Real-time Vision-based Depth Reconstruction with NVidia Jetson

Andrey Bokovoy[1,2], Kirill Muravyev[1,3] and Konstantin Yakovlev[1,3]

***Abstract*— Vision-based depth reconstruction is a challenging problem extensively studied in computer vision but still lacking universal solution. Reconstructing depth from single image is particularly valuable to mobile robotics as it can be embedded to the modern vision-based simultaneous localization and mapping (vSLAM) methods providing them with the metric information needed to construct accurate maps in real scale. Typically, depth reconstruction is done nowadays via fully-convolutional neural networks (FCNNs). In this work we experiment with several FCNN architectures and introduce a few enhancements aimed at increasing both the effectiveness and the efficiency of the inference. We experimentally determine the solution that provides the best performance/accuracy tradeoff and is able to run on NVidia Jetson with the framerates exceeding 16FPS for 320×240 input. We also evaluate the suggested models by conducting monocular vSLAM of unknown indoor environment on NVidia Jetson TX2 in real-time. Open-source implementation of the models and the inference node for Robot Operating System (ROS) are available at `https://github.com/CnnDepth/tx2_fcnn_node`.**

I. INTRODUCTION

Depth reconstruction (estimation) is one of the important problems in mobile robotics, augmented reality, computer aided design etc. The sensors that explicitly provide range measurements such as LIDARs, RGB-D cameras etc., are typically i) expensive, ii) large and heavy, iii) power-demanding, which prevents their widespread usage especially when it comes down to compact mobile robots (like small drones). Thus a strong interest exists in depth estimation using a single camera, as almost every mobile robot is equipped with this sensor. Moreover, there exist data-driven learning-based approaches that are capable of solving monocular vision-based depth reconstruction tasks with suitable (for typical mobile robotics applications) accuracy – see works [1], [2], [3]. Commonly, the main focus of such papers is increasing the accuracy while the performance issues are left out of scope. As a result, the majority of the state-of-the-art methods for depth reconstruction are very resource demanding and need high-performance graphic processing units (GPU) to work in real time. Thus, they are not suitable for creating a fully-autonomous robotic system equipped with a typical embedded computer, even the one that is particularly suitable for image processing with neural networks

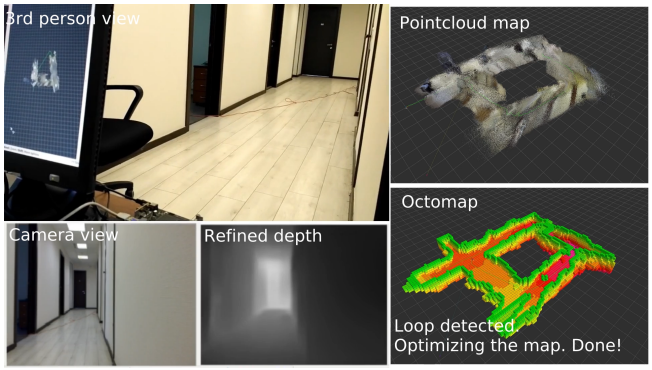

Fig. 1. Monocular vSLAM based on FCNN depth reconstruction and running in real time on NVidia Jetson TX2. This is a screenshot of the video available at: `https://youtu.be/ayjvfzm-C7s`

– NVidia Jetson TX2. On the other hand, there are plenty of reports of this embedded computer being successfully used for autonomous navigation, SLAM etc., but still there is a limited number of papers, e.g. [4], that report a successful usage of single camera deep-learning driven depth estimation that works in real-time on NVidia Jetson TX2. Furthermore, to the best of our knowledge, there are no reproducible results (in terms of open-source code) of FCNNs for real-time embedded vSLAM usage. The foregoing defines the scope of this work. We wish to present a CNN-based depth reconstruction method that i) is accurate enough to be used within the monocular vSLAM pipeline and is equivalent accuracy-wise to the state-of-the-art, ii) is fast enough to work in real time on NVidia Jetson TX2, iii) is open to the community, i.e. comes with a source-code of the ROS-node.

II. RELATED WORK

A. Depth reconstruction from single image

Depth estimation from single image has a prolonged history of studies. Initially such techniques as image pre-processing, feature extraction, edge detection etc. were widely utilized to solve the task. In [5] authors use hardware modification of the camera's lens to make simultaneous image and depth extraction with cost-efficient algorithm. Proposed method exploits the prior knowledge about real images, particularly their statistical distribution [6]. However, in most cases, resultant depth maps require manual correction.

In [7] Markov Random Field (MRF) is used for patch-based depth reconstruction with single image. The original image is divided into a set of patches in different scales. The hand-crafted features are then applied to these patches. Using the statistics from different scales of the patches, MRF

[1]All authors are with the Artificial Intelligence Research Institute, Federal Research Center "Computer Science and Control" of Russian Academy of Sciences, Moscow, Russia.

[2]Andrey Bokovoy is also with the Peoples Friendship University of Russia (RUDN University), Moscow, Russia.

[3]Kirill Muravyev and Konstantin Yakovlev are also with the Moscow Institute of Physics and Technology, Dolgoprudny, Russia.

Corresponding author is Andrey Bokovoy: `bokovoy@isa.ru`

978-1-7281-3605-9/19/$31.00 ©2019 IEEE

models the relationship between the depth of the patch and the neighborhood patches, reconstructing the depth of the whole image.

Not only the knowledge about image statistics may provide the ability to reconstruct the depth map of the single image, but the prior information about the environment. For example, in [8] the information that indoor environment mostly consists of vertical and horizontal lines (walls, floor and etc.) is utilized. This helps to determine the perspective and reconstruct the depth in scenarios where, for example, the robot moves along the corridor. However, it fails in other types of environments.

Recently, deep learning and convolutional neural networks became tools of choice for depth reconstruction as they significantly outperform methods based on hand-crafted features. Fully-convolutional neural networks [9], consisting of encoder and decoder, are the most common architectures to be used for depth reconstruction.

One of the pioneer works in CNN-based depth estimation from single image is [10]. Authors use coarse-scale network to predict the depth of the image at global level (as a dense depth map). Then, local fine-scale network aligns the map with image's local details, such as objects or wall edges.

In [1] authors utilize inverse depth maps, produced by neural network, with known inter-view displacement to achieve unsupervised learning of the CNN, solving the problem of collecting large datasets for training. The training set consists of only 22 600 stereo images, without any data augmentation or usage of pretrained decoder (in this particular case - AlexNet [11]).

In [12] the original up-convolution algorithm is proposed as well as the reverse Huber loss function [13]. This architecture was tested for real-time applications, but the targeted GPU is NVidia GeForce GTX TITAN with 12GB of GPU memory, which is more powerful compared to NVidia Jetson. Notably, this FCNN is used within the SLAM pipeline presented in [14] on desktop PC with Intel Xeon CPU at 2.4GHz with 16GB of RAM and a Nvidia Quadro K5200 GPU with 8GB of VRAM.

Authors of MegaDepth [15] and DenseDepth [16] focus mainly on improvement of FCNN learning phase. In [15] an original loss function is proposed as well as learning strategy and data augmentation techniques. MegaDepth authors focus on improving the quality of the training dataset. They also suggest routines for depth refinement with automatic ordinal labeling and semantic segmentation. Both architectures are too heavy for real-time image processing, especially on embedded systems.

In general, vast majority of the FCNN-depth-reconstruction papers leaves computing constraints out of the scope. We, in contrary, wish to focus on real-time performance on embedded computers that are widely used in modern robotics.

B. Depth reconstruction on embedded systems

One of the most popular embedded computer nowadays for robotics is Raspberry Pi. It is used for autonomous car navigation [17], grid-based path planning [18], object detection [19] etc. Despite the low-power CPU and the absence of GPU, Raspberry is even suitable for some CNN-based tasks. E.g. in [20] object recognition was considered and simple and light architectures were used on par with the Movidius Neural Compute Stick hardware acceleration. Framerates of 0.5-1.5 FPS were achieved. In this work we are targeting framerates of $>$5FPS.

The next popular and more powerful embedded system for robotics is ODROID. In [21][22] it was reported to be used as an on-board computer for semi-dense visual odometry on small quadrotor. Framerate of 5FPS was achieved. In general, the lack of acceleration tools for ODROID forces usage of external hardware for CNN processing.

Finally, some of the platforms were produced recently with deep learning tasks in mind. For example, NVidia Jetson is used for real-time GPU accelerated image processing tasks, including FCNN semantic segmentation [23], object detection [24], image classification [25] etc. Low power consumption and compact size of Jetson make this computer perfectly suitable for mobile robotics and the availability of GPU makes it the research platform of our choice. There is a successful report of running FCNN for depth reconstruction using NVidia Jetson TX2 in real-time [4]. In this work the authors introduce a light-weight encoder-decoder architecture, that was trained with knowledge transfer from a more heavy one. They achieved 30 FPS inference with comparable to state-of-the-art accuracy. However, the vSLAM application is not well-studied (only the scale-drift problem) and the results are not reproducible (in terms of code, TensorRT engine's binaries and etc.). In this work we provide an open-source solution for depth reconstruction and vSLAM.

III. Evaluated architectures

With the focus on on real-time performance under limited computational resources we study different variations of Fully-convolutional Neural Networks for depth reconstruction (see Fig. 2).

Stereotypical FCNN model for depth reconstruction consists of the encoder and the decoder. The former extracts the high-level features from the input image while the latter generates the depth maps from these features. We used ResNet50 (and its cropped version) as the encoder and a few different decoders. On top of that we enhance some blocks of the network to make it work faster while keeping the accuracy at the appropriate level. By combining different encoders, decoders and enhancements we end up with 6 different architectures to evaluate.

A. Encoder

ResNet50 [26] is known to be versatile feature extractor, so we chose it as the encoder. Despite being a deep (50 layers) network with several residual blocks, it's fast enough to operate in real-time. The output of standard ResNet50 for 640x480x3 input is 20x15x2048 feature maps. We further denote this encoder as **Basic**.

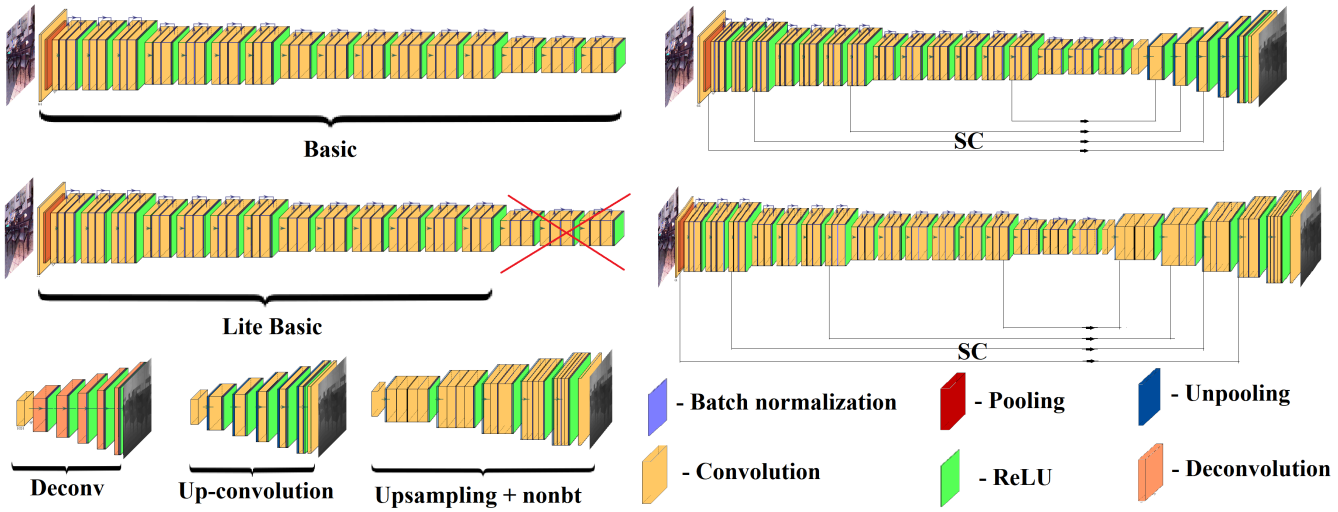

Fig. 2. Visualization of evaluated network architectures.

We also evaluate a light version of ResNet50 which lacks the last stack of residual blocks, so the output for 640x480x3 image is 30x40x1024. This greatly increases the performance of the network while keeping the accuracy at the appropriate level (i.e. sufficient for vSLAM purposes). The cropped version of ResNet50 is referred as **Lite Basic**.

B. Decoder

As the baseline decoder we use the one that is composed of 5 deconvolution blocks (**Deconv**). Each block consists of 5x5 deconvolution + batch normalization + activation (ReLU).

Second, we evaluate the decoder that is composed of the blocks that use the upsampling followed by the non-bottleneck convolution followed by the ReLU. After 5 such blocks we reduce the output to the one channel, i.e. – depth, with 5×5 convolution. The upsampling is performed with the nearest neighbour algorithm. It is followed by the two 3×3 convolutions implemented as the factorized non-bottleneck block, suggested in [27]. It substitutes the conventional 3×3 convolution with a series of 3×1 and 1×3 convolutions that results in faster inference. We denote this decoder as **Upsampling + nonbt**.

Third, we use the **up-convolution** decoder. Each block of this decoder is composed of unpooling + 5×5 convolution + batch normalization + ReLU + Dropout. In total 5 blocks are stacked when the **Basic** encoder is used, 4 – in case of **Lite Basic**.

It was shown in [12] that one can substitute the unpooling + 5×5 convolution with the smaller sized convolutions that are interleaved into the resultant feature map – see Fig. 3. Such approach leads to a faster inference although splitting the 5×5 convolution into 4 parts leads to inconsistent gradients and worse weights optimization during training. To overcome this we suggest using original up-convolution decoder during learning and then transferring the learned weights to the faster architecture (the one that utilizes interleaving). We denote **interl** the unpooling decoder that relies on interleaving at both learning and inference. We denote **interl + T** the decoder with the original (non-interleaved) unpooling + convolution layers used for training and interleaved layers (with the transferred weights) for inference.

C. Shortcuts

We also use shortcuts (or skip connections, referred as **SC**) as the projection from encoder layers to decoder. Despite it does not always improves model's accuracy, it produces more sharpened depths on the objects' edges. For different combinations of encoder-decoder architectures we use different shortcuts. For **Basic + SC** encoder with **Upsampling + nonbt** decoder, we connect the output of last convolution block in every stack of the encoder with respective outputs of decoder blocks (see Fig.2). For **Basic + SC + interl** there are 2 shortcuts that connects the outer and middle blocks of encoder and decoder.

D. Interleaving implementation for faster up-convolution

Interleaving is an approach proposed in [12] that substitutes un-poolling + 5×5 convolution with 4 convolutions which outputs are interleaved into a single feature map. This operation is equivalent in terms of the resultant output but is more computationally efficient.

Our implementation of interleaving differs from the one residing in the authors' repository[1] in the following way: while authors of the original method apply interleaving as a 3-step operation, merging first and second pairs of weights, then merging the results together, we do it in a single operation (see Alg. 1) with respect to multithreading, saving CPU to GPU memory copy time[2].

[1] https://github.com/iro-cp/FCRN-DepthPrediction

[2] Code for Tensorflow and TensorRT is available at https://github.com/CnnDepth/interleave_op.

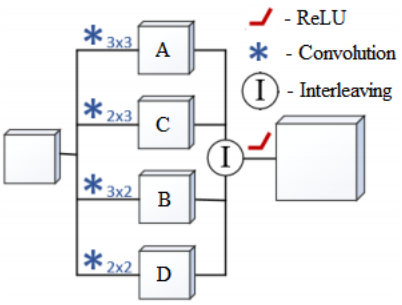

Fig. 3. Faster up-convolution block architecture.

Data: A, B, C, D - input 4D tensors in (N,H,W,C) format, N - batch size, H - image height, W - image width, C - number of channels
Result: Out - interleaved output 4D tensor in (N,H,W,C) format
for $i \leftarrow indexOfElementInBlock$ **do**
  $n_{in} = i/(H * W * C)$;
  $h_{in} = (i \mod (H * W * C))/(W * C)$;
  $w_{in} = (i \mod (W * C)/C))$;
  $c_{in} = (i \mod C)$;
  $index_{in} = n_{in} * H * W * C/4 + (h_{in}/2) * W * C/2 + (w_{in}/2) * C + c_{in}$;
  if $h_{in} iseven$ **then**
    if $w_{in} iseven$ **then**
      $Out[i] = A[index_{in}]$;
    else
      $Out[i] = B[index_{in}]$;
    end
  else
    if $w_{in} iseven$ **then**
      $Out[i] = C[index_{in}]$;
    else
      $Out[i] = D[index_{in}]$;
    end
  end
end

Algorithm 1: Interleaving implementation

E. Loss functions

Running preliminary experiments we discovered, that errors on near and far predicted depths deviate significantly from the mean error. The far-away pixels are less important in the context of autonomous navigation of a mobile robot [28], but the incorrect estimation of depths in the vicinity of the camera may lead to undesirable outcomes (e.g. crashing into the obstacle). To mitigate this issue we suggest using 2 original loss functions.

The first one is a combination of two losses:

$$MSE + REL = \alpha_1 \cdot MSE + \alpha_2 \cdot REL,$$
$$MSE = \frac{1}{H \cdot W} \sum_{i=1}^{H} \sum_{j=1}^{W} (D_{ij}^* - D_{ij})^2,$$
$$REL = \frac{1}{H \cdot W} \sum_{i=1}^{H} \sum_{j=1}^{W} \left(1 - \frac{D_{ij}}{D_{ij}^*}\right)^2,$$

D_{ij}^* – ground-truth pixel depth, D_{ij} – predicted pixel depth, α_1, α_2 – user-defined parameters (we set them to 1 and 2 respectively in our experiments).

The second component of that loss (REL) accounts for the fact that, e.g. the 0.5m error at the distance of 1m is worse than the 0.5m error at the distance 10m.

Another loss function we used is the the BerHu loss [2]:

$$L(D_{ij}^*, D_{ij}) = \begin{cases} |D_{ij}^* - D_{ij}|, & |D_{ij}^* - D_{ij}| < k \\ \dfrac{(D_{ij}^* - D_{ij})^2 + k^2}{2k}, & |D_{ij}^* - D_{ij}| \geq k \end{cases}$$
$$BerHu = \frac{1}{H \cdot W} \sum_{i=1}^{H} \sum_{j=1}^{W} L(D_{ij}^*, D_{ij}).$$

BerHu loss accounts for the same proposition – the model should be more sensitive to the errors within the close range. It needs to be provided with the threshold, k, that accounts for what is "near" and what is "far" and that threshold is fixed during learning. On contrast, we suggest to alter the value of k in the following fashion. At each step during learning phase, we additionally compute two BerHu losses over the pixels lying at depths $[k - \delta; k]$ and $[k; k + \delta]$ and compare them numerically. Then k is shifted towards bigger mean error by $k \pm lr * \delta$.[3]. This allows us to adaptively adjust the closeness threshold while training. In our work, variables δ and lr were set to 1 and 0.01 respectively as initial values. We refer to this function as **aBerHu**.

IV. Experimental results

The considered FCNN architectures were implemented using Tensorflow [30] + Keras [31] frameworks in Python for. Custom interleaving and up-convolution layers were implemented both for CPU and GPU using C/C++ with g++ and nvcc compilers respectively. Learning was performed on the Hybrid high-performance computing cluster of Federal Research Center Computer Science and Control of Russian Academy of Sciences.

For inference tests we consider 2 possible scenarios: 1) fully autonomous depth reconstruction on NVidia Jetson TX2, 2) remote depth reconstruction with mobile PC (laptop). For both scenarios we use accelerated TensorRT framework. All the inference-related part was written in C/C++ using tools available in JetPack software package. All models and respective learned weights are converted to inference engine used by TensorRT. Our PC platform specification is as follows: Intel Core i7 8550, 20GB of RAM, NVidia MX150 GPU with 4GB of memory.

A. Dataset

NYU dataset v2 [32] was used. It consists of more that 400 000 image-depth pairs taken from more then 470 scenes. All raw images were aligned with corresponding depth maps and pre-processed with bilateral filter to fill the missing depth values on the edges of objects. Since there are a lot of similar images in NYU Dataset, we performed random crop for each

[3]Code for adaptive BerHu loss is available at `https://github.com/CnnDepth/fcrn_notebooks`.

TABLE I

EVALUATION OF LOSS FUNCTION FOR NETWORK ARCHITECTURES PRESENTED IN SECTION III. THE VALUES ARE THOSE ORIGINALLY REPORTED BY THE AUTHORS IN THEIR RESPECTIVE PAPER.

	Loss	Input resolution	Decoder	MSE	REL	δ^1	δ^2	δ^3	PC time (s)	Jetson time (s)
Wang et al. [29]	Custom	-	-	0.555	0.220	0.605	0.890	0.970	-	-
Eigen et al. [10]	Custom	304x228	-	0.823	0.215	0.611	0.887	0.971	-	-
Laina et al. [12]	BerHu	304x228	-	0.328	0.127	0.811	0.953	0.988	-	-
Alhashim et al. [16]	Custom	640x480	-	0.152	0.103	0.895	0.980	0.996	-	-
Basic	BerHu	640x480	Deconv	0.467	0.186	0.718	0.929	0.980	0.144	0.152
Basic + SC	BerHu	640x480	Deconv	0.487	0.194	0.695	0.915	0.975	0.521	0.563
Basic + SC	aBerHu	640x480	Upsampling + nonbt	0.440	0.184	0.725	0.932	0.982	0.158	0.215
Basic + SC	MSE + REL	640x480	Upsampling + nonbt	0.419	**0.173**	**0.748**	**0.944**	**0.987**	0.158	0.215
Basic + SC	MSE + REL	320x240	Upsampling + nonbt	**0.408**	0.180	0.746	0.940	0.984	0.049	0.062
Lite Basic + SC	MSE + REL	320x240	Upsampling + nonbt	0.533	0.202	0.687	0.915	0.979	0.035	0.049
Basic + SC + interl	MSE + REL	640x480	Up-convolution	0.514	0.206	0.708	0.912	0.970	0.285	0.328
Basic + SC + interl	MSE + REL	320x240	Up-convolution	0.580	0.215	0.673	0.899	0.965	0.057	0.067
Basic + SC + interl + T	MSE + REL	640x480	Up-convolution	0.445	0.178	0.714	0.939	**0.987**	0.181	0.227
Basic + SC + interl + T	MSE + REL	320x240	Up-convolution	0.495	0.181	0.724	0.940	0.983	0.048	0.061
Lite Basic + interl + T	MSE + REL	640x480	Up-convolution	0.658	0.233	0.642	0.886	0.964	0.101	0.135
Lite Basic + interl + T	MSE + REL	320x240	Up-convolution	0.660	0.236	0.649	0.881	0.960	**0.027**	**0.037**

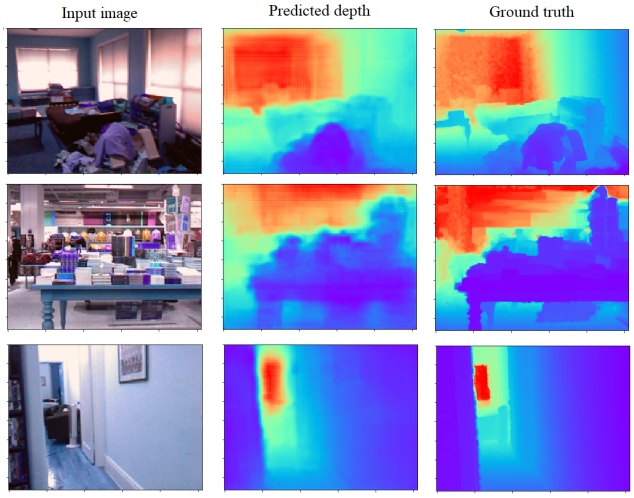

Fig. 4. Visualization of introduced FCNN on NYU Dataset v2.

image-depth pairs, as well as random mirroring and rotations, which led to diversification of dataset.

B. Error metrics

We used the following metrics to measure the accuracy of the depth estimation:

- MSE – mean squared error;
- REL – mean relative error;
- Threshold accuracy δ^i – % of predicted depths D_i* : $\frac{1}{N}\sum_{i=1}^{N} max\left(\frac{D_i*}{D_i}, \frac{D_i}{D_i*}\right) < \delta^i, \delta = 1.25$

C. Results

The results are presented in Table. I. As the best result, we achived 37ms per image inference on NVidia Jetson TX2 for **Lite Basic + interl + T** architecture. The inference speed is comparable to the state-of-the art method for depth-reconstruction with embedded platforms from [4], while the accuracy is slightly worse, but applicable for real-time vSLAM purposes. On the other hand, we've achieved better REL error metrics then some basic architectures presented in [12], [33] and [10], that are focused on offline depth reconstruction, with **Basic + SC** encoder and **Upsampling + nonbt** decoder. Our tests showed, that even 62ms is enough for real-time vSLAM, but odometry may fail during fast translations due to low update rate of the camera images.

Evidently, using the **Lite Basic** encoder gives a notable inference speed improvement, but at the cost of the reduced accuracy.

D. vSLAM evaluation

We implemented the node for Robot Operating System (ROS) for FCNN inference. For localization and mapping we use RTAB-Map [34]. We run both FCNN inference and RTAB-Map on NVidia Jetson TX2. Our experiments show that developed networks are well suited for accurate and fast single-camera simultaneous localization and mapping on embedded platform. The model that provides the best performance/accuracy trade-off is **Basic + SC** encoder paired with **upsampling + nonbt** decoder. It runs on NVidia Jetson's GPU with 16 FPS, while vSLAM operates on CPU, so both algorithms don't interfere (the overall performance is approx. 12FPS). We open-source our FCNN inference implementation with all engines compiled for NVidia Jetson: `https://github.com/CnnDepth/tx2_fcnn_node`. The pipeline may be extended with other methods and algorithms available for ROS.

As shown in Fig. 1, our FCNN inference + RTAB-Map is able to produce high-detailed maps of unknown environment with only single camera. In our previous work [35] we used state-of-the-art feature-based SLAM approach with the enhanced post-processing and still were not able to obtain maps of such quality. The video of the vSLAM evaluation is available at `https://youtu.be/ayjvfzm-C7s`.

V. CONCLUSION AND FUTURE WORK

In this work we have evaluated different FCNN architectures for depth-reconstruction both on PC and NVidia Jetson TX2. We demonstrated, that the proposed models are able to run in real-time with the accuracy comparable to the state-of-the-art. We implemented the proposed methods as a part of ROS framework and tested it with RTAB-Map for the indoor SLAM. The results show that our pipeline is able to produce dense maps on embedded computer in real time.

In future we wish to more thoroughly evaluate the proposed FCNNs within the vSLAM pipeline (e.g. map and pose accuracy estimation), and further use the produced maps for autonomous navigation, e.g. for path planning.

ACKNOWLEDGMENT

This work is supported by the RSF project #16-11-00048 (developing FCNN architectures and evaluating them) and by the "RUDN University Program 5-100" (post-processing of the experimental results).

REFERENCES

[1] R. Garg, V. K. BG, G. Carneiro, and I. Reid, "Unsupervised cnn for single view depth estimation: Geometry to the rescue," in *European Conference on Computer Vision*. Springer, 2016, pp. 740–756.

[2] B. Li, C. Shen, Y. Dai, A. Van Den Hengel, and M. He, "Depth and surface normal estimation from monocular images using regression on deep features and hierarchical crfs," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1119–1127.

[3] C. Godard, O. Mac Aodha, and G. J. Brostow, "Unsupervised monocular depth estimation with left-right consistency," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 270–279.

[4] A. Spek, T. Dharmasiri, and T. Drummond, "Cream: Condensed real-time models for depth prediction using convolutional neural networks," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 540–547.

[5] A. Levin, R. Fergus, F. Durand, and W. T. Freeman, "Image and depth from a conventional camera with a coded aperture," *ACM transactions on graphics (TOG)*, vol. 26, no. 3, p. 70, 2007.

[6] B. A. Olshausen and D. J. Field, "Natural image statistics and efficient coding," *Network: computation in neural systems*, vol. 7, no. 2, pp. 333–339, 1996.

[7] A. Saxena, S. H. Chung, and A. Y. Ng, "3-d depth reconstruction from a single still image," *International journal of computer vision*, vol. 76, no. 1, pp. 53–69, 2008.

[8] E. Delage, H. Lee, and A. Y. Ng, "A dynamic bayesian network model for autonomous 3d reconstruction from a single indoor image," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, vol. 2. IEEE, 2006, pp. 2418–2428.

[9] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.

[10] D. Eigen, C. Puhrsch, and R. Fergus, "Depth map prediction from a single image using a multi-scale deep network," in *Advances in neural information processing systems*, 2014, pp. 2366–2374.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[12] I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari, and N. Navab, "Deeper depth prediction with fully convolutional residual networks," in *2016 Fourth international conference on 3D vision (3DV)*. IEEE, 2016, pp. 239–248.

[13] A. B. Owen, "A robust hybrid of lasso and ridge regression," *Contemporary Mathematics*, vol. 443, no. 7, pp. 59–72, 2007.

[14] K. Tateno, F. Tombari, I. Laina, and N. Navab, "Cnn-slam: Real-time dense monocular slam with learned depth prediction," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 6243–6252.

[15] Z. Li and N. Snavely, "Megadepth: Learning single-view depth prediction from internet photos," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2041–2050.

[16] I. Alhashim and P. Wonka, "High quality monocular depth estimation via transfer learning," *arXiv preprint arXiv:1812.11941*, 2018.

[17] G. S. Pannu, M. D. Ansari, and P. Gupta, "Design and implementation of autonomous car using raspberry pi," *International Journal of Computer Applications*, vol. 113, no. 9, 2015.

[18] A. Andreychuk, A. Bokovoy, and K. Yakovlev, "An empirical evaluation of grid-based path planning algorithms on widely used in robotics raspberry pi platform," in *The 2018 International Conference on Artificial Life and Robotics (ICAROB 2018)*, 2018, pp. 383–386.

[19] V. Pereira, V. A. Fernandes, and J. Sequeira, "Low cost object sorting robotic arm using raspberry pi," in *2014 IEEE Global Humanitarian Technology Conference-South Asia Satellite (GHTC-SAS)*. IEEE, 2014, pp. 1–6.

[20] D. Pena, A. Forembski, X. Xu, and D. Moloney, "Benchmarking of cnns for low-cost, low-power robotics applications," in *RSS 2017 Workshop: New Frontier for Deep Learning in Robotics*, 2017.

[21] C. Forster, M. Pizzoli, and D. Scaramuzza, "Svo: Fast semi-direct monocular visual odometry," in *2014 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2014, pp. 15–22.

[22] M. Faessler, F. Fontana, C. Forster, E. Mueggler, M. Pizzoli, and D. Scaramuzza, "Autonomous, vision-based flight and live dense 3d mapping with a quadrotor micro aerial vehicle," *Journal of Field Robotics*, vol. 33, no. 4, pp. 431–450, 2016.

[23] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello, "Enet: A deep neural network architecture for real-time semantic segmentation," *arXiv preprint arXiv:1606.02147*, 2016.

[24] M. J. Shafiee, B. Chywl, F. Li, and A. Wong, "Fast yolo: a fast you only look once system for real-time embedded object detection in video," *arXiv preprint arXiv:1709.05943*, 2017.

[25] R. LiKamWa, Y. Hou, J. Gao, M. Polansky, and L. Zhong, "Redeye: analog convnet image sensor architecture for continuous mobile vision," in *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3. IEEE Press, 2016, pp. 255–266.

[26] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[27] E. Romera, J. M. Alvarez, L. M. Bergasa, and R. Arroyo, "Erfnet: Efficient residual factorized convnet for real-time semantic segmentation," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 1, pp. 263–272, 2018.

[28] E. Magid and E. Rivlin, "Cautiousbug: a competitive algorithm for sensory-based robot navigation," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, Sep. 2004, pp. 2757–2762 vol.3.

[29] P. Wang, X. Shen, Z. Lin, S. Cohen, B. Price, and A. L. Yuille, "Towards unified depth and semantic prediction from a single image," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 2800–2809.

[30] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 2016, pp. 265–283.

[31] A. Gulli and S. Pal, *Deep Learning with Keras*. Packt Publishing Ltd, 2017.

[32] P. K. Nathan Silberman, Derek Hoiem and R. Fergus, "Indoor segmentation and support inference from rgbd images," in *ECCV*, 2012.

[33] F. Liu, C. Shen, G. Lin, and I. Reid, "Learning depth from single monocular images using deep convolutional neural fields," *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 10, pp. 2024–2039, 2016.

[34] M. Labbé and F. Michaud, "Rtab-map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation," *Journal of Field Robotics*, vol. 36, no. 2, pp. 416–446, 2019.

[35] A. Bokovoy and K. Yakovlev, "Sparse 3D point-cloud map upsampling and noise removal as a vSLAM post-processing step: Experimental evaluation," in *Proceedings of the 3rd International Conference on Interactive Collaborative Robotics (ICR-2018)*. Springer, 2018, pp. 23–33.