative error (Abs Rel) by 56%.

<table>
<thead>
<tr>
<th>KITTI</th>
<th>Multi</th>
<th>$\delta &lt; 1.25$</th>
<th>$\delta &lt; 1.25^2$</th>
<th>$\delta &lt; 1.25^3$</th>
<th>Abs Rel</th>
<th>Sq Rel</th>
<th>RMSE</th>
<th>RMSE log</th>
</tr>
</thead>
<tbody>
<tr>
<td>DORN</td>
<td>N</td>
<td>0.945</td>
<td>0.988</td>
<td>0.996</td>
<td>0.069</td>
<td>0.300</td>
<td>2.857</td>
<td>0.112</td>
</tr>
<tr>
<td>DUSMC</td>
<td>Y</td>
<td>0.617</td>
<td>0.796</td>
<td>0.874</td>
<td>0.083</td>
<td>-</td>
<td>8.879</td>
<td>0.454</td>
</tr>
<tr>
<td>BA-Net</td>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>3.640</td>
<td>0.134</td>
</tr>
<tr>
<td>Ours</td>
<td>Y</td>
<td>0.978</td>
<td>0.993</td>
<td>0.997</td>
<td>0.036</td>
<td>0.190</td>
<td>2.002</td>
<td>0.074</td>
</tr>
</tbody>
</table>

Table 4: Results on the KITTI dataset. We compare to state-of-the-art single-image depth network DORN (Fu et al., 2018) and multiview BA-Net (Tang & Tan, 2018). BA-Net reports results using a different form of the Sq-Rel metric which we denote by †.

4.2 TRACKING EXPERIMENTS

DeepV2D can be turned into a basic SLAM system. Using NYU and ScanNet for training, we test tracking performance on the TUM-RGBD tracking benchmark (Table 5) using sensor depth as input. We achieve a lower translational rmse [m/s] than DeepTAM on most of the sequences. DeepTAM uses optical flow supervision to improve performance, but since our network directly maps optical flow to camera motion, we do not need supervision on optical flow.

We use our global pose optimization in our tracking experiments. We maintain a fixed window of 8 frames during tracking. At each timestep, the pose of the first 3 frames in the window are fixed and the remaining 5 are updated using the motion module. After the update, the start of the tracking window is incremented by 1 frame. We believe our ability to jointly update the pose of multiple frames is a key reason for our strong performance on the RGB-D benchmark.

<table>
<thead>
<tr>
<th></th>
<th>360</th>
<th>desk</th>
<th>desk2</th>
<th>plant</th>
<th>room</th>
<th>rpy</th>
<th>xyz</th>
<th>mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>DVO (Kerl et al., 2013)</td>
<td></td>
<td>0.125</td>
<td>0.037</td>
<td>0.020</td>
<td>0.062</td>
<td>0.042</td>
<td>0.082</td>
<td>0.051</td>
</tr>
<tr>
<td>DeepTAM (Zhou et al., 2018)</td>
<td>0.054</td>
<td>0.027</td>
<td>0.017</td>
<td>0.057</td>
<td>0.039</td>
<td>0.065</td>
<td>0.019</td>
<td>0.040</td>
</tr>
<tr>
<td>DeepTAM (w/o flow) (Zhou et al., 2018)</td>
<td>0.069</td>
<td>0.042</td>
<td>0.025</td>
<td>0.063</td>
<td>0.051</td>
<td>0.070</td>
<td>0.030</td>
<td>0.050</td>
</tr>
<tr>
<td>Ours</td>
<td>0.046</td>
<td>0.034</td>
<td>0.017</td>
<td>0.052</td>
<td>0.032</td>
<td>0.037</td>
<td>0.014</td>
<td>0.033</td>
</tr>
</tbody>
</table>

Table 5: Tracking results in the RGB-D benchmark (translational rmse [m/s]).

5 CONCLUSION

We propose DeepV2D, a deep learning architecture which is built by composing classical geometric algorithms into a fully differentiable pipeline. DeepV2D is flexible and performs well across a variety of tasks and datasets.

REFERENCES


Pierre Mouzon, Pascal Monasse, Renaud Marlet, and Others. Openmvg. an open multiple view geometry library. https://github.com/openMVG/openMVG.


Changchang Wu et al. Visualsfm: A visual structure from motion system. 2011.


A APPENDIX

A.1 LS-OPTIMIZATION LAYER:

In Equation \[3\] we defined the residual error to be:

\[
e_k^{ij}(\xi_i, \xi_j) = r_k - \{[ e^{\xi_j} G_j] e^{\xi_i} G_i \}^{-1} X_k^i - \pi( G_{ij} X_k^i , X_k^i) = \pi^{-1}(x_k, z_k)
\]  

and the objective function as the weighted sum of error terms:

\[
E(\xi) = \sum_{(i,j) \in E} \sum_k e_k^{ij}(\xi_i, \xi_j) \text{diag}(w_k) e_k^{ij}(\xi_i, \xi_j), \quad \text{diag}(w_k) = \begin{pmatrix} w_k^u & 0 \\ 0 & w_k^v \end{pmatrix}
\]

We apply a single Gauss-Newton update to Equation \[9\]. The Gauss-Newton update is computed by solving for the minimum of the second order approximation of the objective function:

\[
\xi^* = -(J^T WJ)^{-1}J^T Wr(\xi_1, ..., \xi_N), \quad J_p = \frac{\partial r_p(\xi)}{\partial \epsilon}|_{\epsilon=0}
\]

where \(r(\xi_1, ..., \xi_N)\) is the stack of residuals and \(J\) is the Jacobian matrix. Each row \(J_i\) is the Jacobian of the \(i^{th}\) residual w.r.t. to each of the parameters. Each \(\xi\) is 6-dimensional, so optimizing over \(N\) poses means we are updating \(6N\) variables.

Let \(r_p = e_k^{ij}(\xi_i, \xi_j)\) be the \(p^{th}\) residual, then

\[
\frac{\partial e_k^{ij}(\xi_i, \xi_j)}{\partial \xi_j}|_{\xi_i=0, \xi_j=0} = \frac{\partial}{\partial \xi_j}[r_k - \{[ e^{\xi_j} G_j] e^{\xi_i} G_i \}^{-1} X_k^i - \pi( G_{ij} X_k^i , X_k^i) ] =
\]

\[
\frac{\partial}{\partial \xi_j} e^{\xi_j}(\xi_i, \xi_j) = \frac{\partial}{\partial \xi_j} \pi(( e^{\xi_i} G_i ) ( e^{\xi_j} G_j )^{-1} X_k^i ) = \frac{\partial}{\partial \xi_j} \pi( e^{\xi_j} ( G_{ij} X_k^i ) )
\]

Likewise, the Jacobian for \(\xi_j\)

\[
\frac{\partial e_k^{ij}(\xi_i, \xi_j)}{\partial \xi_i}|_{\xi_i=0, \xi_j=0} = \frac{\partial}{\partial \xi_i} e^{\xi_j}(\xi_i, \xi_j) = \frac{\partial}{\partial ( G_{ij} X_k^i )} \pi( G_{ij} X_k^i ) \cdot \frac{\partial}{\partial \xi_i} e^{\xi_j}( G_{ij} X_k^i )
\]

where

\[
\frac{\partial e^{\xi_j} X}{\partial \xi}|_{\xi=0} = \begin{bmatrix} 1 & 0 & 0 & 0 & -Z & Y \\ 0 & 1 & 0 & Z & 0 & X \\ 0 & 0 & 1 & -Y & X & 0 \end{bmatrix}
\]

During training, we propagate through the Gauss-Newton update. The update is found by solving the linear system

\[
H\xi = -b, \quad H = J^T WJ, \quad b = J^T Wr(\xi_1, ..., \xi_N)
\]

Since \(H\) is positive definite, we solve Equation \[13\] using Cholesky decomposition. In the backward pass, the gradients can be found by solving another linear system.

\[
\frac{\partial C}{\partial H} = -(H^{-1})(\frac{\partial C}{\partial \xi})^T, \quad \frac{\partial C}{\partial b} = H^{-1}(\frac{\partial C}{\partial \xi})^T
\]
B  Training Details

DeepV2D is implemented in Tensorflow (Abadi et al., 2016). All components of the network are trained from scratch without using any pretrained weights. We use gradient checkpointing (Chen et al., 2016) to reduce memory usage and increase batch size.

When training on NYU and ScanNet, we train with 4 frame video clips. On KITTI, we use 5 frame video clips. The video clips are created by first selecting a keyframe. The other frames are randomly sampled from the set of frames within a specified time window of the keyframe. For example, on NYU, we create the training video by sampling from frames within 1 second of the keyframe.

Training occurs in the following two stages:

**Stage I:** We train the Motion Module using the $L_{\text{motion}}$ loss with RMSProp (Tieleman & Hinton, 2012) and a learning rate of 0.0001. For the input depth, we use the ground truth depth with missing values interpolated. We train Stage I for 20k iterations.

**Stage II:** In stage II, we jointly train the motion and depth modules end-to-end on the combined loss with RMSProp. The initial learning rate is set to .001 and decayed to .0002 after 100k training steps. During the second stage we store depth predictions to be used during the next training epoch. We train Stage II for a total of 120k iterations with a batch size of 2.

**Data Augmentation:** We perform data augmentation by adjusting brightness, gamma, and performing random scaling of the image channels. We also randomly perturb the input camera pose to the Motion Module by sampling small perturbations.

**Timing Information:** On the NYU dataset, our system operates at 440ms per iteration for a 5-frame video. On KITTI, DeepV2D runs at 400ms/iteration.

C  Additional Tracking Information

In Table 6 we report tracking results for all sequences in the Freiburg 1 dataset.

<table>
<thead>
<tr>
<th>Sequence</th>
<th>RGB-D SLAM</th>
<th>DeepTAM</th>
<th>Ours</th>
</tr>
</thead>
<tbody>
<tr>
<td>360</td>
<td>0.119</td>
<td>0.063</td>
<td>0.056</td>
</tr>
<tr>
<td>360(v)</td>
<td>0.125</td>
<td>0.054</td>
<td>0.046</td>
</tr>
<tr>
<td>desk</td>
<td>0.030</td>
<td>0.033</td>
<td>0.029</td>
</tr>
<tr>
<td>desk(v)</td>
<td>0.037</td>
<td>0.027</td>
<td>0.034</td>
</tr>
<tr>
<td>desk2</td>
<td>0.055</td>
<td>0.046</td>
<td>0.041</td>
</tr>
<tr>
<td>desk2(v)</td>
<td>0.020</td>
<td>0.017</td>
<td>0.017</td>
</tr>
<tr>
<td>floor</td>
<td>0.090</td>
<td>0.081</td>
<td>0.064</td>
</tr>
<tr>
<td>plant</td>
<td>0.036</td>
<td>0.027</td>
<td>0.019</td>
</tr>
<tr>
<td>plant(v)</td>
<td>0.062</td>
<td>0.057</td>
<td>0.052</td>
</tr>
<tr>
<td>room</td>
<td>0.048</td>
<td>0.040</td>
<td>0.047</td>
</tr>
<tr>
<td>room(v)</td>
<td>0.042</td>
<td>0.039</td>
<td>0.032</td>
</tr>
<tr>
<td>rpy</td>
<td>0.043</td>
<td>0.046</td>
<td>0.039</td>
</tr>
<tr>
<td>rpy(v)</td>
<td>0.082</td>
<td>0.065</td>
<td>0.037</td>
</tr>
<tr>
<td>teddy</td>
<td>0.067</td>
<td>0.059</td>
<td>0.043</td>
</tr>
<tr>
<td>xyz</td>
<td>0.051</td>
<td>0.019</td>
<td>0.025</td>
</tr>
<tr>
<td>xyz(v)</td>
<td>0.024</td>
<td>0.017</td>
<td>0.016</td>
</tr>
<tr>
<td>Average</td>
<td>0.058</td>
<td>0.043</td>
<td>0.037</td>
</tr>
</tbody>
</table>

Table 6: Per-Sequence tracking results on the RGB-D benchmark evaluated using translational RMSE [m/s]. We outperform DeepTAM and DVO on 12 of the 16 sequences and achieve a lower translational RMSE averaged over all sequences. While DeepTAM requires optical flow supervision to achieve good performance, we do not require supervision on optical flow since the relation between camera motion and optical flow is embedded into our network architecture.
D Camera Pose Ablations

The focus of this work on depth estimation, but we are interested in how different methods for estimating camera pose impact the final performance. In Table 7, we test different methods for estimating camera pose on NYU. In each experiment, we replace the motion module of our trained network with the given alternative, and test the final results. We also report results from MVSNet (trained on NYU) using each SfM implementation.

COLMAP (Schonberger & Frahm, 2016a) and OpenMVG (Moulon et al.) are publicly available SfM implementations. They do not return results on all input sequences, so we only evaluate sequences were they converge without an error. PWCNet+Ceres takes the output of an optical flow network, PWCNet (Sun et al., 2018), and performs joint optimization of depth and pose using the Ceres solver (Agarwal et al., 2012). Finally, we evaluate MVSNet (Yao et al., 2018) when the pose predicted by DeepV2D is given as input. Note that not all SfM implementations converge on all sequences (success rate is reported in parenthesis) and we only evaluate the method on the frames in which it converges.

<table>
<thead>
<tr>
<th>Depth</th>
<th>Motion</th>
<th>Abs-Rel ↓</th>
<th>δ₁ ↑</th>
<th>δ₂ ↑</th>
<th>δ₃ ↑</th>
</tr>
</thead>
<tbody>
<tr>
<td>MVSNet</td>
<td>Identity</td>
<td>0.419</td>
<td>0.382</td>
<td>0.681</td>
<td>0.859</td>
</tr>
<tr>
<td>DeepV2D</td>
<td>Identity</td>
<td>0.362</td>
<td>0.460</td>
<td>0.756</td>
<td>0.901</td>
</tr>
<tr>
<td>MVSNet</td>
<td>COLMAP (274/654)</td>
<td>0.244</td>
<td>0.724</td>
<td>0.857</td>
<td>0.925</td>
</tr>
<tr>
<td>DeepV2D</td>
<td>COLMAP</td>
<td>0.199</td>
<td>0.741</td>
<td>0.878</td>
<td>0.940</td>
</tr>
<tr>
<td>MVSNet</td>
<td>OpenMVG (422/654)</td>
<td>0.181</td>
<td>0.766</td>
<td>0.915</td>
<td>0.965</td>
</tr>
<tr>
<td>DeepV2D</td>
<td>OpenMVG</td>
<td>0.173</td>
<td>0.774</td>
<td>0.913</td>
<td>0.963</td>
</tr>
<tr>
<td>MVSNet</td>
<td>PWC+Ceres (654/654)</td>
<td>0.279</td>
<td>0.851</td>
<td>0.845</td>
<td>0.925</td>
</tr>
<tr>
<td>DeepV2D</td>
<td>PWC+Ceres</td>
<td>0.274</td>
<td>0.664</td>
<td>0.846</td>
<td>0.925</td>
</tr>
<tr>
<td>MVSNet</td>
<td>DeepV2D (654/654)</td>
<td>0.101</td>
<td>0.885</td>
<td>0.970</td>
<td>0.990</td>
</tr>
<tr>
<td>DeepV2D (ours)</td>
<td></td>
<td><strong>0.065</strong></td>
<td><strong>0.952</strong></td>
<td><strong>0.989</strong></td>
<td><strong>0.996</strong></td>
</tr>
</tbody>
</table>

Table 7: Impact of pose estimation method on depth accuracy. Replacing our motion module with SfM degrades performance for both MVSNet and our approach.

We also show results of our method when the motion module is replaced with other methods for estimation motion. In all cases, using SfM results in worse performance. We observe that classical SfM is not robust enough to consistently produce accurate poses, which leads to large errors on the test set. MVSNet performs better using the poses estimated by our network, but still underperforms our full system, showing the importance of differentiable alternation between pose and stereo.
E  ADDITIONAL RESULTS

Figure 6: Visualizations of depth predictions on KITTI dataset.

Figure 7: Additional results on the NYU depth dataset [Silberman et al., 2012] using 7-frame video clips. We show results compared with [Laina et al., 2016] and [Ummenhofer et al., 2017].
F  NETWORK ARCHITECTURES

Figure 8: Motion Module Architecture: The Encoder(left) extracts a dense 1/4 resolution feature map for each of the input images. The Residual Flow Network (right) takes in a pair of feature maps and estimates the residual flow and corresponding weights. This residual flow is estimated with an encoder-decoder network, with skip connections formed by concatenating feature maps. Numbers in parenthesis correspond to the number of output channels for each layer.

Figure 9: Depth Module Architecture: The 2D encoder (top) is applied to each image in the video sequence. The 2D Encoder consists of a series of residual convolutions and 2 Hourglass Networks. The hourglass networks process the incoming features maps as multiple scales. The hourglass network is defined recursively (i.e. HG(n) contains lower resolution hourglass HG(n-1)). We use 4 nested hourglass modules with feature dimension 64-128-192-256. The resulting feature maps from the 2D encoder are used to construct the cost volumes. The 3D matching network (bottom) takes a collection of cost volumes as input. After a 1x1x1 convolutional layer and a 3x3x3 residual convolution, we perform view pooling, which aggregates information over all the frames in the video. The aggregated volume is then processed by a series of 3D hourglass networks, each of which outputs an intermediate depth estimate. The widths of the 3D hourglass is 32-80-128-176.