Detect, Replace, Refine: Deep Structured Prediction For Pixel Wise Labeling

Spyros Gidaris
University Paris-Est, LIGM
Ecole des Ponts ParisTech
spyros.gidaris@imagine.enpc.fr

Nikos Komodakis
University Paris-Est, LIGM
Ecole des Ponts ParisTech
nikos.komodakis@enpc.fr

Abstract

Pixel wise image labeling is an interesting and challenging problem with great significance in the computer vision community. In order for a dense labeling algorithm to be able to achieve accurate and precise results, it has to consider the dependencies that exist in the joint space of both the input and the output variables. An implicit approach for modeling those dependencies is by training a deep neural network that, given as input an initial estimate of the output labels and the input image, it will be able to predict a new refined estimate for the labels. In this context, our work is concerned with what is the optimal architecture for performing the label improvement task. We argue that the prior approaches of either directly predicting new label estimates or predicting residual corrections w.r.t. the initial labels with feed-forward deep network architectures are sub-optimal. Instead, we propose a generic architecture that decomposes the label improvement task to three steps: 1) detecting *the initial label estimates that are incorrect, 2)* replacing *the incorrect labels with new ones, and finally 3)* refining *the renewed labels by predicting residual corrections w.r.t. them. Furthermore, we explore and compare various other alternative architectures that consist of the aforementioned* Detection, Replace*, and* Refine *components. We extensively evaluate the examined architectures in the challenging task of dense disparity estimation (stereo matching) and we report both quantitative and qualitative results on three different datasets. Finally, our dense disparity estimation network that implements the proposed generic architecture, achieves state-of-the-art results in the KITTI 2015 test surpassing prior approaches by a significant margin. We plan to release the Torch [4] code that implements the paper in: https://github.com/gidariss/DRR_struct_pred/.*

1. Introduction

Dense image labeling is a problem of paramount importance in the computer vision community as it encompasses many low or high level vision tasks including stereo matching [37], optical flow [12], surface normals estimation [5], and semantic segmentation [18], to mention a few characteristic examples. In all these cases the goal is to assign a discrete or continuous value for each pixel in the image. Due to its importance, there is a vast amount of work on this problem. Recent methods can be roughly divided into three main classes of approaches.

The first class focuses on developing independent patch classifiers/regressors [31, 29, 30, 18, 7, 20, 23] that would directly predict the pixel label given as input an image patch centered on it or, in cases like stereo matching and optical flow, would be used for comparing patches between different images in order to pick pairs of best matching pixels [19, 36, 37, 38]. Deep convolutional neural networks (DCNNs) [16] have demonstrated excellent performance in the aforementioned tasks thanks to their ability to learn complex image representations by harnessing vast amount of training data [14, 32, 11]. However, despite their great representational power, just applying DCNNs on image patches, does not capture the structure of output labels, which is an important aspect of dense image labeling tasks. For instance, independent feed-forward DCNN patch predictors do not take into consideration the correlations that exist between nearby pixel labels. In addition, feed-forward DCNNs have the extra disadvantages that they usually involve multiple consecutive down-sampling operations (i.e. max-pooling or strided convolutions) and that the top most convolutional layers do not capture factors such as image edges or other fine image structures. Both of the above properties may prevent such methods from achieving precise and accurate results in dense image labeling tasks.

Another class of methods tries to model the joint dependencies of both the input and output variables by use of probabilistic graphical models such as Conditional Random Fields (CRFs) [15]. In CRFs, the dense image labeling task is performed through maximum a posteriori (MAP) inference in a graphical model that incorporates prior knowledge about the nature of the task in hand with pairwise edge potential between the graph nodes of the label variables. For example, in the case of semantic segmentation, those pairwise potentials enforce label consistency among similar or spatially adjacent pixels. Thanks to their ability to jointly model the input-output variables, CRFs have been extensively used in pixel-wise image labelling tasks [13, 25]. Recently, a number of methods has attempted to combine them with the representational power of DCNNs by getting the former (CRFs) to refine and disambiguate the predictions of the later one [27, 2, 39, 3]. Particularly, in semantic segmentation, DeepLab [2] uses a fully connected CRF to post-process the pixel-wise predictions of a convolutional neural network while in CRF-RNN [39], they unify the training of both the DCNN and the CRF by formulating the approximate mean-field inference of fully connected CRFs as Recurrent Neural Networks (RNN). However, a major drawback of most CRF based approaches is that the pairwise potentials have to be carefully hand designed in order to incorporate simple human assumptions about the structure of the output labels Y and at the same time to allow for tractable inference.

A third class of methods relies on a more data-driven approach for learning the joint space of both the input and the output variables. More specifically, in this case a deep neural network gets as input an initial estimate of the output labels and (optionally) the input image and it is trained to predict a new refined estimate for the labels, thus being implicitly enforced to learn the joint space of both the input and the output variables. The network can learn either to predict new estimates for all pixel labels (transform-based approaches) [24, 35, 10, 17], or alternatively, to predict residual corrections w.r.t. the initial label estimates (residual-based approaches) [1]. We will hereafter refer to these methods as *deep joint input-output models*. These are, loosely speaking, related to the CRF models in the sense that the deep neural network is enforced to learn the joint dependencies of both the input image and output labels, but with the advantage of being less constrained about the complexity of the input-output dependencies that it can capture.

Our work belongs to this last category of dense image labeling approaches, thus it is not constrained on the complexity of the input-output dependencies that it can capture. However, here we argue that prior approaches in this category use a sub-optimal strategy. For instance, the transform-based approaches (that always learn to predict new label estimates) often have to learn something more difficult than necessary since they must often simply learn to operate as identity transforms in case of correct initial labels, yielding the same label in their output. On the other hand, for the residual based approaches it is easier to learn to predict zero residuals in the case of correct initial labels, but it is more difficult for them to refine “hard” mistakes that deviate a lot from the initial labels (see figure 1). Due to the above reasons, in our work we propose a deep joint input-output model that decomposes the label estimation/refinement process as a sequence of the following easier to execute operations: 1) *detection* of errors in the input labels, 2) *replacement* of the erroneous labels with new ones, and finally 3) an overall *refinement* of all output labels in the form of residual corrections. Each of the described operations in our framework is executed by a different component implemented with a deep neural network. Even more, those components are embedded in a unified architecture that is fully differentiable thus allowing for an end-to-end learning of the dense image labeling task by only applying the objective function on the final output. As a result of this, we are also able to explore a variety of novel deep network architectures by considering different ways of combining the above components, including the possibility of performing the above operations iteratively, as it is done in [24, 17], thus enabling our model to correct even large, in area, regions of incorrect labels. It is also worth noting that the error detection component in the proposed architecture, by being forced to detect the erroneous pixel labels (given both the input and the initial estimates of the output labels), it implicitly learns the joint structure of the input-output space, which is an important requirement for a successful application of any type of structured prediction model.

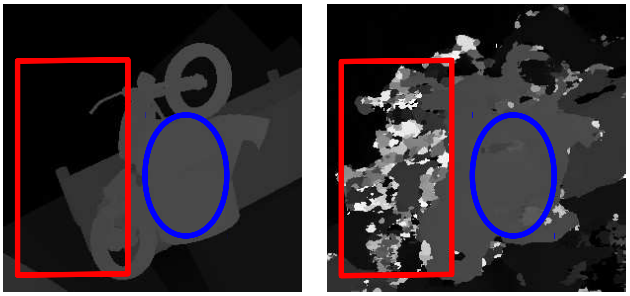

Figure 1: In this figure we visualize two different type of erroneously labeled image regions. On the left hand are the ground truth labels and on the right hand are some initial label estimates. With the red rectangle we indicate a dense concentration of ”hard” mistakes in the initial labels that it is very difficult to be corrected by a residual refinement component. Instead, the most suitable action for such a region is to replace them by predicting entirely new labels for them. In contrast, the blue eclipse indicates an image region with ”soft” label mistakes. Those image regions are easier to be handled by a residual refinement components.

To summarize, our contributions are as follows: **(1)** We propose a deep structured prediction framework for the dense image labeling task, which we call *Detect, Replace, Refine*, that relies on three main building blocks: a) recognizing errors in the input label maps, b) replacing the erroneous labels, and c) performing a final refinement of the output label map. We show that all of the aforementioned steps can be embedded in a unified deep neural network architecture that is end-to-end trainable. **(2)** In the context of the above framework, we also explore a variety of other network architectures for deep joint input-output models that result from utilizing different combinations of the above building blocks. **(3)** We implemented and evaluated our framework on the disparity prediction task and we provide both qualitative and quantitative evidence about the advantages of the proposed approach. **(4)** We show that our disparity estimation model that implements the proposed *Detect, Replace, Refine* architecture achieves state of the art results in the KITTI 2015 test set outperforming all prior published work by a significant margin.

The remainder of the paper is structured as follows: We first describe our structured dense label prediction framework in §2 and its implementation w.r.t. the dense disparity estimation task in §3. Then, we provide experimental results in §4 and we finally conclude the paper in §5.

2. Methodology

Let $X = \{x_i\}_{i=1}^{H \times W}$ be the input image[1] of size $H \times W$, where x_i are the image pixels, and $Y = \{y_i\}_{i=1}^{H \times W}$ be some initial label estimates for this image, where y_i is the label for the i-th pixel. Our dense image labeling methodology belongs on the broader category of approaches that consist of a deep joint input-output model model $F(.)$ that given as input the image X and the initial labels Y, it learns to predict new, more accurate labels $Y' = F(X, Y)$. Note that in this setting the initial labels Y could come from another model $F_0(.)$ that depends only on the image X. Also, in the general case, the pixel labels Y can be of either discrete or continuous nature. In this work, however, we focus on the continuous case where greater variety of architectures can be explored.

The crucial question is what is the most effective way of implementing the deep joint input-output model $F(.)$. The two most common approaches in the literature involve a feed-forward deep convolutional neural network, $F_{DCNN}(.)$, that either directly predicts new labels $Y' = F_{DCNN}(X, Y)$ or it predicts the residual correction w.r.t. the input labels: $Y' = Y + F_{DCNN}(X, Y)$. We argue that both of them are sub-optimal solutions for implementing the $F(.)$ model. Instead, in our work we opt for a decomposition of the task of model $F(.)$ (*i.e.* predicting new, more accurate labels Y') in three different sub-tasks that are executed in sequence.

In the remainder of this section, we first describe the proposed architecture in §2.1, then we discuss the intuition behind it and its advantages in §2.2, and finally we describe other alternative architectures that we explored in §2.3.

2.1. Detect, Replace, Refine architecture

The generic dense image labeling architecture that we propose decomposes the task of the deep joint input-output model in three sub-tasks each of them handled by a different learn-able network component (see Figure 2). Those network components are: the error detection component $F_e(.)$, the label replacement component $F_u(.)$, and the label refinement component $F_r(.)$. The sub-tasks that they perform, are:

Detect: The first sub-task in our generic pipeline is to detect the erroneously labeled pixels of Y by discovering which pixel labels are inconsistent with the remaining labels of Y and the input image X. This sub-task is performed by the error detection component $F_e(.)$ that basically needs to yield a probability map $E = F_e(X, Y)$ of the same size as the input labels Y that will have high probabilities for the "hard" mistakes in Y. These mistakes should ideally be forgotten and replaced with entirely new label values in the processing step that follows (see Figures 3a, 3b, and 3c). As we will see below, the topology of our generic architecture allows the error detection component $F_e(.)$ to learn its assigned task (*i.e.* detecting the incorrect pixel labels) without explicitly being trained for this, e.g., through the use of an auxiliary loss. The error detection function $F_e(.)$ can be implemented with any deep (or shallow) neural network with the only constraint being that its output map E must take values in the range $[0, 1]$.

Replace: In the second sub-task, a new label field U is produced by the convex combination of the initial label field Y and the output of the label replacement component $F_u(.)$: $U = E \odot F_u(X, Y, E) + (1 - E) \odot Y$ (see Figures 3e and 3f). We observe that the error probabilities generated by the error detection component $F_e(.)$ now act as gates that control which pixel labels of Y will be forgotten and replaced by the outputs of $F_u(.)$, which will be all pixel labels that are assigned high probability of being incorrect. In this context, the task of the Replace component $F_u(.)$ is to replace the erroneous pixel labels with new ones that will be in accordance both w.r.t. the input image X and w.r.t. the non-erroneous labels of Y. Note that for this task the Replace component $F_u(.)$ gets as input also the error probability map E. The reason for doing this is to help the Replace component to focus its attention only on those image regions that their labels need to be replaced. The component $F_u(.)$ can be implemented by any neural network that its output has the same size as the input labels Y.

Refine: The purpose of the erroneous label detection and label replacement steps so far was to perform a crude "fix" of the "hard" mistakes in the label map Y. In contrast, the purpose of the current step is to do a final refinement of the entire output label map U, which is produced by the previous steps, in the form of residual corrections: $Y' = U + F_r(X, Y, E, U)$ (see Figures 3g and 3h). Intuitively, the purpose of this step is to correct the "soft" mistakes of the label map U and to better align the output labels Y' with the fine structures in the image X. The Refine component

[1] Here, for simplicity, we consider images defined on a 2D domain, but our framework can be readily applied to images defined on any domain.

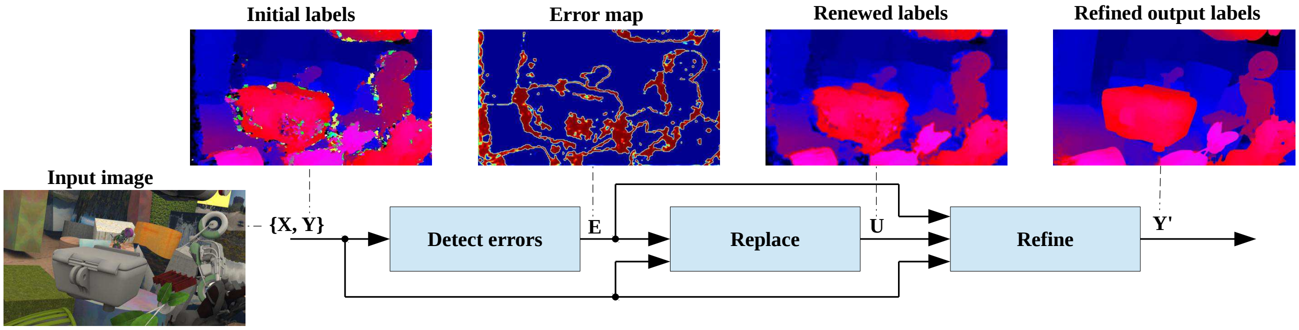

Figure 2: In this figure we demonstrate the generic architecture that we propose for the dense image labeling task. In this architecture the task of the deep joint input-output model is decomposed into three different sub-tasks that are: 1) detection of the erroneous initial labels (based on an estimated error map E) , 2) replacement of the erroneous labels with new ones (leading to a renewed label map U), and then 3) refinement Y' of the renewed label map. The illustrated example is coming from the dense disparity labeling task (stereo matching).

$F_r(.)$ can be implemented by any neural network that its output has the same size as the input labels U.

The above three steps can be applied for more than one iterations which, as we will see later, allows our generic framework to recover a good estimate of the ground truth labels or, in worst case, to yield more plausible results even when the initial labels Y are severely corrupted (see section 4.3.5 in our extended technical report [8]).

To summarize, the workings of our dense labeling generic architecture can be concisely described by the iterative application of the following three equations:

$$E = F_e(X, Y), \tag{1}$$

$$U = E \odot F_u(X, Y, E) + (1 - E) \odot Y, \tag{2}$$

$$Y' = U + F_r(X, Y, E, U). \tag{3}$$

We observe that the above generic architecture is fully differentiable as long as the function components $F_e(.)$, $F_u(.)$, and $F_r(.)$ are also differentiable. Due to this fact, the overall proposed architecture is end-to-end learnable by directly applying an objective function (*e.g.* Absolute Difference or Mean Square Error loss functions) on the final output label maps Y'.

2.2. Discussion

Role of the Detection component $F_e(.)$ and its synergy with the Replace component $F_u(.)$: The error detection component $F_e(.)$ is a key element in our generic architecture and its purpose is to indicate which are the image regions that their labels are incorrect. This type of information is exploited in the next step of label replacement in two ways. Firstly, the Replace component $F_u(.)$ that gets as input the error map E, which is generated by $F_e(.)$, is able to know which are the image regions that their labels needs to be replaced and thus it is able to focus its attention only on those image regions. At this point note that, in equation 7, the error maps E, apart from being given as input attention maps to the Replace component $F_u(.)$, they also act as gates that control which way the information will flow both during the forward propagation and during the backward propagation. Specifically, during the forward propagation case, in the cases that the error map probabilities are either 0 or 1, it holds that:

$$U = \begin{cases} Y, & \text{if } F_e(X, Y) = \mathbf{0}, \\ F_u(X, Y, E), & \text{if } F_e(X, Y) = \mathbf{1}, \end{cases} \tag{4}$$

which basically means that the Replace component $F_u(.)$ is being utilized mainly for the erroneously labelled image regions. Also, during the backward propagation, it is easy to see that the gradients of the replace function w.r.t. the loss L (in the cases that the error probabilities are either 0 or 1) are:

$$\frac{dL}{dF_u(.)} = \begin{cases} \mathbf{0}, & \text{if } F_e(X, Y) = \mathbf{0}, \\ \frac{dL}{dU}, & \text{if } F_e(X, Y) = \mathbf{1}, \end{cases} \tag{5}$$

which means that gradients are back-propagated through the Replace component $F_u(.)$ only for the erroneously labelled image regions. So, in a nutshell, during the learning procedure the Replace component $F_u(.)$ is explicitly trained to predict new values mainly for the erroneously labelled image regions. The second advantage of giving the error maps E as input to the Replace component $F_u(.)$, is that this allows the Replace component to know which image regions contain "trusted" labels that can be used for providing information on how to fill the erroneously labelled regions.

Estimated error probability maps by the Detection component $F_e(.)$: Thanks to the topology of our generic architecture, by optimizing the reconstruction of the ground truth labels $\hat{Y}$, the error detection component $F_e(.)$ implicitly learns to act as a joint probability model for patches of X and Y centered on each pixel of the input image, assigning a high probability of error for patches that do not appear to belong to the joint input-output space (X, Y). In Figures 3c and 3d we visualize the estimated by the Detection component $F_e(.)$ error maps and the ground truth error maps in the context of the disparity estimation task (more visualizations are provided in our extended technical report [8]). It is interesting to note that the estimated error probability maps are very similar to the ground truth error maps despite the fact that we are not explicitly enforcing this behaviour, e.g., through the use of an auxiliary loss.

Error detection component and Highway Networks: Note that the way the Detection component $F_e(.)$ and Replace component $F_u(.)$ interact bears some resemblance to the basic building blocks of the Highway Networks [33] that are being utilized for training extremely deep neural network architectures. Briefly, each highway building block gets as input some hidden feature maps and then predicts transform gates that control which feature values will be carried on the next layer as is and which will be transformed by a non-linear function. There are however some important differences. For instance, in our case the error gate prediction and the label replacement steps are executed in sequence with the latter one getting as input the output of the former one. Instead, in Highway Networks the gate prediction and the non-linear transform of the input feature maps are performed in parallel. Furthermore, in Highway Networks the components of each building block are implemented by simple affine transforms followed by non-linearities and the purpose is to have multiple building blocks stacked one on top of the other in order to learn extremely deep image representations. In contrast, the components of our generic architecture are them selves deep neural networks and the purpose is to learn to reconstruct the input labels Y.

Two stage refinement approach: Another key element in our architecture is that the step of predicting new, more accurate labels Y', given the initial labels Y, is broken in two stages. The first stage is handled by the error detection component $F_e(.)$ and the label replacement component $F_u(.)$. Their job is to correct only the ”hard” mistakes of the input labels Y. They are not meant to correct ”soft” mistakes (*i.e.* errors in the label values of small magnitude). In order to learn to correct those ”soft” mistakes, it is more appropriate to use a component that yields residual corrections w.r.t. its input. This is the purpose of our Refine component $F_r(.)$, in the second stage of our architecture, from which we expect to improve the ”details” of the output labels U by better aligning them with the fine structures of the input images. This separation of roles between the first and the second refinement stages (*i.e.* coarse refinement and then fine-detail refinement) has the potential advantage, which is exploited in our work, to perform the actions of the first stage in lower resolution thus speeding up the processing and reducing the memory footprint of the network. Also, the end-to-end training procedure allows the components in the first stage (*i.e.* $F_e(.)$ and $F_u(.)$) to make mistakes as long as those are corrected by the second stage. This aspect of our architecture has the advantage that each component can more efficiently exploit its available capacity.

2.3. Explored architectures

In order to evaluate the proposed architecture we also devised and tested various others architectures that consist of the same core components as those that we propose. In total, the architectures that are explored in our work are:

***Detect* + *Replace* + *Refine* architecture:** This is the architecture that we proposed in section 2.1.

***Replace* baseline architecture:** In this case the model directly replaces the old labels with new ones: $Y' = Fu(X, Y)$.

***Refine* baseline architecture:** In this case the model predicts residual corrections w.r.t. the input labels: $Y' = Y + Fr(X, Y)$.

***Replace* + *Refine* architecture:** Here the model first replaces the entire label map Y with new values $U = Fu(X, Y)$ and then residual corrections are predicted w.r.t. the updated values U, $Y' = U + Fr(X, Y, U)$.

***Detect* + *Replace* architecture:** Here the model first detects errors on the input label maps $E = F_e(X, Y)$ and then replace those erroneous pixel labels $Y' = E \odot F_u(X, Y, E) + (1 - E) \odot Y$.

***Detect* + *Refine* architecture:** In this case, after the detection of the errors $E = F_e(X, Y)$, the erroneous pixel labels are masked out by setting them to the mean label value l_{mu}, $U = E \odot l_{mu} + (1 - E) \odot Y$. Then the masked label maps are given as input to a residual refinement model $Y' = U + F_r(X, Y, E, U)$. Note that this architecture can also be considered as a specific instance of the general Detect + Replace + Refine architecture where the Replace component $F_u(.)$ does not have any learnable parameters and constantly returns the mean label value, *i.e.*, $F_u(.) = l_{mu}$.

***Parallel* architecture:** Here, after the detection of the errors, the erroneous labels are replaced by the Replace component $F_u(.)$ while the rest labels are refined by the Refine component $F_r(.)$. More specifically, the operations performed by this architecture are described by the following equations:

$$E = F_e(X, Y), \tag{6}$$

$$U_1 = F_u(X, Y, E), U_2 = Y + F_r(X, Y, E), \tag{7}$$

$$Y' = E \odot U_1 + (1 - E) \odot U_2. \tag{8}$$

Basically, in this architecture the components $F_u(.)$ and $F_r(.)$ are applied in parallel instead of the sequential topology that is chosen in the Detect + Replace + Refine architecture.

***Detect* + *Replace* + *Refine* $\times T$:** This is basically the Detect + Replace + Refine architecture but applied iteratively for T iterations. Note that the model implementing this architecture is trained in a multi-iteration manner.

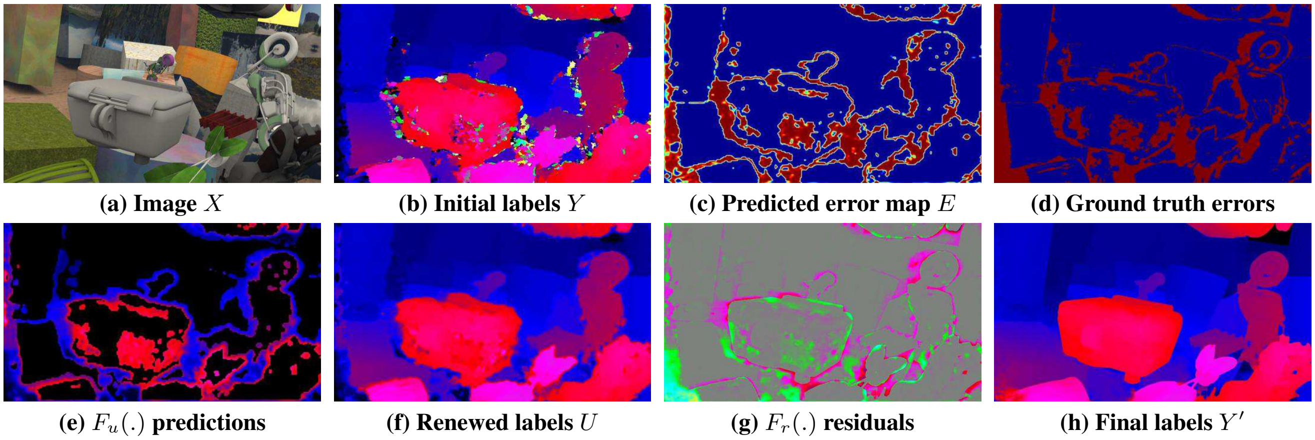

Figure 3: Here we provide an example that illustrates the functions performed by the Detect, Replace, and Refine steps in our proposed architecture. The example is coming from the dense disparity labeling task (stereo matching). Specifically, subfigures **(a)**, **(b)**, and **(c)** depict respectively the input image X, the initial disparity label estimates Y, and the error probability map E that the detection component $F_e(.)$ yields for the initial labels Y. Notice the high similarity of map E with the ground truth error map of the initial labels Y depicted in subfigure **(d)**, where the ground truth error map has been computed by thresholding the absolute difference of the initial labels Y from the ground truth labels with a threshold of 3 pixels (red are the erroneous pixel labels). In subfigure **(e)** we depict the label predictions of the Replace component $F_u(.)$. For visualization purposes we only depict the $F_u(.)$ pixel predictions that will replace the initial labels that are incorrect (according to the detection component) by drawing the remaining ones (*i.e.* those that their error probability is less than 0.5) with black color. In subfigure **(f)** we depict the renewed labels $U = E \odot F_u(X, Y, E) + (1 - E) \odot Y$. In subfigure **(g)** we depict the residual corrections that the Refine component $F_r(.)$ yields for the renewed labels U. Finally, in the last subfigure **(h)** we depict the final label estimates $Y' = U + F_r(X, Y, E, U)$ that the Refine step yields.

3. Detect, Replace, Refine for disparity estimation

In order to evaluate the proposed dense image labeling architecture, as well as the other alternative architectures that are explored in our work, we use the dense disparity estimation (stereo matching) task, according to which, given a left and right image, one needs to assign to each pixel of the left image a continuous label that indicates its horizontal displacement in the right image (disparity). Such a task forms a very interesting and challenging testbed for the evaluation of dense labeling algorithms since it requires dealing with several challenges such as accurately preserving disparity discontinuities across object boundaries, dealing with occlusions, as well as recovering the fine details of disparity maps. At the same time it has many practical applications on various autonomous driving and robot navigation or grasping tasks.

Initial disparities: In all the examined architectures, in order to generate the initial disparity labels Y we used the deep patch matching approach that was proposed by W. Luo *et al.* [19] and specifically their architecture with id 37. We then train our models to reconstruct the ground truth labels given as input only the left image X and the initial disparity labels Y. We would like to stress out that the right image of the stereo pair is not provided to our models. This practically means that the trained models cannot rely only on the image evidence for performing the dense disparity labelling task – since disparity prediction from a single image is an ill-posed problem – but they have to learn the joint space of both input X and output labels Y in order to perform the task.

	$>$ 2 pixel	$>$ 3 pixel	$>$ 4 pixel	$>$ 5 pixel	EPE
Architectures	All	All	All	All	All
Initial labels Y	24.3175	22.9004	21.9140	21.1680	12.0218
Single-iteration results					
Replace (baseline)	12.8007	10.4512	8.8966	7.7467	2.4456
Refine (baseline)	14.5996	12.2246	10.3046	8.7873	2.1235
Replace + Refine	11.1152	9.1821	7.8430	6.8550	2.2356
Detect + Replace	11.6970	9.2419	7.6812	6.6018	2.1504
Detect + Refine	10.5309	8.5565	7.2154	6.2186	1.8210
Parallel	11.0146	8.9261	7.5029	6.4742	2.0241
Detect + Replace + Refine	9.5981	7.9764	6.7895	5.9074	1.8569
Multi-iteration results					
Detect + Replace + Refine x2	**8.8411**	**7.2187**	**6.0987**	**5.2853**	**1.6899**

Table 1: Stereo matching results on the Synthetic dataset.

Network architectures: Due to space limitation we provide extensive implementation details about the network architectures that implement each component of our explored architectures as well as about their training procedure in section 3 of our extended technical report [8]. *Here we want to stress out that during the designing of the explored architectures that implement the joint input-output $F(.)$ model, we took care that all of them have roughly the same number of learnable parameters,* i.e. *the same capacity, thus making their quantitative comparison fair.*

4. Experimental results

In this section we present an experimental evaluation of the proposed architecture as well as of the other explored

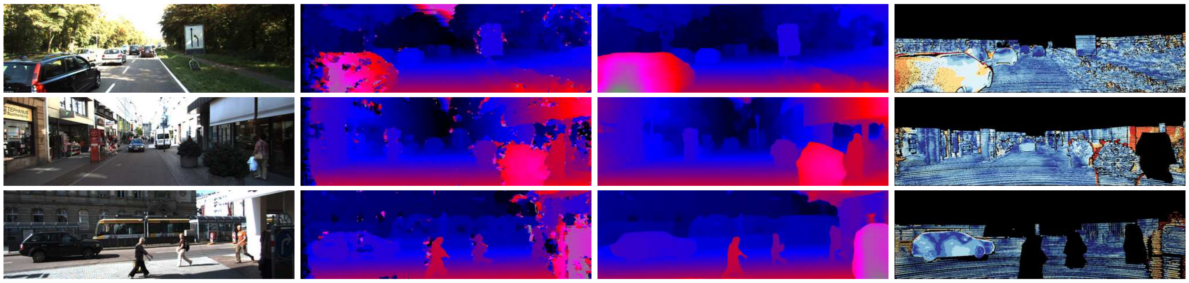

Figure 4: Qualitative results in the validation set of KITTI 2015. From left to right, we depict the left image X, the initial labels Y, the labels Y' that our model estimates, and finally the errors of our estimates w.r.t. ground truth.

	> 2 pixel			> 3 pixel			> 4 pixel			> 5 pixel			EPE		
Architectures	Non-Occ	All	Occ	Non-Occ	All	Occ	Non-Occ	All	Occ	Non-Occ	All	Occ	Non-Occ	All	Occ
Initial labels Y	18.243	26.714	86.125	15.664	23.986	82.330	14.208	22.282	78.758	13.237	21.044	75.579	6.058	8.709	25.598
Single-iteration results															
Replace (baseline)	15.767	21.089	57.197	12.323	16.793	46.303	10.312	14.020	37.922	9.032	12.147	31.770	2.731	3.221	5.818
Refine (baseline)	13.981	19.742	58.039	11.110	16.042	47.732	9.266	13.406	39.218	7.889	11.392	32.467	1.953	2.551	5.665
Replace + Refine	14.262	19.257	52.036	11.297	15.701	43.905	9.552	13.459	37.910	8.408	11.891	33.125	2.292	2.908	6.216
Detect + Replace	15.368	20.984	58.745	11.243	16.169	48.568	8.957	13.176	40.663	7.571	11.179	34.482	2.013	2.676	6.462
Detect + Refine	13.732	19.375	56.383	10.718	15.552	46.281	8.893	12.975	38.197	7.600	11.012	31.478	2.105	2.626	5.389
Parallel	14.917	20.345	57.459	11.363	15.907	46.221	9.234	12.941	37.218	7.840	10.940	30.854	2.012	2.552	5.607
Detect + Replace + Refine	12.845	17.825	50.407	10.096	14.379	41.704	8.285	11.957	34.801	7.057	10.253	29.560	**1.774**	2.368	5.457
Multi-iteration results															
Detect + Replace + Refine x2	**11.529**	**16.414**	**47.922**	**8.757**	**12.874**	**37.977**	**6.997**	**10.482**	**30.634**	**5.911**	**8.916**	**25.514**	1.789	**2.321**	**4.971**

Table 2: Stereo matching results on Middlebury.

	All / All			Noc / All			Runtime
Architectures	D1-bg	D1-fg	D1-all	D1-bg	D1-fg	D1-all	(secs)
Ours	**2.58**	6.04	**3.16**	2.34	4.87	**2.76**	0.4
DispNetC [20]	4.32	**4.41**	4.34	4.11	**3.72**	4.05	0.06
PBCB [28]	**2.58**	8.74	3.61	**2.58**	7.71	3.17	68
Displets v2 [9]	3.00	5.56	3.43	2.73	4.95	3.09	265
MC-CNN [38]	2.89	8.88	3.89	2.48	7.64	3.33	67
SPS-St [34]	3.84	12.67	5.31	3.50	11.61	4.84	2
MBM [6]	4.69	13.05	6.08	4.33	12.12	5.61	0.13

Table 4: Stereo matching results on KITTI 2015 test set.

architectures in the task of dense disparity estimation. For more exhaustive quantitative and qualitative results we refer to section 4 of our extended technical report [8].

4.1. Experimental settings

Training set: In order to train the explored architectures we used the large scale synthetic dataset for disparity estimation that was recently introduced by N. Mayer *et al.* [20] and that includes around $34k$ stereo images. We call this dataset the Synthetic dataset. Also, we enriched this training set with 160 images from the training set of the KITTI 2015 dataset [21, 22][2].

Evaluation sets: We evaluated our architectures on three different datasets. On 2000 images from the test split of the Synthetic dataset, on 40 validation images coming from KITTI 2015 training dataset, and on 15 images from the training set of the Middlebury dataset [26]. Prior to evaluating the explored architectures in the KITTI 2015 validation set, we fine-tuned the models that implement them only on the 160 image of the KITTI 2015 training split.

Evaluation metrics: For evaluation we used the end-point-error (EPE), which is the averaged euclidean distance from the ground truth disparity, and the percentage of disparity estimates that their absolute difference from the ground truth disparity is more than t pixels ($> t$ pixel). Those metrics are reported for the non-occluded pixels (Non-Occ), all the pixels (All), and only the occluded pixels (Occ).

4.2. Quantitative results

In Tables 1, 2, and 3 we report the stereo matching performance of the examined architectures in the Synthetic, Middlebury, and KITTI 2015 evaluation sets correspondingly.

Single-iteration results: We first evaluate all the examined architectures when they are applied for a single iteration. We observe that all of them are able to improve the initial label estimates Y. However, they do not all of them achieve it with the same success. For instance, the baseline models *Replace* and *Refine* tend to be less accurate than the rest models. Compared to them, the *Detect + Replace* and the *Detect + Refine* architectures perform considerably better in two out of three datasets, the Synthetic and the Middlebury datasets. This improvement can only be attributed to the error detection step, which is what it distinguishes them from the baselines, and indicates the importance of having an error detection component in the dense labelling task. Overall, the best single-iteration per-

[2] The entire training set of KITTI 2015 includes 200 images. In our case we split those 200 images in 160 images that were used for training purposes and 40 images that were used for validation purposes

	> 2 pixel			> 3 pixel			> 4 pixel			> 5 pixel			EPE		
Architectures	Non-Occ	All	Occ	Non-Occ	All	Occ	Non-Occ	All	Occ	Non-Occ	All	Occ	Non-Occ	All	Occ
Initial labels Y	8.831	10.649	98.098	6.412	8.253	96.559	5.222	7.059	94.742	4.514	6.339	93.139	1.700	2.457	31.214
Single-iteration results															
Replace (Baseline)	4.997	5.668	37.327	3.329	3.888	27.890	2.452	2.892	19.643	1.924	2.292	15.226	0.858	0.923	3.165
Refine (Baseline)	4.429	5.165	33.028	3.075	3.714	25.107	2.370	2.924	19.610	1.933	2.404	15.978	0.867	0.953	3.384
Replace + Refine	3.963	4.529	**27.411**	2.712	3.209	21.465	2.082	2.507	16.481	1.735	2.098	13.611	0.802	0.865	2.859
Detect + Replace	5.126	5.751	35.554	3.469	4.005	27.656	2.517	2.953	20.519	1.911	2.269	15.947	0.886	0.943	3.108
Detect + Refine	4.482	5.169	34.992	3.054	3.634	26.453	2.328	2.799	19.004	1.865	2.258	14.686	0.863	0.926	2.952
Parallel	5.239	5.952	38.392	3.530	4.139	29.436	2.522	3.017	21.208	1.943	2.338	15.748	0.904	0.962	3.095
Detect + Replace + Refine	3.919	4.610	33.947	2.708	3.294	25.697	2.082	2.570	19.123	1.699	2.112	15.140	0.790	0.858	3.056
Multi-iteration results															
Detect + Replace + Refine x2	**3.685**	**4.277**	28.164	**2.577**	**3.075**	**20.762**	**2.001**	**2.424**	**16.086**	**1.652**	**2.004**	**13.056**	**0.779**	**0.835**	**2.723**

Table 3: Stereo matching results on KITTI 2015 validation set.

formance is achieved by the *Detect + Replace + Refine* architecture that we propose in this paper and combines both the merits of the error detection component and the two stage refinement strategy. Compared to it, the *Parallel* architecture has considerably worse performance, which indicates that the sequential order in the proposed architecture is important for achieving accurate results.

We want to stress out that, since all the explored single iteration architectures have roughly the same capacity (see section 3), the provided experimental results demonstrate that it is better to distribute this capacity in the three described components (Detect, Replace, and Refine) rather than having a single Replace or Refine component.

Multi-iteration results: We also evaluated our best performing architecture, which is the *Detect + Replace + Refine* architecture that we propose, in the multiple iteration case. Specifically, the last entry *Detect + Replace + Refine x2* in Tables 1, 2, and 3 indicates the results of the proposed architecture for 2 iterations and we observe that it further improves the performance w.r.t. the single iteration case. For more than 2 iterations we did not see any further improvement and for this reason we chose not to include those results. Note that in order to train this two iterations model, we first pre-train the single iteration version and then finetune the two iterations version by adding the generated disparity labels from the first iteration in the training set.

KITTI 2015 test set results: We submitted our best solution, which is the proposed *Detect + Replace + Refine* architecture applied for two iterations, on the KITTI 2015 test set evaluation server and we achieved state-of-the-art results in the main evaluation metric, D1-all, surpassing all prior work by a significant margin. The results of our submission, as well as of other competing methods, are reported in Table 4[3]. Note that our improvement w.r.t. the best prior approach corresponds to a more than 10% relative reduction of the error rate. Our total execution time is 0.4 secs, of which around 0.37 secs is used by the patch matching algorithm for generating the initial disparity labels and the rest 0.03 by our *Detect + Replace + Refine x2* architecture (measured in a Titan X GPU). For this submission, after having train the *Detect + Replace + Refine x2* model on the training split (160 images), we further fine-tuned it on both the training and the validation splits (in which we divided the 200 images of KITTI 2015 training dataset).

4.3. Qualitative results

We provide qualitative results from KITTI 2015 validation set in Figure 4. In order to generate them we used the *Detect + Replace + Refine x2* architecture that gave the best quantitative results. We observe that our model is able to recover a good estimate of the actual disparity map even when the initial label estimates are severely corrupted.

5. Conclusions

In our work we explored a family of architectures that performs the structured prediction problem of dense image labeling by learning a deep joint input-output model that (iteratively) improves some initial estimates of the output labels. In this context our main focus was on what is the optimal architecture for implementing this deep model. We argued that the prior approaches of directly predicting the new labels with a feed-forward deep neural networks are sub-optimal and we proposed to decompose the label improvement step in three sub-tasks: 1) detection of the incorrect input labels, 2) their replacement with new labels, and 3) the overall refinement of the output labels in the form of residual corrections. All three steps are embedded in a unified architecture, which we call *Detect + Replace + Refine*, that is end-to-end trainable. We evaluated our architecture in the disparity estimation (stereo matching) task and we report state-of-the-art results in the KITTI 2015 test set.

6. Acknowledgements

This work was supported by the *ANR SEMAPOLIS* project and hardware donation by *NVIDIA*. We would like to thank Sergey Zagoruyko, Francisco Massa, and Shell Xu for their advices with respect to the Torch framework and fruitful discussions.

[3] The link to our KITTI 2015 submission that contains more thorough test set results – both qualitative and quantitative – is: http://www.cvlibs.net/datasets/kitti/eval_scene_flow_detail.php?benchmark=stereo&result=365eacbf1effa761ed07aaa674a9b61c60fe9300

References

[1] J. Carreira, P. Agrawal, K. Fragkiadaki, and J. Malik. Human pose estimation with iterative error feedback. *arXiv preprint arXiv:1507.06550*, 2015. 2

[2] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. *arXiv preprint arXiv:1412.7062*, 2014. 2

[3] L.-C. Chen, A. G. Schwing, A. L. Yuille, and R. Urtasun. Learning deep structured models. In *Proc. ICML*, 2015. 2

[4] R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, number EPFL-CONF-192376, 2011. 1

[5] D. Eigen and R. Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2650–2658, 2015. 1

[6] N. Einecke and J. Eggert. A multi-block-matching approach for stereo. In *2015 IEEE Intelligent Vehicles Symposium (IV)*, pages 585–592. IEEE, 2015. 7

[7] P. Fischer, A. Dosovitskiy, E. Ilg, P. Häusser, C. Hazırbaş, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox. Flownet: Learning optical flow with convolutional networks. *arXiv preprint arXiv:1504.06852*, 2015. 1

[8] S. Gidaris and N. Komodakis. Detect, replace, refine: Deep structured prediction for pixel wise labeling. *arXiv preprint arXiv:1612.04770*, 2016. 4, 5, 6, 7

[9] F. Guney and A. Geiger. Displets: Resolving stereo ambiguities using object knowledge. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4165–4175, 2015. 7

[10] M. Havaei, A. Davy, D. Warde-Farley, A. Biard, A. Courville, Y. Bengio, C. Pal, P.-M. Jodoin, and H. Larochelle. Brain tumor segmentation with deep neural networks. *Medical Image Analysis*, 2016. 2

[11] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015. 1

[12] B. K. Horn and B. G. Schunck. Determining optical flow. *Artificial intelligence*, 17(1-3):185–203, 1981. 1

[13] V. Koltun. Efficient inference in fully connected crfs with gaussian edge potentials. *Adv. Neural Inf. Process. Syst*, 2011. 2

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 1

[15] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the eighteenth international conference on machine learning, ICML*, volume 1, pages 282–289, 2001. 1

[16] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 1

[17] K. Li, B. Hariharan, and J. Malik. Iterative instance segmentation. *arXiv preprint arXiv:1511.08498*, 2015. 2

[18] 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*, pages 3431–3440, 2015. 1

[19] W. Luo, A. G. Schwing, and R. Urtasun. Efficient deep learning for stereo matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5695–5703, 2016. 1, 6

[20] N. Mayer, E. Ilg, P. Häusser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. *arXiv preprint arXiv:1512.02134*, 2015. 1, 7

[21] M. Menze and A. Geiger. Object scene flow for autonomous vehicles. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 7

[22] M. Menze, C. Heipke, and A. Geiger. Joint 3d estimation of vehicles and scene flow. In *ISPRS Workshop on Image Sequence Analysis (ISA)*, 2015. 7

[23] H. Noh, S. Hong, and B. Han. Learning deconvolution network for semantic segmentation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1520–1528, 2015. 1

[24] P. Pinheiro and R. Collobert. Recurrent convolutional neural networks for scene labeling. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 82–90, 2014. 2

[25] C. Russell, P. Kohli, P. H. Torr, et al. Associative hierarchical crfs for object class image segmentation. In *2009 IEEE 12th International Conference on Computer Vision*, pages 739–746. IEEE, 2009. 2

[26] D. Scharstein, H. Hirschmüller, Y. Kitajima, G. Krathwohl, N. Nešić, X. Wang, and P. Westling. High-resolution stereo datasets with subpixel-accurate ground truth. In *German Conference on Pattern Recognition*, pages 31–42. Springer, 2014. 7

[27] A. G. Schwing and R. Urtasun. Fully connected deep structured networks. *arXiv preprint arXiv:1503.02351*, 2015. 2

[28] A. Seki and M. Pollefeys. Patch based confidence prediction for dense disparity map. In *British Machine Vision Conference (BMVC)*, 2016. 7

[29] J. Shotton, M. Johnson, and R. Cipolla. Semantic texton forests for image categorization and segmentation. In *Computer vision and pattern recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008. 1

[30] J. Shotton, T. Sharp, A. Kipman, A. Fitzgibbon, M. Finocchio, A. Blake, M. Cook, and R. Moore. Real-time human pose recognition in parts from single depth images. *Communications of the ACM*, 56(1):116–124, 2013. 1

[31] J. Shotton, J. Winn, C. Rother, and A. Criminisi. Textonboost for image understanding: Multi-class object recognition and segmentation by jointly modeling texture, layout, and context. *International Journal of Computer Vision*, 81(1):2–23, 2009. 1

[32] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 1

[33] R. K. Srivastava, K. Greff, and J. Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015. 5

[34] K. Yamaguchi, D. McAllester, and R. Urtasun. Efficient joint segmentation, occlusion labeling, stereo and flow estimation. In *European Conference on Computer Vision*, pages 756–771. Springer, 2014. 7

[35] F. Yu and V. Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015. 2

[36] S. Zagoruyko and N. Komodakis. Learning to compare image patches via convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4353–4361, 2015. 1

[37] J. Zbontar and Y. LeCun. Computing the stereo matching cost with a convolutional neural network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1592–1599, 2015. 1

[38] J. Žbontar and Y. LeCun. Stereo matching by training a convolutional neural network to compare image patches. *The Journal of Machine Learning Research*, 17(1):2287–2318, 2016. 1, 7

[39] S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. H. Torr. Conditional random fields as recurrent neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1529–1537, 2015. 2