Depth from Monocular Images using a Semi-Parallel Deep Neural Network (SPDNN) Hybrid Architecture

Shabab Bazrafkan¹, Student, IEEE, Hossein Javidnia¹, Student, IEEE, Joseph Lemley², Student, IEEE, Peter Corcoran³, Fellow, IEEE

Abstract

Computing pixel depth values provide a basis for understanding the 3D geometrical structure of an image. As it has been presented in recent research, using stereo images provides an accurate depth due to the advantage of having local correspondences; however, the processing time of these methods are still an open issue. To solve this problem, it has been suggested to use single images to compute the depth values but extracting depth from monocular images requires extracting a large number of cues from the global and local information in the image. This challenge has been studied for a decade and it is still an open problem. Recently the idea of using neural networks to solve this problem has attracted attention. In this paper, we tackle this challenge by employing a Deep Neural Network (DNN) equipped with semantic pixel-wise segmentation utilizing our recently published disparity post-processing method. Four models are trained in this study and they have been evaluated at 2 stages on KITTI dataset. The ground truth images in the first part of the experiment come from the benchmark and for the second part, the ground truth images are considered to be the disparity results from applying a state-of-art stereo matching method. The results of this evaluation demonstrate that using post-processing techniques to refine the target of the network increases the accuracy of depth estimation on individual mono images. The second evaluation shows that using segmentation data as the input can improve the depth estimation results to a point where performance is comparable with stereo depth matching.

Index Terms: Depth Estimation, Deep Convolutional Neural Networks (CNN), Monocular Image

1. Introduction

1.1 Depth Map

Deriving the 3D structure of an object from a set of 2D points is a fundamental problem in computer vision. Most of these conversions from 2D to 3D space are based on the depth values computed for each 2D point. In a depth map, each pixel is defined as the distance between an object visible in the scene and the camera, rather than a colour.

In general depth computation methods are divided into 2 categories:
1. Active methods
2. Passive methods

Active methods involve computing the depth in the scene by interacting with the objects and the environment. There are different types of active methods such as Light based depth estimation which uses the active light illumination to estimate the distance of different objects. Ultrasound based methods is another example of the active method which is quite similar to the time of flight (ToF) method that uses the known speed of light to measure the time an emitted pulse of light takes to arrive to an image sensor.

Passive methods are based on the optical features of the captured images. Passive methods involve extracting the depth information by computational image processing.

Generally, in the category of active methods there are 2 classical techniques: a) Multi-view depth estimation such as depth from stereo and b) Monocular depth estimation.

The research work presented here was funded under the Strategic Partnership Program of Science Foundation Ireland (SFI) and co-funded by SFI and FotoNation Ltd. Project ID: 13/SPP/I2868 on “Next Generation Imaging for Smartphone and Embedded Platforms”. This work is also supported by an Irish Research Council Employment Based Programme Award. Project ID: EBPPG/2016/280

1 S. Bazrafkan is with the Department of Electronic Engineering, College of Engineering, National University of Ireland, Galway, University Road, Galway, Ireland. (e-mail: s.bazrafkan1@nuigalway.ie).
1 H. Javidnia is with the Department of Electronic Engineering, College of Engineering, National University of Ireland, Galway, University Road, Galway, Ireland. (e-mail: h.javidnia1@nuigalway.ie).
2 J. Lemley is with the Department of Electronic Engineering, College of Engineering, National University of Ireland, Galway, University Road, Galway, Ireland. (e-mail: j.lemley2@nuigalway.ie).
3 P. Corcoran is with the Department of Electronic Engineering, College of Engineering, National University of Ireland, Galway, University Road, Galway, Ireland. (e-mail: peter.corcoran@nuigalway.ie).
1.2 Stereo Vision Depth

Stereo matching algorithms can be used to compute depth information from multiple images. By using the calibration information of the cameras, the depth images can be generated. This depth information provides useful data to identify and detect objects in the scene [1].

In recent years, many applications, including time-of-flight [2, 3], structured light [4], and the Kinect were introduced to calculate depth from stereo images. Stereo vision algorithms are generally divided into 2 categories: Local and Global. Local algorithms were introduced as statistical methods that use the local information around a pixel to determine the depth value of the given pixel. These kinds of methods can be used for real-time applications if they are implemented efficiently. Global algorithms try to optimize an energy function to satisfy the depth estimation problem through various optimization techniques [5].

In terms of computation, global methods are more complex than local methods and they are usually impractical for real-time applications. Despite these drawbacks they have an important advantage; in terms of accuracy they are more accurate than local methods. This advantage recently attracted considerable attention in academic literature [6, 8].

A global stereo model is proposed in [6] works by converting the image into a set of 2D triangles with adjacent vertices. Later, the 2D vertices are converted to a 3D mesh by computing the disparity values. To solve the problem of depth discontinuities, a two layer Markov Random Field (MRF) is employed. The layers are fused with an energy function allowing the method to handle the depth discontinuities. The method has been evaluated on the new Middlebury 3.0 benchmark [7] and it was ranked the first by the time of the paper’s publication in terms of the overall accuracy based on the average weight on bad 2.0 index.

Another global stereo matching algorithm, proposed in [8], makes use of texture and edge information of the image. The problem of large disparity differences in small patches in non-textured regions is tackled by using the colour intensity. Also the main matching cost produced by a CNN is augmented using the same colour based cost. The final results are post-processed using a 5x5 median filter and a bilateral filter. This adaptive smoothness filtering technique provides good performance for the algorithm which made it the first in the Middlebury 3.0 benchmark [7] at the time of the paper’s publication.

Many other methods have been proposed for stereo depth such as PMSC [7], GCSVR [7], INTS [9], MDP [10], ICSG [11] which all tried to improve the accuracy of the depth from stereo vision or bring a new method to estimate the depth from a stereo pair. However there is always a trade-off between accuracy and speed for stereo vision algorithms.

Table 1 shows an overview of the average normalized time by number of pixels (sec/megapixels) between most accurate stereo matching algorithms ranked in Middlebury 3.0 benchmark based on the “bad 2.0” metric. The ranking is on the test dense set. This comparison illustrates that “obtaining an accurate depth from a stereo pair requires significant processing power”. These results demonstrate that, today, these methods are too resource intensive for real-time applications like street sensing or autonomous navigation due to their demand for processing resources.

To decrease the processing power of stereo matching algorithms, researchers recently began to work on depth from monocular images. Such algorithms estimate depth from a single camera while keeping the processing power low. The next section gives further details about current research into depth from monocular images.

| Algorithm       | Time/MP (s) | W x H (n disp) | Programming Platform | Hardware                  |
|-----------------|-------------|----------------|----------------------|---------------------------|
| PMSC [7]        | 453         | 1500 x 1000 (<400) | C++                  | i7-6700K, 4GTX TITAN X    |
| MeshStereoExt [6] | 121         | 1500 x 1000 (<400) | C, C++               | 8 Cores-NVIDIA TITAN X    |
| APAP-Stereo [7] | 97.2        | 1500 x 1000 (<400) | Matlab+Mex           | i7 Core 3.5GHz, 4 Cores   |
| NTDE [8]        | 114         | 1500 x 1000 (<400) | n/a                  | i7 Core, 2.2 GHz Geforce GTX TITAN X |
| MC-CNN-act [12] | 112         | 1500 x 1000 (<400) | n/a                  | NVIDIA GTX TITAN Black    |
| MC-CNN+RBS [13] | 140         | 1500 x 1000 (<400) | C++                  | Intel(R) Xeon(R) CPU ES-1650 0, 3.2GHz, 6 Cores-32GB RAM-NVIDIA GTX TITAN X |
| SNP-RSM [7]     | 258         | 1500 x 1000 (<400) | Matlab               | i5, 4590 CPU, 3.3 GHz     |
| MCCCCN_Layout [7] | 262        | 1500 x 1000 (<400) | Matlab               | i7 Core, 3.5GHz           |
| MC-CNN-6x [12] | 1.26        | 1500 x 1000 (<400) | n/a                  | NVIDIA GTX TITAN X        |
| LPU [7]         | 3523        | 1500 x 1000 (<400) | Matlab               | Core i5, 4 Cores-2xGTX 970 |
| MDP [10]        | 58.5        | 1500 x 1000 (<400) | n/a                  | 4 i7 Cores, 3.4 GHz       |
| MeshStereo [6]  | 54          | 1500 x 1000 (<400) | C++                  | i7-2600, 3.4GHz, 8 Cores  |
| SOU4P-net [7]   | 678         | 1500 x 1000 (<400) | n/a                  | i7 Core, 3.2GHz GTX 980   |
| INTS [9]        | 127         | 1500 x 1000 (<400) | C, C++               | i7 Core, 3.2 GHz          |
1.3 Deep Learning

DNN (Deep Neural Networks) are one of the most recent approaches in pattern recognition science which are able to handle highly non-linear problems in classification and regression. These models use consecutive non-linear signal processing units in order to mix and re-orient their input data to give the most representative results in the output. The DNN structure learns from the input that it encounters and then it generalizes what it learns into the data samples it has never seen before [16]. The typical deep neural network model is made from several convolutional layers followed by pooling and fully connected layers accompanied by different regularization tasks. Each of these units is as follows:

**Convolutional Layer:** This layer typically convolves the 3D image “I” with the 4D kernel “W” and adds a 3D bias term “b” to it. The output is given by:

\[ P = I * W + b \] (1)

where * operator is nD convolution. During the training process, the kernel and bias parameters are updated in a way that optimizes the error function of the network output.

**Pooling Layer:** The pooling layer applies a (usually) non-linear transform (Note that the average pooling is a linear transform, but the more popular max-pooling operation is non-linear) on the input image which reduces the spatial size of the data representation after the operation. It is common to put a pooling layer between two consecutive convolutional layers. Reducing the spatial size leads to less computational load and also prevents over-fitting. The reduced spatial size also provides a certain amount of translation invariance.

**Fully Connected Layer:** Fully connected layers are exactly same as classical Neural Network (NN) layers where all the neurons in a layer are connected to all the neurons in their subsequent layer. The neurons give the summation of their input multiplied by their weights passed through their activation functions.

**Regularization:** In general, regularization is proposed to prevent the overfitting inside the network. One can train a more complex network (more parameters) with regularization and prevent over-fitting. Different kinds of regularization have been proposed. The most important ones are weight regularization, drop-out [17], and batch normalization [18]. Each regularization technique is suitable for one or more specific applications and no one technique works for every task.

### Table: Deep Learning Results

| Method      | Input Size | Language | GPU/Processor       |
|-------------|------------|----------|---------------------|
| GCSVR [7]   | 1500 x 1000 <= 400 | C++ | i7 Core, 2.8GHz-Nvidia GTX 660Ti |
| JMR [7]     | 1500 x 1000 <= 400 | C++ | Core i7, 3.6 GHz-GTX 980 |
| LCU [7]     | 750 x 500 <= 200 | Matlab, C++ | 1 Core Xeon CPU, E5-2690, 3.00 GHz |
| TMAP [14]   | 1500 x 1000 <= 400 | Matlab | i7 Core, 2.7GHz |
| SPS [7]     | 3000 x 2000 <= 800 | C, C++ | 1 i7 Core, 2.8GHz |
| IDR [15]    | 1500 x 1000 <= 400 | CUDA C++ | NVIDIA GeForce TITAN Black |

1.4 Monocular Vision Depth

Depth estimation from a single image is a fundamental problem in computer vision and has potential applications in robotics, scene understanding and 3D reconstruction. This problem remains challenging because there are no reliable cues for inferring depth from a single image. For example, temporal information and stereo correspondences are missing from such images.

As the result of the recent research, deep Convolutional Neural Networks (CNN) are setting new records for various vision applications. A deep convolutional neural field model for estimating depths from a single image has been presented in [19] by reformulating the depth estimation into a continuous conditional random field (CRF) learning problem. The CNN employed in this research was composed of 5 convolutional and 4 fully-connected layers. At the first stage of the algorithm the input image was over-segmented into superpixels. The cropped image patch centred on its centroid was used as input to the CNN. For a pair of neighbouring superpixels, a number of similarities were considered and was used as the input to the fully connected layer. The output of these 2 parts was then used as input to the CRF loss layer. As a result, the time required for estimating the depth from a single image using the trained model decreased to 1.1 seconds on a desktop PC equipped with a NVIDIA GTX 780 GPU with 6GB memory.

It has been found that the superpixelling technique of [19] is not a good choice to initialize the disparity estimation from mono images because of the lack of the monocular visual cues such as texture variations and gradients, defocus, colour/haze in some parts of the image. To solve this issue a MRF learning algorithm has been implemented to capture some of these monocular cues [20]. The captured cues were integrated with a stereo system to obtain better depth estimation than the stereo system alone. This method uses a fusion of stereo
+ mono depth estimation. At small distances, the algorithm relies more on stereo vision, which is more accurate than monocular vision. However, at further distances, the performance of stereo degrades; and the algorithm relies more on monocular vision.

1.5 Paper Overview

In this paper a hybrid DNN is presented to estimate depth from mono cameras based on the approach presented in [21]. The depth map from the stereo sets are estimated using the same approach as [21] and these they are used as the target to train the network while using information from a single image (the left image in stereo set) as input. Four models are trained and evaluated to estimate the depth from single camera images. These models are trained with two different inputs with one or two channels. In first case, the input is simply the original image. In the second case the first channel is the original image and the second channel is its segmentation map. For each of these two cases one of two different targets are used. Specifically, these targets were the stereo depth maps with post-processing or without post-processing as explained in [21]. The technical details of each model are presented in the next section. Fig. 1 shows the general overview of the trained models in this paper.

![Figure 1. The overview of the trained models in this paper](image)

2. Enabling Deep Learning Methodology used in this Work

2.1 A Brief Description of Our Post-Processing Method

In [21] a guided joint filter is presented which is based on the mutual information of the RGB image (used as a reference image) and the estimated depth image. This approach has been used to increase the accuracy of the depth estimation in stereo vision by preserving the edges and corners in the depth map and filling the missing parts. The method was compared with top 8 depth estimation methods in Middlebury benchmark [7] at the time the paper was authored. Seven metrics including Mean Square Error (MSE), Root Mean Square Error (RMSE), Peak Signal-to-Noise Ratio (PSNR), Signal-to-Noise Ratio (SNR), Mean Absolute Error (MAE), Structural Similarity Index (SSIM) and Structural Dissimilarity Index (DSSIM) were used to evaluate the performance of each method. The evaluation ranked the method as 1st in 5 metrics and 2nd and 3rd in other metrics.

The same approach is used in this paper to compute the depth from stereo set. Two of the models in this paper are using this method to compute the depth as one of the inputs of the network.

2.2 Semantic Pixel-Wise Image Segmentation

One of the principle ideas in this paper is to investigate how semantic segmentation can help a neural network to better learn and improve depth estimation in a 2D image scene. For this purpose SegNet [22, 23] is employed as the underlying DNN architecture.

SegNet is one of the most successful recent implementations of DNN for semantic pixel-wise image segmentation and has supplanted other configurations of Fully Convolutional Network (FCN) both by accuracy and simplicity in implementation. SegNet uses the convolutional layers of the VGG16 network as the encoder of the network and eliminates the fully connected layers thus reducing the number of trainable parameters from 134M to just 14.7M which represents a reduction of 90% in the number of parameters to be trained. The encoder portion of SegNet consists of 13 convolutional layers with ReLU nonlinearity followed by max-pooling (2x2 window) and stride 2 in order to implement a non-overlapping sliding window. This consecutive max-pooling and striding, results in a network configuration that is highly robust to translation in the input image but has the drawback of losing spatial resolution of the data.

This loss of spatial resolution is not beneficial in segmentation tasks where it is necessary to preserve the boundaries from the input image in the segmented output. To overcome this problem, the following solution has been considered in our work: as most of the spatial resolution information is lost in the max-pooling operation, saving the information of the max-pooling indices and using this information in the decoder part of the network preserves the high frequency information of the input image from the encoder part and passes them to the decoder where the un-pooling operation is applied to the image.

Note that for each layer in the encoder portion of the network there is a corresponding decoder layer. In other words, the idea of SegNet is that wherever max-pooling is applied to the input data, the indices of the feature space are saved at the maximum value. Therefore, for each position of the sliding window of max-pooling it just
stores 2 bits to save the index of the maximum value. Later these indices will be employed to make a sparse feature space before the de-convolution step in each layer while applying the un-pooling step in the decoder part. A batch normalization layer [18] is placed after each convolutional layer to avoid overfitting and to promote faster convergence. Decoder filter banks are not tied to corresponding encoder filters and are trained independently in the SegNet architecture.

Originally SegNet has been trained using Stochastic Gradient Descent (SGD) with learning rate 0.1 and momentum 0.9. In this paper, the Caffe implementation of SegNet has been employed for training purposes [24]. The gray-scale CamVid road scene database (360×480) [25] has been used in the training step

3. Methodology Enhancements - Proposed Approach

3.1 Semi-Parallel Deep Neural Network (SPDNN)

This paper is the first to introduce the SPDNN concept inspired from graph optimization techniques. In this method several deep neural networks are parallelized and merged in a novel way that facilitates the advantages of each. The final model is trained for the problem, our observations show that using this method reduces overfitting and gives provides superior convergence compared to alternatives.

The merging of multiple networks using SPDNN is next described in the context of the current depth mapping problem. In this particular problem 8 different networks were originally designed and trained for the given task. These are documented in Fig. 2 – Fig. 9 and will be described in the next section. None of these networks on their own gave useful results on the depth analysis problem. However it was noticed that each network tended to perform well in certain aspects, but lost out due to failures in other portions of an input image. This led to the idea that it might be advantageous to find a way to combine multiple individual networks in a way to facilitate training them in a parallelised architecture. In principle parallelizing them in a smart way and training them concurrently will force the networks to converge to better combined output.

3.2 Individual SegNet Networks for Depth Analysis

The network shown in Fig. 2 is a deep fully convolutional neural network (A fully convolutional neural network is a network wherein all the layers are convolutional layers) with no pooling and no padding. Therefore no information loss occurs inside the network since there is no bottleneck or data compression; this network is able to preserve the details of the input samples. But the main problem is that this model is unable to find big objects and coarse features of the image. In order to solve this problem three other networks have been designed as shown in Fig. 3 – Fig. 5. These three networks take advantage of the max-pooling layers to gain transition invariance and also to recognize bigger objects and coarser features inside the image. These networks use 2×2, 4×4 and 8×8 max-pooling operators respectively. Larger pooling kernels allow coarser features to be detected by the network. The main problem with these networks was that the spatial details vanished as a result of data compression in pooling layers.

After several attempts of designing different networks, the observations show that in order to estimate the depth from an image, the network needs to see the whole image at some stage as one object. So as to do that it requires the kernel to be the same size as the image in at least one layer, which is equivalent to a fully connected layer inside the network.

One problem with fully connected layers is that every neuron in every layer is connected. Due to the computationally prohibitive nature training networks that use of such connection mechanisms, and their tendency to cause overfitting, it is desirable to reduce the number of these connections. Adding fully connected layers result in a very tight bottleneck which seems to be crucial for the depth estimation task but also causes the majority of the details of the input data to be lost. In Fig. 6 – Fig. 9 the networks with fully connected layers are shown. These networks correspond to networks in Fig. 2 – Fig. 5 but with convolutional layers replaced with fully connected layers in the right hand side of the network. Using different pooling sizes before the fully connected layer will cause the network to extract different levels of features but all these configurations introduce loss of detail in the input samples.

Each of these eight configurations has their own advantages and shortcomings from missing the coarse features to missing the details and thus an inability of the network to fully utilize the whole input as a single sample. None of these designs converged to a reasonable depth estimation model.

The main idea of the SPDNN method is to mix and merge these networks and generate a single model which includes all the layers of the original models in order to be able to preserve the details and also detect the bigger objects in the scene for depth estimation task.
Figure 2. Top row: network 1, Bottom row: graph corresponds to network 1.

Figure 3. Top row: network 2, Bottom row: graph corresponds to network 2.

Figure 4. Top row: network 3, Bottom row: graph corresponds to network 3.

Figure 5. Top row: network 4, Bottom row: graph corresponds to network 4.

Figure 6. Top row: network 5, Bottom row: graph corresponds to network 5.
The proposed SPDNN idea uses graph contraction to merge several neural networks. The first step is to turn each network into a graph in which it is necessary to consider each layer of the network as a node in the graph. Each graph starts with the input node and ends with output node. The nodes in the graph are connected based on the connections in the corresponding layer of the other neural networks. Note that the pooling and un-pooling layers are not considered as a node in the graph but their properties will stay with the graph labelling which will be explained later.

In Fig. 2 – Fig. 9 the networks and corresponding compressed graphs are show. Two properties are assigned to each node in the graph. The first property is the layer structure, and the second one is the distance of the current node to the input node. To convert the network into a graph a labelling scheme is required for nodes. The proposed labelling scheme uses different signs for different layer structures, C for convolutional layer (for example 3C mean a convolutional layer with 3x3 kernel), F for fully connected layer (for example 30F means a fully connected layer with 30 neurons) and P for pooling property (for example 4P means that the data has been pooled by the factor of 4 in this layer).

4. The SPDNN Parallelization Methodology

4.1 Graph Contraction

The mechanism by which convolution layers combine is explained later in the paper.

The problem with parallelizing networks is that, having the same structure of layers with the same distance from the input might lead all the layers to converge to the same values. For example the first layer in all of the networks shown in Fig. 2 – Fig. 9 is a 2D convolutional layer with a 3x3 kernel. The mechanism by which graph contraction is used to merge these layers into a single layer, thus avoiding redundancy in the network structure, is explained later in the paper.
Some properties like convolutional and fully connected layers occur in a specific node, but pooling and unpooling operations will stick with the data to the next layers. In other words, the pooling property stays with the data except when an unpooling or a fully connected layer is reached. For example a node with the label \((3C8P, 4)\) corresponds to a convolutional layer with a \(3\times 3\) kernel, the \(8P\) portion of this label indicates that the data has undergone \(8\times 8\) pooling and the \(4\) at the end indicates that this label is at a distance of \(4\) form the input layer. In Fig. 2 – Fig. 9 under each network the corresponding graph with assigned labels is illustrated.

The next step is to put all these graphs in a parallel format sharing a single input and single output node. Fig. 10 shows the graph in this step.

![Parallelized version of the graphs shown in Fig. 2 – Fig. 9 sharing a single input node and single output node](image)

In order to merge layers with the same structure and the same distance from input node, the nodes with the exact same properties are labelled with the same letters. For example all the nodes with properties \((3C, 1)\) are labelled with letter A, and all the nodes with the properties \((3C2P, 4)\) are labelled as K and so on.

The next step is to apply the graph contraction on parallelized graph. In the graph contraction procedure the nodes with the same label are merged to a single node while saving their connections to the previous/next nodes. For instance all the nodes with label A are merged into one node, but its connection to the input node and also nodes B, C, D and E are preserved. The contracted version of the graph in Fig. 10 is shown in Fig. 11.

![Contracted version of the big graph shown in Fig. 10](image)

Afterwards, the graph has to be converted back to the neural network structure. In order to do this process, the preserved structural properties of each node are used. For example node C is a \(3\times 3\) convolutional layer which has experienced a pooling operation. Note that the pooling quality will be recalled from the original network.

The concatenation layer is used in the neural network in order to implement the nodes wherein several other nodes lead to one node. For example in nodes N and O the outputs of nodes J, K, L and M are concatenated with the pooling qualities taken from their original networks.
4.2 The Combined Model/Architecture

In the final model presented in Fig. 13, the input image is first processed in 4 parallel fully convolutional sub-networks with different pooling sizes. This provides the advantages of using different networks with different pooling sizes at the same time. The output of these 4 sub-networks are concatenated in 2 different forms; One to pool the larger images to be the same size as the smallest image in the previous part and the other one is to unpool the smaller images of the previous part to be the same size as the largest image.

After merging these outputs, the data is led to 2 different networks. One is the fully convolutional network to deepen the learning and release more abstract features of the input and the other network is an auto-encoder network with different architecture for encoder and decoder.

The observations show that it is essential for the network to see the whole image as one input at some stage in order to be able to estimate the depth; this requires putting a convolutional layer with a filter having the same size as the image which is the same as using a fully connected layer. As previously mentioned, having a fully connected layer in the network is crucial for reasonable estimation of image’s depth which is provided in the bottleneck of the auto encoder. The results from the auto encoder and the fully convolutional sub-network are again merged in order to give a single output after applying a one channel convolutional layer.

In order to regularize the network, prevent overfitting and increase the convergence, batch normalization [18] is applied after every convolutional layer and the drop-out technique [17] is used in fully connected layers. The experiments in this paper show that using weight regularization in the fully connected layers gives slower convergence. That’s why this technique is not used in the final version of the model. All the nonlinearities in the network are the ReLU non-linearity which is widely used in deep neural networks except the output layer which took advantage of the sigmoid nonlinearity. The value repeating technique was used in the un-pooling layer due to non-specificity of the corresponding pooled layer in the decoder part of the auto-encoder sub-network.

The value repeating technique, illustrated in Fig. 12, involves repeating the value from the previous layer in order to obtain the un-pooled image. The figure shows the 2×2 un-pooling and the process is the same for other un-pooling sizes.

![Figure 12. Repeating technique used in un-pooling layers.](image_url)

4.3 SPDNN: How it works and why it’s effective

One might ask why the SPDNN approach is effective and what is the difference between this approach and other mixing approaches? Here the model designed by the SPDNN scheme is investigated in the forward and back propagation steps. The key component is in the back-propagation step where the parameters in parallel layers influence each other. These two steps are described below:

1- Forward propagation: Consider the network designed by the SPDNN approach shown in Fig. 14. This network is made of 5 sub-networks. In this figure just the general view of the network is shown and the layers in detail are ignored since the main goal is to show the information flow inside the whole network.

![Figure 13. The converted network from graph shown in Fig. 11.](image_url)
When the input samples are fed into the network, the data can travel inside the network through three different paths shown in Fig. 15. At this stage the parallel networks are blind to each other, i.e., the networks placed in parallel don’t share any information with each other. As shown in Fig. 15 the data traveling in sub-net 1 and sub-net 2 are not influenced by each other since they don’t share any path together, as in sub-net 3 and sub-net 4.

2- Back propagation: while training the network, the loss function calculated based on the error value at the output of the neural network is a mixed and merged function of the error value correspond to every data path in the network. In the backpropagation step the parameters inside the network update based on this mixed loss values, i.e., this value back-propagates throughout the whole network as it is shown in Fig. 16. Therefore, at this stage of training, each subnetwork is influenced by the error value from every data path shown in Fig. 16. This illustrates the way that each subnetwork is trained to reduce the error of its own path and also the error from the mixture of all paths.

The main difference between the SPDNN approach and other mixing approaches like the voting approach lies in the backpropagation step where different sub-nets are influenced by the error of each other and try to compensate each other’s shortcomings by reducing the final mixed error value. Meanwhile in the voting approach different classifiers are trained independent of each other and they are not communicating to reduce their total error value.

4.4 Training Set

In this paper, KITTI Stereo 2012, 2015 [26] are used for training purposes. The disparity of each set is estimated by the method which has been explained earlier in the paper. The left image of the sets along with the estimated disparity map is used to train the network. Each image is flipped vertically, horizontally and each vertically flipped image is flipped horizontally as well in order to augment the database to cover all the possibilities of the disparity in case of camera rotation. In total 33,096 images are used in this research. 70% of the initial set is considered for training, 20% for validation and 10% for test purposes. Each model is trained for two sets of input samples and two sets of output targets. Therefore we have 4 different experiments as follows:

1. First Model: Input: Left RGB Image + Pixel-wise Segmented Image. Target: Post-Processed Disparity.
2. Second Model: Input: Left RGB Image. Target: Post-Processed Disparity.
3. Third Model: Input: Left RGB Image + Pixel-wise Segmented Image. Target: Disparity.
4. Fourth Model: Input: Left RGB Image. Target: Disparity.

The images are resized to 80×264 pixels during the whole process. Training is done on a standard desktop with an NVIDIA GTX 1080 GPU with 8GB memory.

4.5 Experimental Results

In all our experiments the mean square error value between the output of the network and the target values has been used as the loss function and the Nestrov momentum technique [27] with learning rate 0.01 and momentum 0.9 has been used to train the network. The Train Loss and Validation Loss for each of these experiments are shown in Fig. 17 and Fig. 18 respectively. These figures show that using the Post-Processed Disparity as target results in lower loss values which means that the network was able to learn better features in those experiments.

The evaluation in this paper has been done in 2 parts. In the first part the results of the trained models are compared against the ground truth provided by the KITTI benchmark.

In the second part of our evaluation we considered the disparity maps computed by our recent stereo matching technique [21] as ground truth. The primary goal was to compare this ground truth with the disparity maps from the trained models. A secondary goal was to evaluate the performance of the stereo matching technique on road sensing applications.

For evaluation purposes, 8 metrics including PSNR, MSE, RMSE, SNR, MAE, Structural Similarity Index (SSIM) [28], Universal Quality Index (UQI) [29] and Pearson Correlation Coefficient (PCC) [30] are used. Table 2 shows the results of the first part of our experiments, the numerical comparison of the trained models against the ground truth provided by the benchmark. The best value for each metric is presented in bold. Fig. 19 – Fig. 21 represent the colour coded disparity maps computed by the trained models using the proposed DNN where the dark red and dark blue parts represent closest and furthest points to the camera respectively. For visualization purposes all of the images presented in this section are upsampled using Joint Bilateral Upsampling [31].

| Table 2. Numerical Comparison of the Models – Experiment 1 |
|----------------------------------------------------------|
| After With SegNet | After Without SegNet | Before With SegNet | Before Without SegNet |
| PSNR            | 14.3424             | 13.7677             | 13.8333             | 13.8179             |
| MSE             | 0.0382              | 0.0436              | 0.0435              | 0.0439              |
| RMSE            | 0.1937              | 0.2069              | 0.206              | 0.2066              |
| SNR             | 4.4026              | 3.8279              | 6.1952             | 6.1798              |
| MAE             | 0.1107              | 0.1212              | 0.1236             | 0.1234              |
| SSIM            | 0.9959              | 0.9955              | 0.9955             | 0.9955              |
| UQI             | 0.9234              | 0.9252              | 0.9053             | 0.9064              |
| PCC             | 0.7687              | 0.8485              | 0.7702             | 0.7729              |
The numerical comparison of the trained models for the second part of our experiment is shown in Table 3. The colour coded disparity maps computed by the trained models using the proposed DNN are illustrated in Fig. 22 – Fig. 24.
Table 3. Numerical Comparison of the Models – Experiment 2

|               | After With SegNet | After Without SegNet | Before With SegNet | Before Without SegNet |
|---------------|-------------------|----------------------|-------------------|-----------------------|
| PSNR          | 15.0418           | 14.1895              | 13.3819           | 14.0491               |
| MSE           | 0.0378            | 0.0447               | 0.0535            | 0.0441                |
| RMSE          | 0.1854            | 0.203                | 0.2223            | 0.2039                |
| SNR           | 8.822             | 7.9696               | 5.4271            | 6.0943                |
| MAE           | 0.1442            | 0.1581               | 0.1673            | 0.153                 |
| SSIM          | 0.9952            | 0.9943               | 0.994             | 0.9951                |
| UQI           | 0.8401            | 0.8369               | 0.7951            | 0.8178                |
| PCC           | 0.8082            | 0.795                | 0.704             | 0.6919                |

Figure 22. Estimated Disparity Maps from the Trained Models – Example 1

Figure 23. Estimated Disparity Maps from the Trained Models – Example 2
These results provide evidence that estimating depth map from monocular images/cameras by employing DNN is effective. We show that trained models estimate depth values with a similar accuracy to the depth values computed by stereo matching methods while providing the advantage of ~0.23 second computational time (on NVIDIA GTX 1080 GPU).

Table 4 shows the numerical evaluation of our stereo matching technique against the ground truth images used in the first part of the experiment. Clearly the comparison of these numbers with Table 2 shows that higher accuracy is achievable by using the stereo matching technique but the differences are not particularly significant.

Table 4. Numerical Comparison of the Models – Experiment 2

|            | After Post-Processing | Before Post-Processing |
|------------|----------------------|------------------------|
| PSNR       | 14.8234              | 14.1384                |
| MSE        | 0.0351               | 0.043                  |
| RMSE       | 0.1845               | 0.2017                 |
| SNR        | 4.8836               | 6.5003                 |
| MAE        | 0.1017               | 0.1213                 |
| SSIM       | 0.9966               | 0.9955                 |
| UQI        | 0.9353               | 0.9069                 |
| PCC        | 0.823                | 0.7797                 |

5. Conclusions & Future Work

In this paper we have presented a deep neural network to train a highly accurate model for estimating depth from monocular images. In total 4 models have been trained. Pixel-wise segmentation and our post-processing technique have been used to provide inputs for 2 of the models. The KITTI benchmark has been used for training and experimental purposes. Each model has been evaluated in 2 sections, first against the ground truth provided by the benchmark and second against the disparity maps computed by the stereo matching method. The results showed that a slightly higher accuracy can be obtained by employing the stereo matching technique but our results demonstrate that there is not a big difference between the depths from the models trained by proposed DNN and the values computed by stereo matching. It is also worth pointing out an important advantage of these models which is the processing time of ~0.23 second, making it suitable for providing depth estimation in real time. Using pixel-wise segmentation as one of the inputs of the network has slightly increased the performance of the models in terms of accuracy but it brought some artifacts such as wrong depth patches on a surfaces. The evaluation results also illustrate the higher accuracy of the models where post-processed disparity was used as the target in training procedure and it is an evidence for the performance of our post-processing method.

References

1. D. Nair. (Aug 24, 2016). 3D Imaging with NI LabVIEW. Available: http://www.ni.com/white-paper/14103/en/
2. C. Niclass, M. Soga, H. Matsubara, and S. Kato, "A 100m-Range 10-Frame/s 340x96-Pixel Time-of-Flight Depth Sensor in 0.18-μm CMOS," in ESSCIRC (ESSCIRC), 2011 Proceedings of the, 2011, pp. 107-110.
3. C. Niclass, K. Ito, M. Soga, H. Matsubara, I. Aoyagi, S. Kato, et al., "Design and characterization of a 256x64-pixel single-photon imager in CMOS for a MEMS-based laser scanning time-of-flight sensor," Optics Express, vol. 20, pp. 11863-11881, 2012/05/21 2012.
4. D. Scharstein and R. Szeliski, "High-accuracy stereo depth maps using structured light," in Computer Vision and Pattern Recognition. 2003. Proceedings. 2003 IEEE Computer Society Conference on, 2003, pp. I-195-I-202 vol.1.
5. R. K. Gupta and S.-Y. Cho, "A Correlation-Based Approach for Real-Time Stereo Matching," in Advances in Visual Computing; 6th International Symposium, ISVC 2010, Las Vegas, NV, USA, November 29 – December 1, 2010, Proceedings, Part II, G. Bebis, R. Boyle, B. Parvin, D. Koracin, R.
Chung, R. Hammound, et al., Eds., ed Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 129-138.
6. C. Zhang, Z. Li, Y. Cheng, R. Cai, H. Chao, and Y. Rui, "MeshStereo: A Global Stereo Model with Mesh Alignment Regularization for View Interpolation," in 2015 IEEE International Conference on Computer Vision (ICCV), 2015, pp. 2057-2065.
7. Middlebury Stereo Evaluation - Version 3. Available: http://vision.middlebury.edu/stereo/eval3/
8. K. R. Kim and C. S. Kim, "Adaptive smoothness constraints for efficient stereo matching using texture and edge information," in 2016 IEEE International Conference on Image Processing (ICIP), 2016, pp. 3429-3433.
9. X. Huang, Y. Zhang, and Z. Yue, "Image-Guided Non-Local Dense Matching with Three-Steps Optimization," ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci., vol. III-3, pp. 67-74, 2016.
10. A. Li, D. Chen, Y. Liu, and Z. Yuan, "Coordinating Multiple Disparity Proposals for Stereo Computation," in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, USA, 2016, pp. 4022-4030.
11. M. Shahbazi, G. Sohn, J. Théau, and P. Ménard, "Revisiting Intrinsic Curves for Efficient Dense Stereo Matching," ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci., vol. III-3, pp. 123-130, 2016.
12. J. Žbontar and Y. LeCun, "Stereo matching by training a convolutional neural network to compare image patches," J. Mach. Learn. Res., vol. 17, pp. 2287-2318, 2016.
13. J. T. Barron and B. Poole, "The Fast Bilateral Solver," arXiv:1511.03296, vol.abs/1511.03296, 2016.
14. E. T. Psota, J. Kowalczyk, M. Mittek, P. L. C, and rez, "MAP Disparity Estimation Using Hidden Markov Trees," in 2015 IEEE International Conference on Computer Vision (ICCV), 2015, pp. 2219-2227.
15. J. Kowalczyk, E. T. Psota, and L. C. Perez, "Real-Time Stereo Matching on CUDA Using an Iterative Refinement Method for Adaptive Support-Weight Correspondences," IEEE Transactions on Circuits and Systems for Video Technology, vol. 23, pp. 94-104, 2013.
16. B. Yegnanarayana, Artificial neural networks: PHI Learning Pvt. Ltd., 2009.
17. N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," J. Mach. Learn. Res., vol. 15, pp. 1929-1958, 2014.
18. S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," arXiv:1502.03167, vol.abs/1502.03167, 2015.
19. 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, pp. 2024-2039, 2016.
20. A. Saxena, J. Schulte, and A. Y. Ng, "Depth estimation using monocular and stereo cues," in Proceedings of the 20th international joint conference on Artificial intelligence, Hyderabad, India, 2007, pp. 2197-2203.
21. H. Javidinia and P. Corcoran, "A Depth Map Post-Processing Approach Based on Adaptive Random Walk With Restart," IEEE Access, vol. 4, pp. 5509-5519, 2016.
22. V. Badrinarayanan, A. Kendall, and R. Cipolla, "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation," arXiv:1511.00561, vol.abs/1511.00561, 2015.
23. A. Kendall, V. Badrinarayanan, and R. Cipolla, "Bayesian segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding," arXiv preprint arXiv:1511.02680, 2015.
24. A. Kendall, V. Badrinarayanan, and R. Cipolla. Caffe Implementation of SegNet. Available: https://github.com/alexkendall/caffe-segnet
25. G. J. Brostow, J. Shotton, J. Fauqueur, and R. Cipolla, "Segmentation and Recognition Using Structure from Motion Point Clouds," in Computer Vision – ECCV 2008: 10th European Conference on Computer Vision, Marseille, France, October 12-18, 2008, Proceedings, Part I, D. Forsyth, P. Torr, and A. Zisserman, Eds., ed Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 44-57.
26. M. Menze and A. Geiger, "Object scene flow for autonomous vehicles," in 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 3061-3070.
27. I. Sutskever, J. Martens, G. E. Dahl, and G. E. Hinton, "On the importance of initialization and momentum in deep learning," ICML (3), vol. 28, pp. 1139-1147, 2013.
28. W. Zhou, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," IEEE Transactions on Image Processing, vol. 13, pp. 600-612, 2004.
29. W. Zhou and A. C. Bovik, "A Universal Image Quality Index," IEEE Signal Processing Letters, vol. 9, pp. 81-84, 2002.
30. K. Pearson, "Note on regression and inheritance in the case of two parents," Proceedings of the Royal Society of London, vol. 58, pp. 240-242, 1895.
31. J. Kopf, M. F. Cohen, D. Lischinski, and M. Uyttendaele, "Joint bilateral upsampling," ACM Trans. Graph., vol. 26, p. 96, 2007 s