Deep learning for 3-D Scene Reconstruction and Modeling

Yu Huang

Yu.Huang07@gmail.com

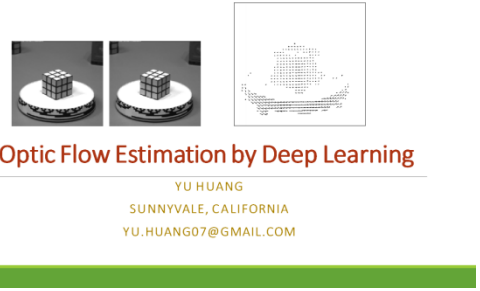
Sunnyvale, CA

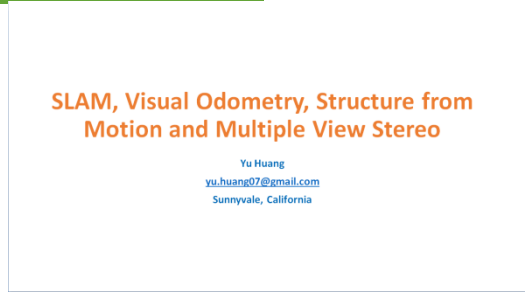
Outline

- Background
- LIFT: Learned Invariant Feature Transform
- Detect/Match Keypoints with Deep Architectures
- MatchNet
- Universal Correspondence Network
- Depth Prediction using a Multi-Scale Deep Network
- Deeper Depth Prediction with Fully Convol. ResNet
- Depth Estimation with Left-Right Consistency
- Depth Estimation: Geometry to the Rescue
- Deep Depth From Focus
- Deep Learning for Monocular Depth Map Prediction
- Scene Parsing with Perspective Understanding
- PoseNet
- Generate Object Models with CNNs
- 3D View Synthesis by Deep NN
- Multi-view 3D Models from Single Images
- View Synthesis by Appearance Flow
- Perspective Transformer Nets
- 3D-R2N2
- SurfNet
- Face reconstruction
- Deep3D
- DeepStereo
- Novel View Generation from Deep NN
- SfMNet
- DEMON
- Learning of Depth and Ego-Motion from Video
- CNN-SLAM
- Neural SLAM
- Semantic scene completion from depth image
- **Appendix**: Introduction of Deep Learning

Background and Problems

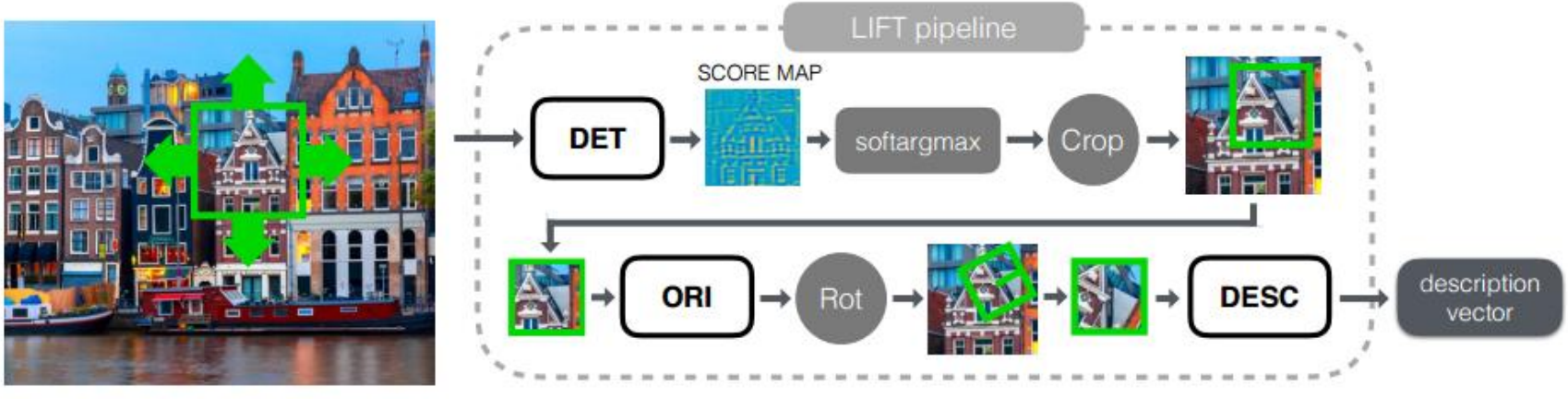
- Low level vision: Feature description and matching;
- Image-based relocalization;
- 3-D geometric object modeling;
- Image-based modeling: view synthesis;
- 2D-to-3D: depth/disparity prediction;
- Structure from Motion and Scene Flow;
- Visual odometry: Pose estimation;
- Multiple view reconstruction: Depth fusion;
- SLAM;





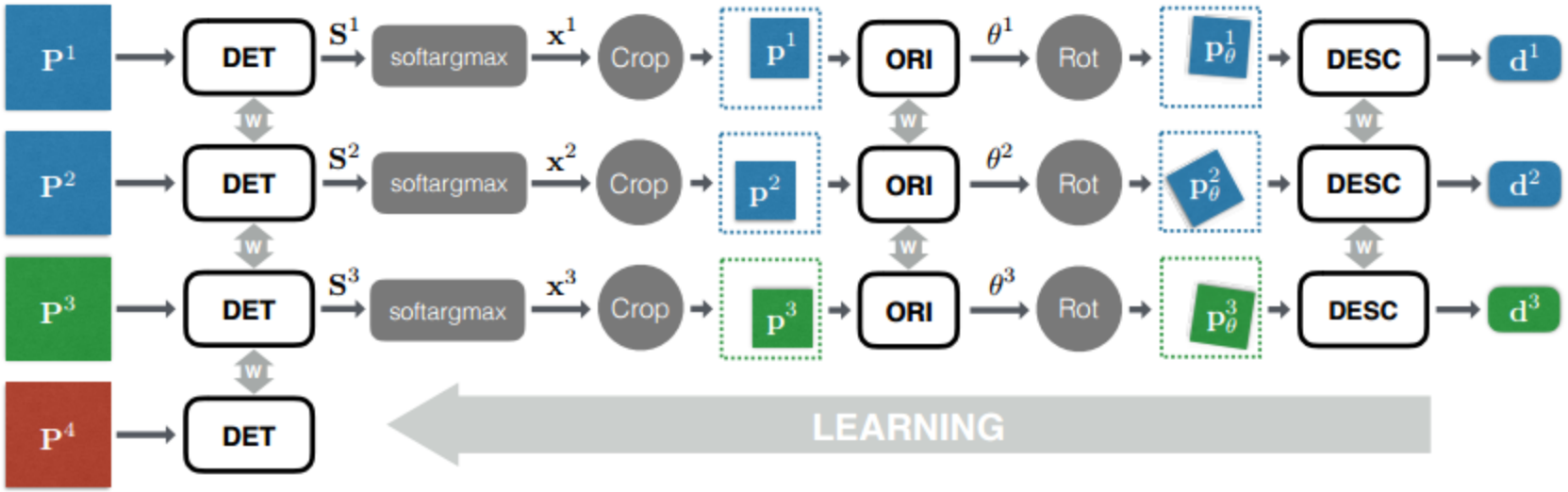
LIFT: Learned Invariant Feature Transform

- A Deep Network architecture that implements the full feature point handling pipeline, that is, detection, orientation estimation, and feature description.
- Learn to do all 3 in a unified manner while preserving end-to-end differentiability.



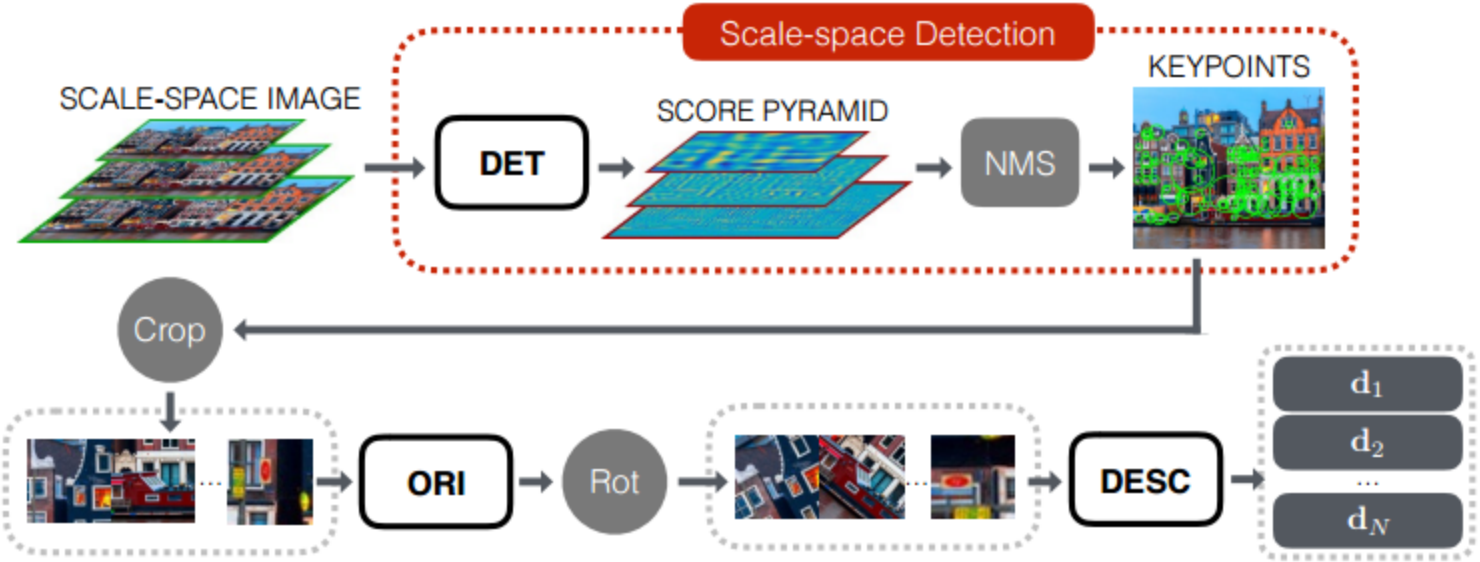
The integrated feature extraction pipeline. It consists of 3 major components: the Detector, the Orientation Estimator, and the Descriptor. They are tied together with differentiable operations to preserve end-to-end differentiability.

LIFT: Learned Invariant Feature Transform



The Siamese training architecture with 4 branches, which takes as input a quadruplet of patches: Patches P 1 and P 2 correspond to different views of the same physical point, and are used as positive examples to train the Descriptor; P 3 shows a different 3D point, which serves as a negative example for the Descriptor; and P 4 contains no distinctive feature points and is only used as a negative example to train the Detector. Given a patch P, the Detector, the softargmax, and the Spatial Transformer layer Crop provide all together a smaller patch p inside P. p is then fed to the Orientation Estimator, which along with the Spatial Transformer layer Rot, provides the rotated patch p_θ that is processed by the Descriptor to obtain the final description vector d.

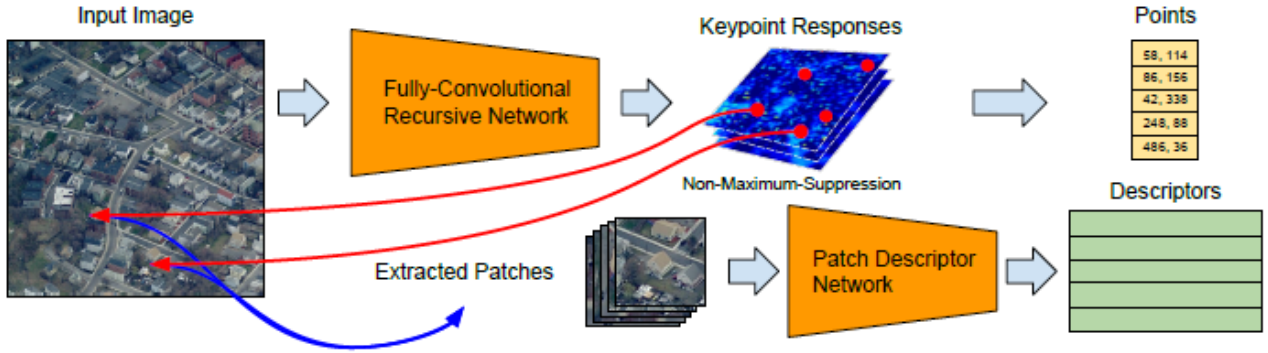
LIFT: Learned Invariant Feature Transform



As the Orientation Estimator and the Descriptor only require evaluation at local maxima, decouple the Detector and run it in scale space with traditional NMS (non-maximal suppression) to obtain proposals for the two other components.

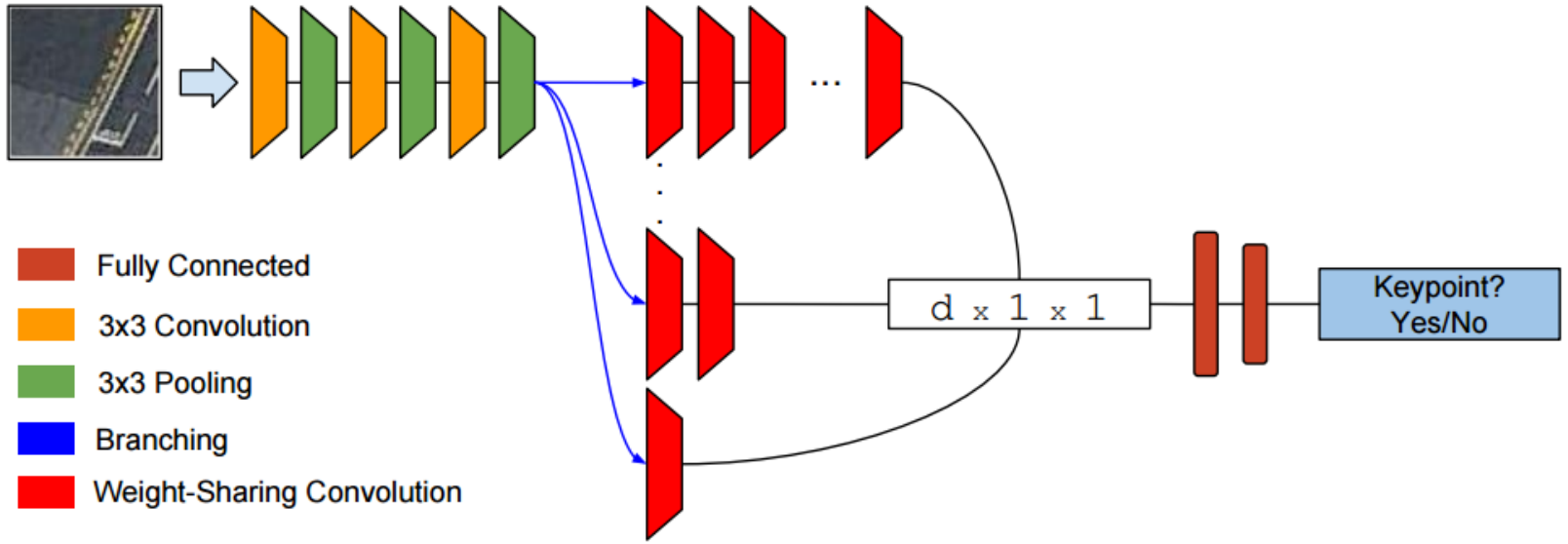
Learning to Detect and Match Keypoints with Deep Architectures

- Learning to detect and describe keypoints from images leveraging deep architectures.
- The model learns from this vast dataset to identify and describe meaningful keypoints.



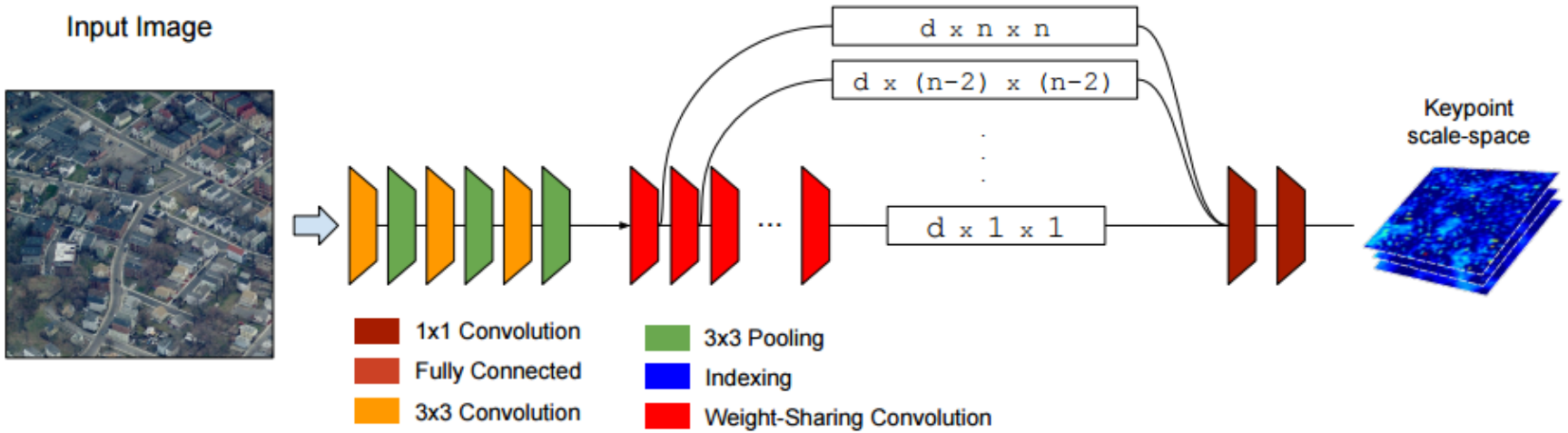
Given an image, a fully convolutional recursive network outputs a scale-pyramid of keypoint responses, which are used to extract patches. Then, the patches are described by a patch descriptor network.

Learning to Detect and Match Keypoints with Deep Architectures



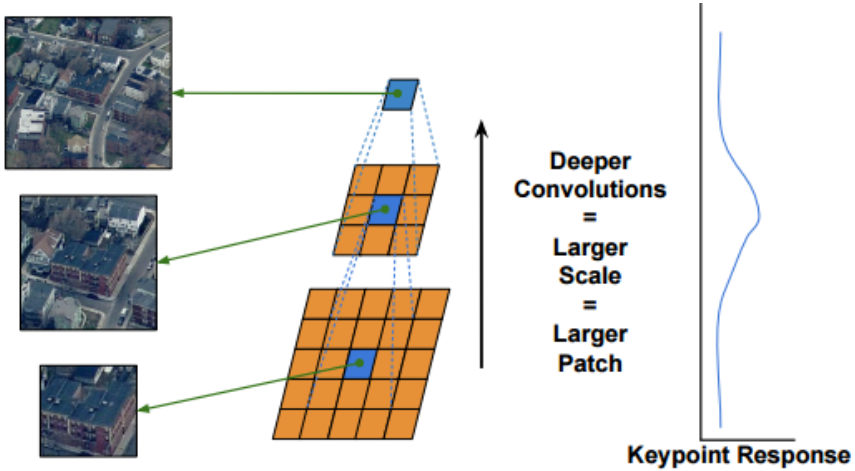
Training architecture for multiscale keypoint detection network. First, patches pass through a set of convolutions and pooling layers. Then, a recursive convolution is applied until the feature map dimension is 1 × 1. A scale-dependent branch is chosen for each patch determining the number of recursive convolutions. Finally, two FCLs lead into a binary keypoint classifier.

Learning to Detect and Match Keypoints with Deep Architectures

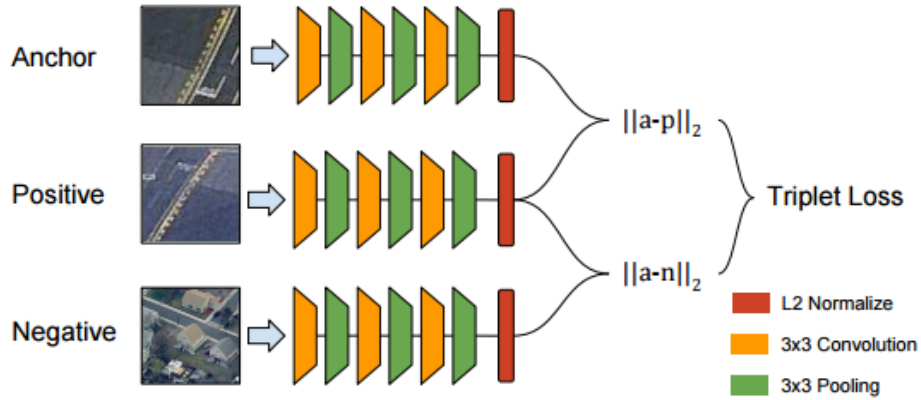


Inference architecture for multiscale keypoint detection network. First, an input image is passed through a set of convolutions and pooling layers. Then, a recursive convolution is applied until the feature map dimension is 1×1. After each recursive convolution, compute the keypoint feature map. The output feature maps resemble a keypoint scale-space.

Learning to Detect and Match Keypoints with Deep Architectures



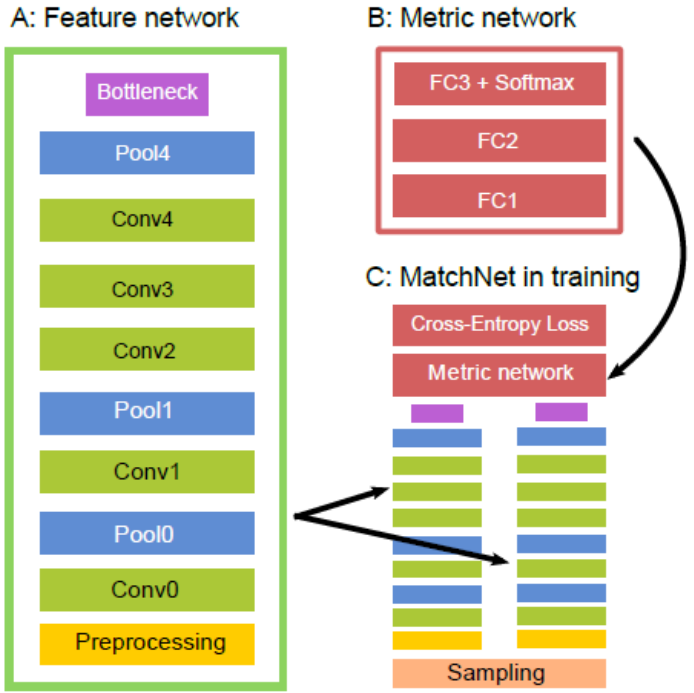
Convolutions later in the detection network correspond to larger patch sizes. The keypoint feature map with the highest response indicates the best keypoint scale.



Training architecture of the keypoint description triplet network. Three patches are passed through channels which share weights to rank their euclidean distances in the feature space.

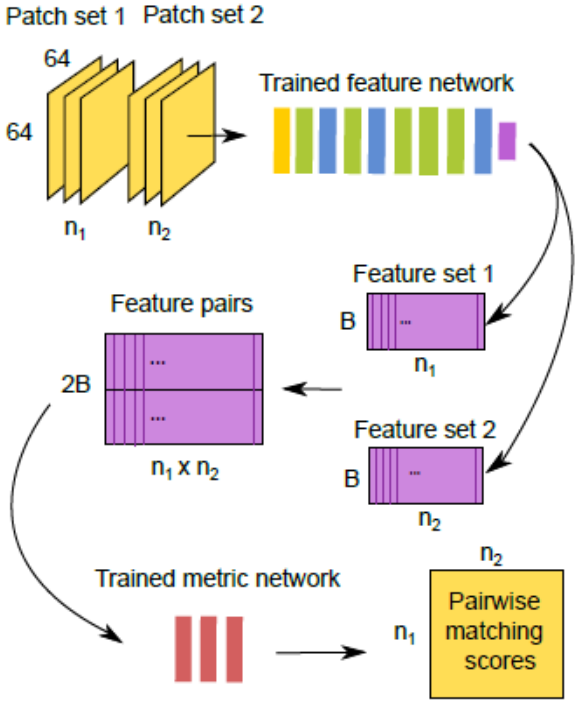
MatchNet: Unifying Feature and Metric Learning for Patch-Based Matching

- A unified approach to combining both for training a patch matching system, MatchNet;
- A deep convolutional network that extracts features from patches and a network of three fully connected layers that computes a similarity between the extracted features.
- Train MatchNet on standard datasets and employ an input sampler to augment the training set with synthetic exemplar pairs that reduce overfitting.



A: The feature network used for feature encoding, with an optional bottleneck layer to reduce feature dimension.
B: The metric network used for feature comparison.
C: In training, the feature net is applied as two “towers” on pairs of patches with shared parameters.
Output from the two towers are concatenated as the metric network’s input. The entire network is jointly trained on labeled patch-pairs generated from the sampler to minimize the cross-entropy loss. In prediction, the two sub-networks (A and B) are conveniently used in a two-stage pipeline.

MatchNet: Unifying Feature and Metric Learning for Patch-Based Matching



MatchNet is disassembled during prediction

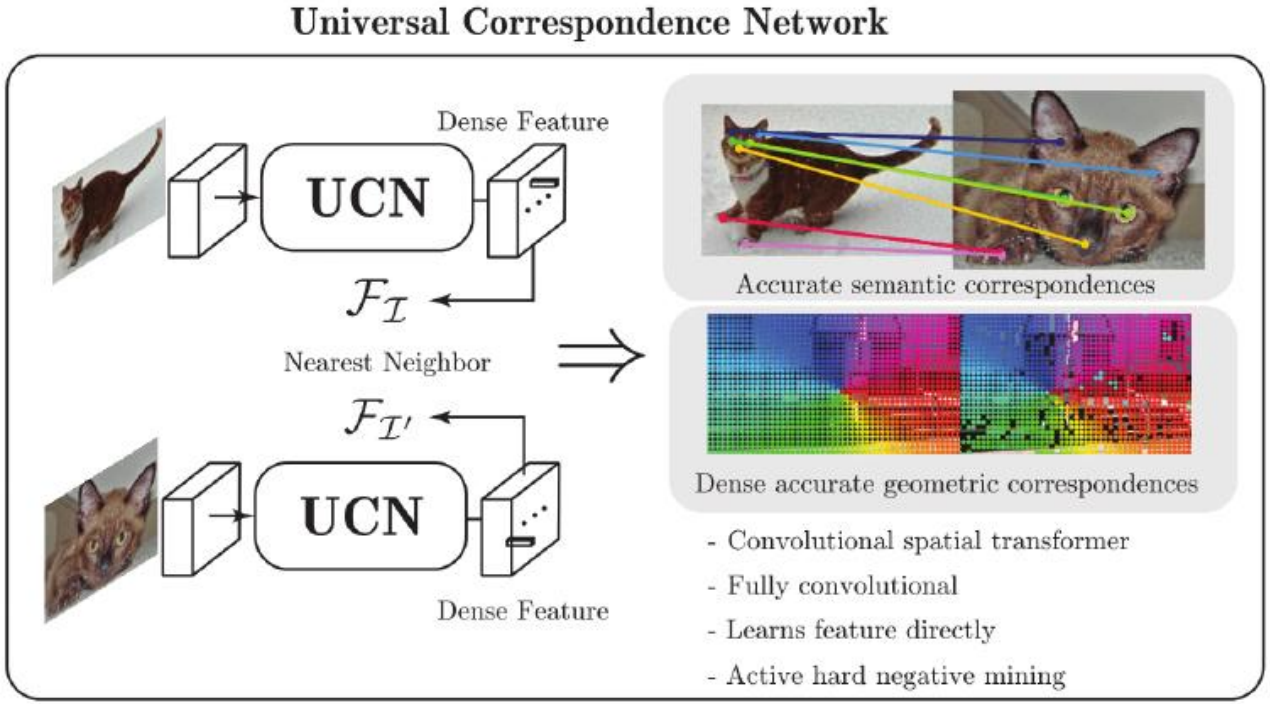
Table 1. Layer parameters of MatchNet. The output dimension is given by (height × width × depth). PS: patch size for convolution and pooling layers; S: stride. Layer types: C: convolution, MP: max-pooling, FC: fully-connected. We always pad the convolution and pooling layers so the output height and width are those of the input divided by the stride. For FC layers, their size B and F are chosen from: $B \in \{64, 128, 256, 512\}$, $F \in \{128, 256, 512, 1024\}$. All convolution and FC layers use ReLU activation except for FC3, whose output is normalized with Softmax (Equation 2).

Name	Type	Output Dim.	PS	S
Conv0	C	$64 \times 64 \times 24$	7×7	1
Pool0	MP	$32 \times 32 \times 24$	3×3	2
Conv1	C	$32 \times 32 \times 64$	5×5	1
Pool1	MP	$16 \times 16 \times 64$	3×3	2
Conv2	C	$16 \times 16 \times 96$	3×3	1
Conv3	C	$16 \times 16 \times 96$	3×3	1
Conv4	C	$16 \times 16 \times 64$	3×3	1
Pool4	MP	$8 \times 8 \times 64$	3×3	2
Bottleneck	FC	B	-	-
FC1	FC	F	-	-
FC2	FC	F	-	-
FC3	FC	2	-	-

Universal Correspondence Network

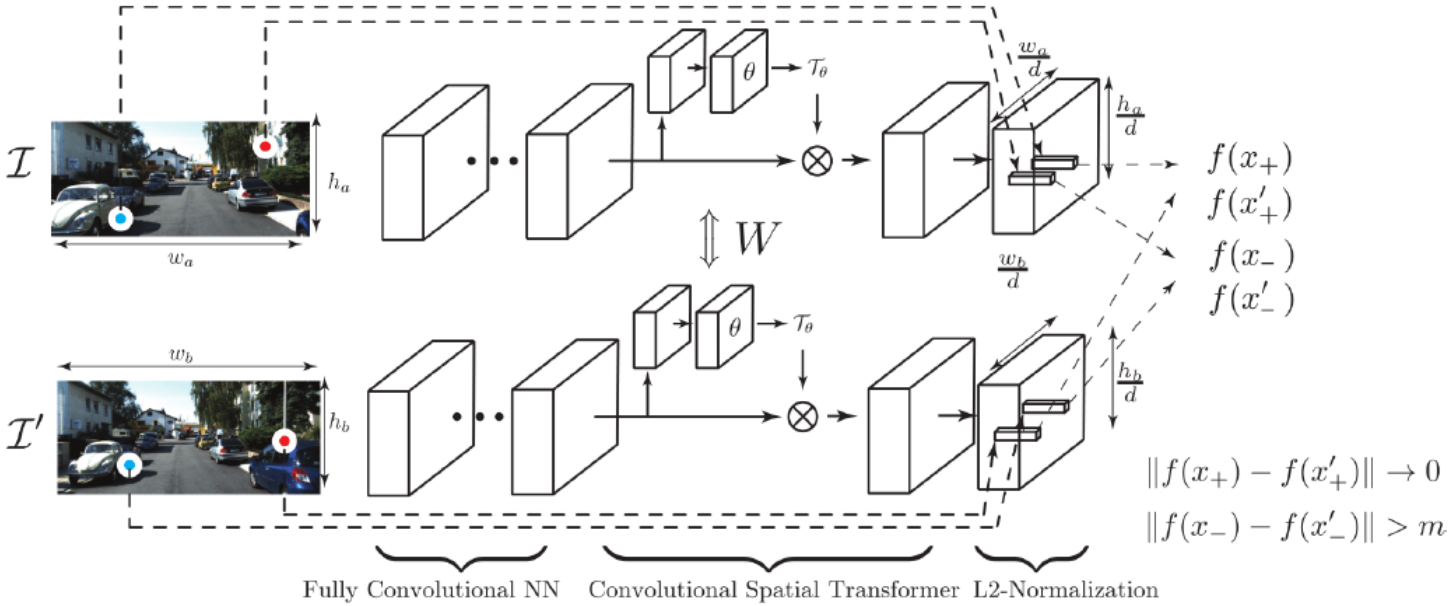
- A deep learning framework for accurate visual correspondences for both geometric and semantic matching, spanning across rigid motions to intra-class shape or appearance variations.
- Deep metric learning directly learns a feature space that preserves either geometric or semantic similarity.
- Fully convolutional architecture, along with a correspondence contrastive loss allows faster training, accurate gradient computation through the use of thousands of examples per image pair and faster testing with O(n) feed forward passes for n keypoints, instead of O(n2) for typical patch similarity methods.
- A convolutional spatial transformer to mimic patch normalization in traditional features like SIFT, which is shown to dramatically boost accuracy for semantic correspondences across intra-class shape variations.

Universal Correspondence Network



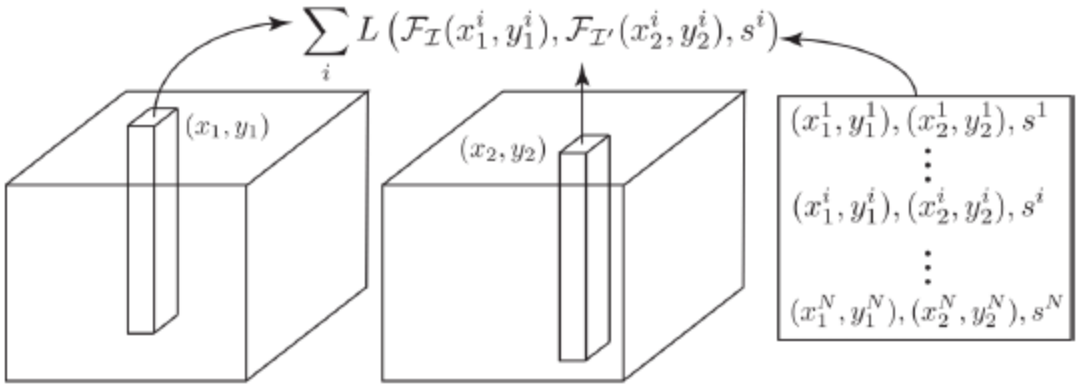
The Universal Correspondence Network accurately and efficiently learns a metric space for geometric correspondences, dense trajectories or semantic correspondences.

Universal Correspondence Network



The network is fully convolutional, consisting of a series of convolutions, pooling, nonlinearities and a convolutional spatial transformer, followed by channel-wise L2 normalization and correspondence contrastive loss.

Universal Correspondence Network



Correspondence contrastive loss takes three inputs: two dense features extracted from images and a correspondence table for positive and negative pairs.

Depth Prediction using a Multi-Scale Deep Network

- Two deep network stacks: one that makes a coarse global prediction based on the entire image, and another that refines this prediction locally;
- Apply a scale-invariant error to help measure depth relations rather than scale;

- Augment training data with online random transformations (scale, rotation, translation, flips, color).
- Baseline for comparison: Make3D.

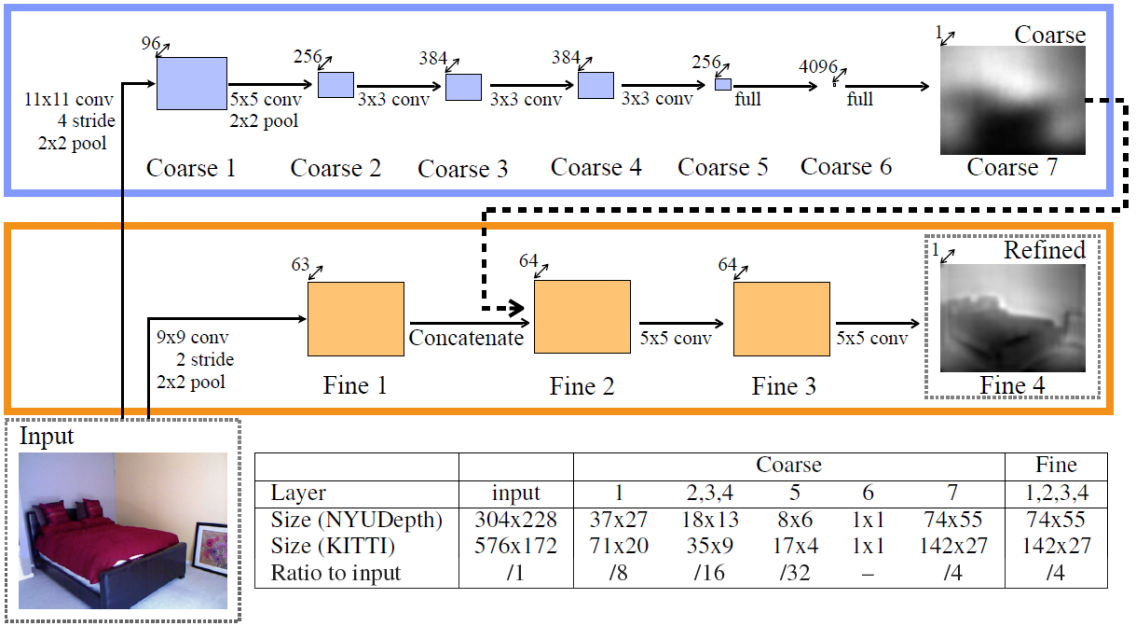
$$D(y, y^*) = \frac{1}{n}\sum_{i=1}^{n}(\log y_i - \log y_i^* + \alpha(y, y^*))^2$$

$$\text{where } \alpha(y, y^*) = \frac{1}{n}\sum_i(\log y_i^* - \log y_i)$$

$$d_i = \log y_i - \log y_i^*$$

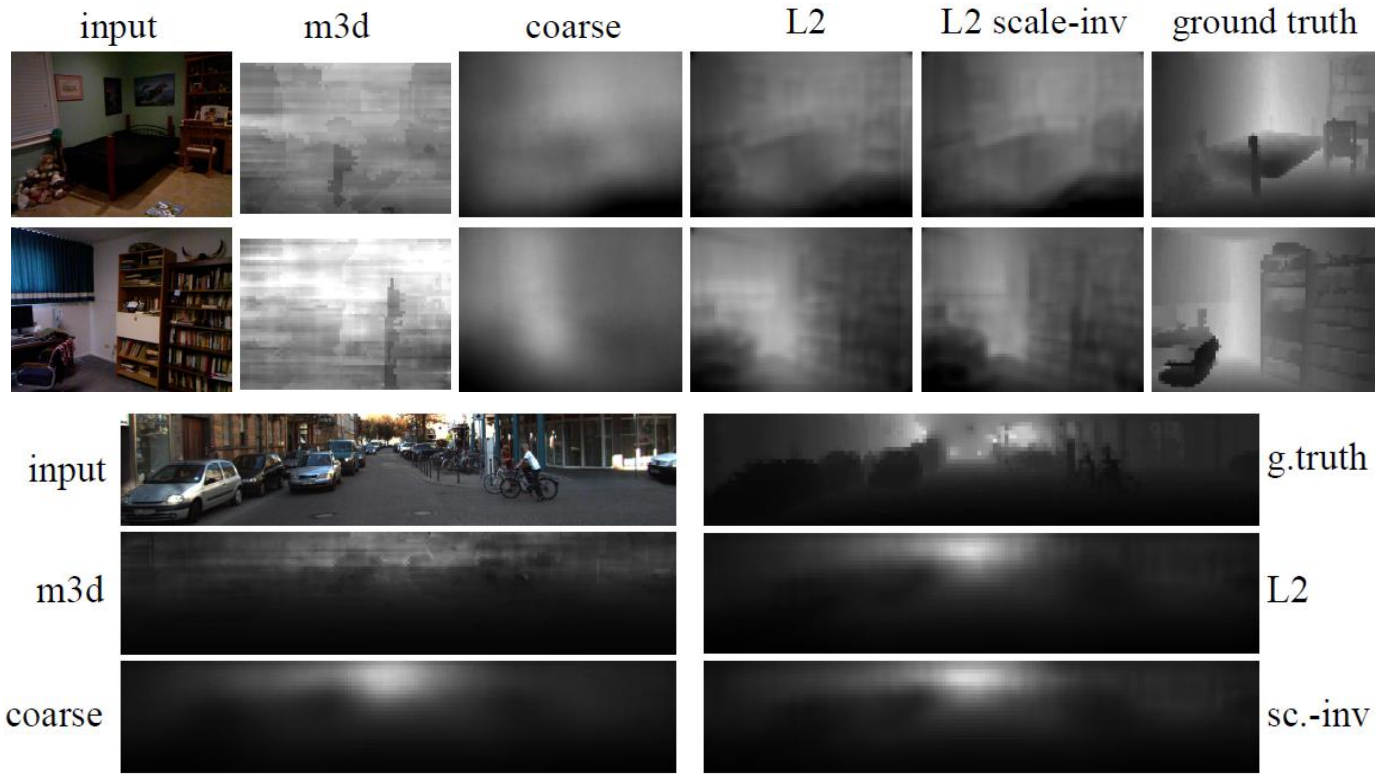
$$D(y, y^*) = \frac{1}{n^2}\sum_{i,j}\left((\log y_i - \log y_j) - (\log y_i^* - \log y_j^*)\right)^2$$

$$= \frac{1}{n}\sum_i d_i^2 - \frac{1}{n^2}\sum_{i,j} d_i d_j = \frac{1}{n}\sum_i d_i^2 - \frac{1}{n^2}\left(\sum_i d_i\right)^2$$

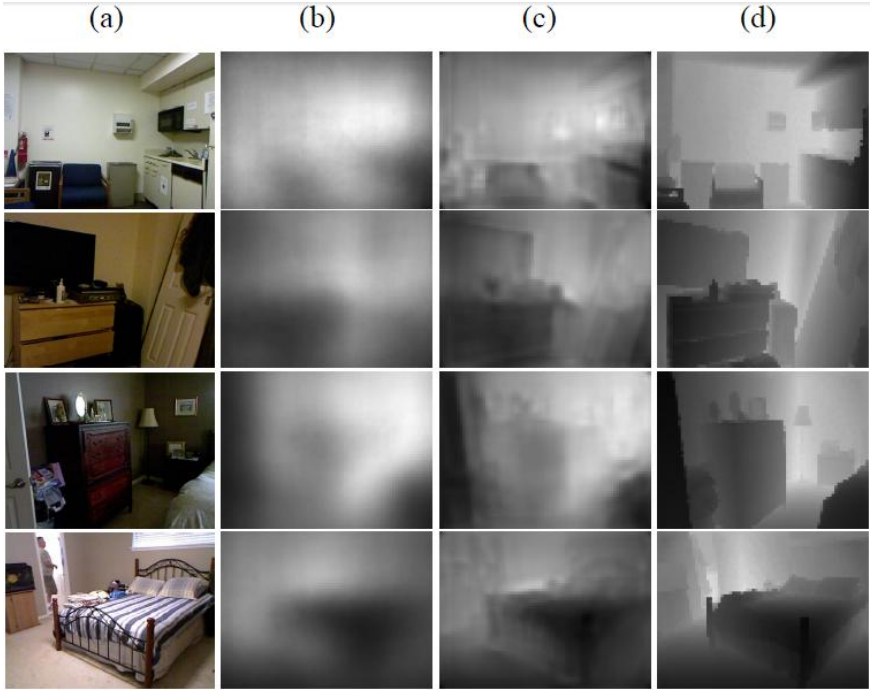


		Coarse					Fine
Layer	input	1	2,3,4	5	6	7	1,2,3,4
Size (NYUDepth)	304x228	37x27	18x13	8x6	1x1	74x55	74x55
Size (KITTI)	576x172	71x20	35x9	17x4	1x1	142x27	142x27
Ratio to input	/1	/8	/16	/32	–	/4	/4

Depth Prediction using a Multi-Scale Deep Network



Depth Prediction using a Multi-Scale Deep Network



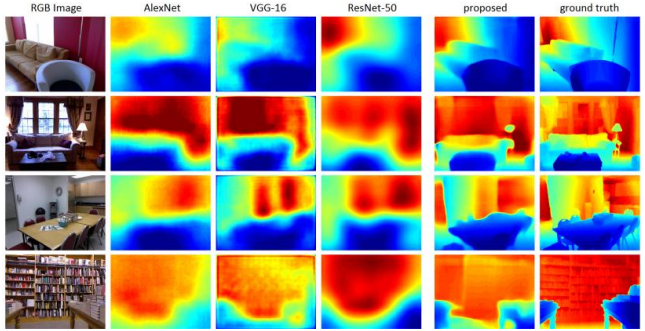
(a) input, (b) output of coarse network, (c) refined output of fine network, (d) ground truth.

Deeper Depth Prediction with Fully Convolutional Residual Networks

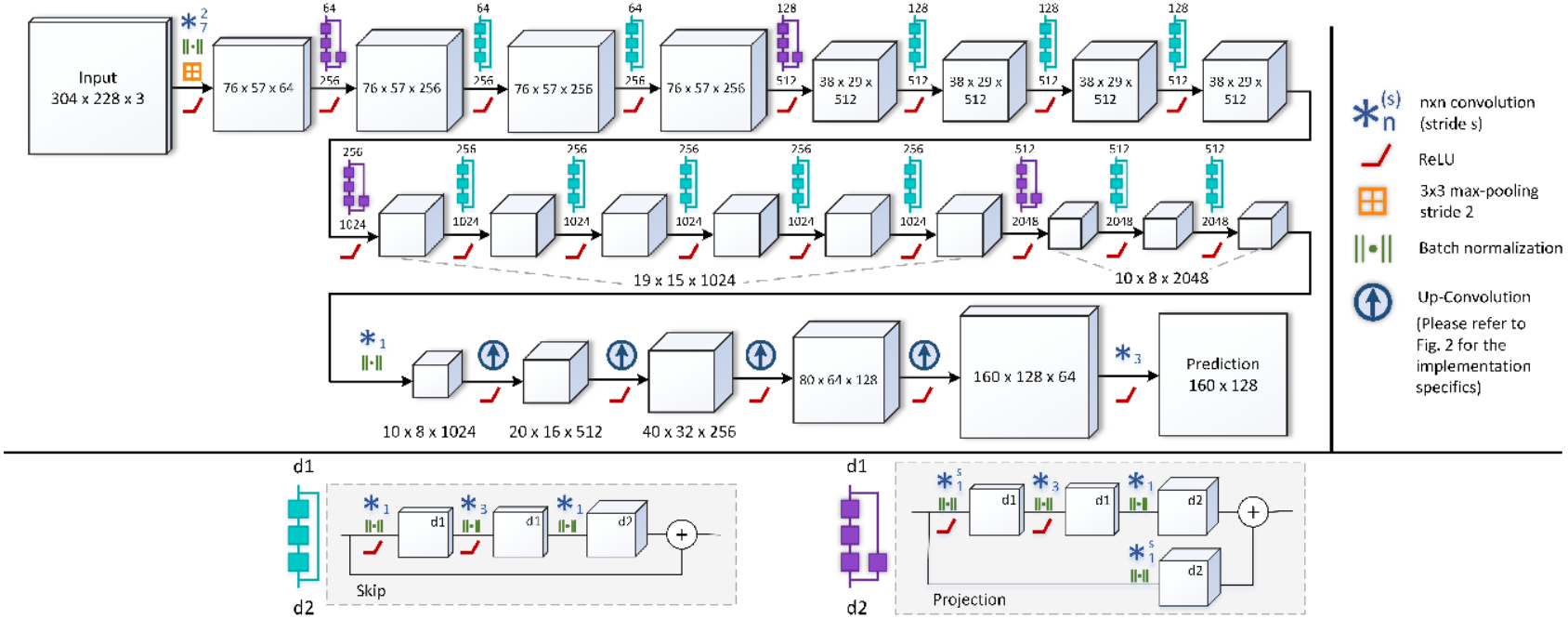
- A fully convolutional architecture, encompassing residual learning, to model the ambiguous mapping btw monocular images and depth maps, trained end-to-end.
- Efficiently learn feature map up-sampling within the network: For optimization, the reverse Huber loss driven by the value distributions commonly present in depth maps.
- Not rely on post-processing techniques, such as CRFs or other additional refinement steps.
- Source codes: https://github.com/iro-cp/FCRN-DepthPrediction.

The reverse Huber (berHu) as loss function

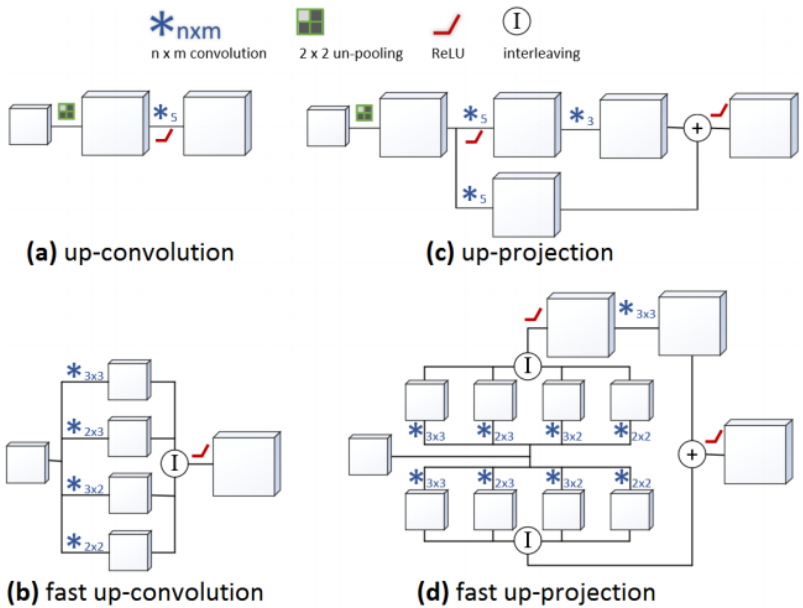
$$\mathcal{B}(x) = \begin{cases} |x| & |x| \leq c, \\ \frac{x^2+c^2}{2c} & |x| > c. \end{cases}$$



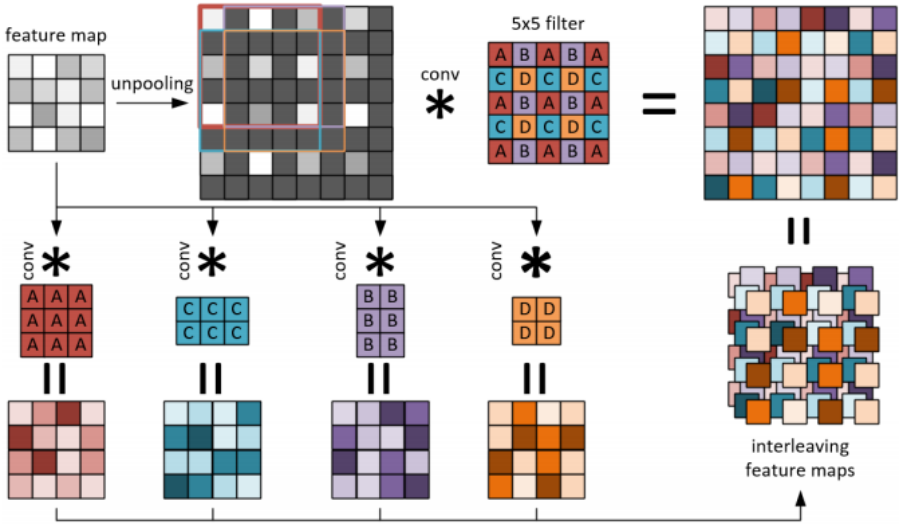
Deeper Depth Prediction with Fully Convolutional Residual Networks



Deeper Depth Prediction with Fully Convolutional Residual Networks



From up-convolutions to up-projections.

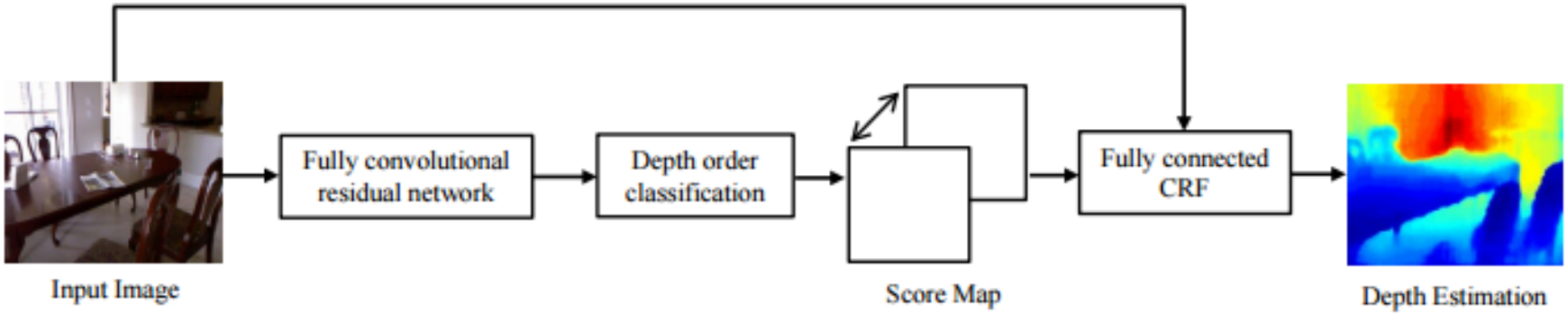


Faster up-convolutions. Convolving the original feature map with the 4 differently composed filters and interleaving them to obtain the same output, while avoiding zero multiplications.

Estimating Depth from Monocular Images as Classification Using Deep Fully Convolutional Residual Networks

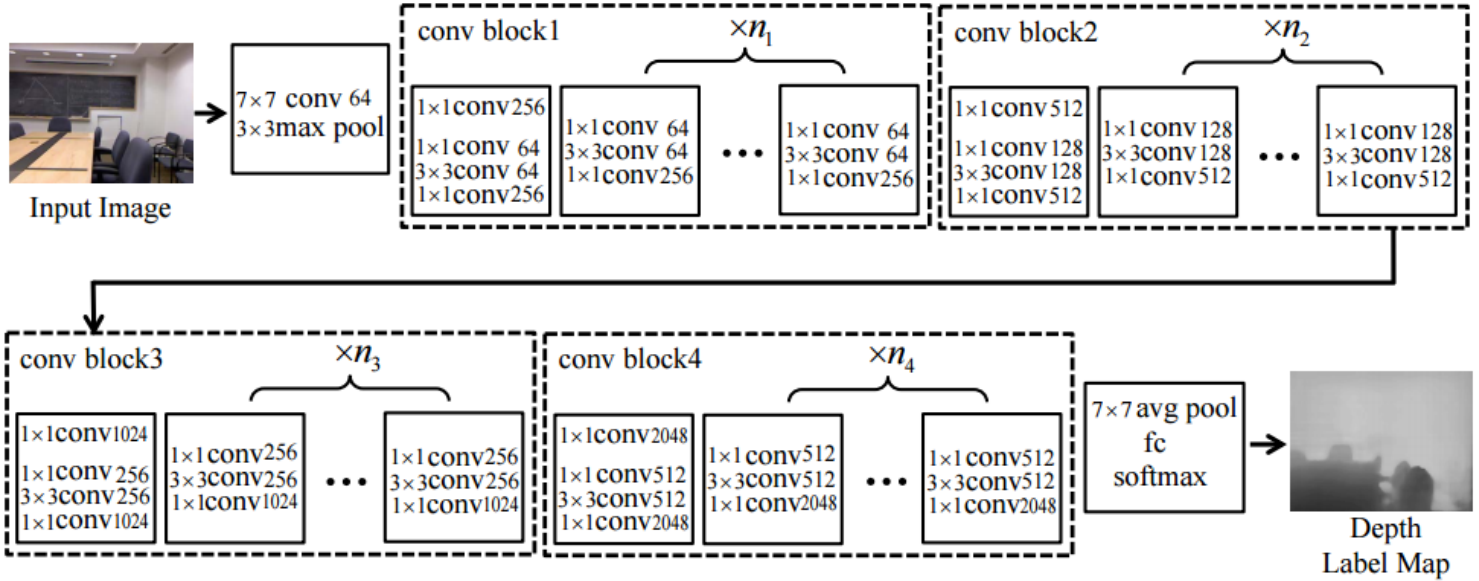
- Formulate depth estimation as a pixel-wise classification task.
- Specifically, discretize the continuous depth values into multiple bins and label the bins according to their depth range.
- Train fully convolutional deep residual networks to predict the depth label of each pixel.
- Performing discrete depth labeling instead of continuous depth value regression allows to predict a confidence in the form of prob. distribution.
- Apply fully-connected conditional random fields (CRF) as a post processing step to enforce local smoothness interactions, which improves the results.

Estimating Depth from Monocular Images as Classification Using Deep Fully Convolutional Residual Networks



An overview of depth estimation model. It takes as input an RGB image and output score maps. Fully-connected CRFs are then applied to obtain the final depth estimation.

Estimating Depth from Monocular Images as Classification Using Deep Fully Convolutional Residual Networks



Network architecture. The input image is fed into a convolution layer, a max pooling layer and 4 convolution blocks. The value of [n1, n2, n3, n4] is [2, 3, 22, 2] for the 101-layer network architecture and [2, 7, 35, 2] for the 152-layer network architecture. The last 3 layers are an average pooling layer, a full-connected layer with number of channels equal to the number of labels and a softmax layer. The output map is downsampled by a factor of 8.

Estimating Depth from Monocular Images as Classification Using Deep Fully Convolutional Residual Networks

The energy function of the fully-connected CRF is the sum of unary potential U and pairwise potential V :

$$E(\mathbf{D}) = \sum_i U(D_i) + \sum_{i,j} V(D_i, D_j),$$

$$U(D_i) = L(D_i) = -\log(P(D_i|z_i)).$$

$$\sum_{i,j} V(D_i, D_j) = \Delta(D_i, D_j) \sum_{s=1}^{M} w_s \cdot k^s(\mathbf{f}_i, \mathbf{f}_j),$$

$$L = -\frac{1}{\sum_{i=1}^{N} 1\{P(D_i^*|z_i) < t\}} \times \sum_{i=1}^{N} \sum_{D=1}^{B} 1\{P(D_i^*|z_i) < t\} H(D_i^*, D) \log(P(D|z_i))$$

$$\Delta(D_i, D_j) = |D_i - D_j|$$

Each k^s is the Gaussian kernel depends on features (denoted as f) extracted for pixel i and j and is weighted by parameter w_s.

$$w_1 \exp\left(-\frac{\|p_i - p_j\|^2}{2\sigma_\alpha^2} - \frac{\|I_i - I_j\|^2}{2\sigma_\beta^2}\right) + w_2 \exp\left(-\frac{\|p_i - p_j\|^2}{2\sigma_\gamma^2}\right)$$

Unsupervised Monocular Depth Estimation with Left-Right Consistency

- Replacing use of explicit depth data during training with easier-to-obtain binocular stereo footage;
- Training objective that enables CNN to learn to perform single image depth estimation, despite the absence of ground truth depth data.
- Exploiting epipolar geometry constraints, generate disparity images by training the network with an image reconstruction loss.
- A training loss that enforces consistency between the disparities produced relative to both the left and right images, leading to improved performance and robustness compared to existing

$$C_s = \alpha_{ap}(C^l_{ap} + C^r_{ap}) + \alpha_{ds}(C^l_{ds} + C^r_{ds}) + \alpha_{lr}(C^l_{lr} + C^r_{lr}), \quad (1)$$

loss C_s at each output scale s

$$C^l_{ds} = \frac{1}{N}\sum_{i,j} |\partial_x d^l_{ij}| e^{-\|\partial_x I^l_{ij}\|} + |\partial_y d^l_{ij}| e^{-\|\partial_y I^l_{ij}\|}. \quad (3)$$

disparity smoothness loss

$$C^l_{ap} = \frac{1}{N}\sum_{i,j} \alpha \frac{1 - \mathrm{SSIM}(I^l_{ij}, \tilde{I}^l_{ij})}{2} + (1-\alpha)\left\| I^l_{ij} - \tilde{I}^l_{ij} \right\|. \quad (2)$$

photometric image reconstruction cost

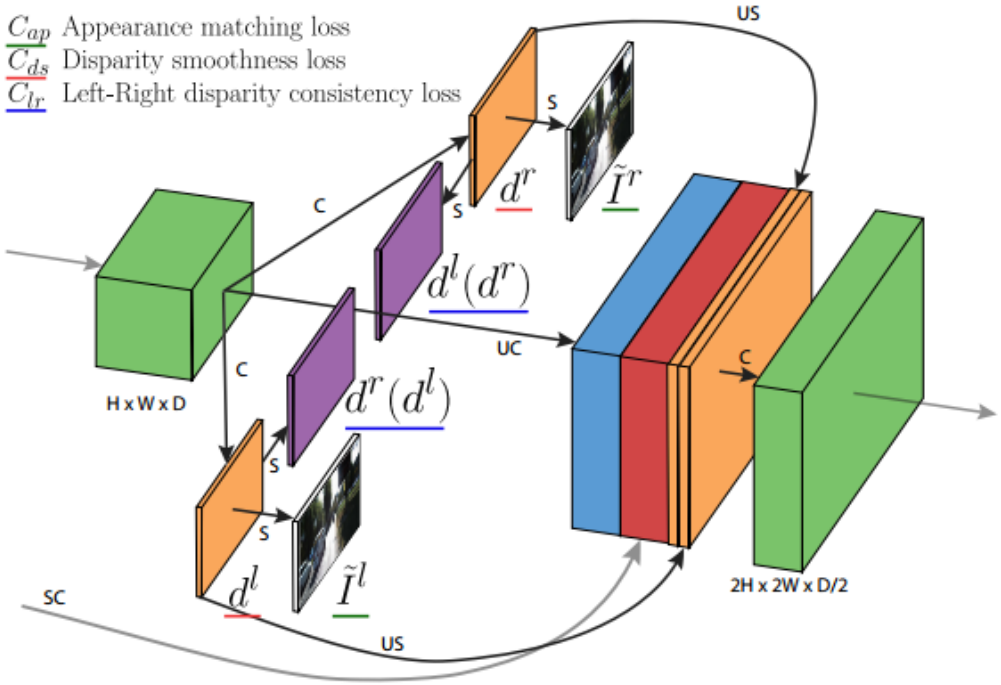
$$C^l_{lr} = \frac{1}{N}\sum_{i,j} \left| d^l_{ij} - d^r_{ij + d^l_{ij}} \right|. \quad (4)$$

left-right disparity consistency penalty

Unsupervised Monocular Depth Estimation with Left-Right Consistency

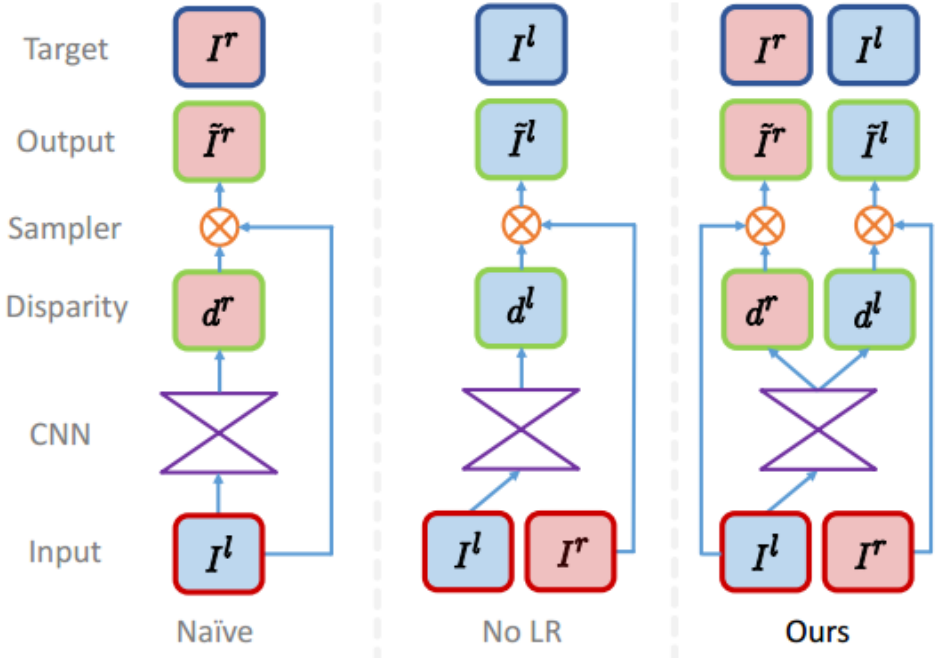
The loss module outputs left and right disparity maps, d_l and d_r. The loss combines smoothness, reconstruction, and left-right disparity consistency terms. This same module is repeated at each of the 4 different output scales.

C: Convolution
UC: Up-Convolution
S: Bilinear Sampling
US: Up-Sampling
SC: Skip Connection

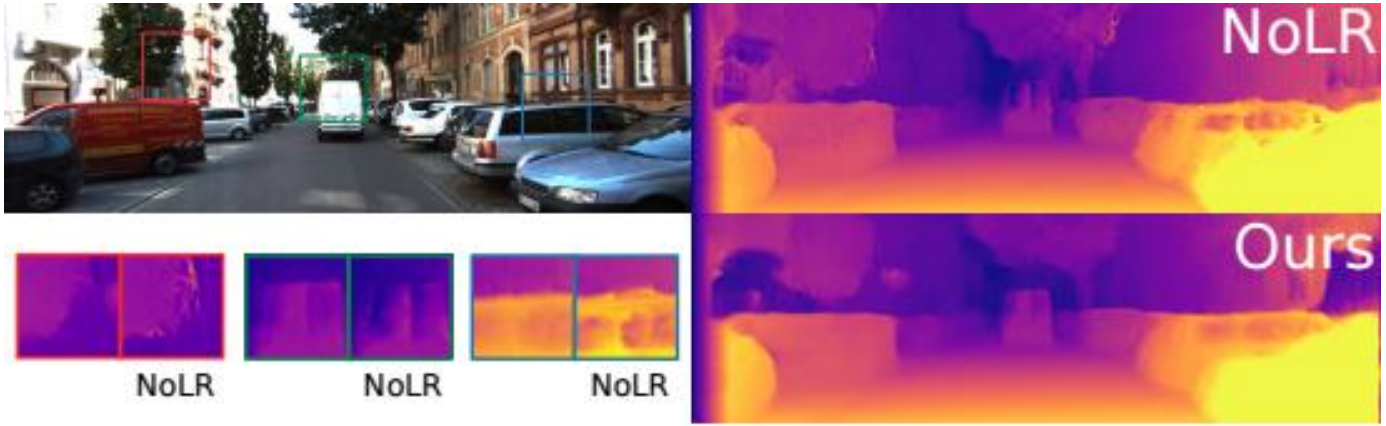


Unsupervised Monocular Depth Estimation with Left-Right Consistency

✓Sampling strategies for backward mapping.
✓ With naıve sampling the CNN produces a disparity map aligned with the target instead of the input.
✓No LR corrects for this, but suffers from artifacts.
✓Instead, this approach uses the left image to produce disparities for both images, improving quality by enforcing mutual consistency.



Unsupervised Monocular Depth Estimation with Left-Right Consistency

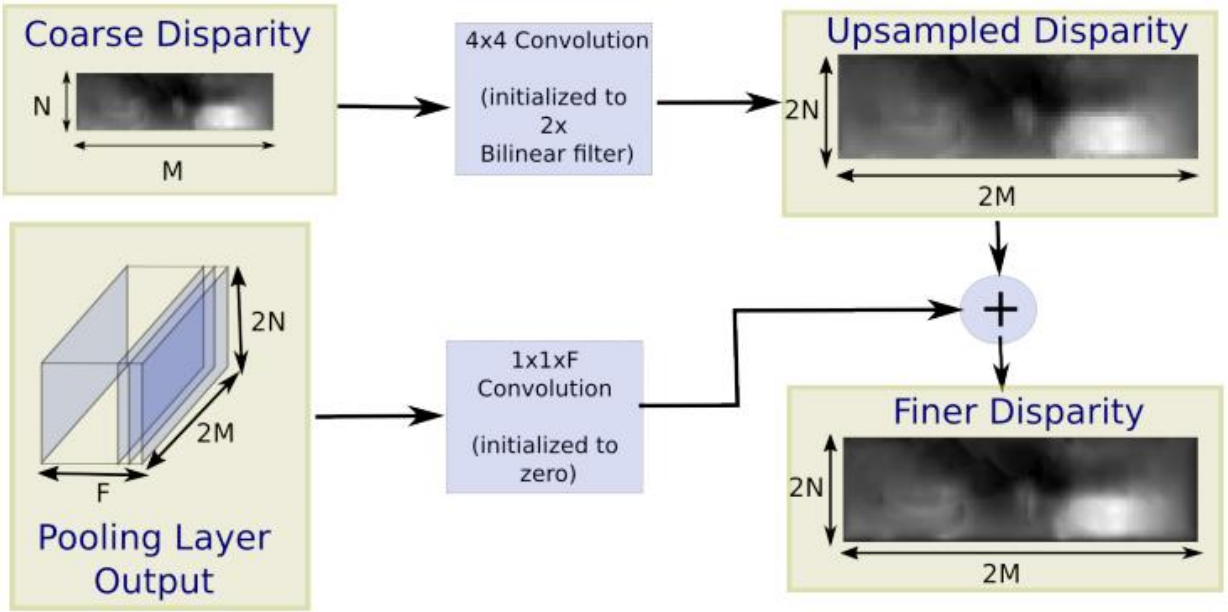


Comparison between with and without the left-right consistency. The consistency term produces better results on the object boundaries. (Note: Both without post-processing).

Unsupervised CNN for Single View Depth Estimation: Geometry to the Rescue

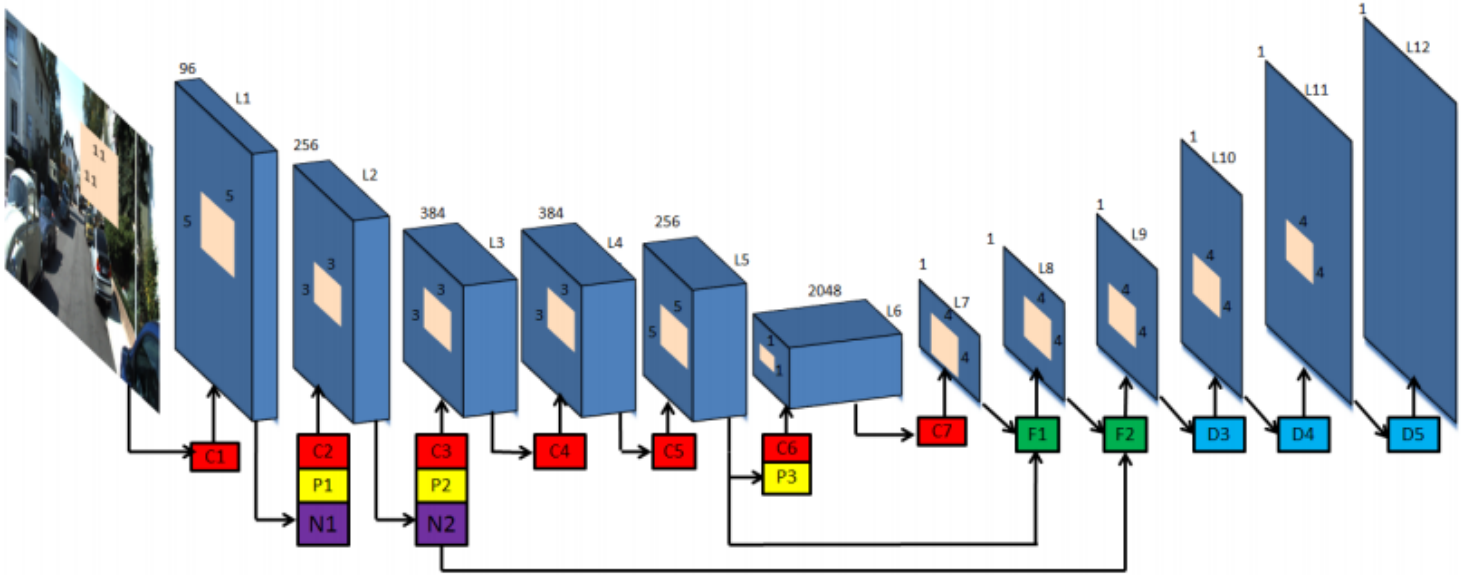
- A unsupervised framework to learn a deep convolutional neural network for single view depth prediction, without requiring a pre-training stage or annotated ground-truth depths.
- Training the network in a manner analogous to an autoencoder: At training time apply a pair of images, source and target, with small, known camera motion between the two such as a stereo pair.
- Train the convolutional encoder for the task of predicting the depth map for the source image.
- Explicitly generate an inverse warp of the target image using the predicted depth and known inter-view displacement, to reconstruct the source image; the photometric error in the reconstruction is the reconstruction loss for the encoder.
- The acquisition of this training data is considerably simpler than for equivalent systems, requiring no manual annotation, nor calibration of depth sensor to camera.

Unsupervised CNN for Single View Depth Estimation: Geometry to the Rescue



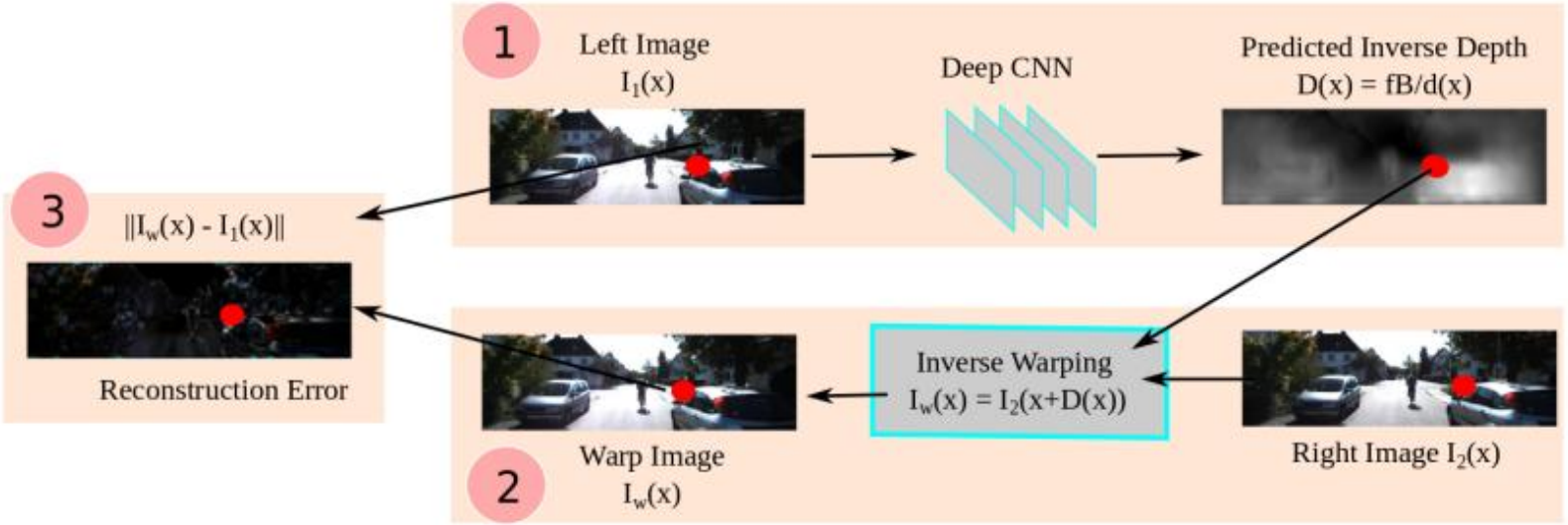
Coarse-to-fine stereo with CNN with results on a sample validation instance: a convolution based upsampling architecture proposed to mimic the coarse-to-fine stereo estimations.

Unsupervised CNN for Single View Depth Estimation: Geometry to the Rescue



Network architecture: The blocks C (red), P (yellow), L (dark blue), F (green), D (blue) correspond to convolution, pooling, local response normalization, FCN and upsampling layers respectively. The FCN blocks F1 and F2 upsample the predictions from layers (L7, L8) and combine it with the input of the pooling layers P3 and P2 respectively.

Unsupervised CNN for Single View Depth Estimation: Geometry to the Rescue

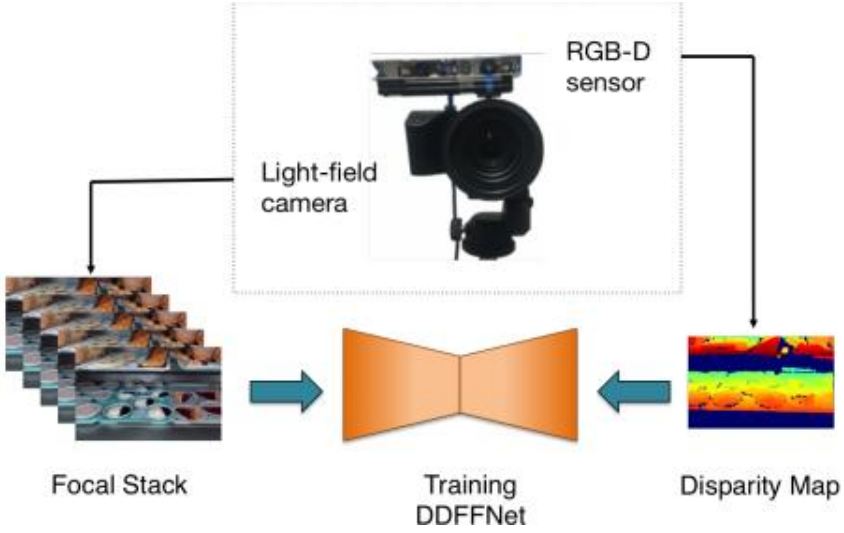


A stereopsis based auto-encoder setup: the encoder is a traditional convolutional neural network with stacked convolutions and pooling layers and maps the left image of the rectified stereo pair into its depth map. The decoder explicitly forces the encoder output to be disparities (scaled inverse depth) by synthesizing a backward warp image by moving pixels from right image along the scan-line.

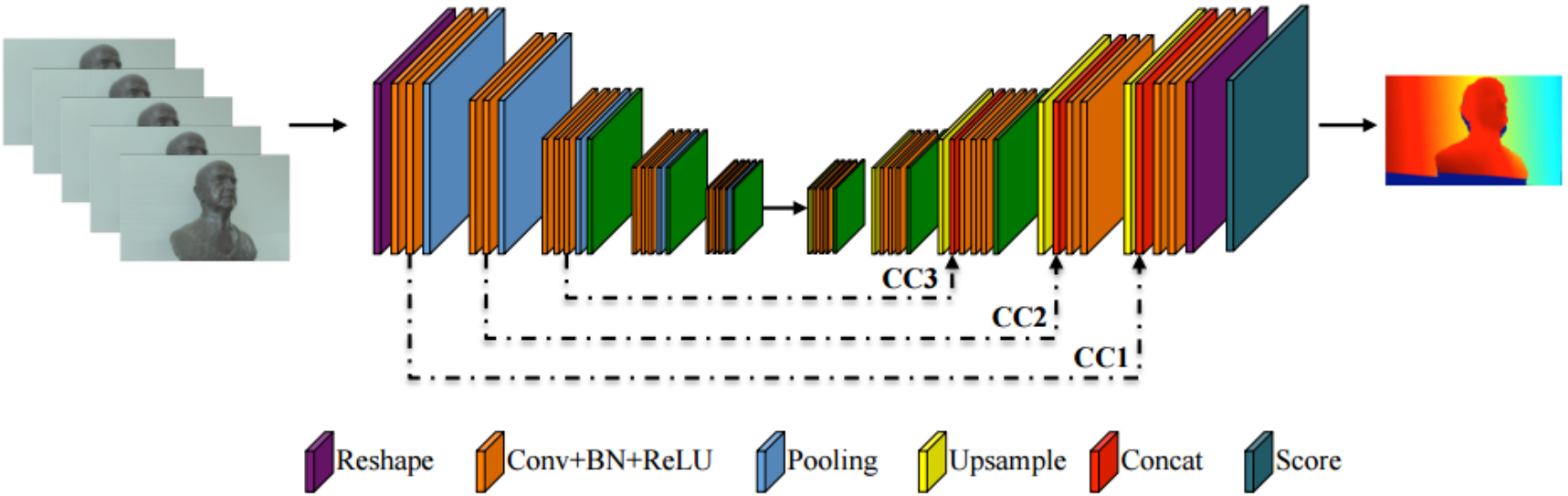
Deep Depth From Focus

- ‘Deep Depth From Focus (DDFF)’ as the end-to-end learning approach.
- A real-scene indoor benchmark composed of 4D light-field images obtained from a plenoptic camera and ground truth depth obtained from a registered RGB-D sensor.

Experimental setup. RGB-D sensor on top of a plenoptic camera in order to capture calibrated ground truth depth maps and light-field images from which to create focal stacks. These two inputs will be used to train DDFNet.



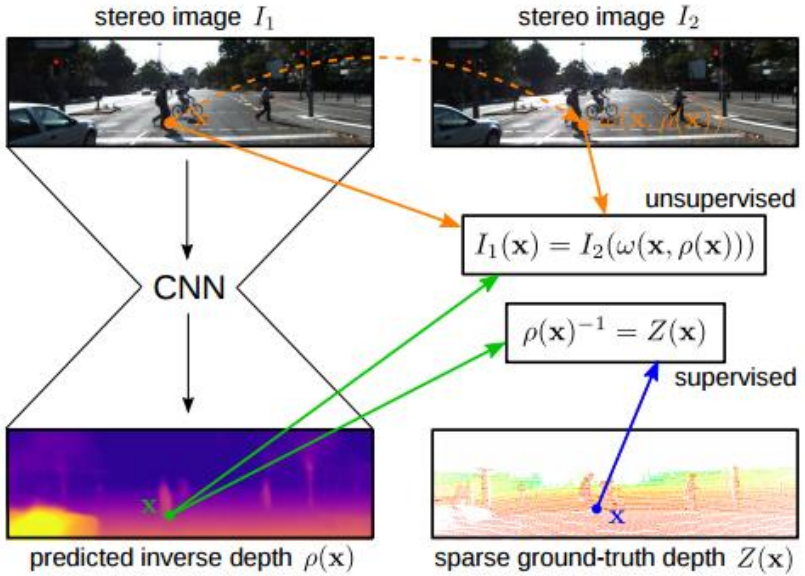
Deep Depth From Focus



DDFFNet. Proposed encoder-style architecture that takes in a focal stack and produces a disparity map. Here several architectural modifications, namely CC connections, Upsample, i.e. Unpool, BL and UpConv.

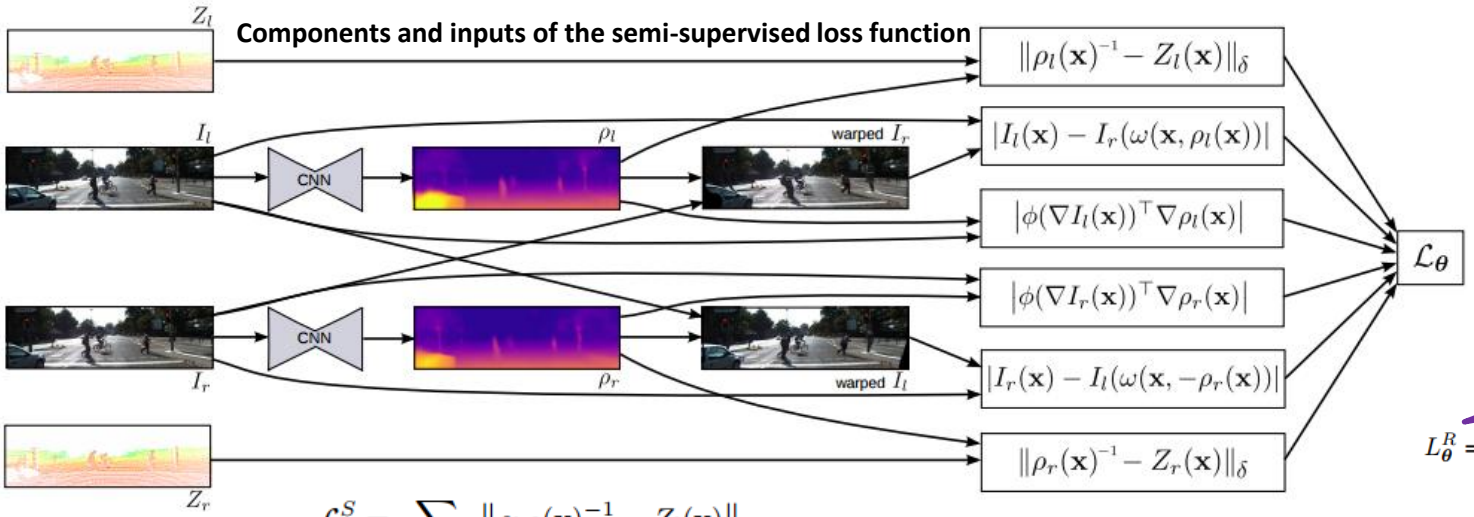
Semi-Supervised Deep Learning for Monocular Depth Map Prediction

- In the context of monocular depth map prediction, it is barely possible to determine dense ground truth depth images in realistic dynamic outdoor environments.
- When using LiDAR sensors, noise is present in the distance measurements, calibration btw sensors cannot be perfect, and measurements are typically much sparser than the camera images.
- An approach to depth map prediction from monocular images that learns in a semi-supervised way.
- While using sparse ground-truth depth for supervised learning, also enforce the deep network to produce photo-consistent dense depth maps in a stereo setup using a direct image alignment loss.



Concurrently train a CNN from unsupervised and supervised depth cues to achieve state-of-the-art performance in single image depth prediction.

Semi-Supervised Deep Learning for Monocular Depth Map Prediction



Regularization loss

$$L^R_{\boldsymbol{\theta}} = \sum_{i \in \{l,r\}} \sum_{\mathbf{x} \in \Omega} \left| \phi(\nabla I_i(\mathbf{x}))^\top \nabla \rho_i(\mathbf{x}) \right|$$

Supervised loss

$$\mathcal{L}^S_{\boldsymbol{\theta}} = \sum_{\mathbf{x} \in \Omega_{Z,l}} \left\| \rho_{l,\boldsymbol{\theta}}(\mathbf{x})^{-1} - Z_l(\mathbf{x}) \right\|_\delta + \sum_{\mathbf{x} \in \Omega_{Z,r}} \left\| \rho_{r,\boldsymbol{\theta}}(\mathbf{x})^{-1} - Z_r(\mathbf{x}) \right\|_\delta$$

Unsupervised loss

$$\mathcal{L}^U_{\boldsymbol{\theta}} = \sum_{\mathbf{x} \in \Omega_{U,l}} |(\mathbf{G}_\sigma * I_l)(\mathbf{x}) - (\mathbf{G}_\sigma * I_r)(\omega(\mathbf{x}, \rho_{l,\boldsymbol{\theta}}(\mathbf{x})))| + \sum_{\mathbf{x} \in \Omega_{U,r}} |(\mathbf{G}_\sigma * I_r)(\mathbf{x}) - (\mathbf{G}_\sigma * I_l)(\omega(\mathbf{x}, -\rho_{r,\boldsymbol{\theta}}(\mathbf{x})))|,$$

Semi-Supervised Deep Learning for Monocular Depth Map Prediction

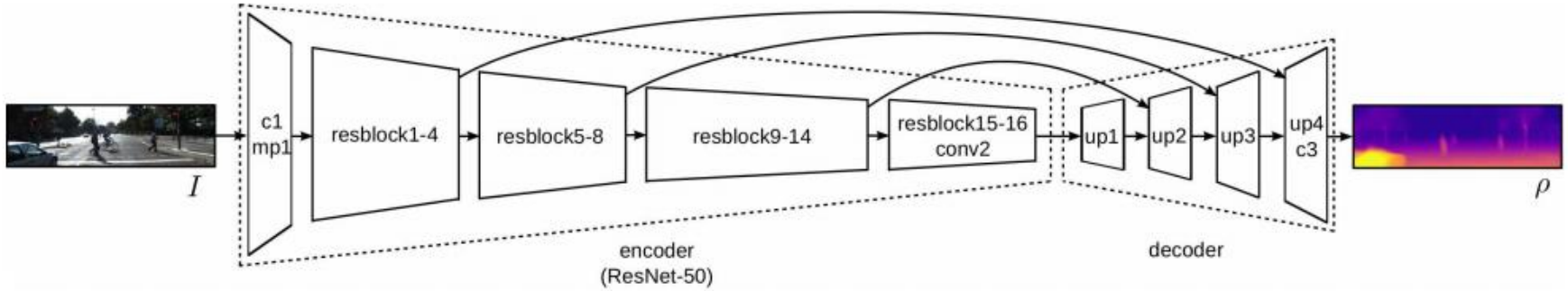
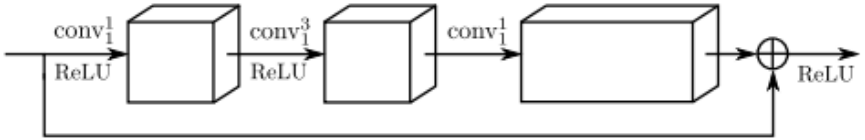
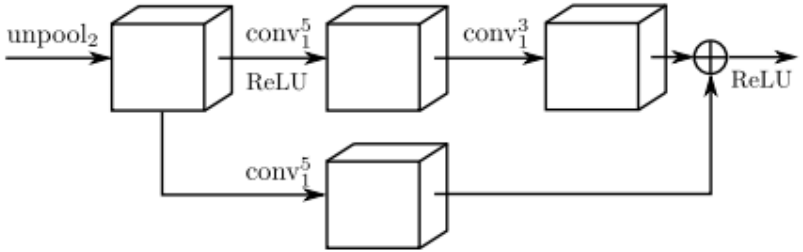


Illustration of the deep residual encoder-decoder architecture (c1, c3, mp1 abbreviate conv1, conv3, and max pool1, respectively). Skip connections from corresponding encoder layers to the decoder facilitate fine detailed depth map prediction.

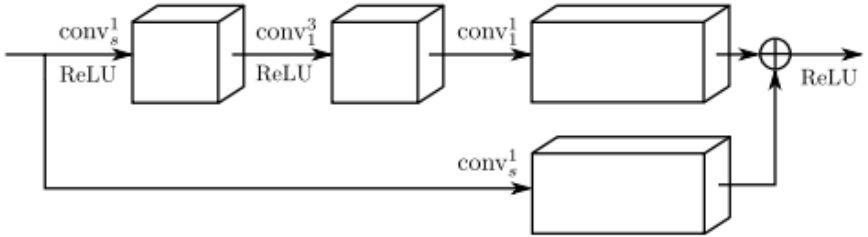
Semi-Supervised Deep Learning for Monocular Depth Map Prediction



Type 1 residual block resblock^1_s with stride s = 1. The residual is obtained from 3 successive convolutions. The residual has the same number of channels as the input.



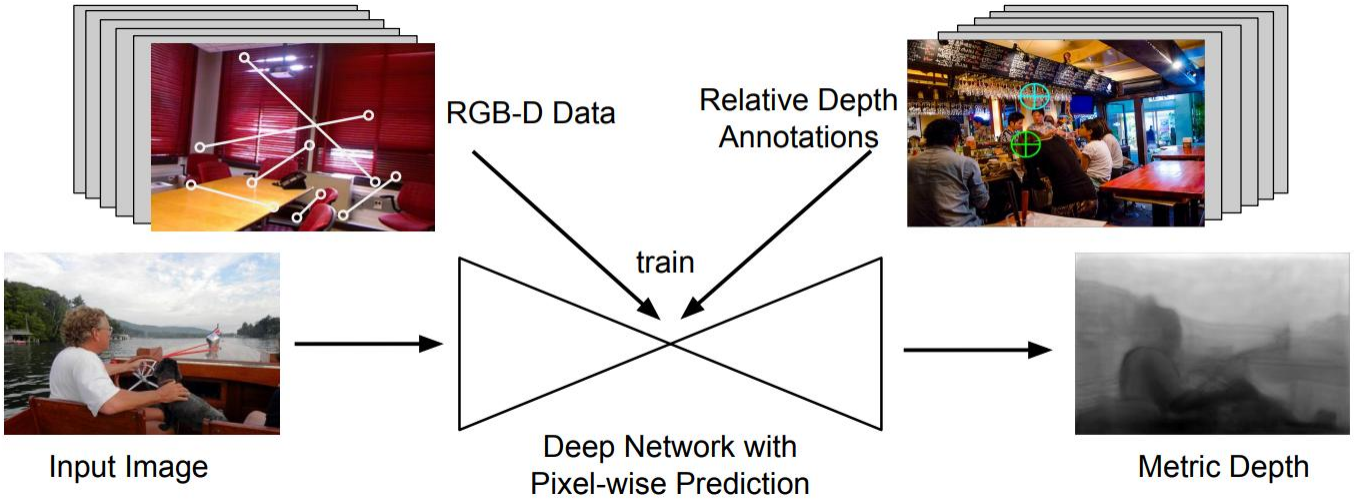
Schematic illustration of the upprojection residual block. It unpools the input by a factor of 2 and applies a residual block which reduces number of channels by a factor of 2.



Type 2 residual block resblock^2_s with stride s. The residual is obtained from 3 successive convolutions, while the 1st convolution applies stride s. An additional convolution applies the same stride s and projects the input to the number of channels of the residual.

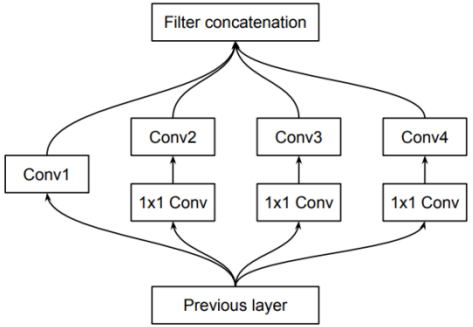
Single-Image Depth Perception in the Wild

- Recover depth from a single image taken in unconstrained settings.
- Dataset “Depth in the Wild”: images in the wild annotated with relative depth between pairs of random points.
- Learning to estimate depth using annotations of relative depth.



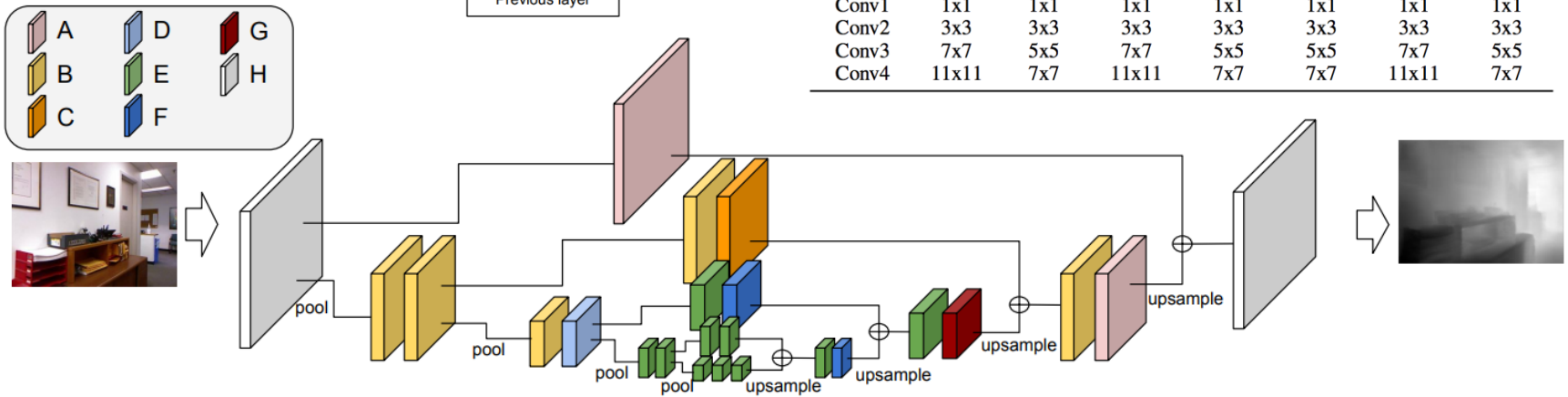
Single-Image Depth Perception in the Wild

Variant of Inception Module used below.



Parameters for each type of layer in NN. Conv1 to Conv4 are sizes of the filters used in the components of Inception module. Conv2 to 4 share the same number of input and is specified in Inter Dim.

Block Id	A	B	C	D	E	F	G
#In/#Out	128/64	128/128	128/128	128/256	256/256	256/256	256/128
Inter Dim	64	32	64	32	32	64	32
Conv1	1x1	1x1	1x1	1x1	1x1	1x1	1x1
Conv2	3x3	3x3	3x3	3x3	3x3	3x3	3x3
Conv3	7x7	5x5	7x7	5x5	5x5	7x7	5x5
Conv4	11x11	7x7	11x11	7x7	7x7	11x11	7x7

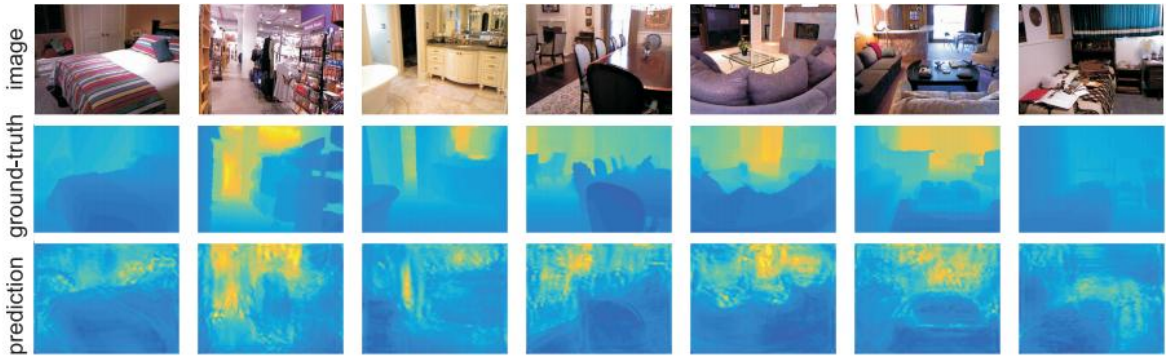


Network design. Each block represents a layer. Blocks sharing the same color are identical. The ⊕ sign denotes the element-wise addition. Block H is a convolution with 3x3 filter. All other blocks denote the **Inception** module.

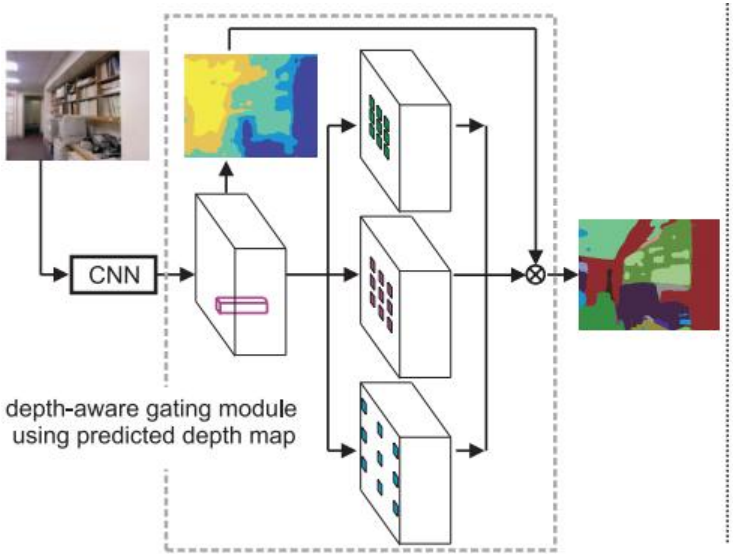
Single-Image Depth Perception in the Wild

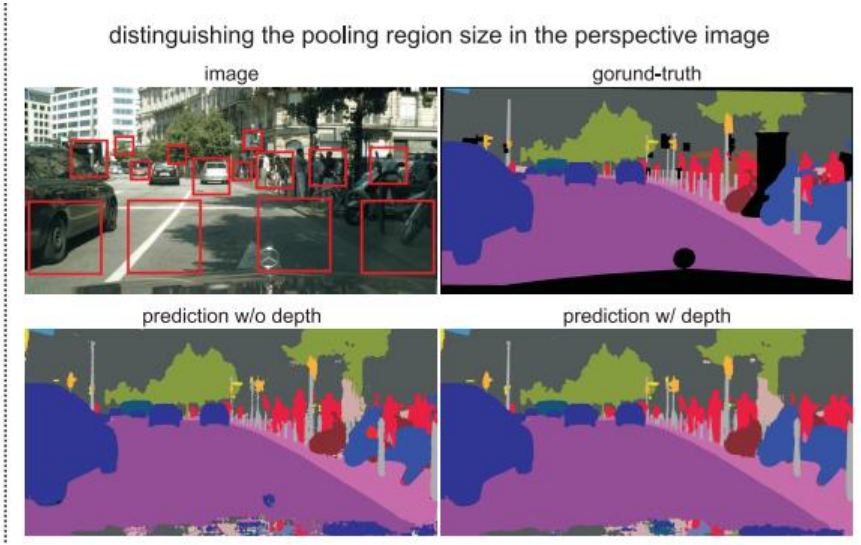
Recurrent Scene Parsing with Perspective Understanding in the Loop

- A **depth-aware gating module** that adaptively chooses the pooling field size in a convol. net architecture according to the object scale so that small details can be preserved for objects at distance and a larger receptive field can be used for objects nearer to the camera.
- The depth gating signal is provided from stereo disparity or directly from a single image.
- A recurrent CNN trained in an end-to-end fashion to perform **semantic segmentation**.
- The recurrent module iteratively refines the segmentation results, leveraging the depth estimate and output prediction from the previous loop.



Recurrent Scene Parsing with Perspective Understanding in the Loop

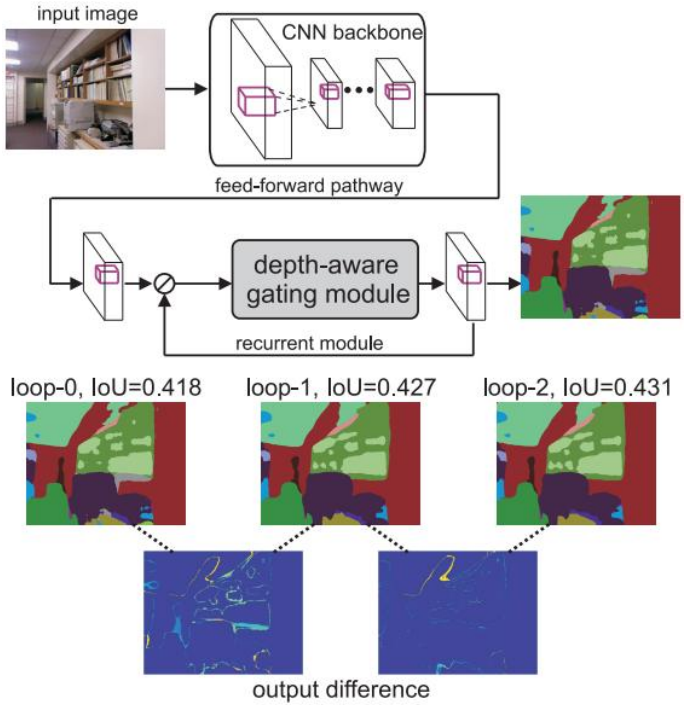




Left: depth-aware gating module **using the predicted depth map**. Right: an example image with ground-truth, segmentation result w/o the depth gating module. Rectangles overlayed on the image indicate pooling field sizes which are adapted based on the local depth estimate. Using depth-gated pooling yields more accurate segment label predictions by avoiding pooling across small multiple distant objects.

Recurrent Scene Parsing with Perspective Understanding in the Loop

The input to our recurrent module is the concatenation of the feature map from an intermediate layer of the feed-forward pathway with the prior recurrent prediction. The recurrent module utilizes depth-aware gating which carries out both depth regression and quantized prediction. Updated depth predictions at each iteration gate pooling fields used for semantic segmentation. This recurrent update of depth estimation increases the flexibility and representation power of the system yielding improved segmentation. It illustrates the prediction prior to, and after two recurrent iterations for a particular image. It visualizes the difference in predictions between consecutive iterations and note the gains in accuracy at each iteration as measured by average intersection-over-union (IoU) benchmark performance.



PoseNet: A Convolutional Network for Real-Time 6-DOF Camera Relocalization

- **PoseNet**: A robust and real-time monocular 6 DOF relocalization system.
- It trains a CNN to regress 6-DOF camera pose from a RGB image in an end-to-end manner with no need of additional engineering or graph optimisation.
- The algorithm can operate indoors and outdoors in real time, 5ms per frame.
- This is achieved by an efficient 23 layer deep convnet, image plane regression.
- It localizes from high level features and is robust to difficult lighting, motion blur and different camera intrinsics where point based SIFT registration fails.
- The pose feature that is produced generalizes to other scenes allowing to regress pose with only a few dozen training examples.

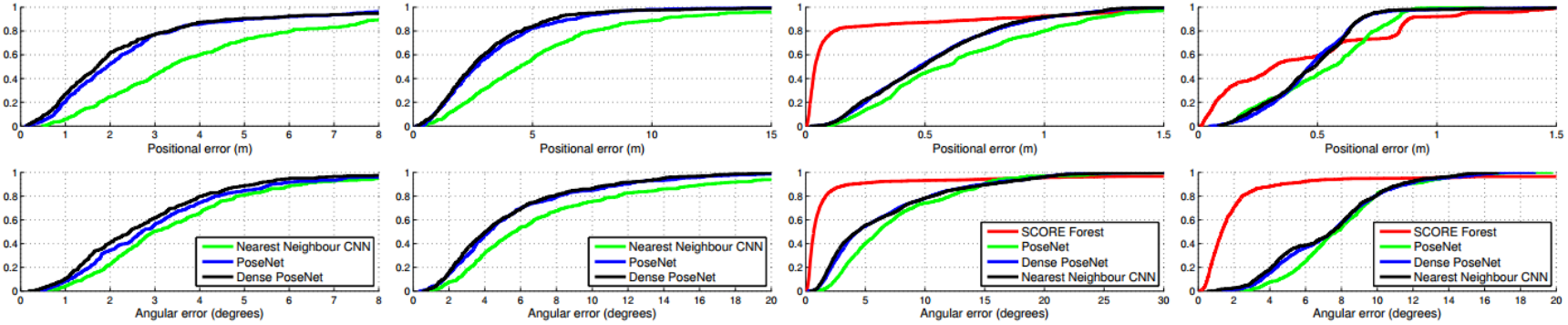
PoseNet: A Convolutional Network for Real-Time 6-DOF Camera Relocalization

- To regress pose, train the convnet on Euclidean loss using stochastic gradient descent with the following objective loss function:

$$loss(I) = \|\hat{\mathbf{x}} - \mathbf{x}\|_2 + \beta \left\| \hat{\mathbf{q}} - \frac{\mathbf{q}}{\|\mathbf{q}\|} \right\|_2$$

  where β is a scale factor chosen to keep the expected value of position and orientation errors to be approximately equal.
- Use GoogLeNet as a basis for developing the pose regression network.
  - Replace all 3 softmax classifiers with affine regressors.
  - The softmax layers were removed and each final FCL was modified to output a pose vector of 7-dimensions representing position (3) and orientation (4).
  - Insert another FCL before the final regressor of feature size 2048.
  - This was to form a localization feature vector which may then be explored for generalisation.
  - At test time also normalize the quaternion orientation vector to unit length.

PoseNet: A Convolutional Network for Real-Time 6-DOF Camera Relocalization

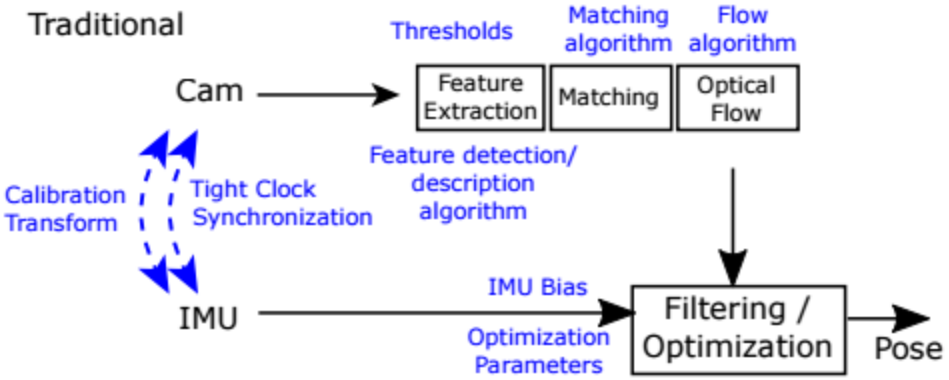


(a) King's College (b) St Mary's Church (c) Pumpkin (d) Stairs

Localization performance for both position and orientation as a cumulative histogram of errors. The regression convnet outperforms the NN feature matching. Comparing to the RGB-D SCoRe Forest approach shows that this CNN method is competitive, but outperformed by a more expensive depth approach. The method does perform better on the hardest few frames, above the 95th percentile, with the worst error lower than the worst error from the SCoRe approach.

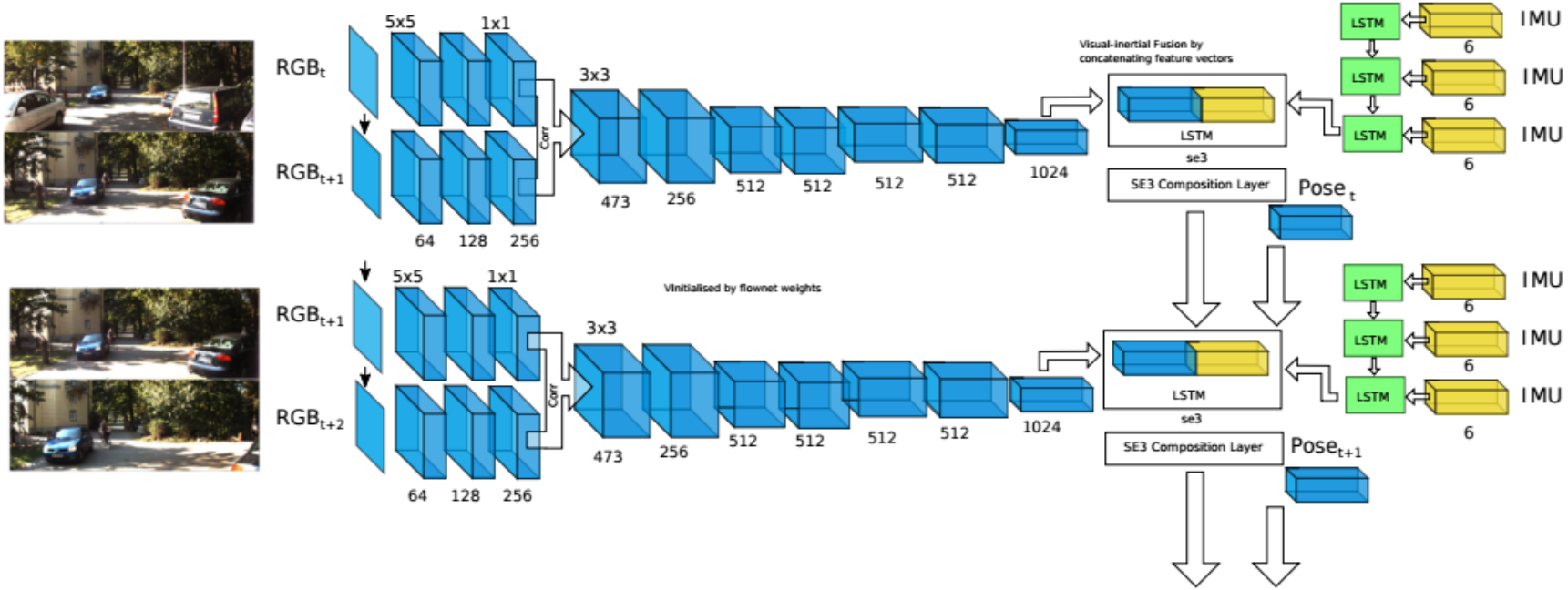
VINet: Visual-Inertial Odometry as a Sequence-to-Sequence Learning Problem

- On-manifold sequence-to-sequence learning approach to motion estimation using visual and inertial sensors.



Comparison btw a standard visual-inertial odometry framework and the learning-based approach.

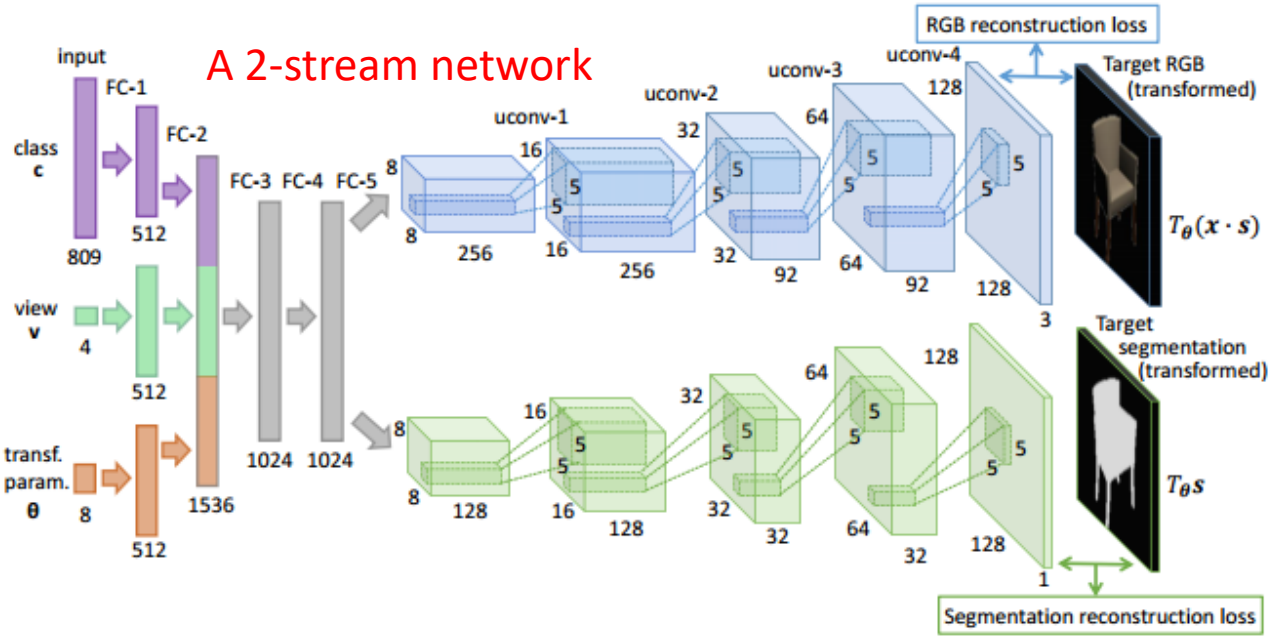
VINet: Visual-Inertial Odometry as a Sequence-to-Sequence Learning Problem



VINet architecture for visual-inertial odometry. LSTM processing the pose output at camera-rate and an IMU LSTM processing data at the IMU rate.

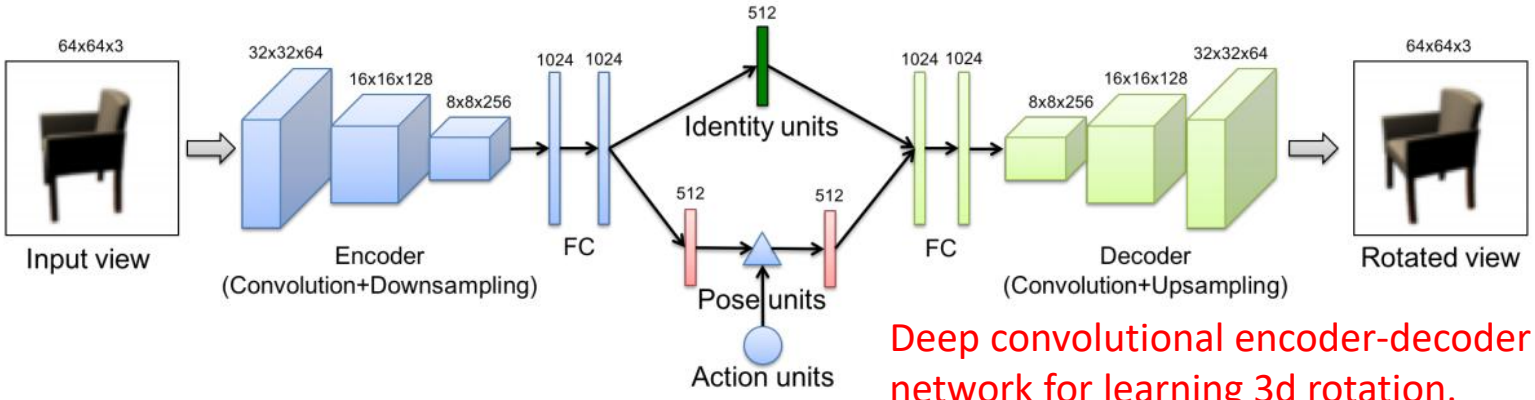
Learning to Generate Chairs, Tables and Cars with Convolutional Networks

- Train generative 'up-convolutional' neural networks which are able to generate images of objects given object style, viewpoint, and color.



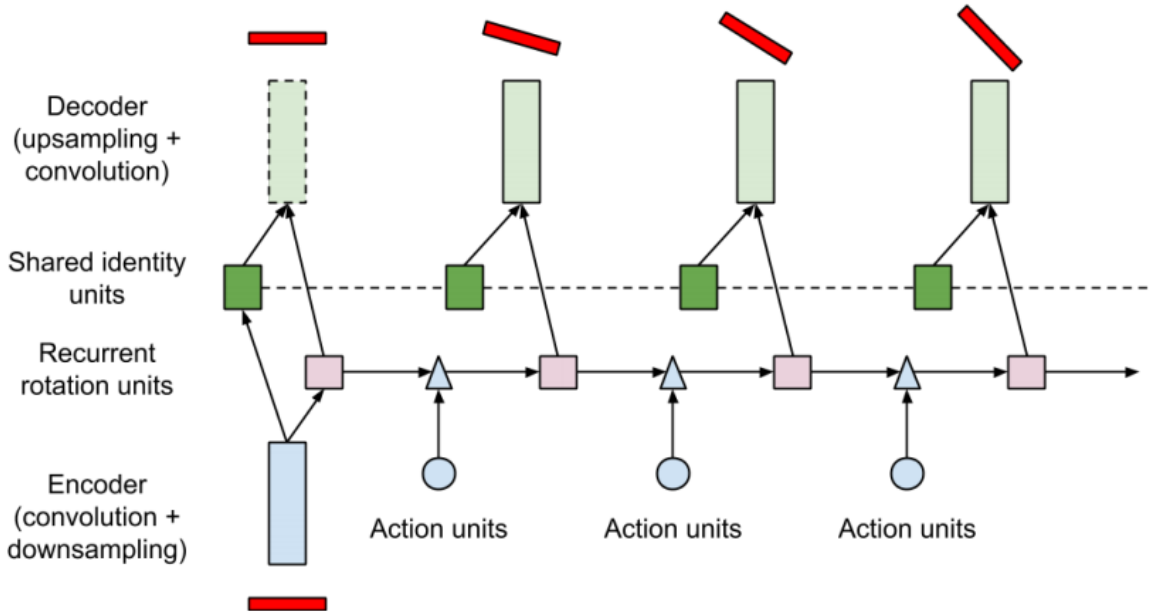
Weakly-supervised Disentangling with Recurrent Transformations for 3D View Synthesis

- Train a neural network when restricting the attention to specific object categories for which to gather ample training data.
- A recurrent convolutional encoder-decoder network that is trained end-to-end on the task of rendering rotated objects starting from a single image.
- The recurrent structure allows the model to capture long-term dependencies along a sequence of transformations.



Deep convolutional encoder-decoder network for learning 3d rotation.

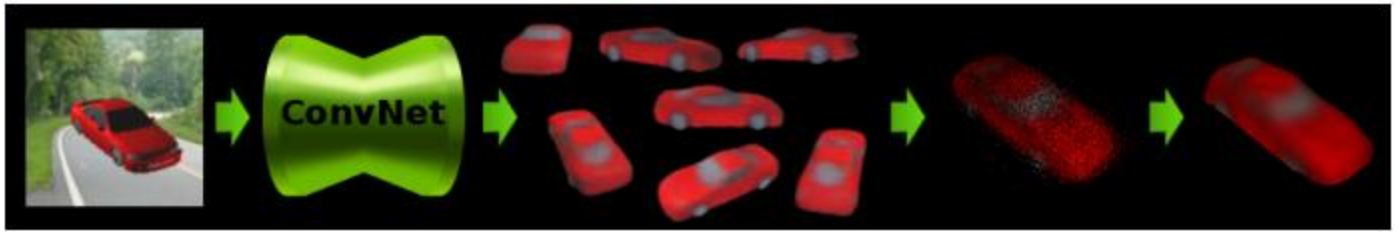
Weakly-supervised Disentangling with Recurrent Transformations for 3D View Synthesis



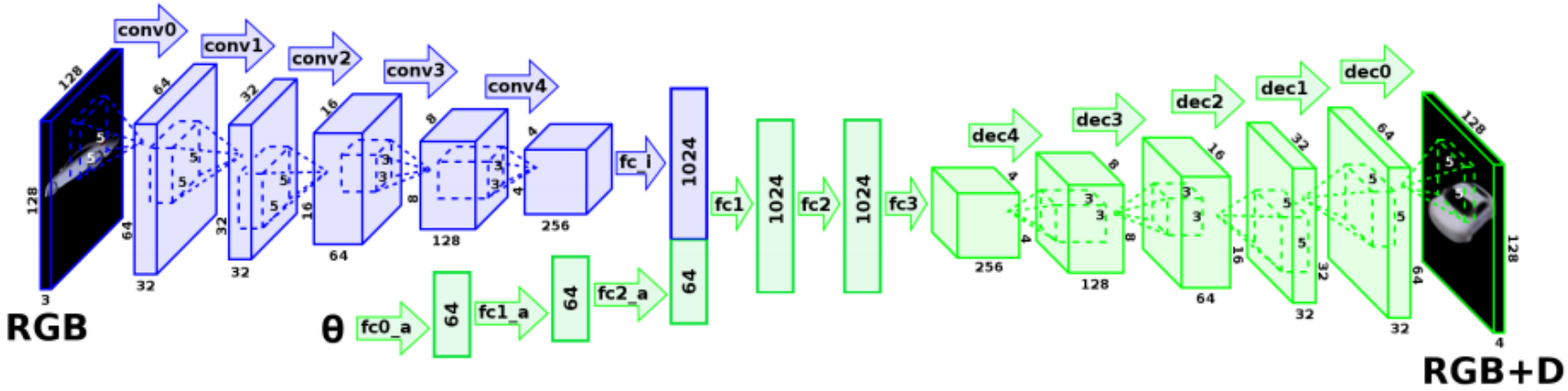
Unrolled recurrent convolutional network for learning to rotate 3d objects. The convolutional encoder and decoder have been abstracted out, represented here as vertical rectangles.

Multi-view 3D Models from Single Images with a Convolutional Network

- A convolutional network capable of inferring a 3D representation of a previously unseen object given a single image of this object.
- Concretely, the network can predict an RGB image and a depth map of the object as seen from an arbitrary view.
- Several of these depth maps fused together give a full point cloud of the object.
- The point cloud can in turn be transformed into a surface mesh.
- The network is trained on renderings of synthetic 3D models of cars and chairs.
- It successfully deals with objects on cluttered background and generates reasonable predictions for real images of cars.



Multi-view 3D Models from Single Images with a Convolutional Network

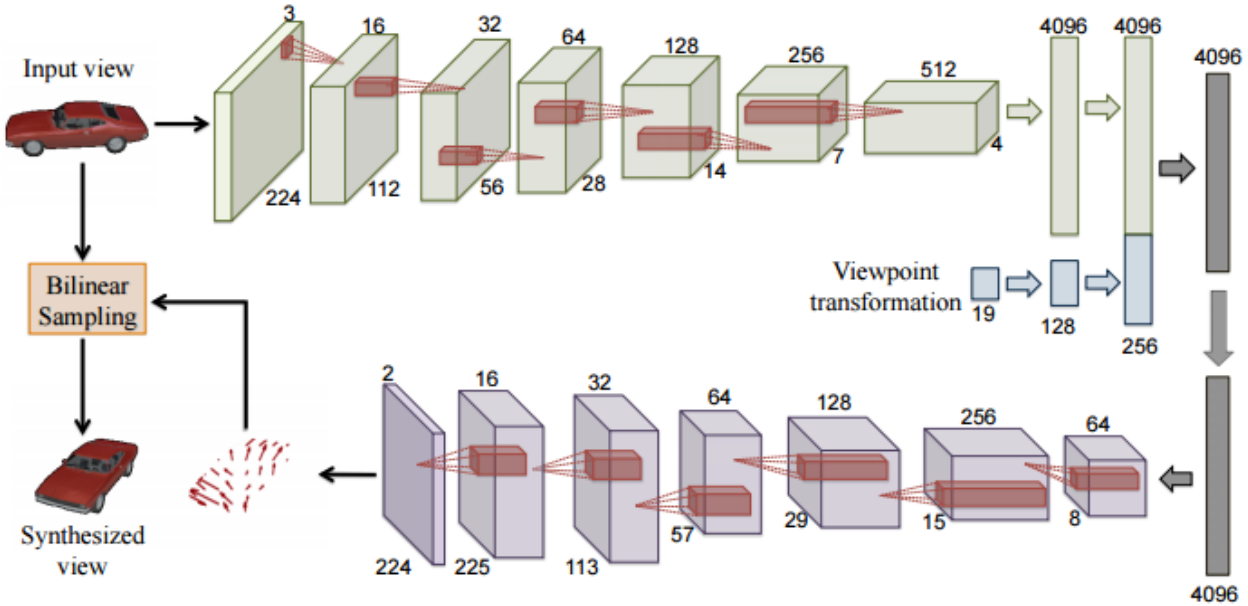


Architecture of the network. The encoder turns an input image into an abstract 3D representation. The decoder processes the angle, modifies the encoded hidden representation accordingly, and renders the final image together with the depth map.

View Synthesis by Appearance Flow

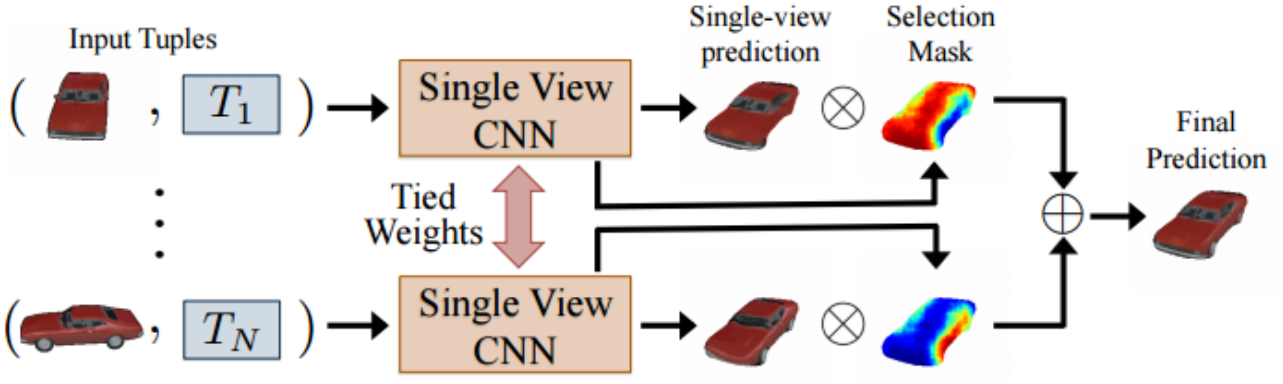
- Novel view synthesis: given an input image, synthesizing new images of the same object or scene observed from arbitrary viewpoints.
- A learning task, instead of learning to synthesize pixels from scratch, learn to copy them from the input image.
- It exploits the observation that the visual appearance of different views of the same instance is highly correlated, and such correlation could be explicitly learned by training a CNN to predict appearance flows – 2-D coordinate vectors specifying which pixels in the input view could be used to reconstruct the target view.
- Furthermore, the framework easily generalizes to multiple input views by learning how to optimally combine single-view predictions.

View Synthesis by Appearance Flow



Overview of the single-view network architecture. An encoder-decoder framework where the input view and the desired viewpoint transformation are first encoded via several convolution and fully-connected layers, and then a decoder consisting of two fully-connected and six up-sampling convolution layers outputs an appearance flow field, which in conjunction with the input view yields the synthesized view through a bilinear sampling layer. All the layer weights are learned end-to-end through backpropagation.

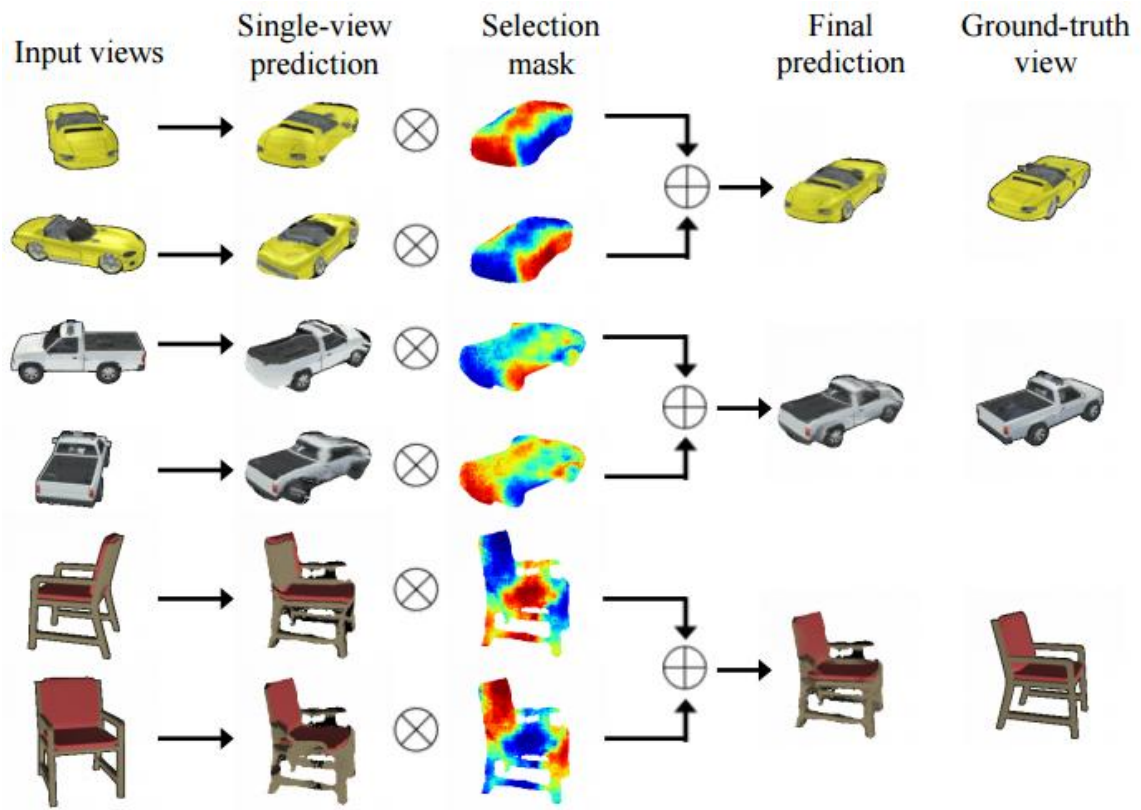
View Synthesis by Appearance Flow



Overview of the multi-view network architecture (⊗: per-pixel product, ⊕: per-pixel normalized sum). For each input view, use a single-view CNN with shared weights to independently predict the target view as well as a per-pixel selection/confidence mask. The final target view prediction is obtained by linearly combining the predictions from each view weighted by the selection masks.

View Synthesis by Appearance Flow

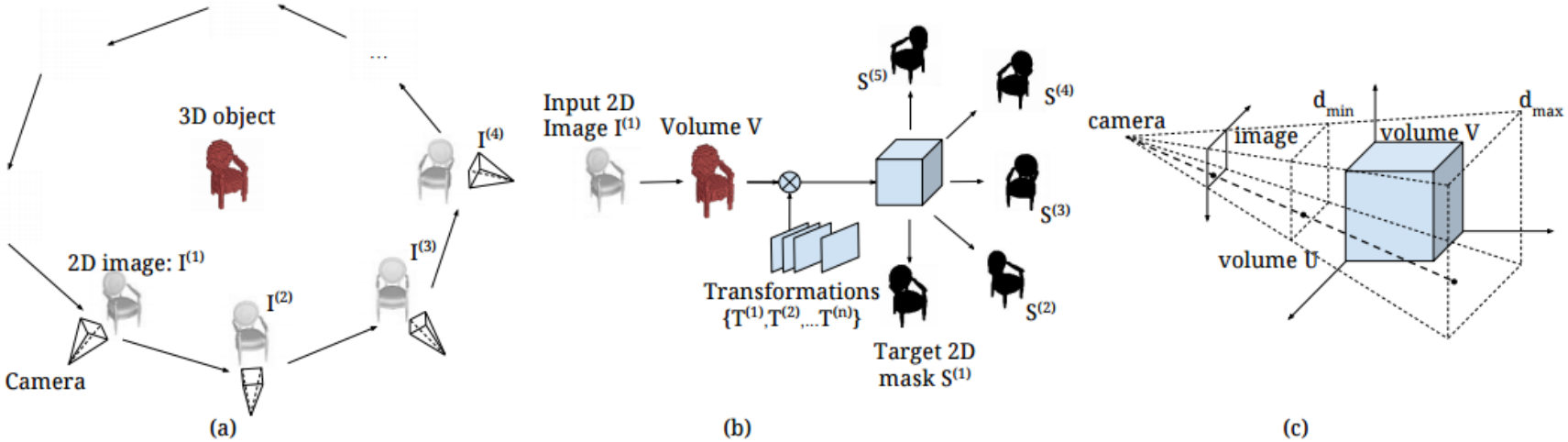
View synthesis examples using the multi-view network. Typically, the final prediction is more similar to the ground-truth than any independent prediction.



Perspective Transformer Nets: Learning Single-View 3D Object Reconstruction

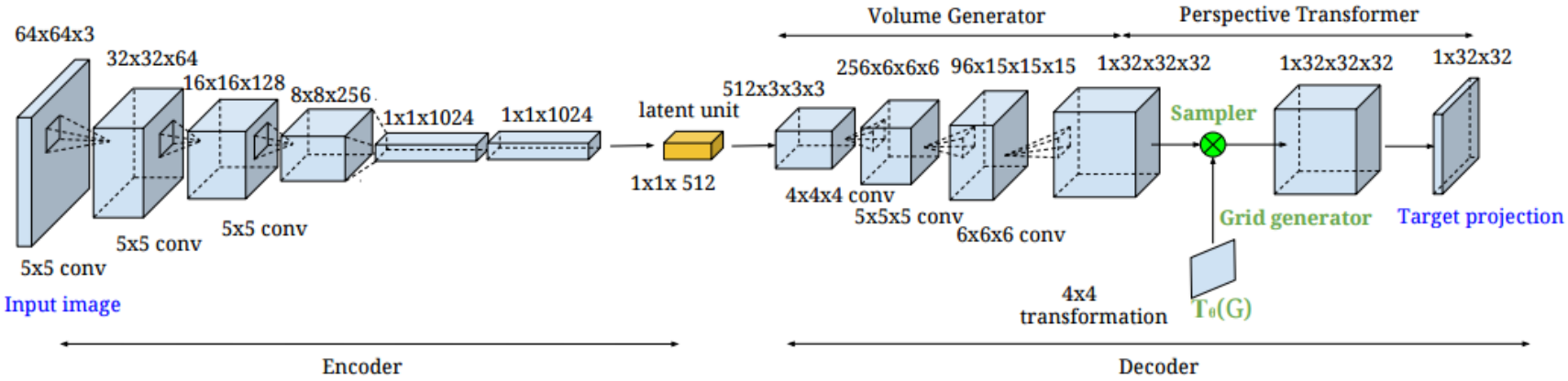
- Single-view 3D object reconstruction from a learning agent's perspective.
- Formulate the learning process as an interaction btw 3D and 2D representations and propose an encoder-decoder network with a novel projection loss defined by the perspective transformation.
- The projection loss enables the unsupervised learning using 2D observation without explicit 3D supervision.
- Generating 3D volume from a single 2D image: (1) learning from single-class objects; (2) learning from multi-class objects and (3) testing on novel object classes.

Perspective Transformer Nets: Learning Single-View 3D Object Reconstruction



(a) understanding 3D object from learning agent's perspective; (b) single-view 3D volume reconstruction with perspective transformation. (c) perspective projection.

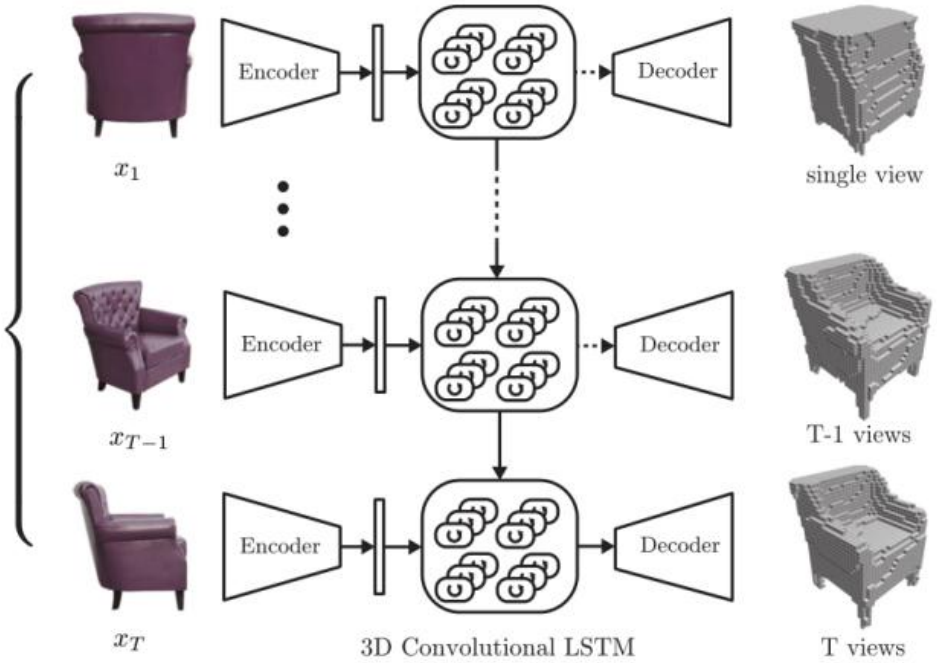
Perspective Transformer Nets: Learning Single-View 3D Object Reconstruction



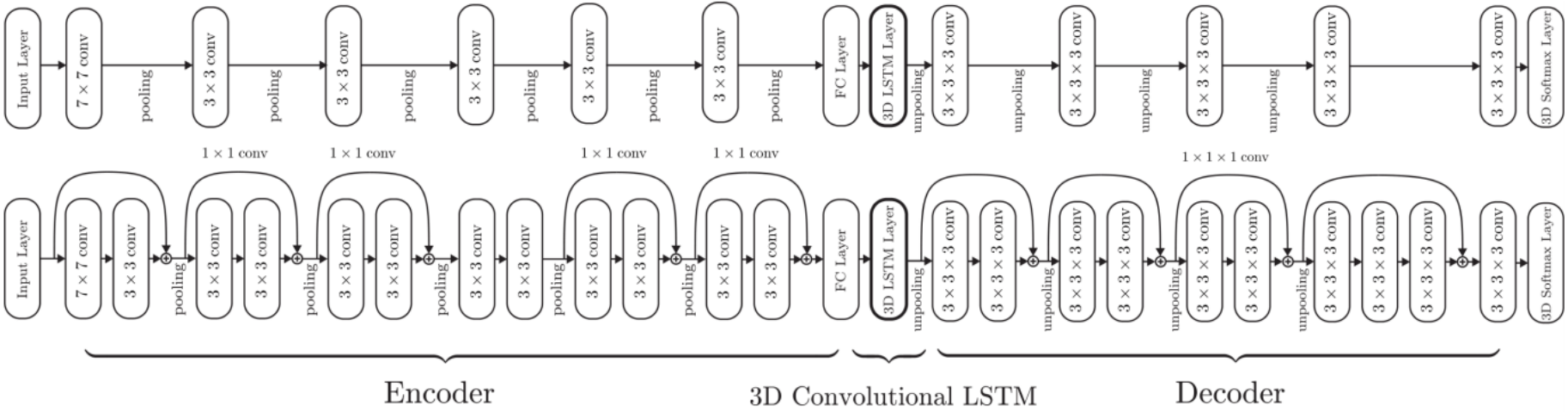
Network Architecture: a 2D convol. encoder, a 3D up-convol. decoder and a perspective transformer networks. The 2D convol. encoder consists of 3 convol. layers, followed by 3 FCLs(convol. layers have 64, 128 and 256 channels with fixed filter size of 5 × 5; the 3 FCLs have 1024, 1024 and 512 neurons, respectively). The 3D convol. decoder consists of one FCL, followed by 3 convol. layers (the FCL have 3 × 3 × 3 × 512 neurons; convol. layers have 256, 96 and 1 channels with filter size of 4 × 4 × 4, 5 × 5 × 5 and 6 × 6 × 6). For perspective transformer networks, used perspective transformation to project 3D volume to 2D silhouette where the transformation matrix is parametrized by 16 variables and sampling grid is set to 32 × 32 × 32.

3D-R2N2: A Unified Approach for Single and Multi-view 3D Object Reconstruction

- A NN architecture, 3D Recurrent Reconstruction Neural Network (3D-R2N2).
- The network learns a mapping from images of objects to their underlying 3D shapes from a large collection of synthetic data.
- The network takes in one or more images of an object instance from arbitrary viewpoints and outputs a reconstruction of the object in the form of a 3D occupancy grid.
- It does not require any image annotations or object class labels for training or testing.

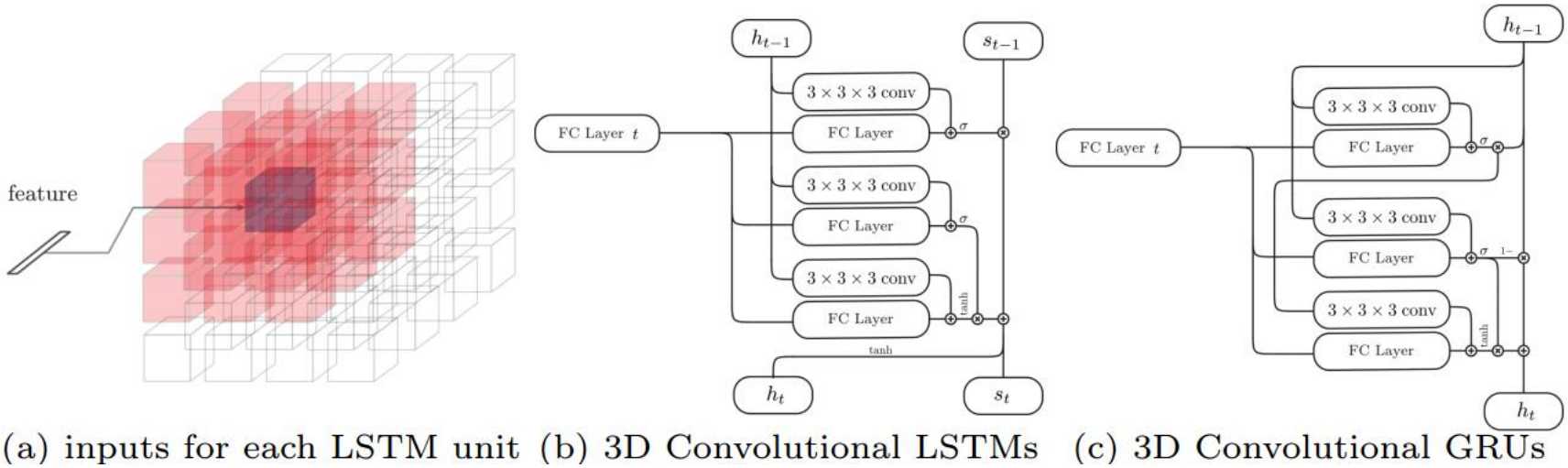


3D-R2N2: A Unified Approach for Single and Multi-view 3D Object Reconstruction



Network architecture: Each 3D-R2N2 consists of an encoder, a recurrence unit and a decoder. After every convolution layer, place a LeakyReLU nonlinearity. The encoder converts a 127 × 127 RGB image into a low-dimensional feature which is then fed into the 3D-LSTM. The decoder then takes the 3D-LSTM hidden states and transforms them to a final voxel occupancy map. After each convolution layer is a LeakyReLU. Two versions of 3D-R2N2: (top) a shallow network and (bottom) a deep residual network.

3D-R2N2: A Unified Approach for Single and Multi-view 3D Object Reconstruction



(a) inputs for each LSTM unit (b) 3D Convolutional LSTMs (c) 3D Convolutional GRUs

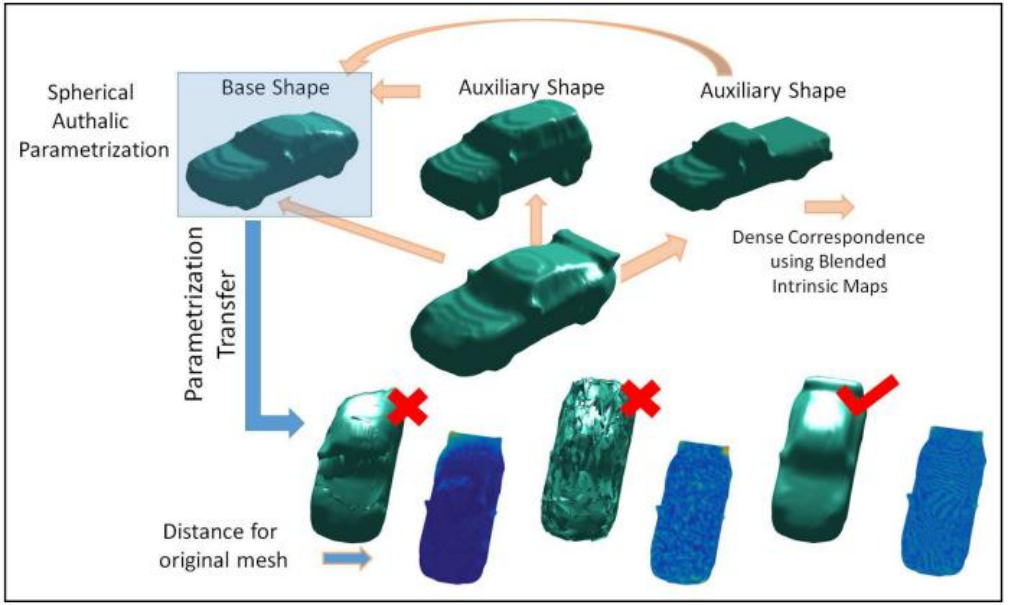
(a) At each time step, each unit in the 3D-LSTM receives the same feature vector from the encoder and the hidden states from its neighbors by a 3×3×3 convolution ($Ws * h_{t-1}$) as inputs. Two versions of 3D-LSTMs: (b) 3D-LSTMs without output gates and (c) 3D Gated Recurrent Units (GRUs).

SurfNet: Generating 3D shape surfaces using deep residual networks

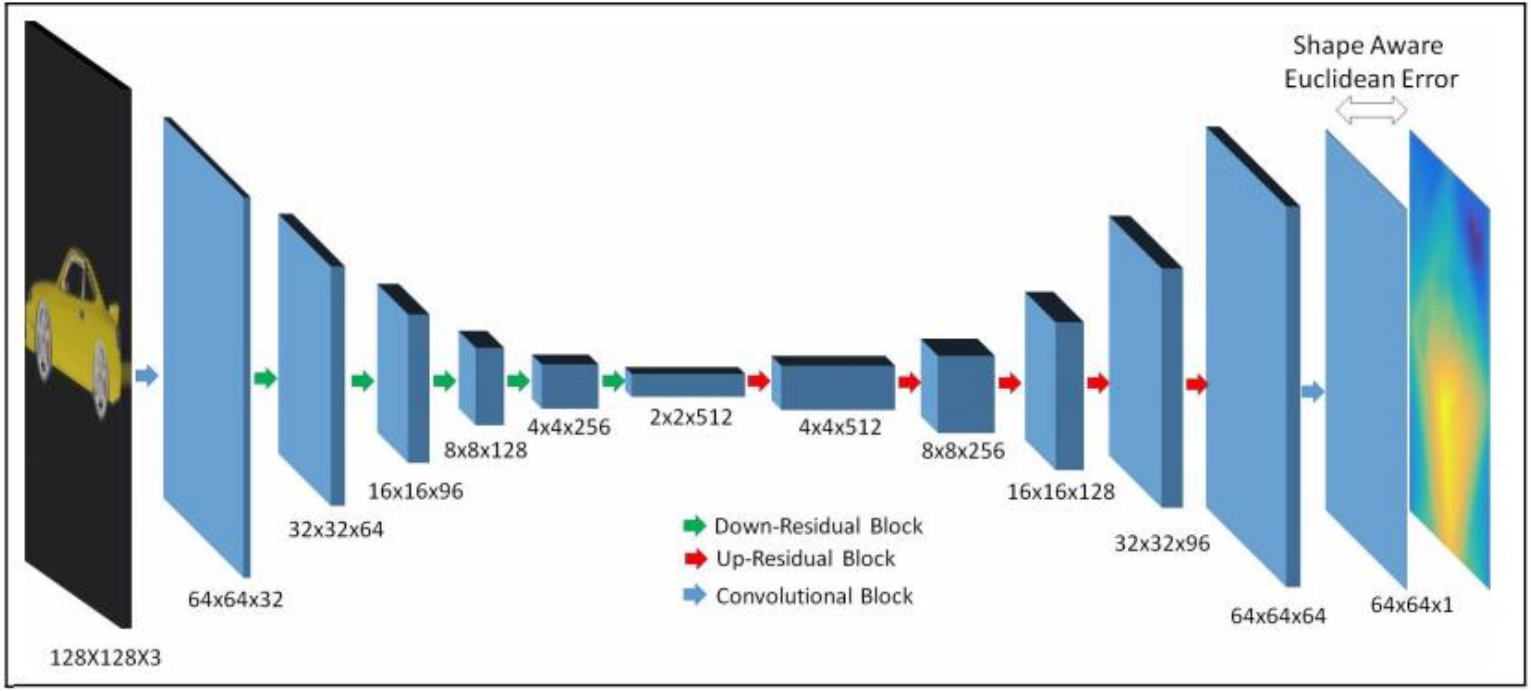
- Lifting convolution operators from the traditional 2D to 3D results in high computational overhead with little additional benefit as most of the geometry information is contained on the surface boundary.
- Generate the 3D shape surface of rigid/non-rigid shapes using deep CNNs.
- Create consistent ‘geometry images’ as the shape surface of a category of 3D objects, then use it for category-specific shape surface generation from a parametric representation or an image by developing novel extensions of deep residual networks for the task of geometry image generation.
- It learns a meaningful representation of shape surfaces allowing it to interpolate between shape orientations and poses, invent new shape surfaces and reconstruct 3D shape surfaces from previously unseen images.

SurfNet: Generating 3D shape surfaces using deep residual networks

Correspondences btw mesh model and exemplar shapes to create consistent geometry images. The geometry image created using correspondences btw central mesh model and axillary shape on top right shows best surface reconstruction lower than a threshold error, and subsequently used for training the neural network.

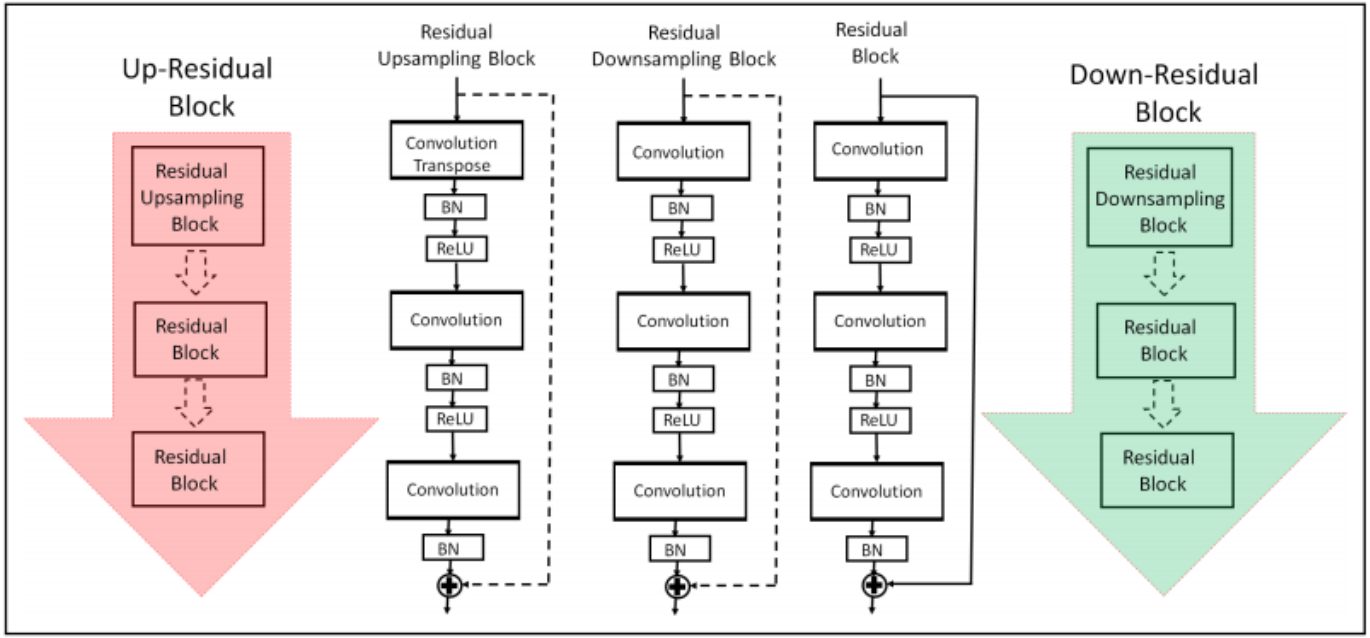


SurfNet: Generating 3D shape surfaces using deep residual networks



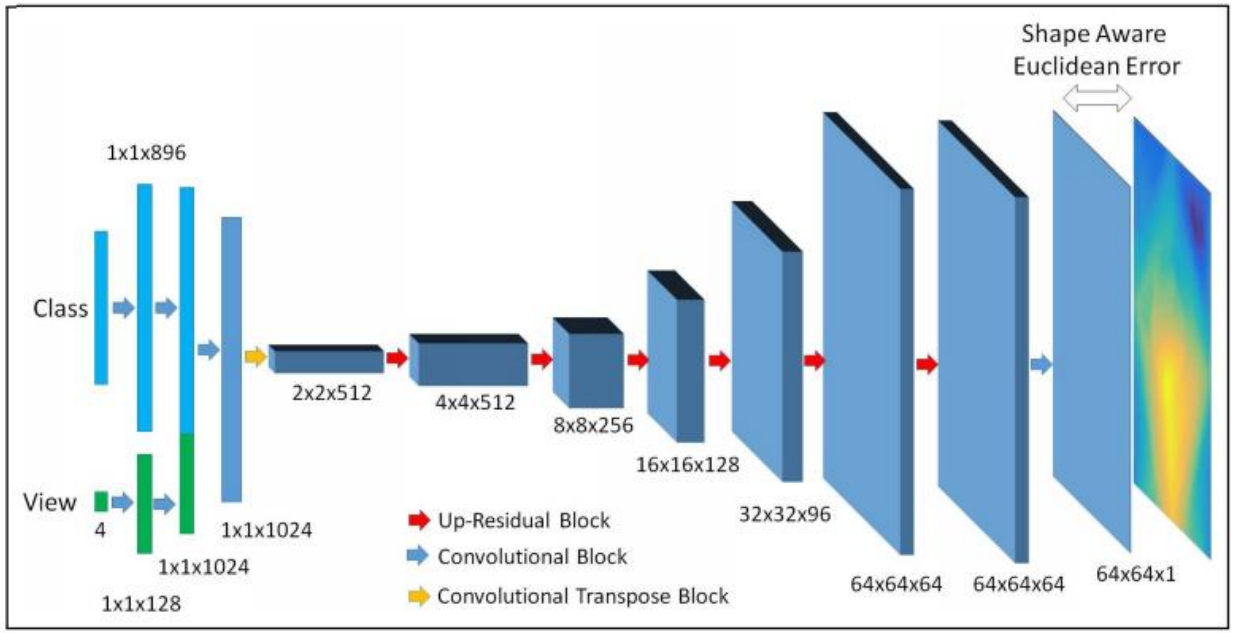
Architecture for generating a geometry image feature channel from an image.

SurfNet: Generating 3D shape surfaces using deep residual networks



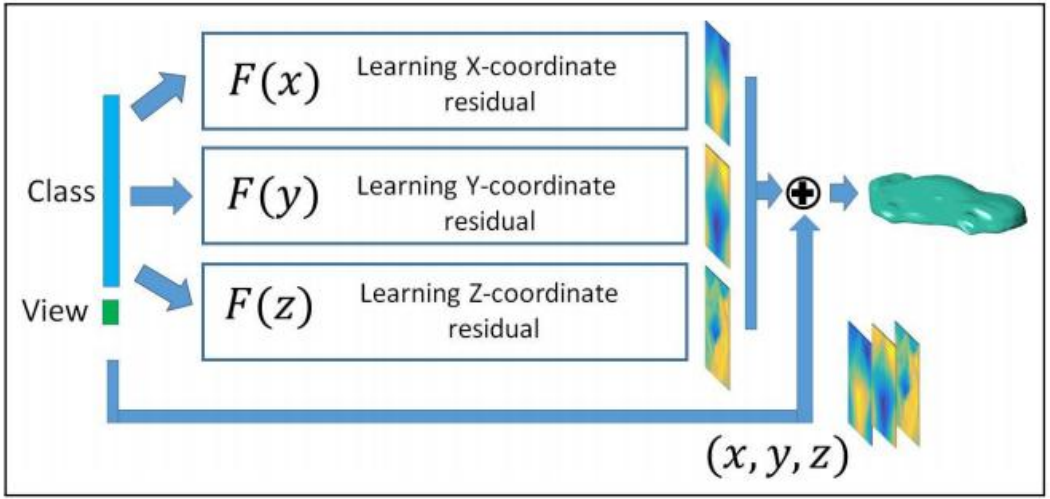
The up and down residual building blocks of our network architecture composed of upsampling, downsampling and standard residual blocks shown in center.

SurfNet: Generating 3D shape surfaces using deep residual networks



Network architecture for generating a geometry image feature channel for rigid shapes from a one hot-encoded class label and view angle (in analogy to pose) parameters.

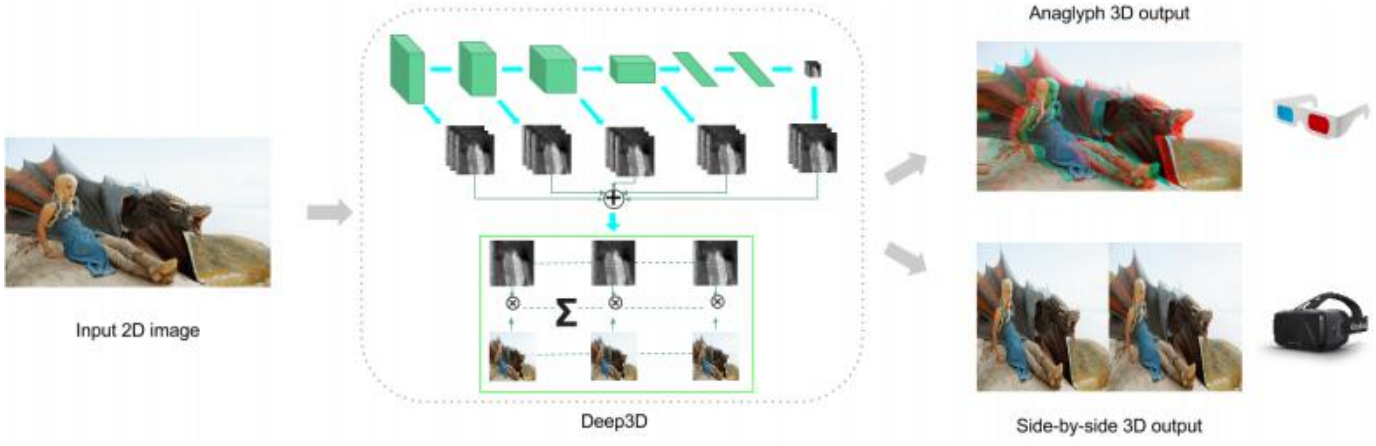
SurfNet: Generating 3D shape surfaces using deep residual networks



Pipeline for generating a surface plot of a rigid shape from class and view parameters by summing the residual geometry image of x, y, z coordinates to the base geometry image.

Deep3D: Fully Automatic 2D-to-3D Video Conversion with Deep CNN

- Use deep NNs for automatically converting 2D videos and images to stereoscopic 3D format.
- In contrast to previous automatic 2D-to-3D conversion algorithms, this approach is trained end-to-end directly on stereo pairs extracted from 3D movies.
- This training scheme could exploit more data and significantly increases performance.

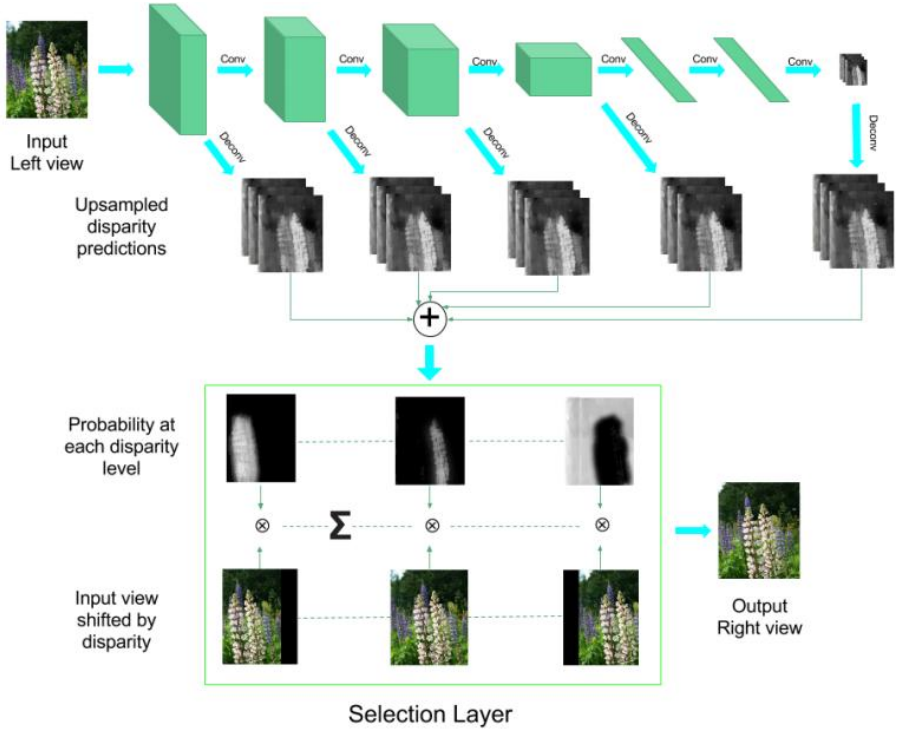


Deep3D: Fully Automatic 2D-to-3D Video Conversion with Deep CNN

- Deep3D, a fully automatic 2D-to-3D conversion algorithm that takes 2D images or video frames as input and outputs 3D stereo image pairs.
- The stereo images can be viewed with 3D glasses or head-mounted VR displays.
- Deep3D is trained directly on stereo pairs from a dataset of 3D movies to minimize the pixel-wise reconstruction error of the right view when given the left view.
- Internally, the Deep3D network estimates a probabilistic disparity map that is used by a differentiable depth image-based rendering layer to produce the right view.
- Thus Deep3D does not require collecting depth sensor data for supervision.

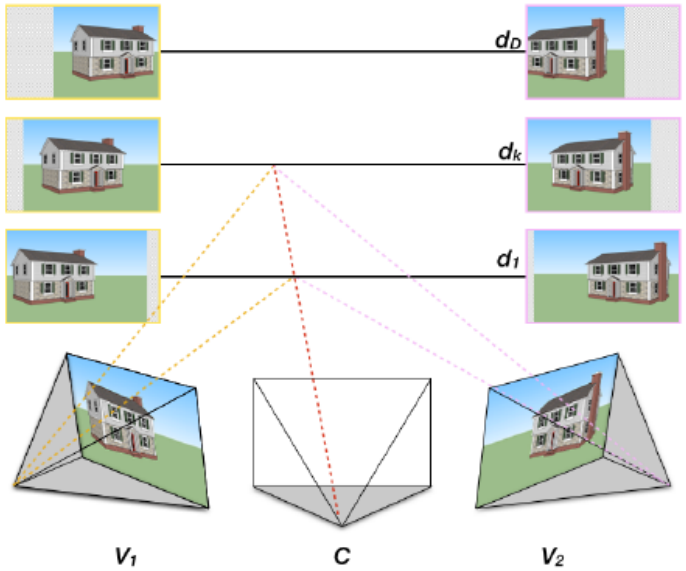
Deep3D: Fully Automatic 2D-to-3D Video Conversion with Deep CNN

Deep3D model architecture. The model combines information from multiple levels and is trained end-to-end to directly generate the right view from the left view. The base network predicts a probabilistic disparity assignment which is then used by the selection layer to model Depth Image-Based Rendering (DIBR) in a differentiable way. This also allows implicit in-painting.

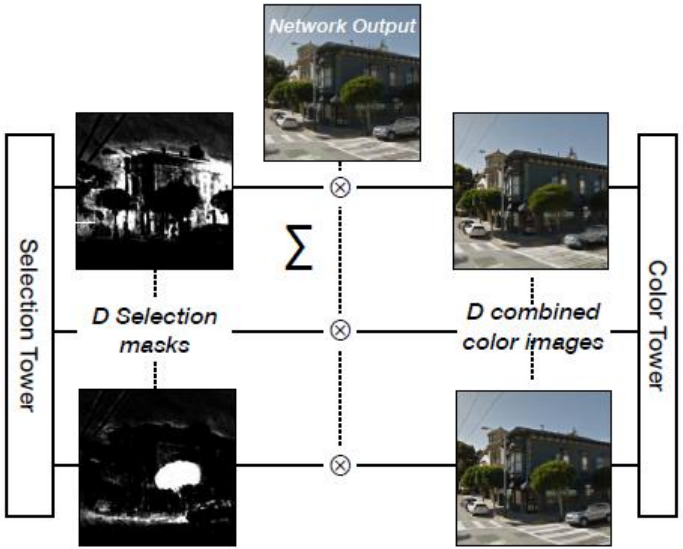


DeepStereo: Learning to Predict New Views from the World's Imagery

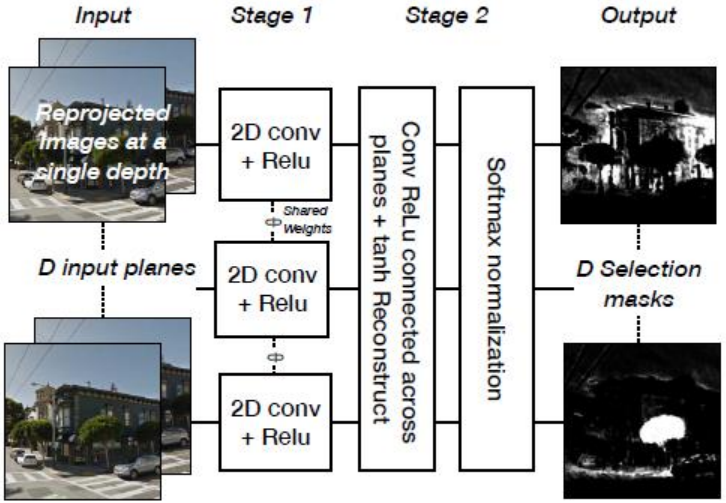
- This work performs new view synthesis directly from pixels, trained from a large number of posed image sets.
- End-to-end learning: the pixels from neighboring views of a scene are presented to the network, which then directly produces the pixels of the unseen view;
- A set of 3D plane sweep volumes as input; a plane sweep volume (PSV) consists of a stack of images reprojected to the target camera C;
- Each image I_k in the stack is reprojected into C at a set of varying depths d Ξ {d_1, d_2, …d_D} to form a plane sweep volume $V_C^k = \{P_1^k, P_2^k, ..., P$ where P^k_i refers to the reprojected image I_k at depth d_i.
- Create a separate plane sweep volume V^k_C for each input image I_k.
- Each voxel $v^k_{i,j,z}$ in each plane sweep volume V^k_C has RGB and A (alpha) components.
- A given output pixel depends only on a small column of voxels from each of the per-source PSVs.



DeepStereo: Learning to Predict New Views from the World's Imagery

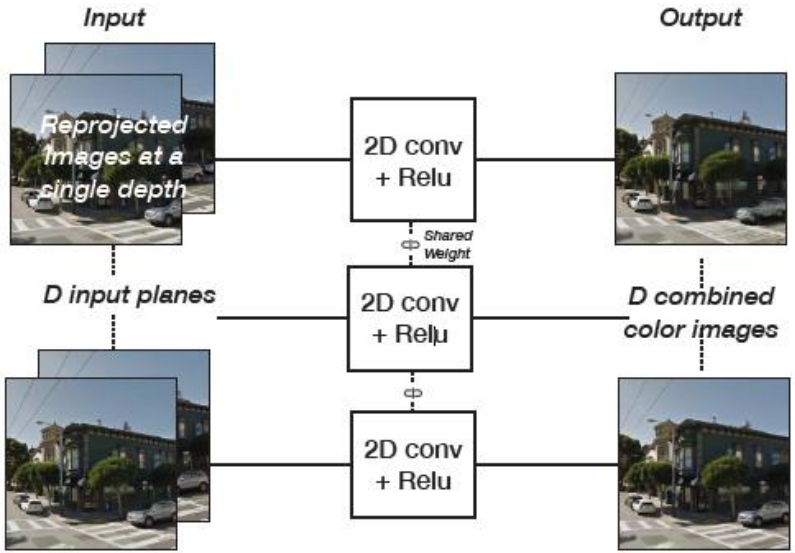


Basic architecture with both selection and color towers



The selection tower: compute features that are independent of depth; model interactions btw depth planes, connected across depth planes.

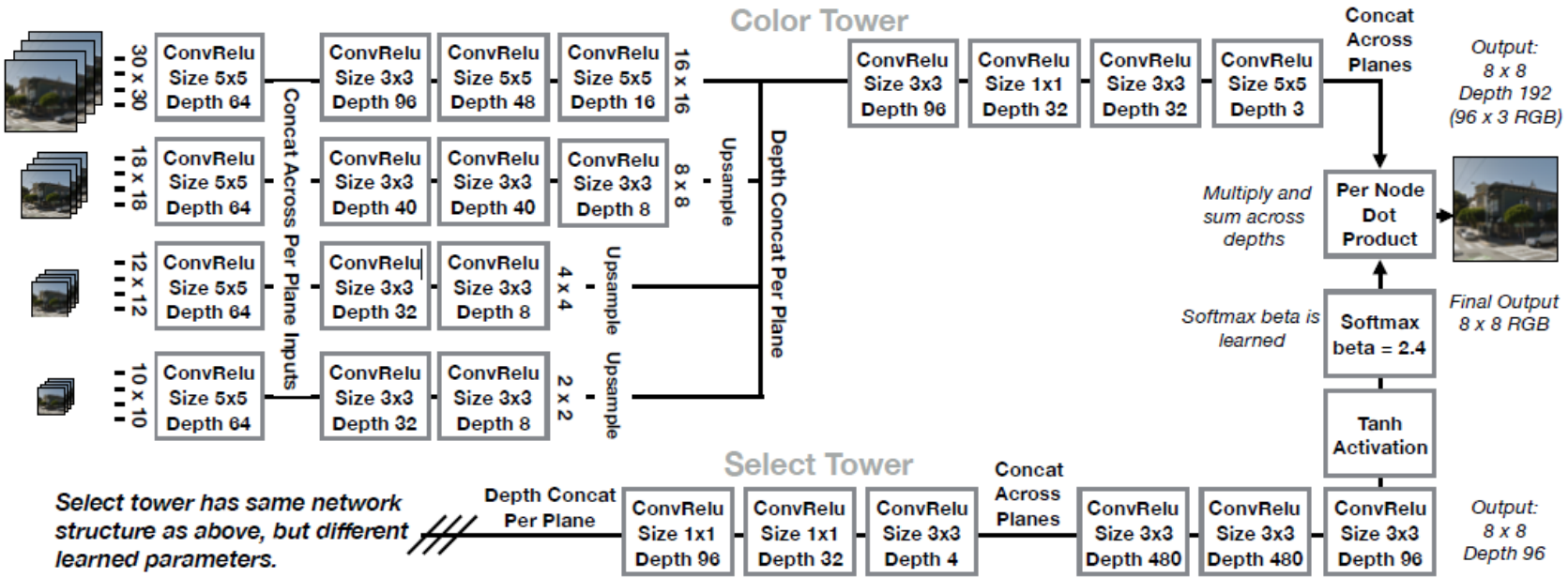
DeepStereo: Learning to Predict New Views from the World's Imagery



The color tower: learns to combine and warp pixels across the sources.

- **Multi-resolution patches**. Rather than predict a full image at a time predict the output image patch-by-patch.
  - Four different resolutions: Each resolution is first processed independently by several layers and then up-sampled and concatenated before entering the final layers.
- **Training**. Taking a posed set of images, leaving one image out, and predicting it from the remaining ones.
  - Uses subsets of N+1 images during training;
  - The center image is used as the target and the other N are used as input.
  - The network was trained to produce 8x8 patches from overlapping input patches of size 30x30;
  - Trained by Adagrad with an initial learning rate of 0.0001.

DeepStereo: Learning to Predict New Views from the World's Imagery

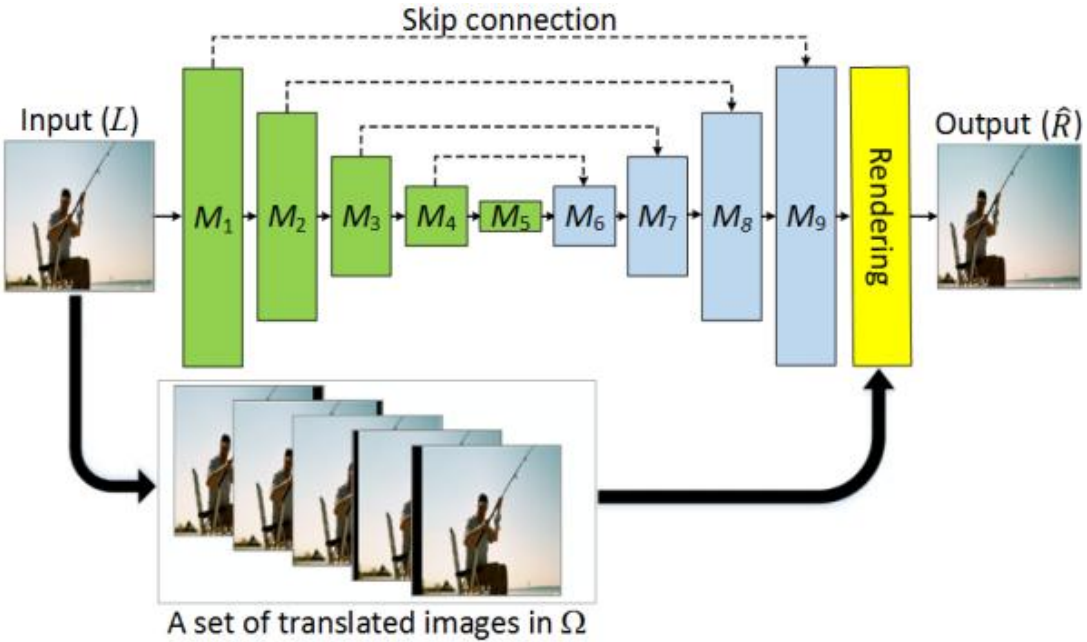


Full network diagram: The initial stages of both the color and selection towers are the same structure, but do not share parameters.

Efficient and Scalable View Generation from a Single Image using Fully Convolutional Networks

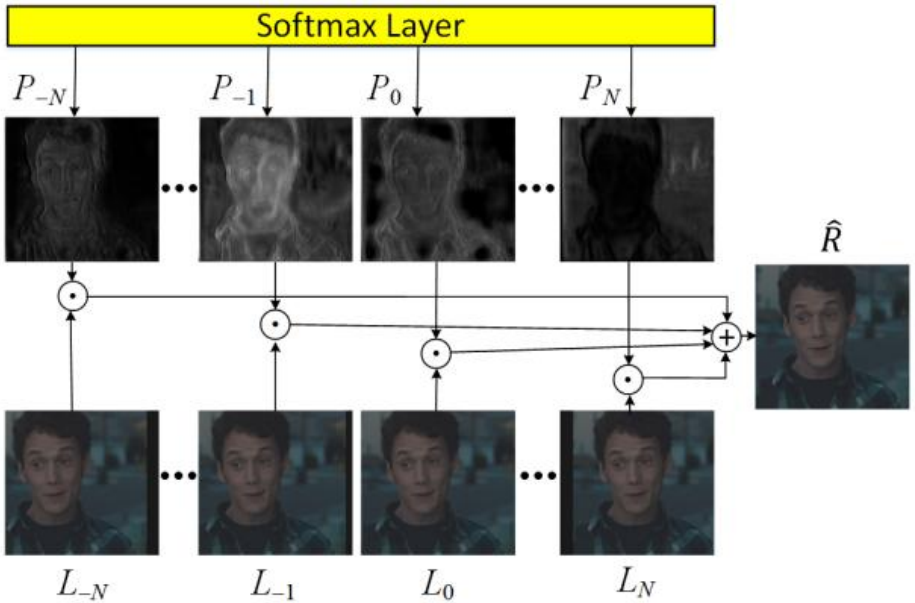
- Exploit two FCN architectures for Single-image-based view generation (SIVG).
- To train the solutions, a large dataset of 2M stereoscopic images is built.
- DeepViewren: based on combination of an FCN and a view-rendering network.
  - It generates competitive accuracy to the state of the art, however, with the fastest processing speed of all. That is x5 times faster speed and x24 times lower memory consumption compared to the state of the art.
- DeepViewdec: consists of decoupled networks for luminance and chrominance signals.
  - It has much higher accuracy, but with x2.5 times faster speed and x12 times lower memory consumption.

Efficient and Scalable View Generation from a Single Image using Fully Convolutional Networks



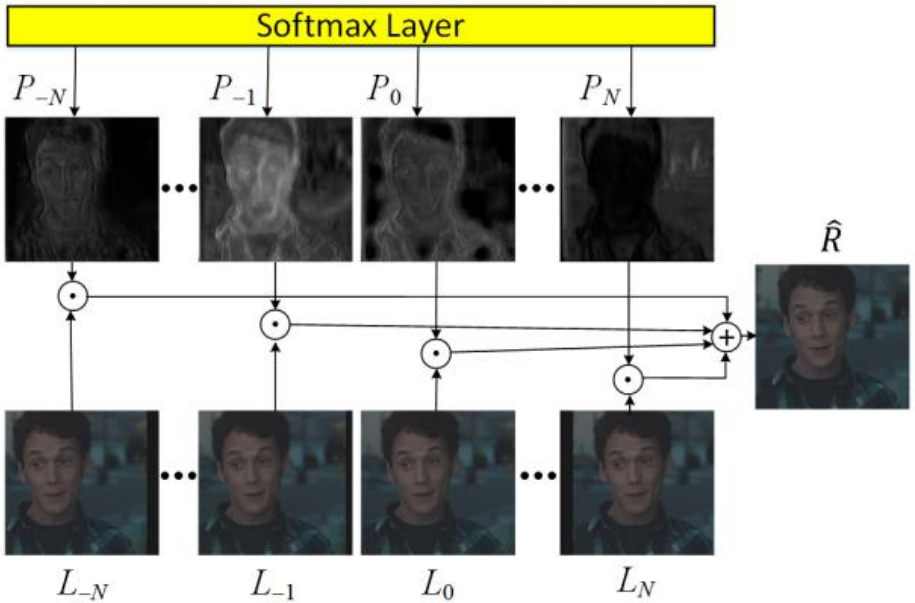
DeepViewgen architecture. The encoding network extracts low, middle and high level features from the input image and transfers them to the decoding network. After decoding, the rendering network generates probabilistic disparity maps and estimates the right-image.

Efficient and Scalable View Generation from a Single Image using Fully Convolutional Networks



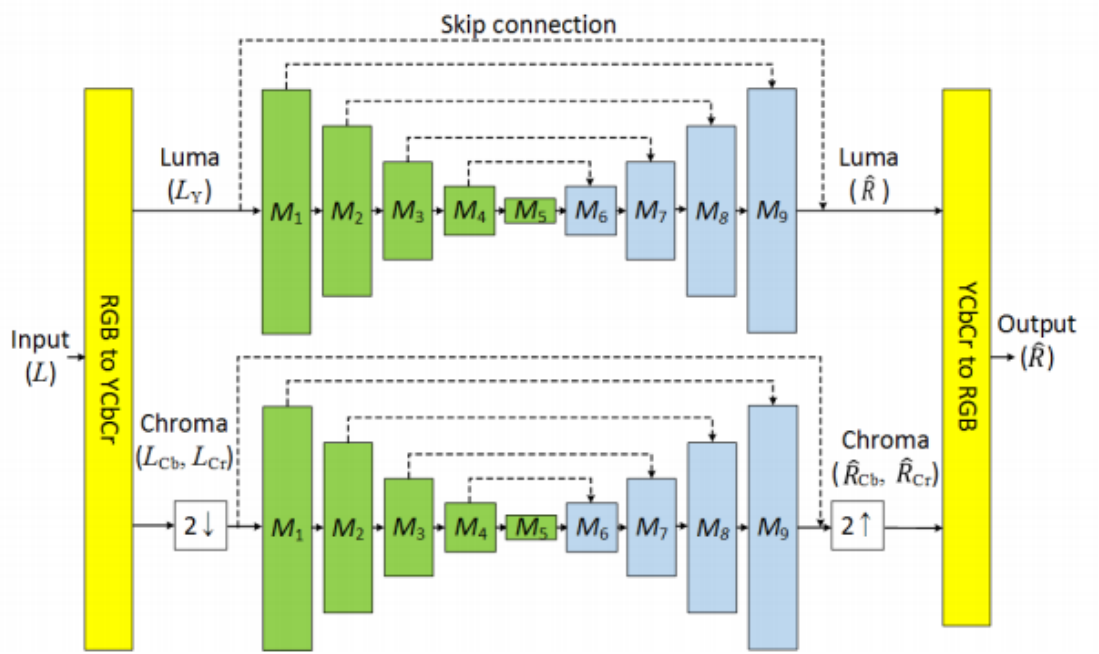
The rendering network. The softmax layer normalizes the output of the decoding network to probabilistic values over channels (P_i, $i \in \Omega$). Here, # channels is identical to # values in a disparity range $\Omega = \{-N, -N + 1, ..., 0, 1, ..., N\}$. The final right-view image $\hat{R}$ is synthesized by pixel-wise multiplication between P and their correspondingly-translated left-images L.

Efficient and Scalable View Generation from a Single Image using Fully Convolutional Networks



The rendering network. The softmax layer normalizes the output of the decoding network to probabilistic values over channels (P_i, $i \in \Omega$). Here, # channels is identical to # values in a disparity range $\Omega = \{-N, -N + 1, ..., 0, 1, ..., N\}$. The final right-view image $\hat{R}$ is synthesized by pixel-wise multiplication between P and their correspondingly-translated left-images L.

Efficient and Scalable View Generation from a Single Image using Fully Convolutional Networks

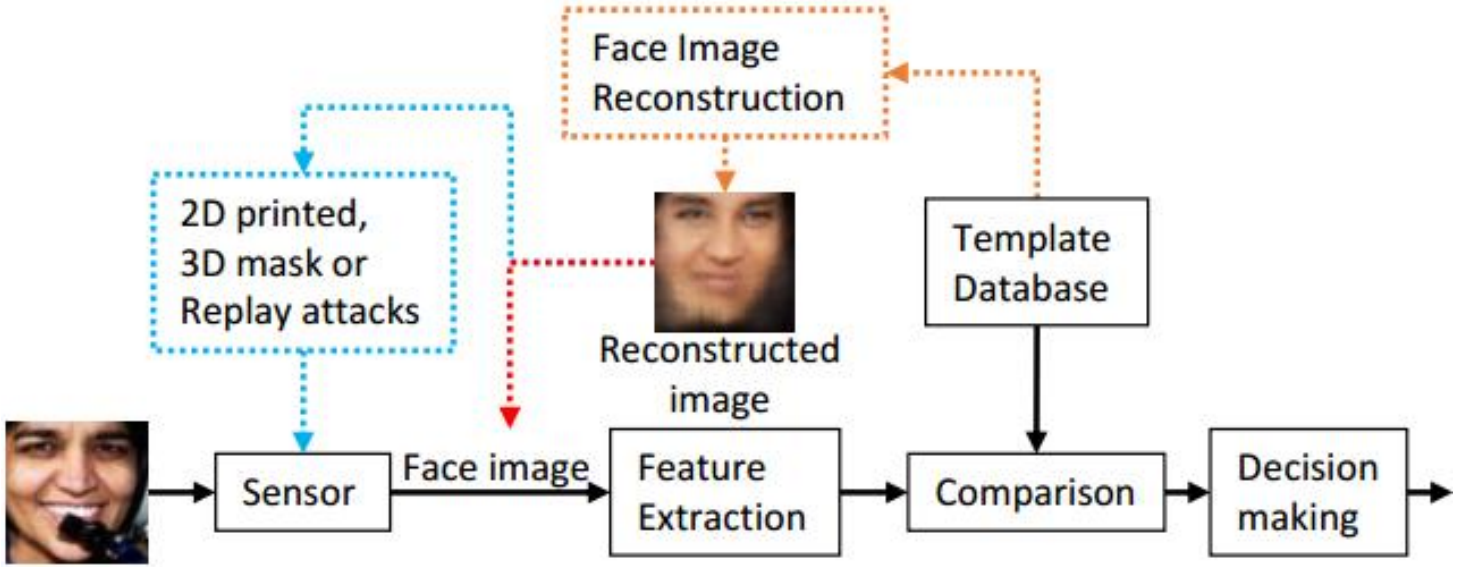


DeepViewdec architecture. Two decoupled networks having the same architecture, i.e., luminance (Y) and chrominance (Cb, Cr) network. Each network is trained separately.

Face Image Reconstruction from Deep Templates

- To what extent face templates derived from deep networks can be inverted to obtain the original face image.
- Deep templates, extracted by deep networks under image reconstruction attack.
- A de-convolutional neural network (D-CNN) to reconstruct images of faces from their deep templates.
- The reconstruction method was evaluated using type-I (comparing the reconstructed images against the original face images used to generate the deep template) and type-II (comparing the reconstructed images against a different face image of the same subject) attacks.
- A three-trial attack for each target face image using three face images reconstructed from three different D-CNNs.

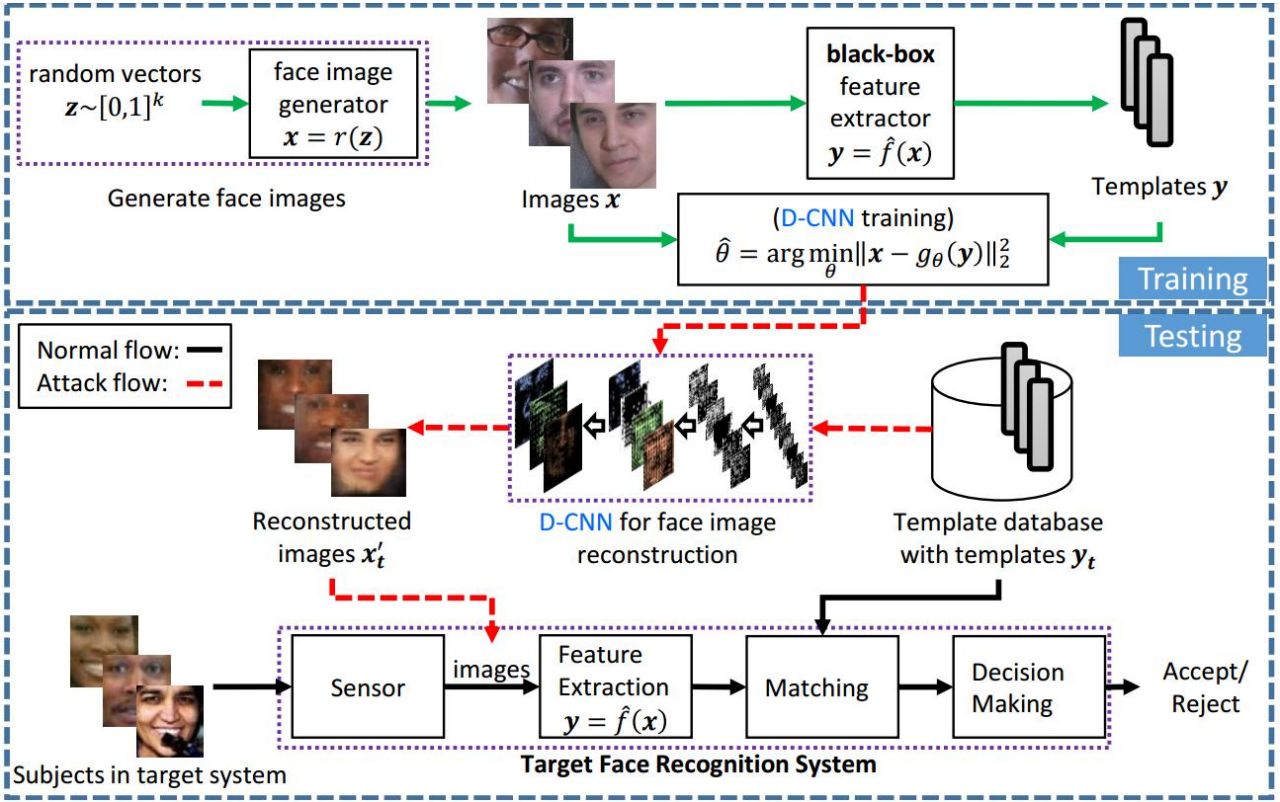
Face Image Reconstruction from Deep Templates



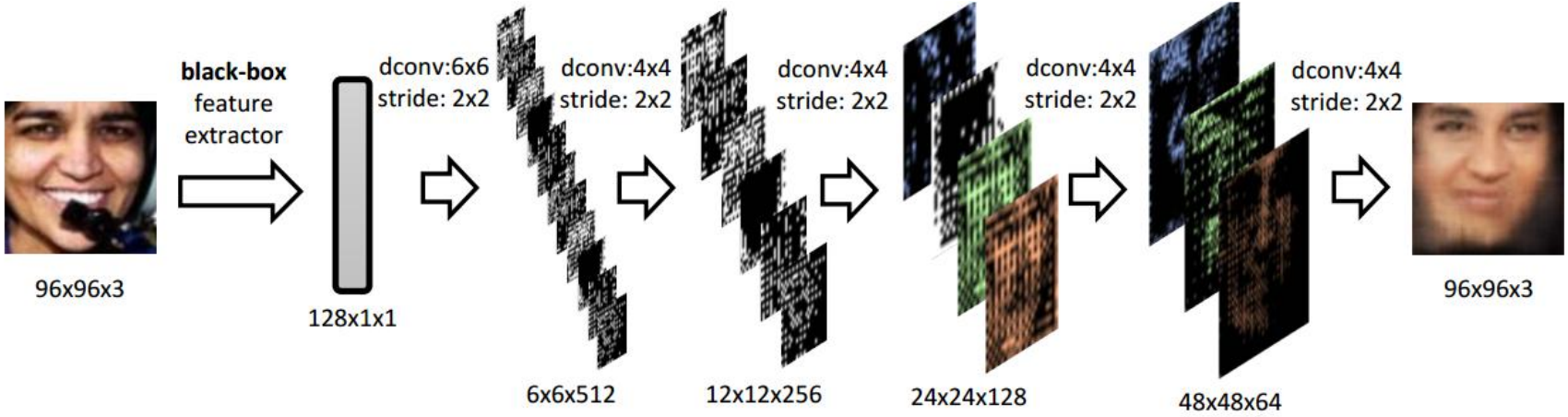
Face recognition system vulnerability to **image reconstruction** attacks. Face image of a target subject is reconstructed from a template to gain system access by either (a) creating a fake face (for example, a 2D printed image or 3D mask) or (b) inserting a reconstructed face into the feature extractor.

Face Image Reconstruction from Deep Templates

An overview of the template security study of deep network based face recognition systems under image reconstruction attack.



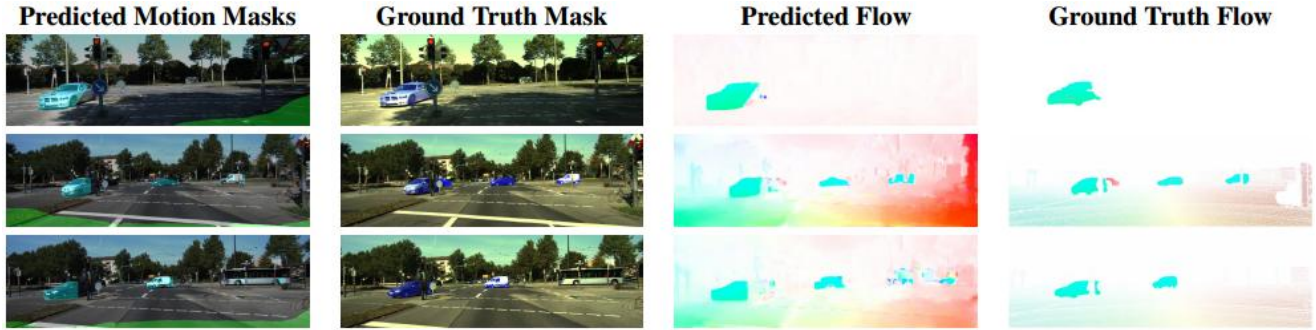
Face Image Reconstruction from Deep Templates



The de-convolutional neural network (D-CNN) for reconstructing face images from the corresponding face templates. The numbers (w × h × c) below each layer denote its width, height, and number of channels, respectively.

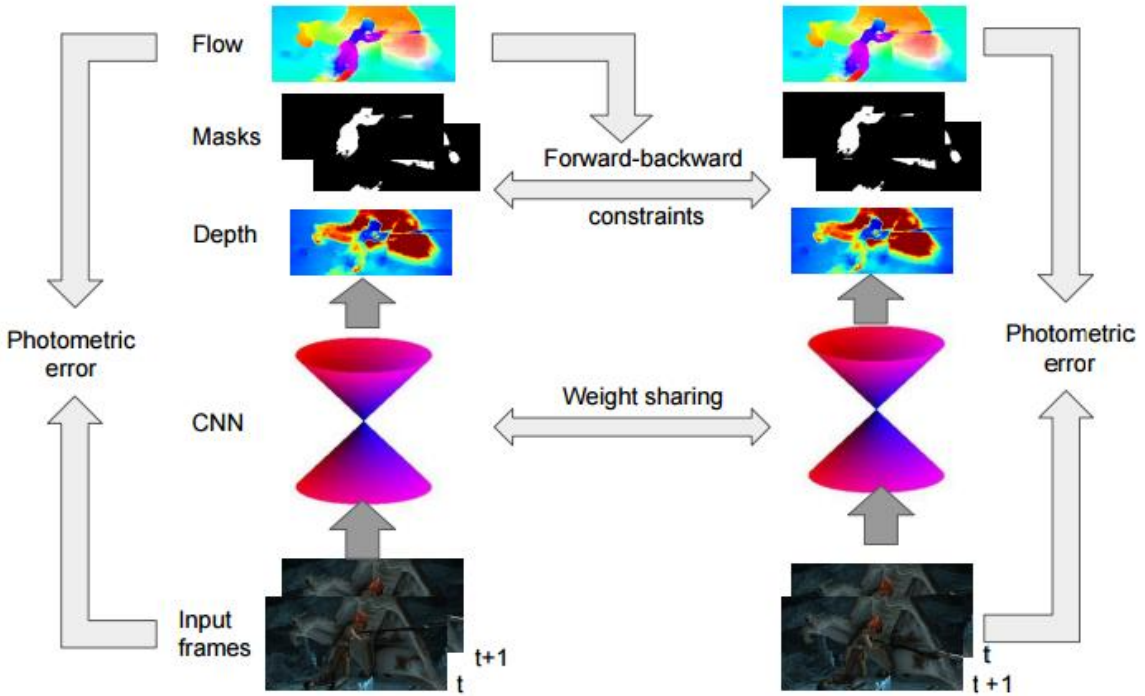
SfM-Net: Learning of Structure and Motion from Video

- SfM-Net, a geometry-aware NN for motion estimation in videos that decomposes frame pixel motion in terms of scene and object depth, camera motion and 3D object rotations and translations.
- Given a sequence of frames, SfM-Net predicts depth, segmentation, camera and rigid object motions, converts those into a dense frame-to-frame motion field (optical flow), differentiably warps frames in time to match pixels and back-propagates.
- The model can be trained with various degrees of supervision: 1) self-supervised by the reprojection photometric error (completely unsupervised), 2) supervised by ego-motion (camera motion), or 3) supervised by depth (e.g., as provided by RGBD sensors).



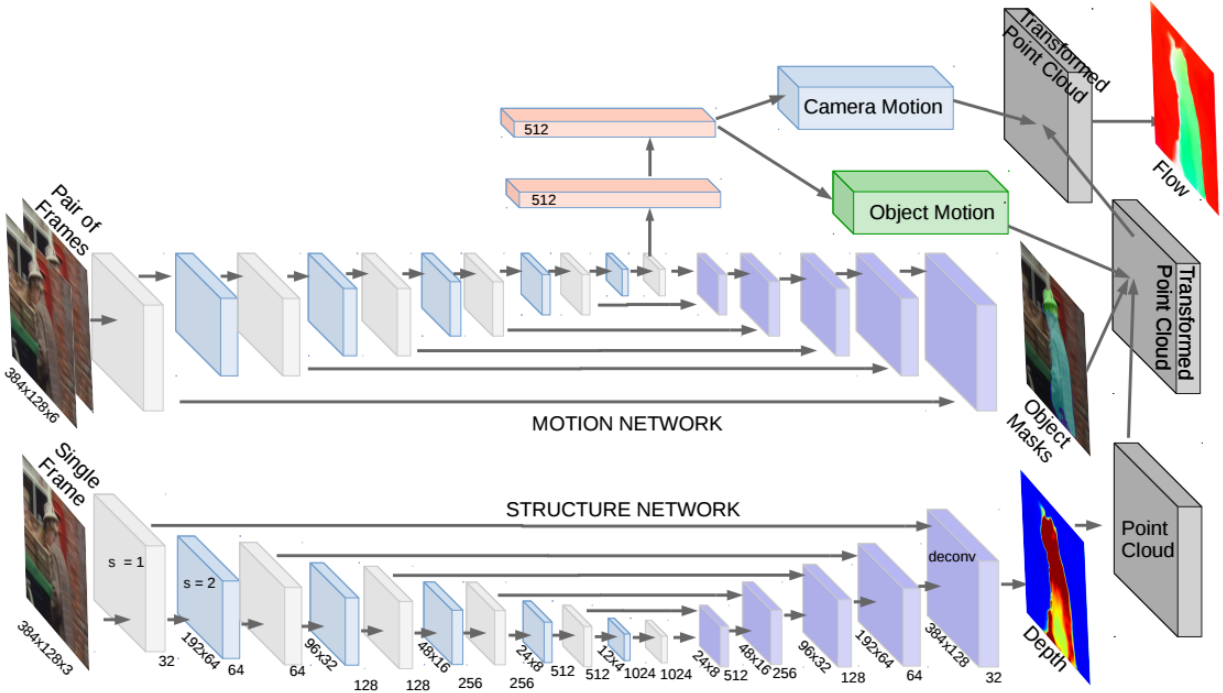
SfM-Net: Learning of Structure and Motion from Video

SfM-Net: Given a pair of frames as input, the model decomposes frame pixel motion into 3D scene depth, 3D camera rotation and translation, a set of motion masks and corresponding 3D rigid rotations and translations. It back-projects the resulting 3D scene flow into 2D optical flow and warps accordingly to match pixels from one frame to the next. Forward-backward consistency checks constrain the estimated depth.



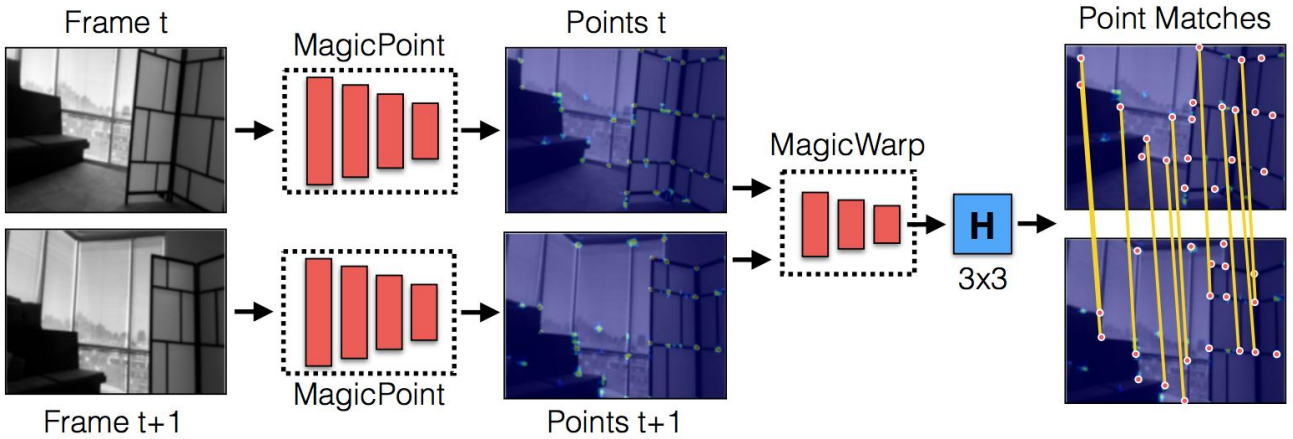
SfM-Net: Learning of Structure and Motion from Video

SfM-Net architecture. For each pair of consecutive frames, a conv/deconv sub-network predicts depth while another predicts a set of segmentation masks. The coarsest feature maps of the motion-mask encoder are further decoded through FCL towards 3D rotations and translations for the camera and the K segmentations. The predicted depth is converted into a per frame point-cloud using estimated or known camera intrinsics.

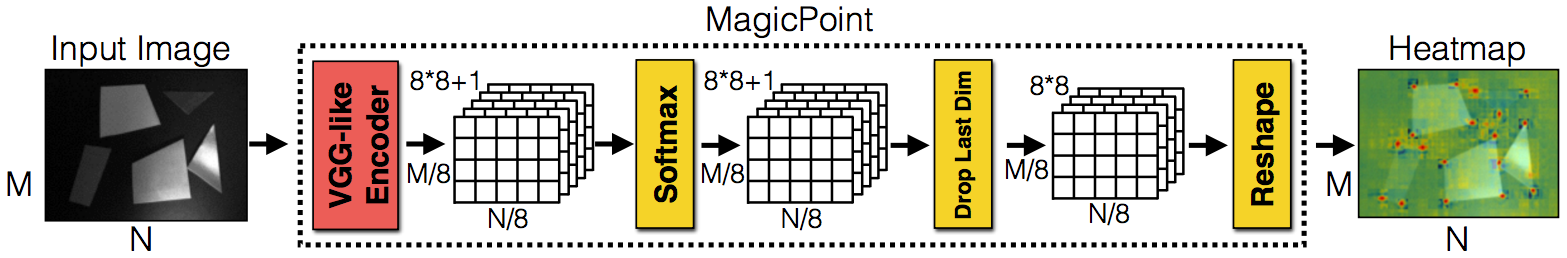


Toward Geometric Deep SLAM

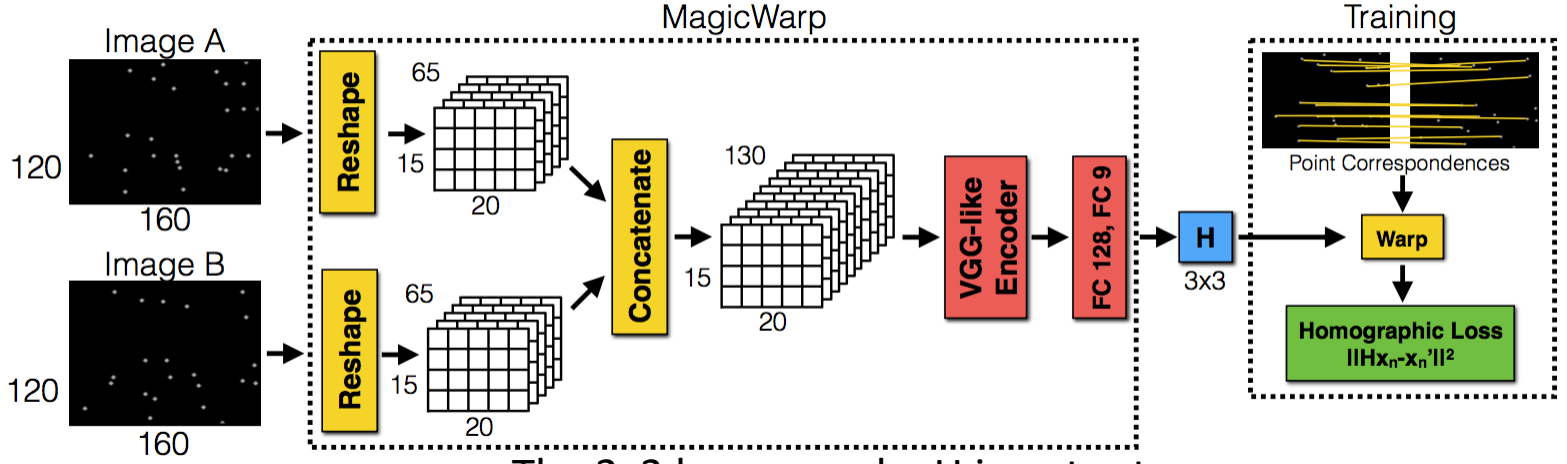
- Point tracking powered by two CNNs: 1st network, MagicPoint, operates on single images and extracts salient 2D points; 2nd network, MagicWarp, operates on pairs of point images, and estimates the homography.
- Note: The extracted points are “SLAM-ready” because they are by design isolated and well-distributed throughout the image.



Toward Geometric Deep SLAM



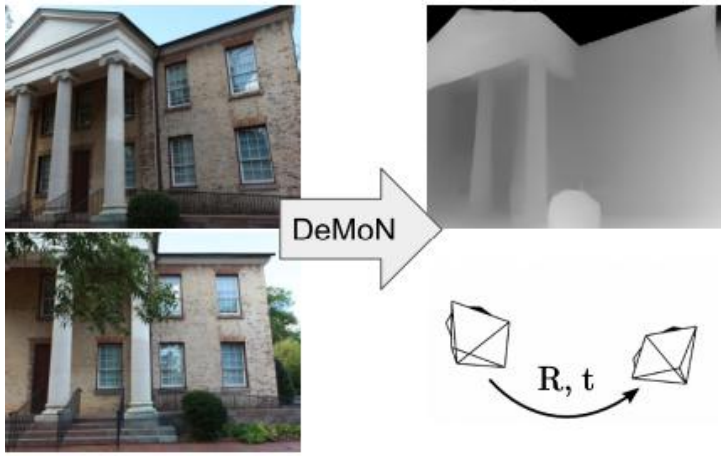
outputs a "point-ness" probability for each pixel.



The 3x3 homography H is output.

DeMoN: Depth and Motion Network for Learning Monocular Stereo

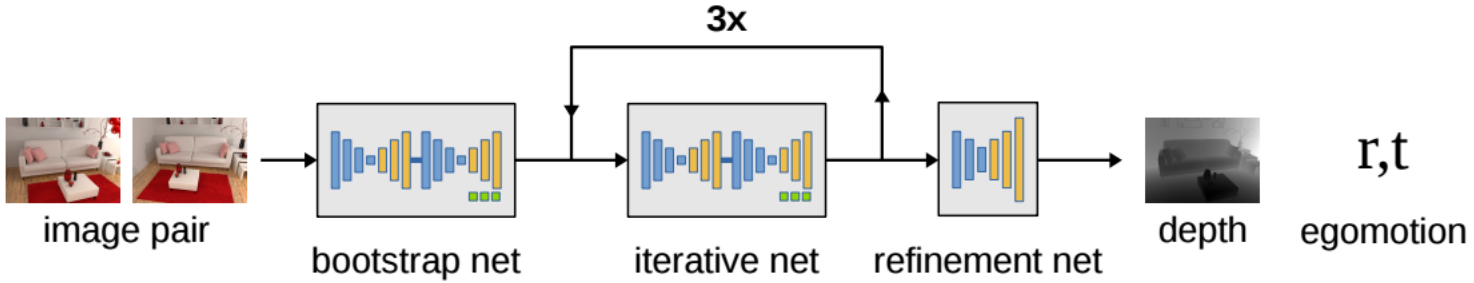
- Structure from motion as a learning problem: train a convolutional network end-to-end to compute depth and camera motion from successive, unconstrained image pairs.
- The architecture is composed of multiple stacked **encoder-decoder** networks, the core part being an **iterative network** that is able to improve its own predictions.
- The network estimates not only depth and motion, but additionally surface normals, optical flow btw the images and confidence of the matching.
- A training loss based on spatial relative differences.
- The architecture consists of 3 main components: the bootstrap net, the iterative net and the refinement net.

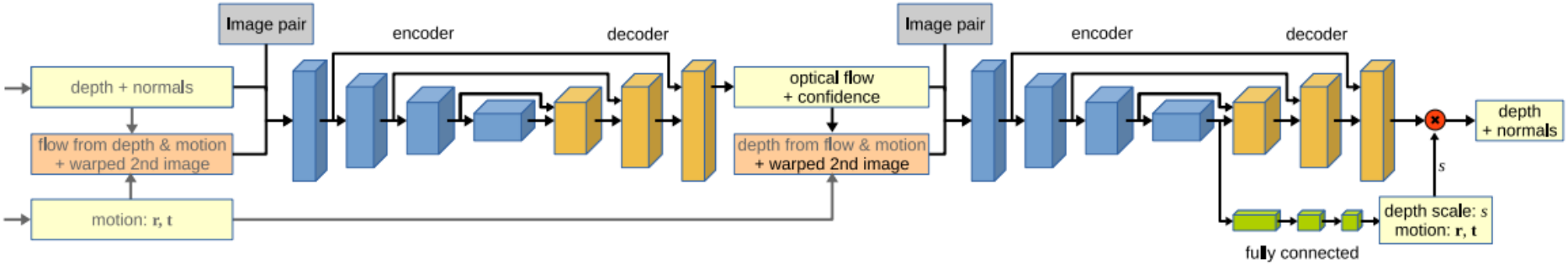


DeMoN: Depth and Motion Network for Learning Monocular Stereo

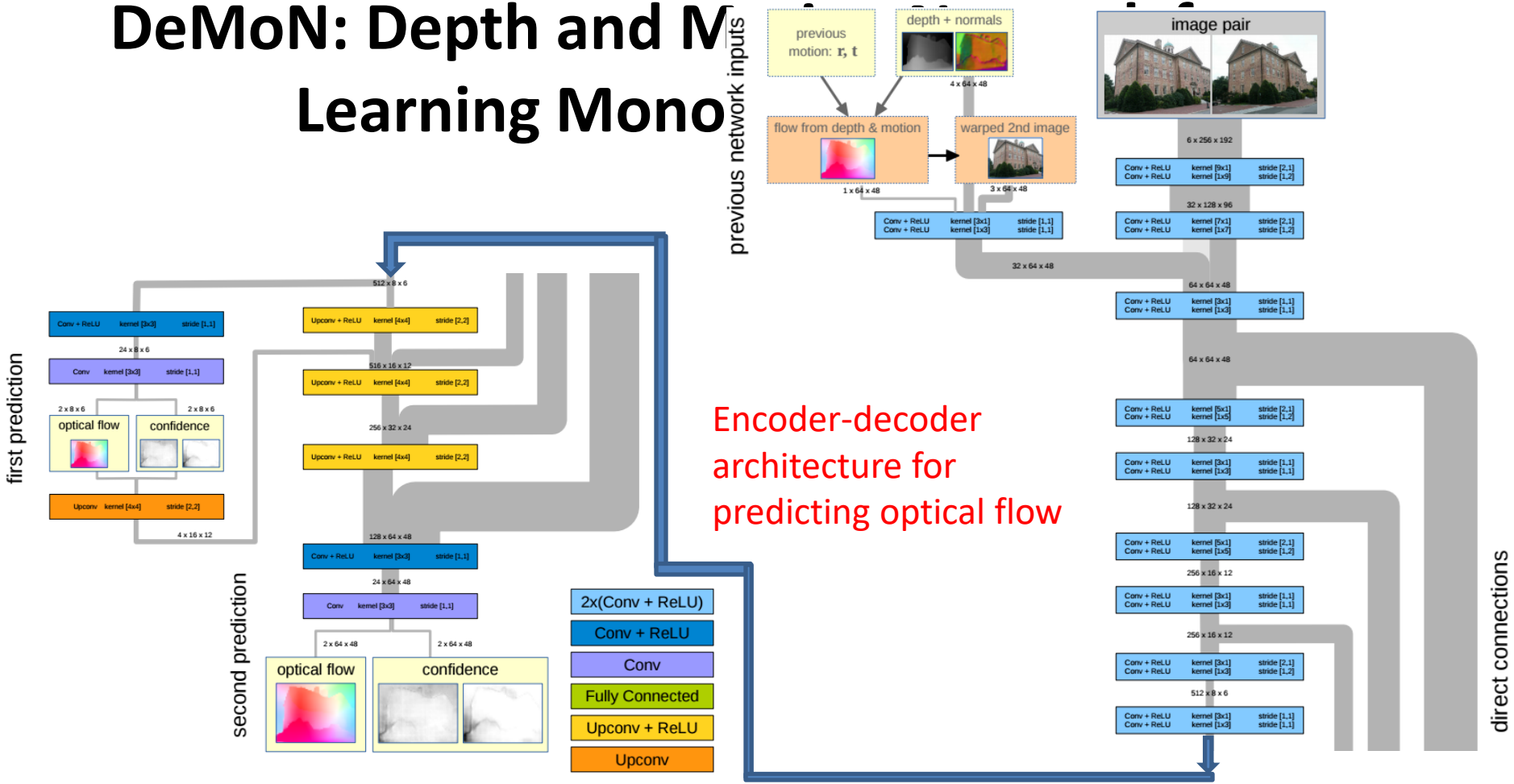
- The first two components are pairs of encoder-decoder networks, where the first one computes optical flow while the second one computes depth and camera motion.
- The bootstrap component gets the image pair as input and outputs the initial depth and motion estimates.
  - Internally, first an encoder-decoder network computes optical flow and a confidence map for the flow;
  - The second encoder-decoder takes as input the optical flow, its confidence, the image pair, and the second image warped with the estimated flow field.
- The iterative net is applied recursively to successively refine the estimates of the previous iteration.
  - The iterative net is trained to improve existing depth, normal, and motion estimates.
  - The architecture of this encoder-decoder pair is identical to the bootstrap net, but it takes additional inputs.
- The last is a encoder-decoder network that generates the final up-sampled and refined depth map.
  - The full resolution first image and the nearest-neighbor-upsampled depth and normal field as input, the final refinement net upscales the predictions to the full input image resolution (256 × 192).

DeMoN: Depth and Motion Network for Learning Monocular Stereo

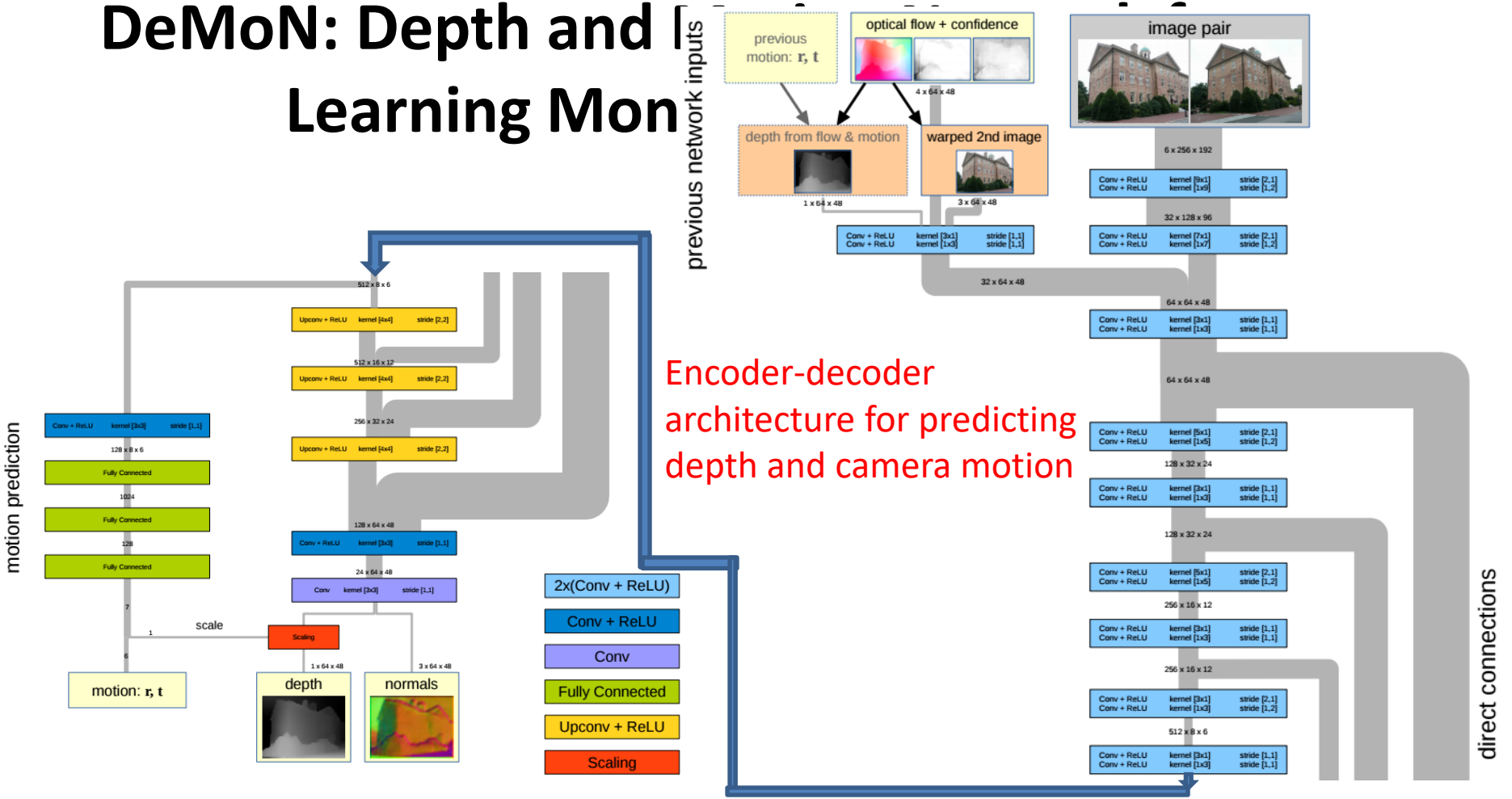




DeMoN: Depth and M
Learning Mono

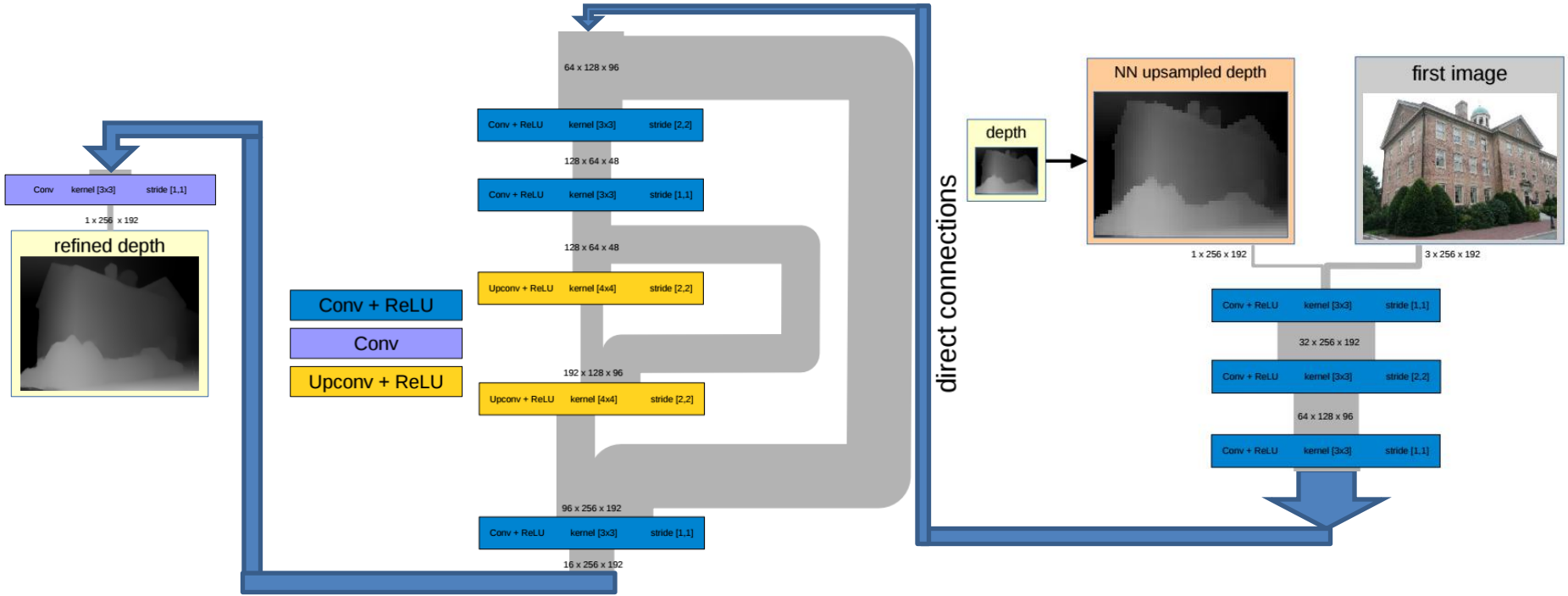


DeMoN: Depth and [illegible] Learning Mon



Encoder-decoder architecture for predicting depth and camera motion

DeMoN: Depth and Motion Network for Learning Monocular Stereo



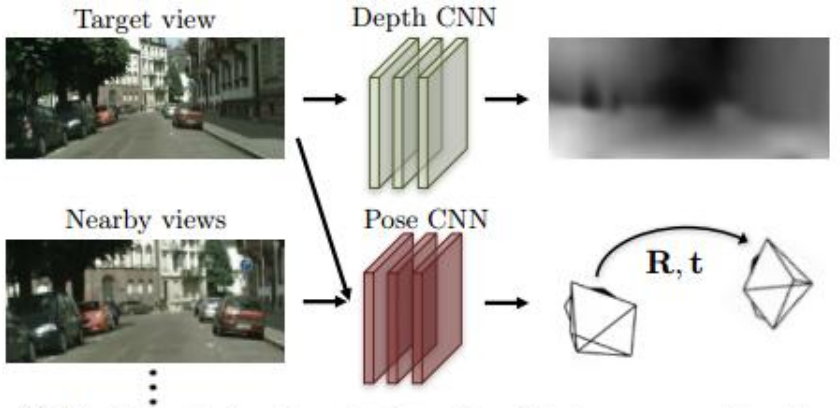
Encoder-decoder architecture for refining the depth prediction

Unsupervised Learning of Depth and Ego-Motion from Video

- Unsupervised learning framework for the task of monocular depth and camera motion estimation;
- End-to-end learning approach with view synthesis as the supervisory signal, requiring only monocular video sequences for training.
- Single-view depth and multi-view pose networks, with a loss based on warping nearby views to the target using the computed depth and pose.
- Coupled by the loss during training, but can be applied independently at test time.

Unsupervised Learning of Depth and Ego-Motion from Video

(a) Training: unlabeled video clips.

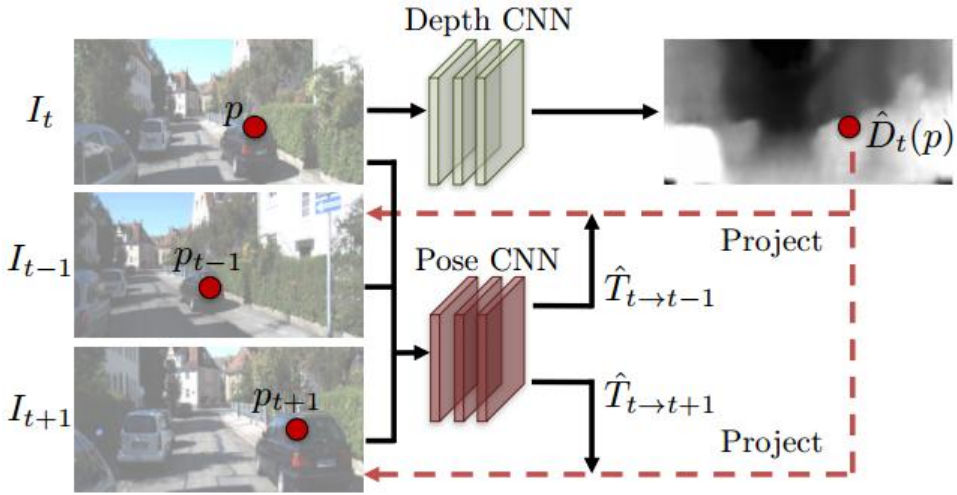


(b) Testing: single-view depth and multi-view pose estimation.

The training data to the system consists solely of unlabeled image sequences capturing scene appearance from different viewpoints, where the poses of the images are not provided.

The training procedure produces two models that operate independently, one for single-view depth prediction, and one for multi-view camera pose estimation.

Unsupervised Learning of Depth and Ego-Motion from Video



Overview of the supervision pipeline based on view synthesis. The **depth network** takes only the target view as input, and outputs a per-pixel depth map. The **pose network** takes both the target view and the nearby/source views as input, and outputs the relative camera poses. The outputs of **both networks** are then used to inverse warp the source views to reconstruct the target view, and the photometric reconstruction loss is used for training the CNNs. By utilizing view synthesis as supervision, able to train the entire framework in an unsupervised manner from videos.

Unsupervised Learning of Depth and Ego-Motion from Video

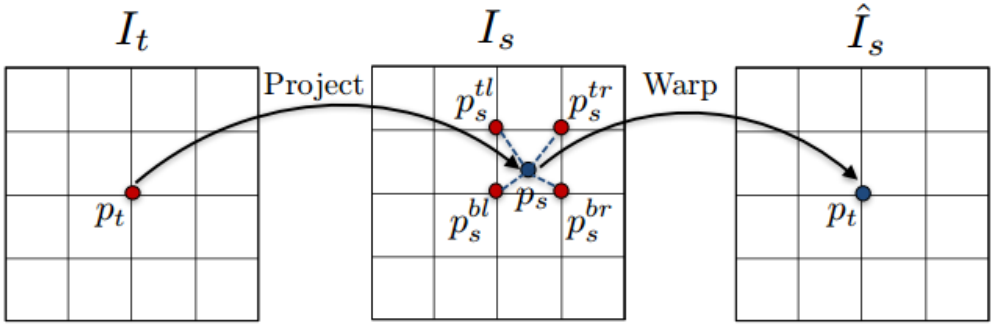
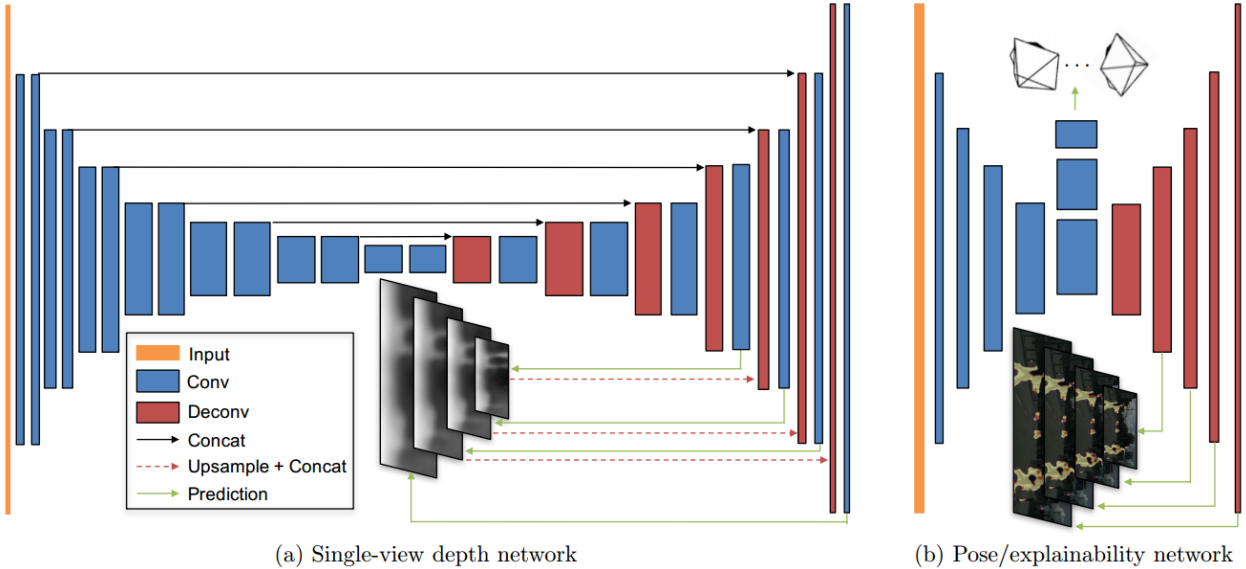


Illustration of the differentiable image warping process. For each point pt in the target view, first project it onto the source view based on the predicted depth and camera pose, and then use bilinear interpolation to obtain the value of the warped image at location pt.

Unsupervised Learning of Depth and Ego-Motion from Video

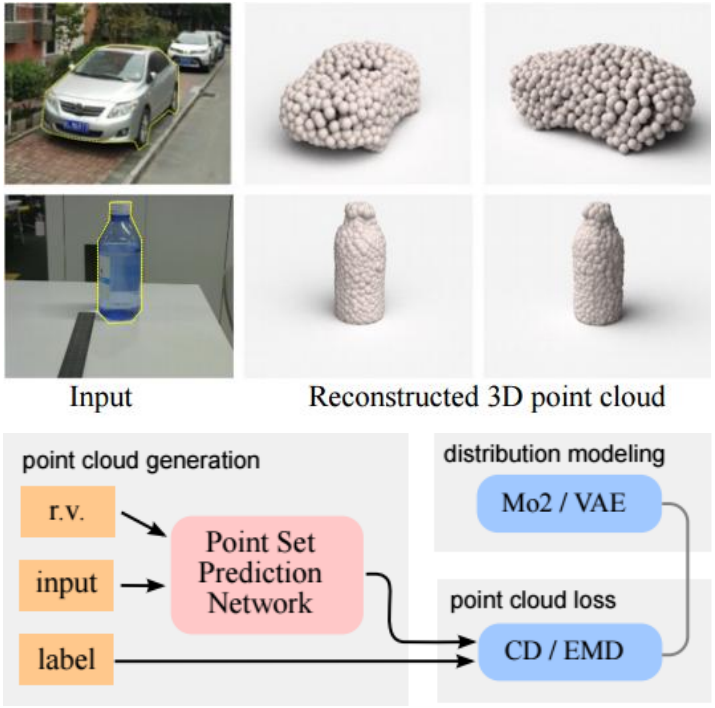


(a) Single-view depth network (b) Pose/explainability network

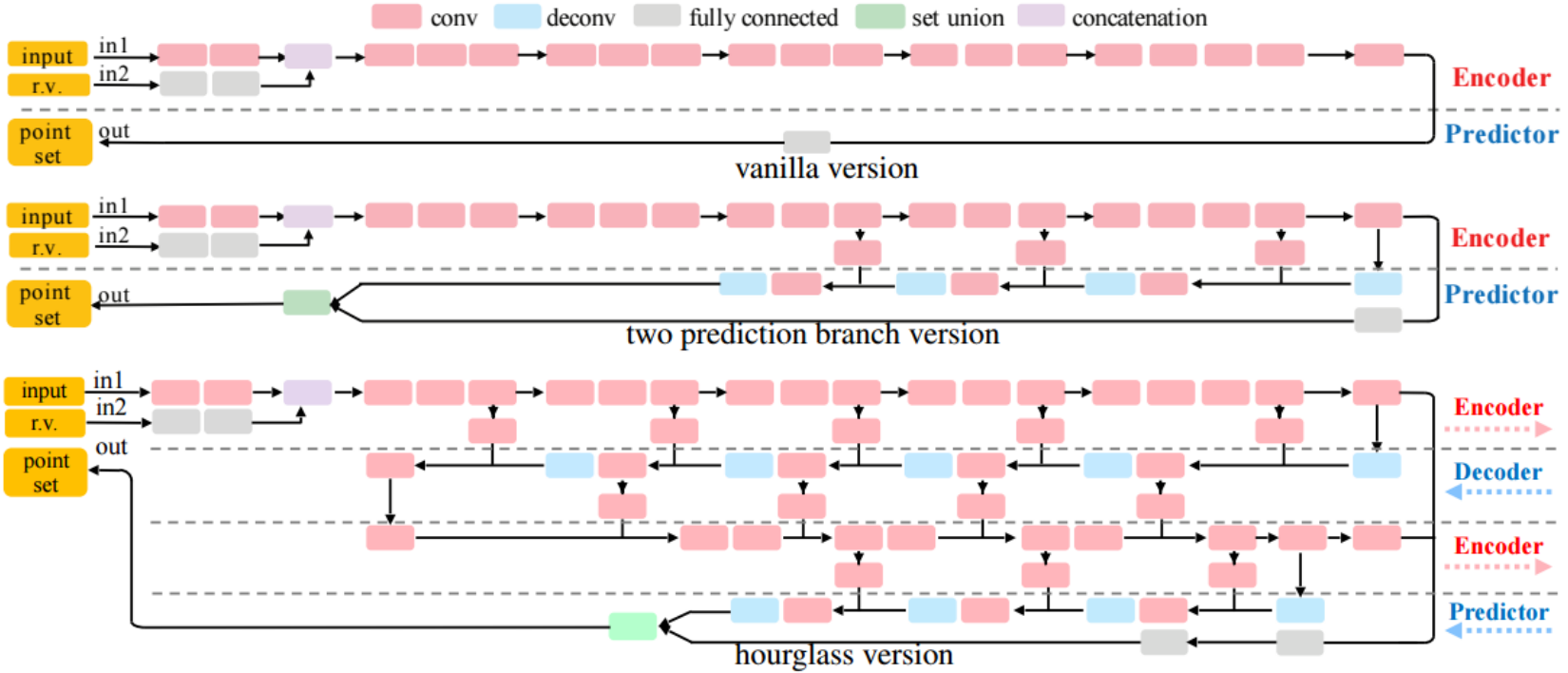
Network architecture for depth/pose/explainability prediction modules. The width and height of each rectangular block indicates the output channels and the spatial dimension of the feature map at the corresponding layer respectively, and each reduction/increase in size indicates a change by the factor of 2.

A Point Set Generation Network for 3D Object Reconstruction from a Single Image

- Address the problem of 3D reconstruction from a single image, generating a straight-forward form of output – point cloud coord.s
- Along with this problem arises a unique and interesting issue, that the ground truth shape for an input image may be ambiguous.
- The solution is a conditional shape sampler, capable of predicting multiple plausible 3D point clouds from an input image.

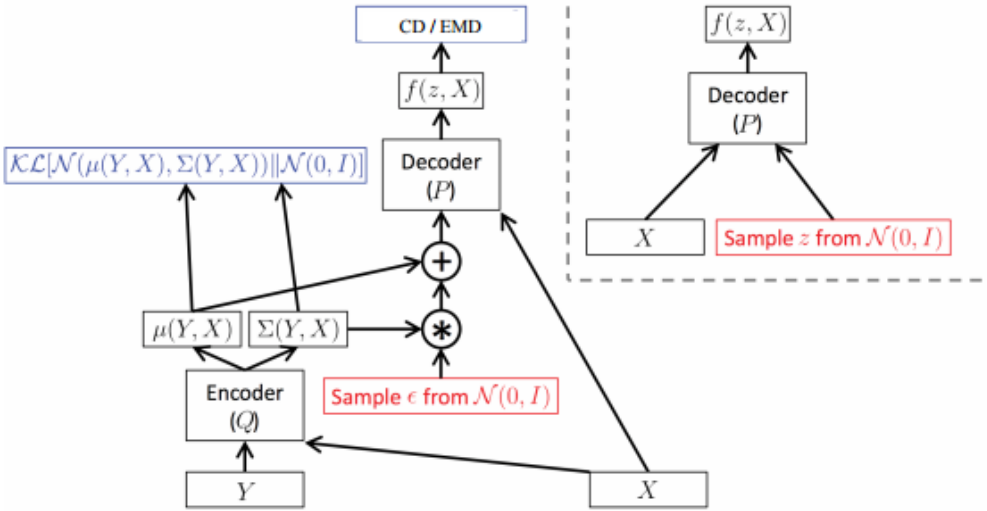


A Point Set Generation Network for 3D Object Reconstruction from a Single Image



PointOutNet structure

A Point Set Generation Network for 3D Object Reconstruction from a Single Image

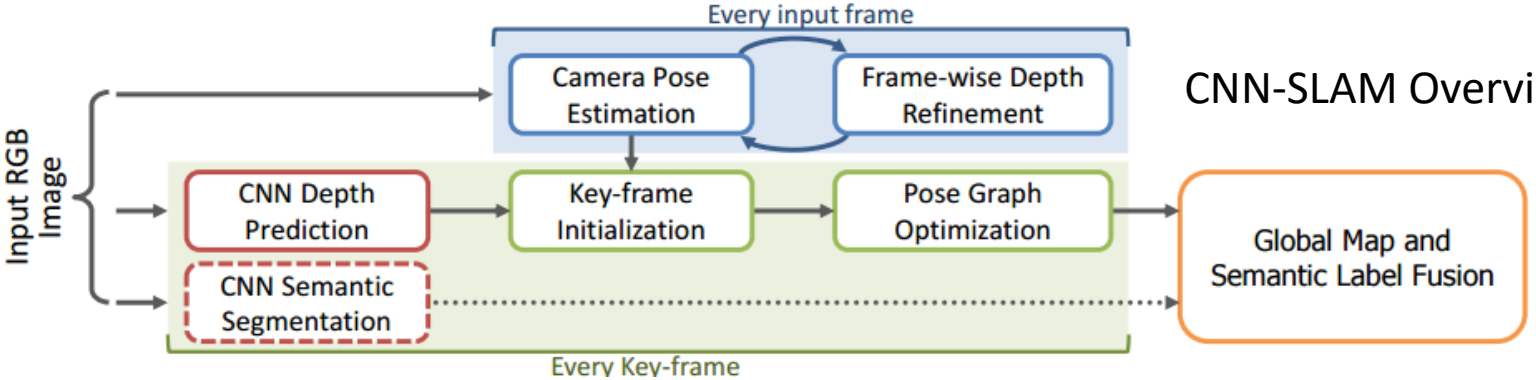


Network for conditional VAE shape sampler P(S|X). Left: a training-time conditional VAE implemented as a feed forward NN. Here, Y is the volumetric form of the ground truth shape S, whereas f(z, X) is the point cloud form of the predicted shape for S. Right: the same model at test time.

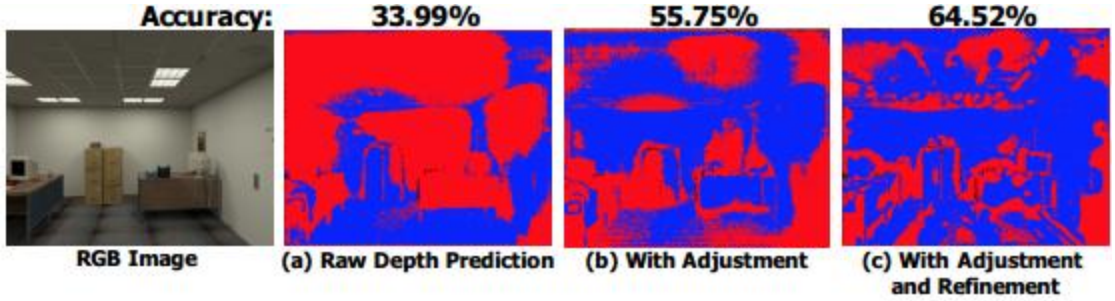
CNN-SLAM: Real-time dense monocular SLAM with learned depth prediction

- Given the depth prediction from CNNs, it can be deployed for accurate and dense monocular reconstruction.
- CNN-predicted dense depth maps are naturally fused together with depth measurements obtained from direct monocular SLAM.
- The fusion scheme privileges depth prediction in image locations where mono SLAM approaches tend to fail, e.g. along low-textured regions, and vice-versa.
- Depth prediction can estimate the absolute scale of the reconstruction, overcoming one of the major limitations of monocular SLAM.
- Fuse semantic labels, obtained from a single frame, with dense SLAM, yielding semantically coherent scene reconstruction from a single view.

CNN-SLAM: Real-time dense monocular SLAM with learned depth prediction



CNN-SLAM Overview

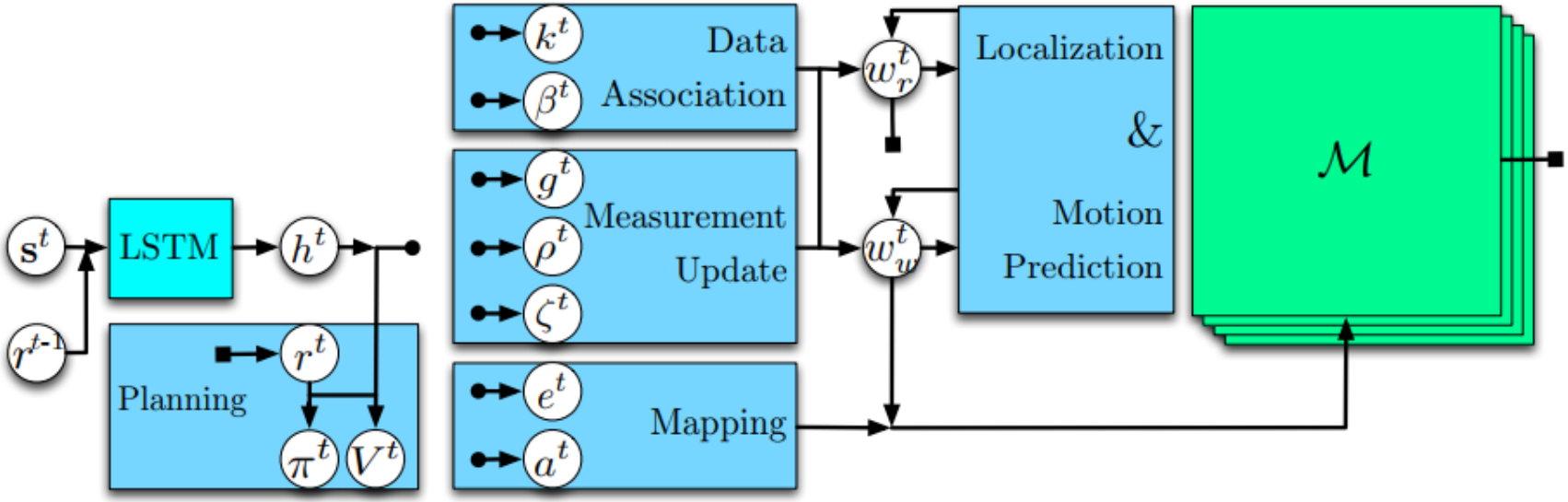


depth estimation accuracy

Neural SLAM

- Learn representations of a global map from sensor data, to aid their exploration in new environments.
- Simultaneous Localization and Mapping (SLAM) into the soft attention based addressing of external memory architectures, in which the external memory acts as an internal representation of the environment.
- This structure encourages the evolution of SLAM-like behaviors inside a completely differentiable deep NN.
- Reinforcement learning to successfully explore new environments where long-term memory is essential.

Neural SLAM



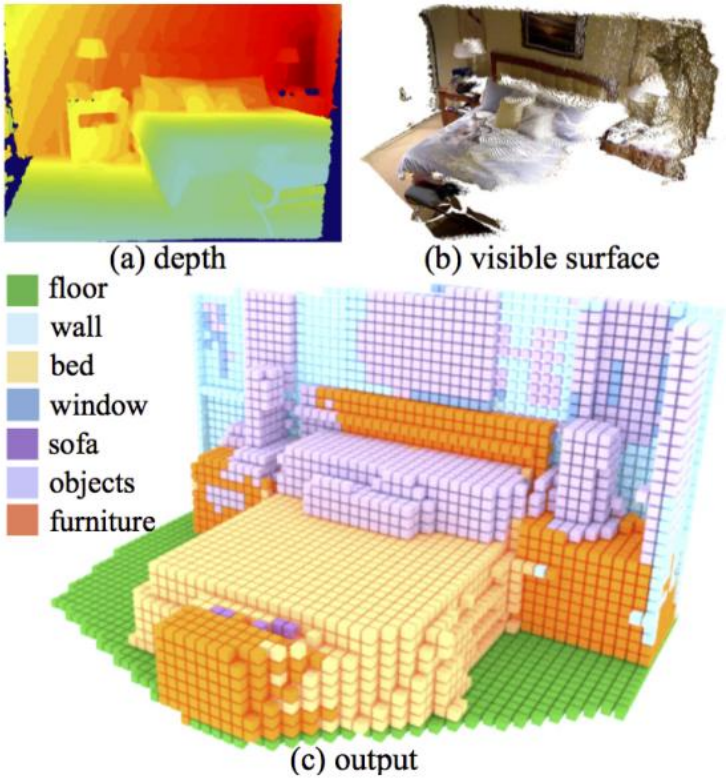
Neural-SLAM model architecture

Semantic Scene Completion from a Single Depth Image

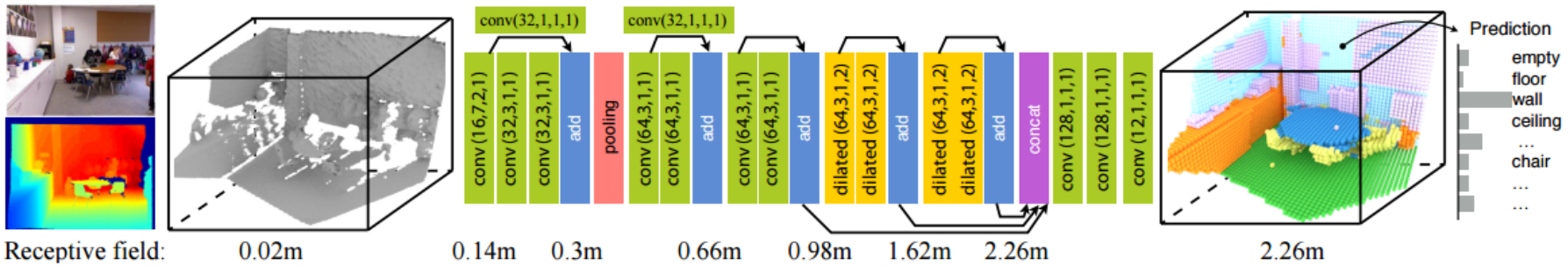
- A task for producing a complete 3D voxel representation of volumetric occupancy and semantic labels for a scene from a single-view depth map observation.
- To leverage the coupled nature of scene completion and semantic labeling of depth maps, design the semantic scene completion network (SSCNet), an end-to-end 3D convolnet that takes a single depth image as input and simultaneously outputs occupancy and semantic labels for all voxels in the camera view frustum.
- A dilation-based 3D context module to efficiently expand the receptive field and enable 3D context learning.
- To train our network, construct SUNCG - a manually created large-scale dataset of synthetic 3D scenes with dense volumetric annotations.

Semantic Scene Completion from a Single Depth Image

Semantic scene completion. (a) Input single-view depth map (b) Visible surface from the depth map. (c) Semantic scene completion result: the model jointly predicts volumetric occupancy and object categories for each of the 3D voxels in the view frustum. Note: the entire volume occupied by the bed is predicted to have the bed category.

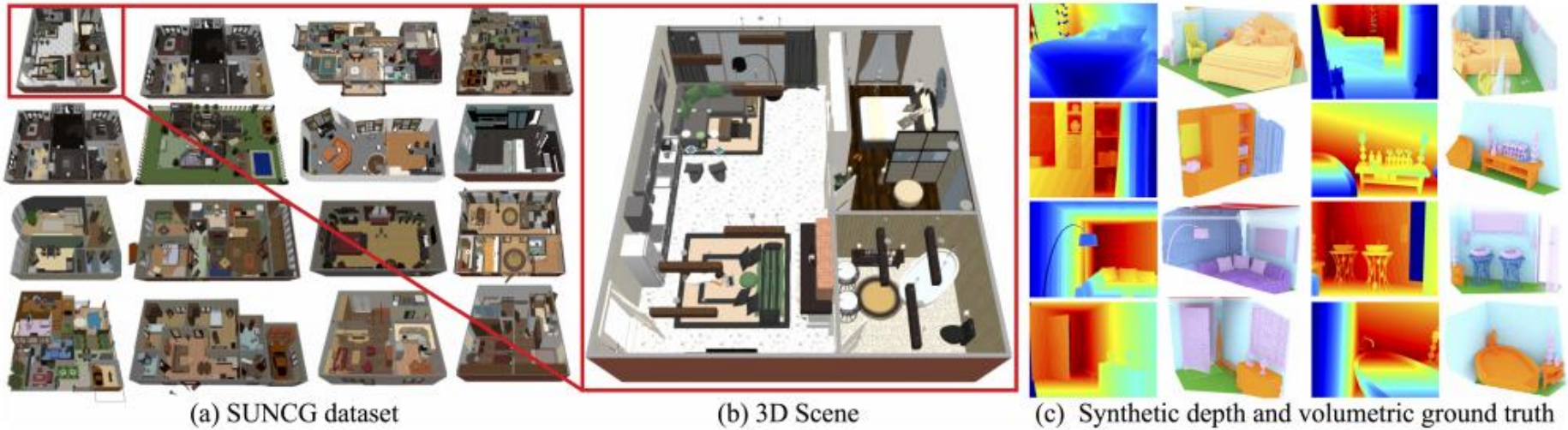


Semantic Scene Completion from a Single Depth Image



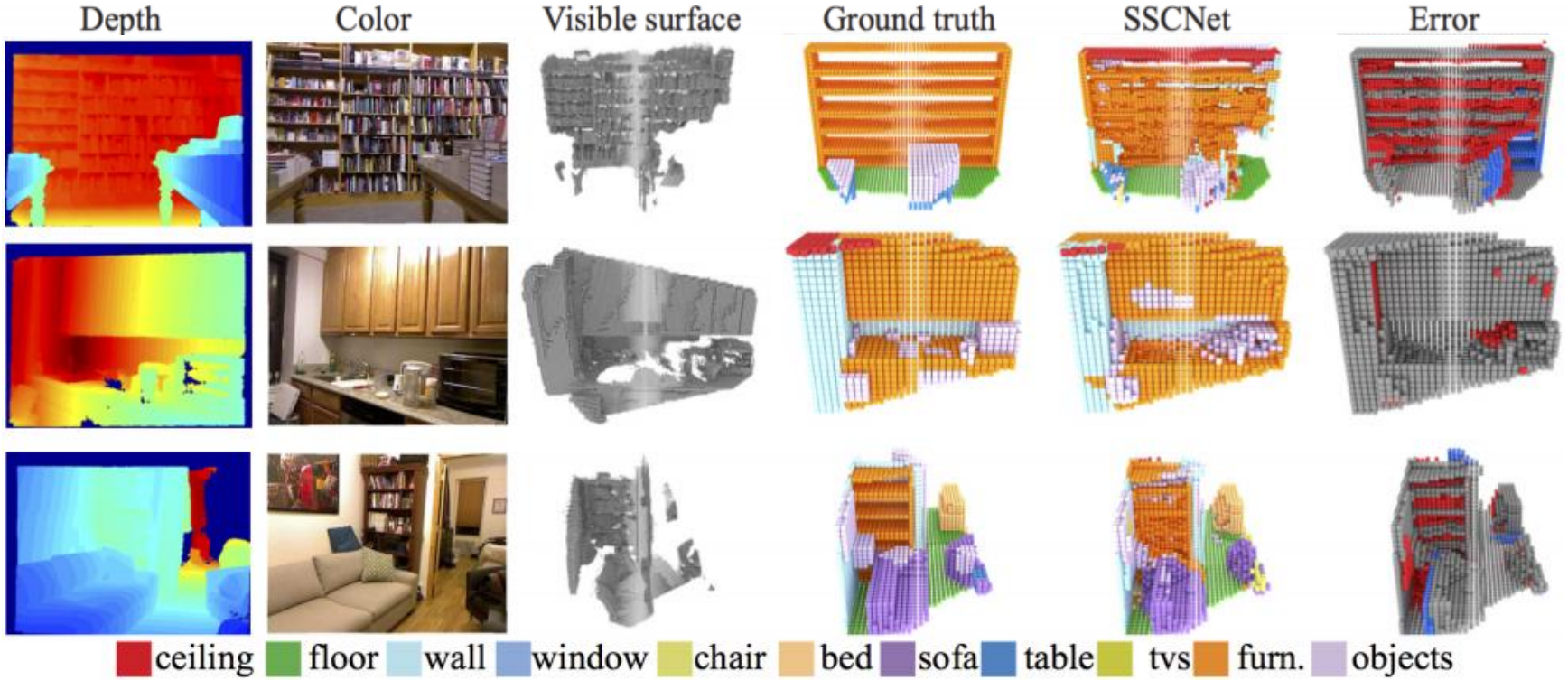
SSCNet: Semantic scene completion network. Taking a single depth map as input, the network predicts occupancy and object labels for each voxel in the view frustum. The convolution parameters are shown as (number of filters, kernel size, stride, dilation).

Semantic Scene Completion from a Single Depth Image



Synthesizing Training Data. Collect a large-scale synthetic 3D scene dataset to train the network. For each of the 3D scenes, select a set of camera positions and generate pairs of rendered depth images and volumetric ground truth as training examples.

Semantic Scene Completion from a Single Depth Image

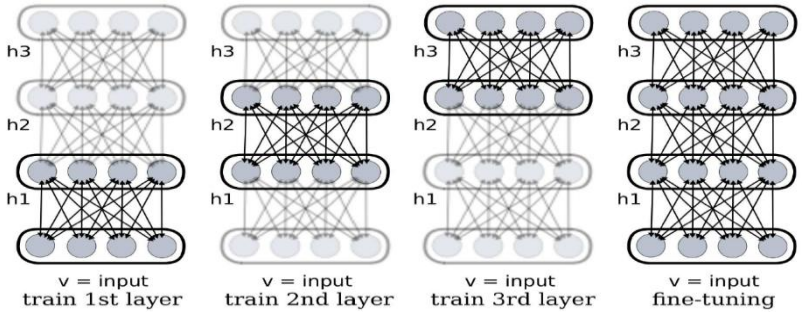


Appendix:

Deep Learning

Deep Learning

- **Representation learning** attempts to automatically learn good features or representations;
- **Deep learning** algorithms attempt to learn **multiple levels** of representation of increasing complexity/abstraction (intermediate and high level features);
- Become effective via **unsupervised** pre-training + **supervised** fine tuning;
  - Deep networks trained with back propagation (without unsupervised pre-training) perform **worse** than shallow networks.
- Deal with the curse of dimensionality (smoothing & sparsity) and over-fitting (unsupervised, regularizer);
- Semi-supervised: structure of manifold assumption;
  - **labeled** data is scarce and **unlabeled** data is abundant.



Why Deep Learning?

- Supervised training of **deep** models (e.g. many-layered Nets) is too hard (optimization problem);
    - Learn prior from unlabeled data;
- Shallow models are not for learning high-level **abstractions**;
    - Ensembles or forests do not learn features first;
    - Graphical models could be deep net, but mostly not.
- Unsupervised learning could be “local-learning”;
    - Resemble boosting with each layer being like a weak learner
- Learning is weak in directed graphical models with many **hidden** variables;
    - Sparsity and regularizer.
- Traditional **unsupervised** learning methods aren’t easy to learn multiple **levels** of representation.
    - Layer-wised unsupervised learning is the solution.
- Multi-task learning (transfer learning and self taught learning);
- Other issues: scalability & parallelism with the burden from **big data**.

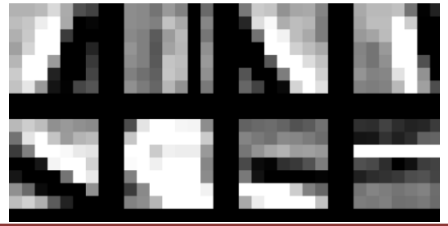
Deep Feature Learning

- Hand-crafted features:
  - Needs expert knowledge
  - Requires time-consuming hand-tuning
  - (Arguably) one limiting factor of computer vision systems
- Key idea of feature learning:
  - Learn statistical structure or correlation from unlabeled data
  - The learned representations used as features in supervised and semi-supervised settings
- Hierarchical feature learning
  - Deep architectures can be representationally efficient.
  - Natural progression from the low level to the high level structures.
  - Can share the lower-level representations for multiple tasks in computer vision.

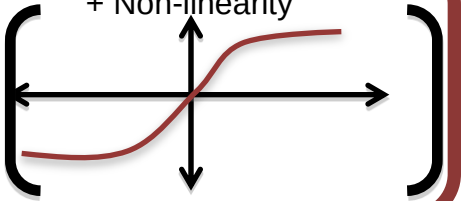
Feature Learning Architectures

Pixels / Features

Filtering with Dictionary
(patch/tiled/convolutional)



+ Non-linearity



Normalization between feature responses

(Group) Sparsity

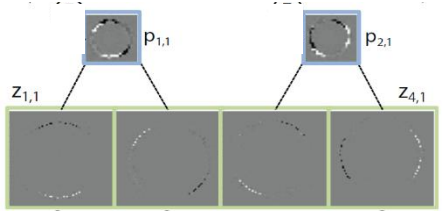
ℓ_1

Max / Softmax

Local Contrast Normalization (Subtractive & Divisive)

Not an exact separation

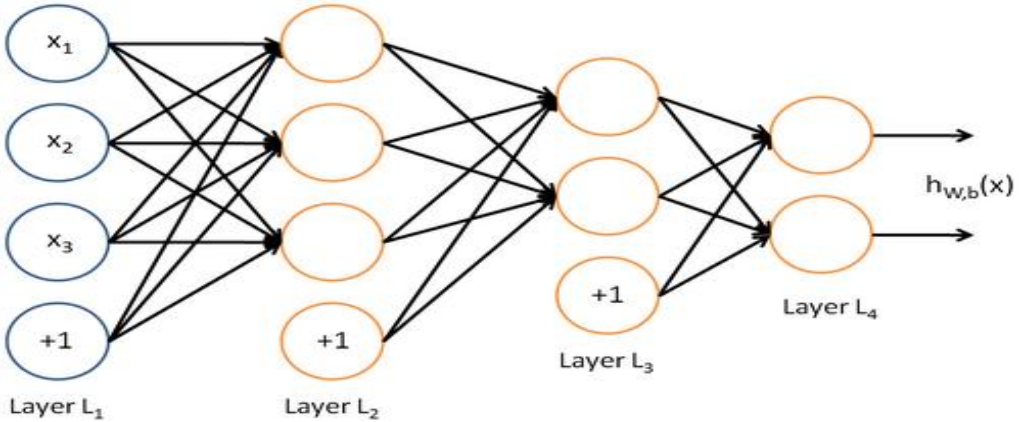
Spatial/Feature
(Sum or Max)



Features

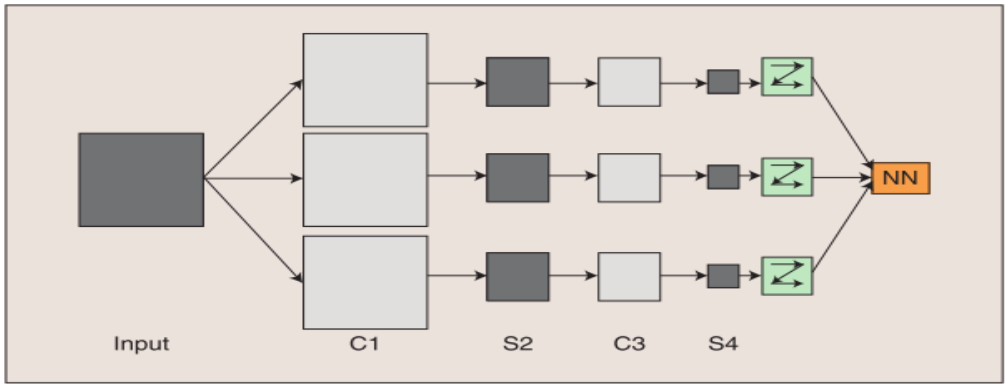
Multi Layer Neural Network

- A neural network = running several **logistic** regressions at the same time;
  - Neuron=logistic regression or…
- Calculate error derivatives (gradients) to refine: **back propagate** the error derivative through model (the chain rule)
  - **Online** learning: stochastic/incremental gradient descent;
  - **Batch** learning: conjugate gradient descent.



Convolutional Neural Networks (CNN)

- CNN is a special kind of multi-layer NNs applied to 2-d arrays (usually images), based on spatially localized neural input;
  - local receptive fields(shifted window), shared weights (weight averaging) across the hidden units, and often, spatial or temporal sub-sampling;
  - Related to generative MRF/discriminative CRF:
    - **CNN=Field of Experts MRF=ML inference in CRF**;
  - Generate 'patterns of patterns' for pattern recognition.
- Each layer combines (merge, smooth) patches from previous layers
  - Pooling /Sampling (e.g., max or average) filter: compress and smooth the data.
  - Convolution filters: (translation invariance) unsupervised;
  - **Local contrast normalization**: increase sparsity, improve optimization/invariance.



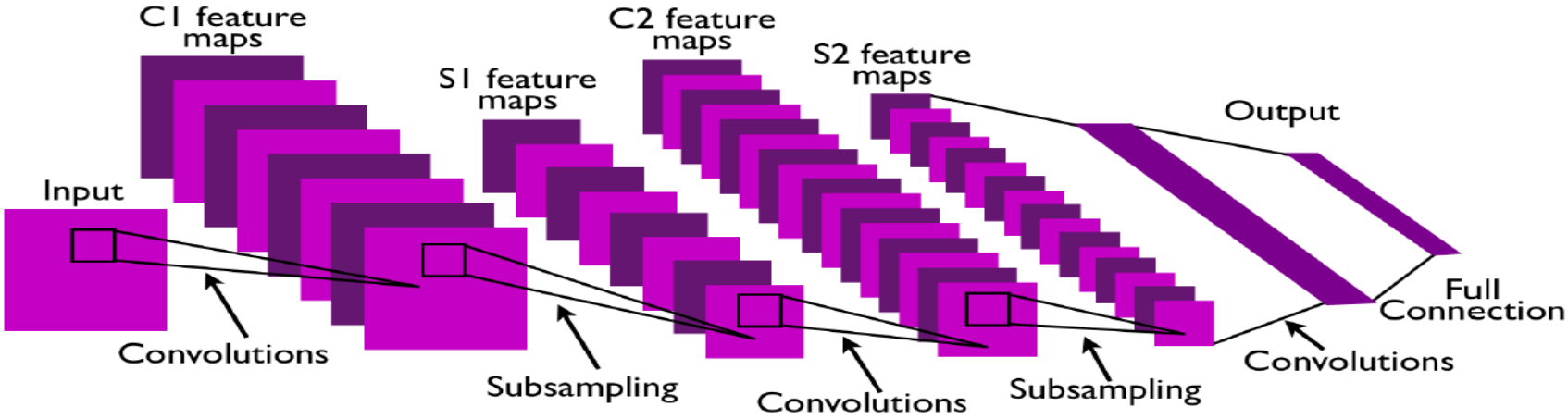
C layers convolutions,
S layers pool/sample

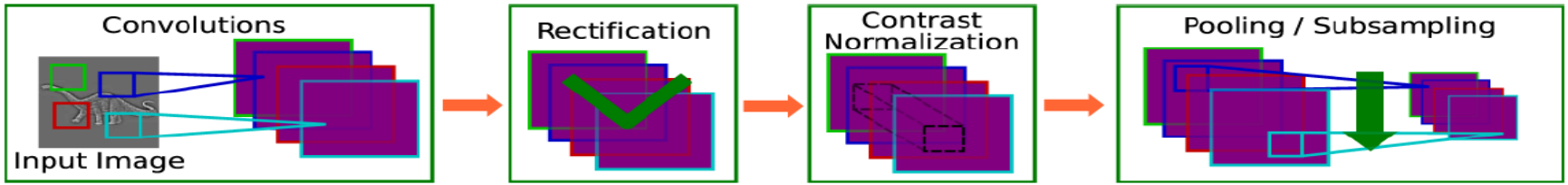
Convolutional Neural Networks (CNN)

- Convolutional Networks are trainable multistage architectures composed of multiple stages;
- Input and output of each stage are sets of arrays called *feature maps;*
- At output, each feature map represents a particular feature extracted at all locations on input;
- Each stage is composed of: a filter bank layer, a non-linearity layer, and a feature pooling layer;
- A ConvNet is composed of 1, 2 or 3 such 3-layer stages, followed by a classification module;
  - A fully connected layer: softmax transfer function for posterior distribution.
- Filter: A trainable filter (kernel) in filter bank connects input feature map to output feature map;

$$y_j = b_j + \sum_i k_{ij} * x_i$$ * is discrete convolution operator

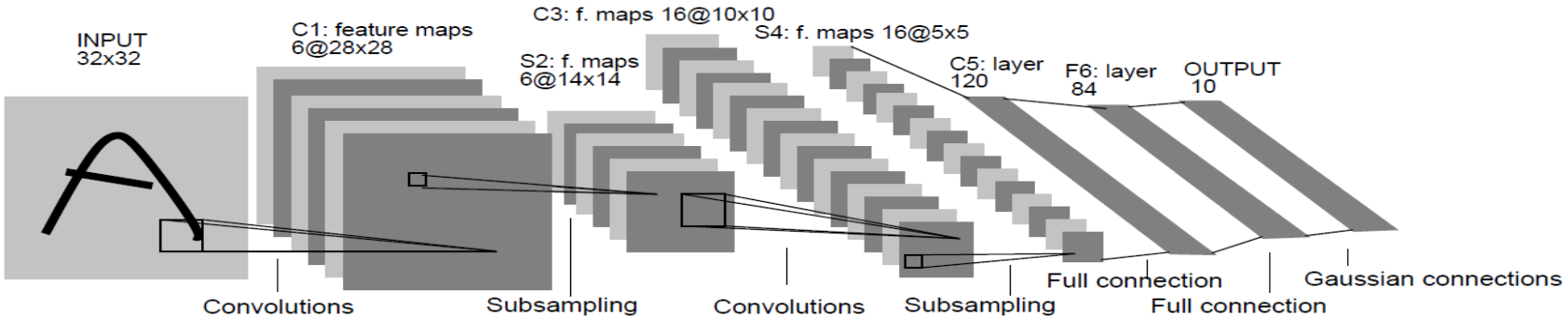
- Nonlinearity: a pointwise sigmoid tanh() or a rectified sigmoid abs(g_i•tanh()) function;
  - In rectified function, g_i is a trainable gain parameter, might be followed a contrast normalization N;
- Feature pooling: treats each feature map separately -> a reduced-resolution output feature map;
- Supervised training is performed using a form of SGD to minimize the prediction error;
  - Gradients are computed with the back-propagation method.
- Unsupervised pre-training: predictive sparse decomposition (PSD), then supervised fine-tuning.


C1 feature maps
S1 feature maps
C2 feature maps
S2 feature maps
Output
Input
Full Connection
Convolutions
Subsampling
Convolutions
Subsampling
Convolutions


Convolutions
Input Image
Rectification
Contrast Normalization
Pooling / Subsampling

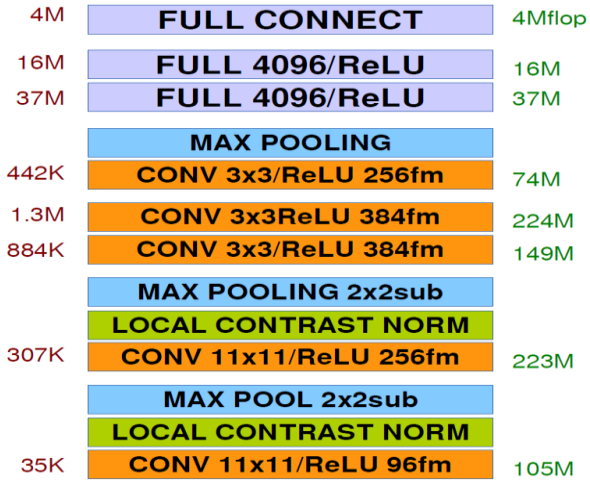
LeNet (LeNet-5)

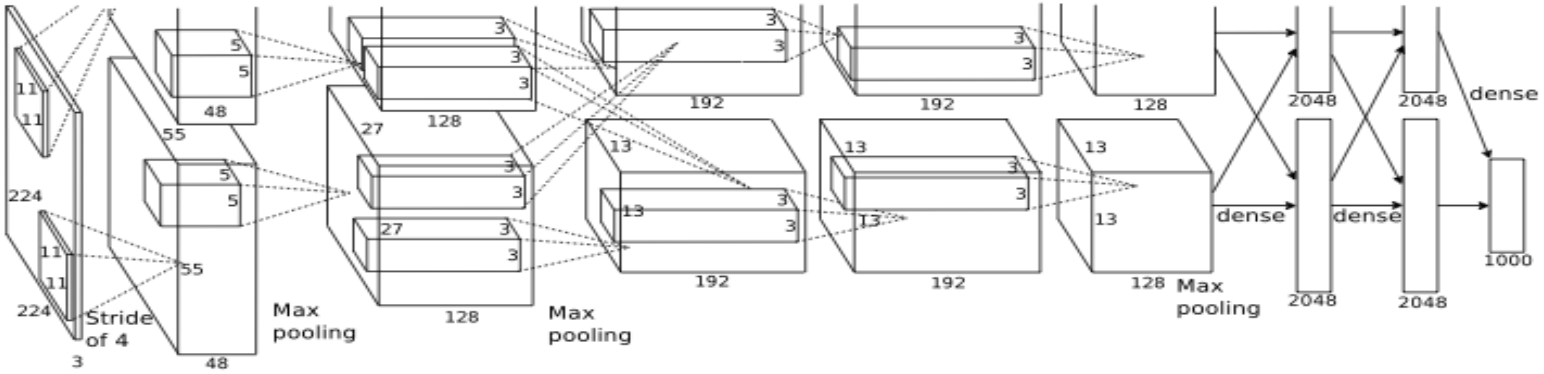
- A layered model composed of convolution and subsampling operations followed by a holistic representation and ultimately a classifier for handwritten digits;
- Local receptive fields (5x5) with local connections;
- Output via a RBF function, one for each class, with 84 inputs each;
- Learning by Graph Transformer Networks (GTN);



AlexNet

- A layered model composed of convol., subsample., followed by a holistic representation and all-in-all a landmark classifier;
- Consists of 5 convolutional layers, some of which followed by max-pooling layers, 3 fully-connected layers with a final 1000-way softmax;
- Fully-connected “FULL” layers: linear classifiers/matrix multiplications;
- ReLU are rectified-linear nonlinearities on layer output, can be trained several times faster;
- Local normalization scheme aids generalization;
- Overlapping pooling slightly less prone to overfitting;
- Data augmentation: artificially enlarge the dataset using label-preserving transformations;
- Dropout: setting to zero output of each hidden neuron with prob. 0.5;
- Trained by SGD with batch # 128, momentum 0.9, weight decay 0.0005.

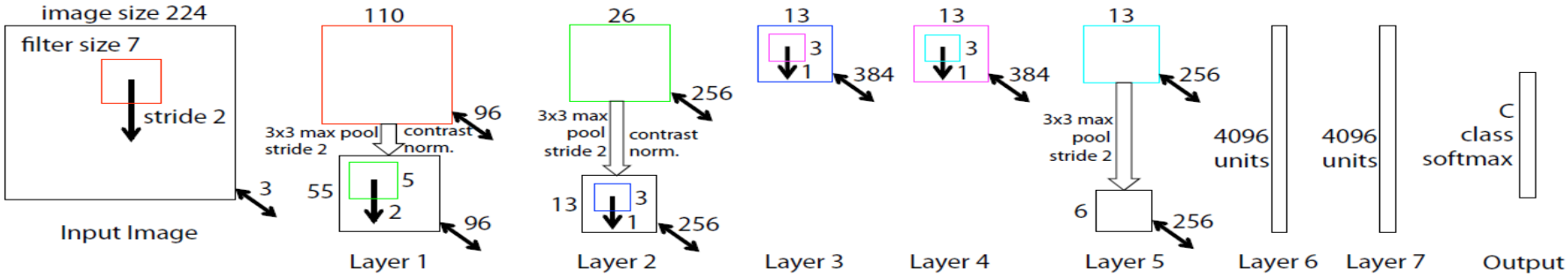




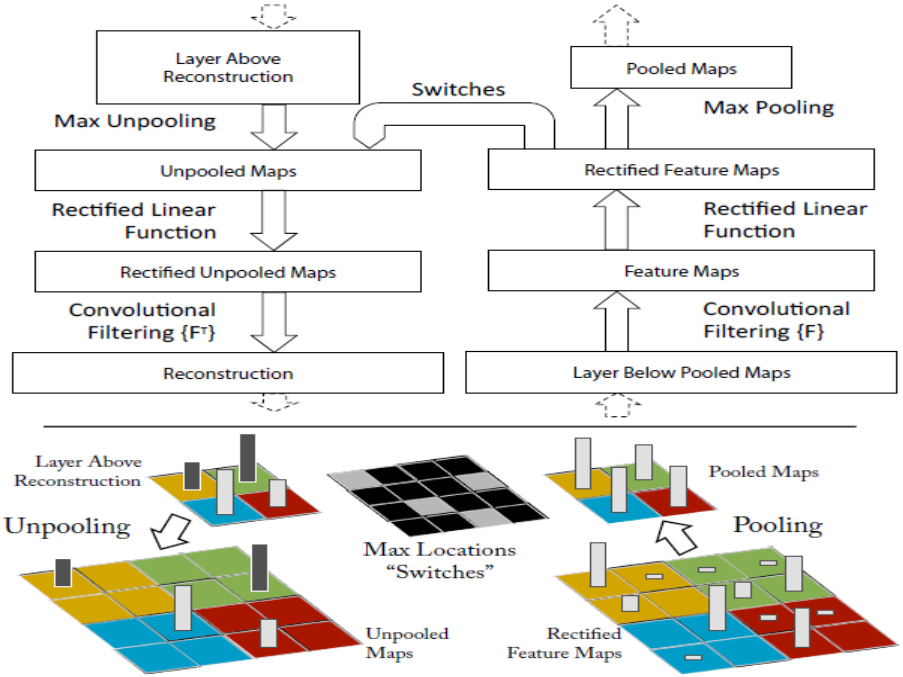
The network’s input is 150,528-dimensional, and the number of neurons in the network’s remaining layers is given by 253,440–186,624–64,896–64,896–43,264-4096–4096–1000.

MattNet

- Matthew Zeiler from the startup company “Clarifai”, winner of ImageNet Classification in 2013;
- Preprocessing: subtracting a per-pixel mean;
- Data augmentation: downsampled to 256 pixels and a random 224 pixel crop is taken out of the image and randomly flipped horizontally to provide more views of each example;
- SGD with min-batch # 128, learning rate annealing, momentum 0.9 and dropout to prevent overfitting;
- 65M parameters trained for 12 days on a single Nvidia GPU;
- Visualization by layered DeconvNets: project the feature activations back to the input pixel space;
  - Reveal input stimuli exciting individual feature maps at any layer;
  - Observe evolution of features during training;
  - Sensitivity analysis of the classifier output by occluding portions to reveal which parts of scenes are important;
- DeconvNet attached to each of ConvNet layer, unpooling uses locations of maxima to preserve structure;
- Multiple such models were averaged together to further boost performance;
- Supervised pre-training with AlexNet, then modify it to get better performance (error rate 14.8%).



Architecture of an eight layer ConvNet model. Input: 224 by 224 crop of an image (with 3 color planes). # 1-5 layers Convolution: 96 filters, 7x7, stride of 2 in both x and y. Feature maps: (i) via a rectified linear function, (ii) 3x3 max pooled (stride 2), (iii) contrast normalized 55x55 feature maps. # 6-7 layers: fully connected, input in vector form (6x6x256 = 9216 dimensions). The final layer: a C-way softmax function, C - number of classes.

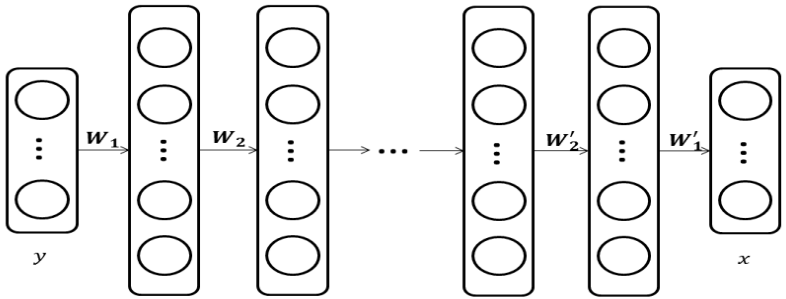


Top: A deconvnet layer (left) attached to a convnet layer (right). The deconvnet will reconstruct approximate version of convnet features from the layer beneath.

Bottom: Unpooling operation in the deconvnet, using switches which record the location of the local max in each pooling region (colored zones) during pooling in the convnet.

Stacked Denoising Auto-Encoder

- Denoising Auto-Encoder: Multilayer NNs with target output=input;
- Auto-encoder learns the salient variation like a nonlinear PCA;
- Stack many (may be sparse) auto-encoders in succession and train them using **greedy layer-wise unsupervised learning**
    - Drop the decode layer each time
    - Performs better than stacking RBMs**;**
- **Supervised training** on the last layer using final features;
- (option) Supervised training on the entire network to fine- tune all weights of the neural net;
- Empirically not quite as accurate as DBNs.



RNN: Recurrent Neural Network

- A nonlinear dynamical system that maps sequences to sequences;
- Parameterized with three weight matrices and three bias vectors;
- RNNs are fundamentally difficult to train due to their nonlinear iterative nature;
  - The derivative of the loss function can be exponentially large with respect to the hidden activations;
  - RNN suffers also from the vanishing gradient problem.
- Back Propagation Through Time (**BPTT**):
  - "Unfold" the recurrent network in time, by stacking identical copies of the RNN, and redirecting connections within the network to obtain connections between subsequent copies;
  - It's hard to be used where online adaption is required as the entire time series must be used.
- Real-Time Recurrent Learning (**RTRL**) is a forward-pass only algorithm that computes the derivatives of the RNN w.r.t. its parameters at each timestep;
  - Unlike BPTT, RTRL maintains the exact derivative of the loss so far at each timestep of the forward pass, without a backward pass and the need to store the past hidden states;
  - However, the computational cost of RTRL is prohibitive and more memory than BPTT as well.
- Success in Application: Speech Recognition and Handwriting recognition.

LSTM: Long Short-Term Memory

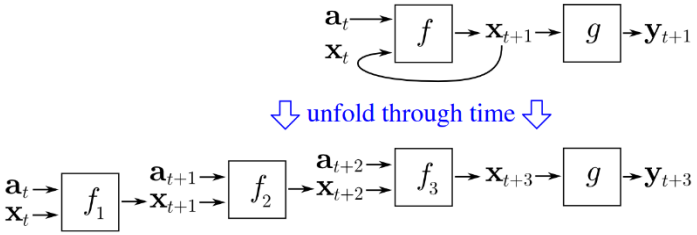
- An RNN structure that elegantly addresses the vanishing gradients problem using “memory units”;
- These linear units have a self-connection of strength 1 and a pair of auxiliary “gating units” that control the flow of information to and from the unit;
- Let N be the number of memory units of the LSTM. At each timestep t, the LSTM maintains a set of vectors as below, whose evolution is governed by the following equations:

variable name	description
i_t^g	$[0,1]^N$-valued vector of input gates
i_t	$[-1,1]^N$-valued vector of inputs to the memory units
o_t	$[0,1]^N$-valued vector of output gates
f_t	$[0,1]^N$-valued vector of the forget gates
v_t	$\mathbb{R}^v$-valued input vector
h_t	$[-1,1]^h$-valued conventional hidden state
m_t	$\mathbb{R}^N$-valued state of the memory units
$\tilde{m}_t$	$\mathbb{R}^N$-valued memory state available to the rest of the LSTM
z_t	the output vector

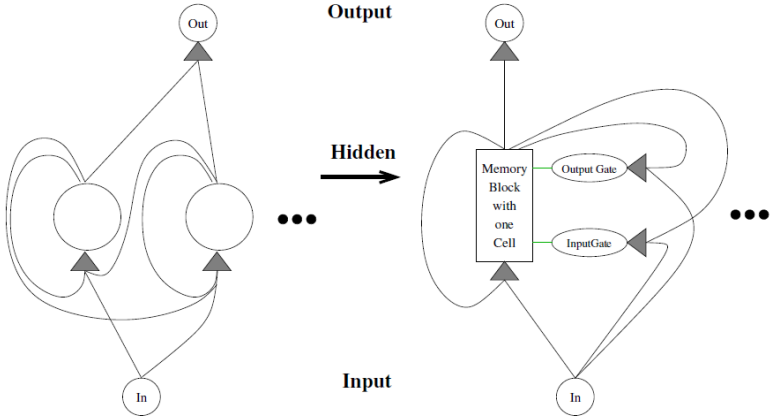
$$
\begin{aligned}
h_t &= \tanh(W_{hh}h_{t-1} + W_{hv}v_t + W_{hm}\tilde{m}_{t-1}) \\
i_t^g &= \text{sigmoid}(W_{igh}h_t + W_{igv}v_t + W_{igm}\tilde{m}_{t-1}) \\
i_t &= \tanh(W_{ih}h_t + W_{iv}v_t + W_{im}\tilde{m}_{t-1}) \\
o_t &= \text{sigmoid}(W_{oh}h_t + W_{ov}v_t + W_{om}\tilde{m}_{t-1}) \\
f_t &= \text{sigmoid}(b_f + W_{fh}h_t + W_{fv}v_t + W_{fm}\tilde{m}_{t-1}) \\
m_t &= m_{t-1} \odot f_t + i_t \odot i_t^g \quad \text{the input gate allows the memory unit to be updated} \\
\tilde{m}_t &= m_t \odot o_t \quad \text{the output gate determines if information can leave the unit} \\
z_t &= g(W_{yh}h_t + W_{ym}\tilde{m}_t)
\end{aligned}
$$

- ... are highly complex, making them tedious to implement;
- Note: Theano has LSTM module.

LSTM: Long Short-Term Memory



From Simple RNN to BPTT



Left: RNN with one fully connected hidden layer;
Right: LSTM with memory blocks in hidden layer.

Gated Recurrent Unit

- GRU is a variation of RNN, adaptively capturing dependencies of different time scales with each recurrent unit;
- GRU uses gating units as well to modulate the flow of information inside the unit, but without a memory cells.
- GRU doesn't control degree to which its state is exposed, but exposes the whole state each time;
- Different from LSTM:

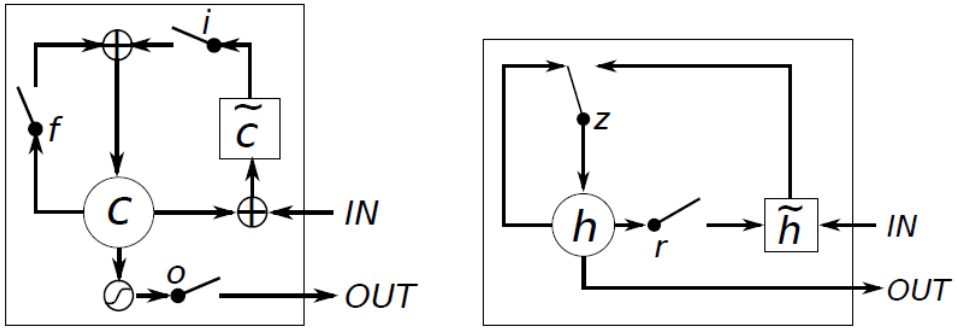
sequence $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_T)$

activation $h_t^j = (1 - z_t^j) h_{t-1}^j + z_t^j \tilde{h}_t^j$, update gate $z_t^j = \sigma\left(W_z \mathbf{x}_t + U_z \mathbf{h}_{t-1}\right)^j$

candidate activation $\tilde{h}_t^j = \tanh\left(W \mathbf{x}_t + U\left(\mathbf{r}_t \odot \mathbf{h}_{t-1}\right)\right)^j$ reset gate $r_t^j = \sigma\left(W_r \mathbf{x}_t + U_r \mathbf{h}_{t-1}\right)^j$

  - GRU expose its full content without control;
  - GRU controls the information flow from the previous activation when computing the new, candidate activation, but does not independently control the amount of the candidate activation being added (the control is tied via the update gate).

- Shared virtues with LSTM: the additive component of their update from t to t + 1;
  - Easy for each unit to remember the existence of a specific feature in the input stream for a long series of steps;
  - Effectively creates shortcut paths that bypass multiple temporal steps, which allow the error to be back-propagated easily without too quickly vanishing.



(a) Long Short-Term Memory (b) Gated Recurrent Unit

Stochastic Gradient Descent (SGD)

- The general class of estimators that arise as minimizers of sums are called M-estimators;
  - Where are stationary points of the likelihood function (or zeroes of its derivative, the score function)?
- Online gradient descent samples a subset of summand functions at every step;
  - The true gradient of is approximated by a gradient at a single example;
  - Shuffling of training set at each pass.
- There is a compromise between two forms, often called "mini-batches", where the true gradient is approximated by a sum over a small number of training examples.
- STD converges almost surely to a global minimum when the objective function is convex or pseudo-convex, and otherwise converges almost surely to a local minimum.

Back Propagation

- **Back propagation** is a multi-layer network training method
  - We find parameters W, to minimize an error $E(f(x_0,w),y_0) = -\log(f(x_0,w)- y_0)$.
  - For this we will do iterative gradient descent:

$$w(t) = w(t-1) - \lambda * \frac{-\partial E}{\partial w}(t)$$

  - Error propagation
    - Forward propagation of a training pattern's input through the multilayer network to generate the output activations;
    - Backward propagation of the output activations (logistic or soft-max) through the multiplayer network using the pattern target to generate deltas of all output and hidden units (**the chain rule**);

$$\frac{\partial E}{\partial y_{l-1}} = \frac{\partial E}{\partial y_l} \times \frac{\partial y_l(w,y_{l-1})}{\partial y_{l-1}}$$
$$\frac{\partial E}{\partial w_l} = \frac{\partial E}{\partial y_l} \times \frac{\partial y_l(w,y_{l-1})}{\partial w_l}$$

  - Weight update
    - Multiply its output delta and input activation to get the weight gradient;
    - Subtract a ratio (i.e. the learning rate) of the gradient from the weight.

Loss function

- Euclidean loss is used for regressing to real-valued lables [-inf,inf];
- Sigmoid cross-entropy loss is used for predicting K independent probability values in [0,1];
- Softmax (normalized exponential) loss is predicting a single class of K mutually exclusive classes;
  - Generalization of the logistic function that "squashes" a *K*-dimensional vector of arbitrary real values z to a *K*-dimensional vector of real values σ(z) in the range (0, 1).

$$\sigma(\mathbf{z})_j = \frac{e^{\mathbf{z}_j}}{\sum_{k=1}^{K} e^{\mathbf{z}_k}} \quad \text{for } j=1,\ldots,K.$$

  - The predicted probability for the *j*'th class given a sample vector **x** is $\mathbf{x} \mapsto \mathbf{x}^{\mathrm{T}}\mathbf{w}_1, \ldots, \mathbf{x} \mapsto \mathbf{x}^{\mathrm{T}}\mathbf{w}_K$

$$P(y = j|\mathbf{x}) = \frac{e^{\mathbf{x}^{\mathrm{T}}\mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^{\mathrm{T}}\mathbf{w}_k}}$$

- Sigmoidal or Softmax normalization is a way of reducing the influence of extreme values or outliers in the data without removing them from the dataset.

Variable Learning Rate

- Too large learning rate
  - cause oscillation in searching for the minimal point
- Too slow learning rate
  - too slow convergence to the minimal point
- Adaptive learning rate
  - At the beginning, the learning rate can be large when the current point is far from the optimal point;
  - Gradually, the learning rate will decay as time goes by.
- Should not be too large or too small:
  - annealing rate $\alpha(t)=\alpha(0)/(1+t/T)$
  - $\alpha(t)$ will eventually go to zero, but at the beginning it is almost a constant.

Variable Momentum

- Classical Momentum (CM) is a technique for accelerating gradient descent that accumulates a velocity vector in directions of persistent reduction in the objective across iterations: given the objective function $f(\theta)$,

$$V_{t+1} = \mu V_t - \epsilon \nabla f(\theta_t), \quad \theta_{t+1} = \theta_t + V_{t+1},$$

  With $\epsilon > 0$ as learning rate, $\mu \in [0,1]$ as momentum coefficient and $\nabla f(\theta_t)$ as gradient at θ_t;
- Nesterov's Accelerated Gradient (NAG) is also a 1st order optimization method with better convergence rate guarantee than gradient descent;

$$V_{t+1} = \mu V_t - \epsilon \nabla f(\theta_t + \mu V_t), \quad \theta_{t+1} = \theta_t + V_{t+1},$$

- For convex objectives, momentum-based methods outperform SGD in the early or transient stages of optimization, however equally effective in the final stage;
- Hessian-free (HF) methods and truncated Newton methods work by optimizing a local quadratic model of the objective via the linear conjugate gradient (CG) algorithms;
  - If CG terminated after just one step, HF becomes equivalent to NAG;

AdaGrad/AdaDelta

- AdaGrad: asymptotically sublinear regret, adapt learning rate for each weight based on historical info.:

$$\Delta W_{ij}(t+1) = -\frac{\gamma}{\sqrt{\sum_1^{t+1}(\frac{\partial E}{\partial w_{ij}}(\tau))^2}} * \frac{\partial E}{\partial w_{ij}}(t+1)$$

  - Normalizes each coordinate of gradient by the historical (previous iterations) magnitude of that coordinate;
  - Frequently occurring features in the gradients get small learning rates and infrequent features get higher ones;
  - Sensitive to initial conditions, continual decay of learning rate.
- AdaDelta: accumulate the denominator over last k gradients (a sliding window):

$$\alpha(t+1) = \sum_{t-k+1}^{t+1}(\frac{\partial E}{\partial w}(\tau))^2$$

$$\Delta W(t+1) = -\frac{\gamma}{\sqrt{\alpha(t+1)}} * \frac{\partial E}{\partial w}(t+1) \ .$$

  - This requires to keep last k gradients; instead it use a simpler formula:

$$\beta(t+1) = \rho * \beta(t) + (1-\rho) * (\frac{\partial E}{\partial w}(t+1))^2$$

$$\Delta W(t+1) = -\frac{\gamma}{\sqrt{\beta(t+1)+\epsilon}} * \frac{\partial E}{\partial w}(t+1) \ .$$

  - Avoid AdaGrad's weakness.

Data Augmentation for Overfitting

- The easiest and most common method to **reduce overfitting** on image data is to artificially **enlarge the dataset** using label-preserving transformations;
- Perturbing an image I by transformations that leave the underlying class unchanged (e.g. cropping and flipping) in order to generate additional examples of the class;
- Two distinct forms of data augmentation:
  - image translation
  - horizontal reflections
  - changing RGB intensities

Weight Decay for Overfitting

- Weight decay or L2 regularization adds a penalty term to the error function, a term called the regularization term: the negative log prior in Bayesian justification, $C = C_0 + \frac{\lambda}{2n} \sum_w w^2$
  - Weight decay works as rescaling weights in the learning rule, but bias learning still the same;
  - Prefer to learn small weights, and large weights allowed if improving the original cost function;
  - A way of compromising btw finding small weights and minimizing the original cost function;
- In a linear model, weight decay is equivalent to ridge (Tikhonov) regression;
- L1 regularization: the weights not really useful shrink by a constant amount toward zero;
  - Act like a form of feature selection;
  - Make the input filters cleaner and easier to interpret;

  $C = C_0 + \frac{\lambda}{n} \sum_w |w|.$
- L2 regularization penalizes large values strongly while L1 regularization ;
- Markov Chain Monte Carlo (MCMC): simulating a Markov chain whose equilibrium distr. is the posterior distribution for weights & hyper-parameters;
- Hybrid Monte Carlo: gradient and sampling.

Early Stopping for Overfitting

- Steps in early stopping:
  - Divide the available data into training and validation sets.
  - Use a large number of hidden units.
  - Use very small random initial values.
  - Use a slow learning rate.
  - Compute the validation error rate periodically during training.
  - Stop training when the validation error rate "starts to go up".
- Early stopping has several advantages:
  - It is fast.
  - It can be applied successfully to networks in which the number of weights far exceeds the sample size.
  - It requires only one major decision by the user: what proportion of validation cases to use.
- Practical issues in early stopping:
  - How many cases do you assign to the training and validation sets?
  - Do you split the data into training and validation sets randomly or by some systematic algorithm?
  - How do you tell when the validation error rate "starts to go up"?

Dropout and Maxout for Overfitting

- Dropout: set the output of each hidden neuron to zero w.p. 0.5.
  - Motivation: Combining many different models that share parameters succeeds in reducing test errors by approximately averaging together the predictions, which resembles the bagging.
  - The units which are “dropped out” in this way do not contribute to the forward pass and do not participate in back propagation.
  - So every time an input is presented, the NN samples a different architecture, but all these architectures share weights.
  - This technique reduces complex co-adaptations of units, since a neuron cannot rely on the presence of particular other units.
  - It is, therefore, forced to learn more robust features that are useful in conjunction with many different random subsets of the other units.
  - Without dropout, the network exhibits substantial overfitting.
  - Dropout roughly doubles the number of iterations required to converge.
- Maxout takes the maximum across multiple feature maps;

Thanks!