Supplementary Material

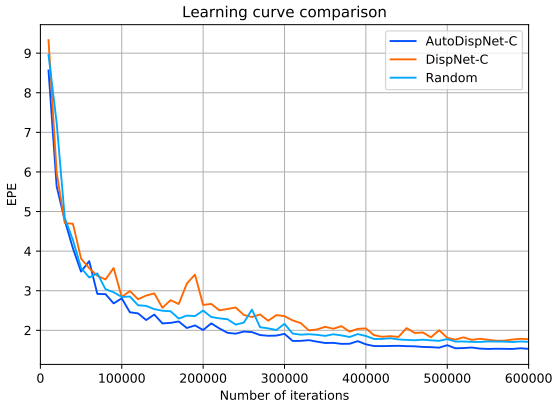


Figure 1: Learning curve comparison. We compare the learning curve of AutoDispNet-C with the baseline DispNet-C and an architecture built with random cells sampled from our search space (denoted by Random). The evolution of EPE over number of iterations is shown for the FlyingThings3D dataset (test split).

1. Learning curves

Figure 1 shows the learning curve (evolution of EPE over number of iterations) of AutoDispNet-C. In addition to the baseline (DispNet-C), we also compare the learning curve of a random cell architecture. We randomly sample cells from the search space and stack them in the same fashion as AutoDispNet-C (see section 6 of main paper). We sample four times and build four different random architectures. After training, we pick the best random architecture based on the validation performance on FlyingThings3D. All networks are trained using the same settings as the baseline. Learning curve of the best random architecture is shown in Figure 1. We observe that AutoDispNet-C clearly outperforms the random architecture and the baseline. We also observe that the random architecture is comparable to the baseline. Our observation is similar to Liu *et al*. [7] on classification, where they also report a surprisingly strong performance for random architectures.

2. Performance of smaller networks

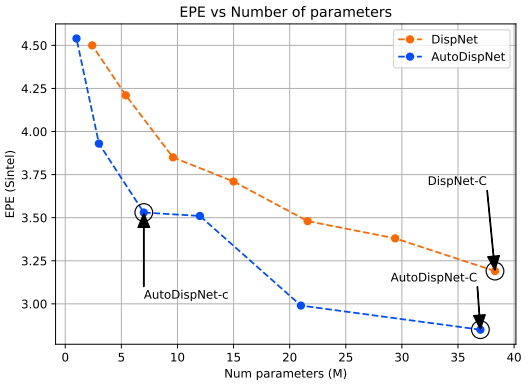
We train networks of reduced capacities for both AutoDispNet-C and DispNet-C architectures. For DispNet-C smaller networks are obtained by multiplying the number of channels for each layer by fixed factor (similar to [3]). Smaller variants of AutoDispNet-C are obtained by reducing the number of channels (C_{init}) for the first cell. A comparison of EPE vs number of parameters and EPE vs FLOPS is shown in Figure 2.

3. Optimizing the refinement network

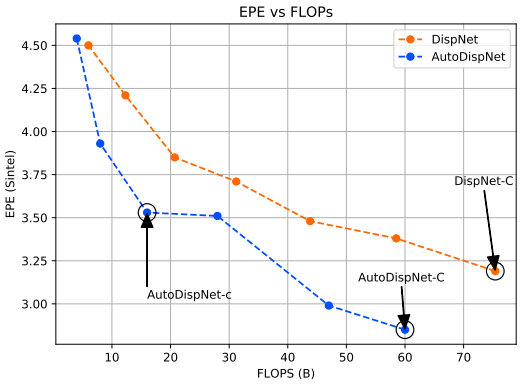
Network stack		**EPE**
C	S	(Sintel)
Dense-DARTS	reuse cells	2.30
Dense-DARTS	Dense-DARTS	2.32
Dense-DARTS + BOHB	reuse cells + hyperparams	2.14
Dense-DARTS + BOHB	reuse cells + BOHB	2.16

Table 1: We show the results of optimizing cells and hyperparameters of the refinement network in a stack containing two networks (AutoDispNet C and S). First row shows a network where cells for the first network are learned using Dense-DARTS and the refinement network reuses these cells. In the second row, we learn new cell structures of the refinement network using Dense-DARTS. In the third row, we learn cells for the first network and tune the hyperparameters using BOHB. In this case, the refinement network reuses both cells and hyperparameters. In the fourth row, we learn new hyperparameters for the refinement network using BOHB but still use the same cell structures as the first network.

In a stacked setting, the refinement network predicts the residual for correcting errors in predictions from the previous network. Since this task is different from predicting disparity from scratch, we trained a search network to learn specialized cells for the refinement task. However, we found that learning cells for the refinement network did not improve performance over reusing cells learned for the first network. The same argument can also be made for optimizing hyperparameters of the refinement network using BOHB. Surprisingly, even BOHB did not yield improve-



(a) EPE vs Params



(b) EPE vs FLOPS

Figure 2: Performance of smaller networks. We compare the test performance of smaller DispNet and AutoDispNet architectures. In Figure 2a, we see that AutoDispNet architectures have a lower error with reduced number of parameters compared to the baseline. A similar trend is observed on comparing the EPE with respect to FLOPS (Figure 2b).
The EPE is shown for the Sintel dataset.

ments over reusing hyperparameters learned for the first network. We show our experimental results in Table 1. We conjecture that the refinement task is much simpler than estimating disparity from scratch and optimizing cells or hyperparameters is trivial in this case.

4. Finetuning on the KITTI dataset

For finetuning on KITTI, we optimize the learning rate and weight decay coefficient using BOHB for the first network in the stack. For running BOHB, we take all samples from KITTI'12 and KITTI'15 datasets and use 70% of the mixture for training. The remaining 30% of the samples are used for validation. We ran BOHB in parallel on 5 GPU workers for a total number of 10 SuccessiveHalving iterations. We used the default BOHB settings with $\eta = 3$ and budgets $10k$, $30k$ and $90k$ mini-batch iterations. For each budget the learning rate is annealed to zero using a cosine schedule. Figure 3 shows the EPE of all sampled configurations throughout the optimization procedure. The optimized hyperparameters are then used to finetune the successive networks in the stack. For the last network, we add two more decoding stages to go to full resolution. Here, we use transposed convolutions instead of upsampling cells because applying the cell structure at higher resolutions becomes computationally expensive.

5. Single view depth estimation

To evaluate on single view depth estimation, we used the proposed extension of DARTS and compare our results with the competitive method by Laina *et al.* [5], which uses a

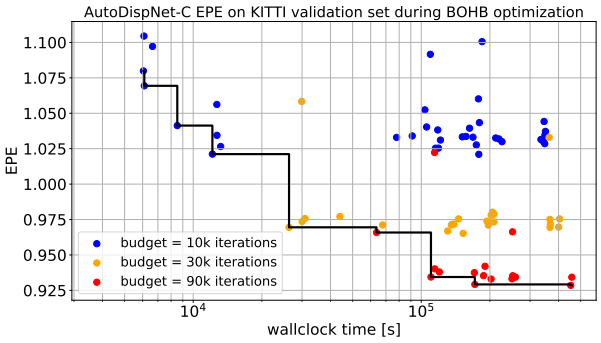


Figure 3: Hyperparameter optimization on KITTI. AutoDispNet-C EPE of all sampled configurations on the different budgets throughout the BOHB optimization procedure. The black line shows the best performing configurations (incumbent) as a function of time.

ResNet based encoder-decoder with hand-designed upsampling blocks. For a fair comparison, we evaluated both architectures by training them on a subset of the SUN3D dataset using the same hyperparameters and loss function. Please note that in this setting, the siamese part of the network is replaced with a single stream. The extracted architecture is then fine-tuned on $\sim 10,000$ samples from the NYU train dataset using BOHB (optimizing the learning rate and weight decay).

Algorithm 1: Hyperband pseudocode

input : min/max budgets b_{min}, b_{max}, η

1 $s_{max} = \lfloor log_\eta \frac{b_{max}}{b_{min}} \rfloor$;
// Begin HB outerloop
2 **for** $s \in \{s_{max}, s_{max} - 1, ..., 0\}$ **do**
3 $N = \lceil \frac{s_{max}+1}{s+1} \cdot \eta^s \rceil$;
4 sample N configurations $C = \{c_1, c_2, ..., c_N\}$;
// Initial budget for SH
5 $b = \eta^{-s} \cdot b_{max}$;
// Start SH innerloop
6 **while** $b \leq b_{max}$ **do**
// Evaluate all configurations in C for the given budget
7 $L = \{\tilde{f}(c, b) | c \in C\}$;
// Keep only the best $\lfloor |C/\eta| \rfloor$ ones
8 $C = top\_k(C, L, \lfloor |C/\eta| \rfloor)$;
// Increase budget by a factor of η
9 $b = \eta \cdot b$;
10 **end**
11 **end**

6. More details on BOHB

BOHB [1] combines Bayesian Optimization (BO) and Hyperband (HB) [6] in order to exhibit strong anytime and final performance. BOHB follows the same strategy as HB to allocate resources to configurations calling the SuccessiveHalving (SH) [4] subroutine repeatedly on its inner loop. Refer to Algorithm 1 for a pseudo-code for Hyperband.

On the outerloop HB samples uniformly N random configurations from the hyperparameter search space (lines 3-4). Afterwards, SH evaluates these N configurations (line 7) on the smallest available budget for this outerloop iteration (line 5) and advances the best $1/\eta$ performing configurations (line 8) to evaluate on a higher budget (increased by a factor of η; line 9). This process goes on until the maximum available budget is reached (line 6). As an example, suppose SH starts with a maximum $N = 27$ number of sampled hyperparameter configurations for training a neural network with a minimum budget of $b_{min} = 1$ epoch (first SH innerloop in Figure 4). With an $\eta = 3$ the next iteration of SH would start the best $N/\eta = 9$ configurations evaluated on some validation set with the second budget $\eta \cdot b_{min} = 3$ epochs. This will continue until only one configuration is evaluated for $b_{max} = 27$ epochs.

In order to account for the very aggressive evaluations with many configurations on the smallest budget (as done in the first SH innerloop), HB resets SH to start with a smaller degree of aggressiveness, i.e. evaluating the new sampled configurations on a larger initial budget (lines 3-5 in Algorithm 1; illustrated in the second innerloop of Figure 4). Nevertheless, the number of configurations N sampled in every HB outerloop iteration (line 3 in Algorithm 1) is chosen such that the same total budget is assigned to each SH run.

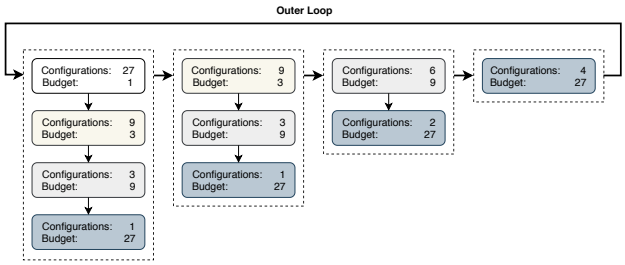


Figure 4: Hyperband inner and outer loops. Hyperband runs SuccessiveHalving on its inner loop with a initial budget and number of starting configurations determined on its outer loop such that the total budget in every SuccessiveHalving run is the same.

Even though BOHB relies on HB to balance the number of configurations it evaluates and the resources assigned to each configuration, it replaces the random sampling in line 3 of Algorithm 1 by a model-based sampling, where the model is build by the configurations evaluated so far. The strong final performance of BOHB arises from the model-based guided search, which effectively focuses more attention to regions in space where good configurations lie.

7. Hyperparameter importance

In order to assess the importance of hyperparameters over the whole search space we analyze our BOHB results using functional analysis of variance (fANOVA; [2]). This method allows us to quantify how much of the performance variance in the configuration space is explained by single hyperparameters, by marginalizing performances over all possible values that other hyperparameters could have taken. These estimates stem from a random forest model fit on all configurations evaluated on specific budgets during the BOHB optimization procedure.

For the hyperparameter optimization conducted on the FlyingThings3D dataset we observe from Figure 5 that the learning rate remains much more important than the weight decay across the first two budgets ($16k$ and $50k$ iterations). For the highest budget of $150k$ iterations, the importance of the weight decay hyperparameter becomes larger, however it is still dominated by the learning rate. Notice the optimal value that BOHB determines for each hyperparameter in our space (gray dashed line in Figure 5). Interestingly, for smaller budgets (i.e. less training iterations) AutoDispNet-C models trained with a small learning rate and high weight decay value (this has a small importance though) perform better on average. As the budget increases the a higher learning rate and a smaller weight decay value are preferred.

We observe similar results when optimizing the learn-

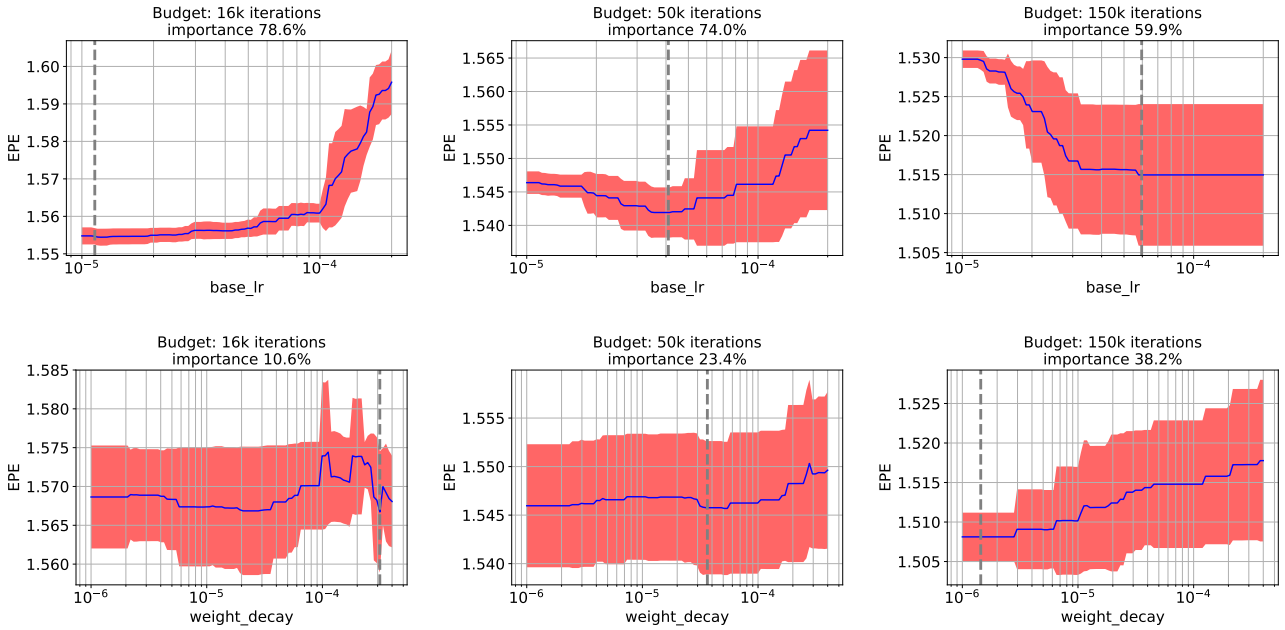


Figure 5: fANOVA plots for all the budgets we run BOHB on the FlyingThings3D dataset. The solid blue line represents the estimated mean EPE (+/- 1std shown by red shaded areas) as a function of hyperparameters as modelled by the random forest we fit to the observations. The importance on top of each plot indicates the fraction of the total variance explained by the individual choice, while the dashed gray line the optimal value as determined by BOHB.

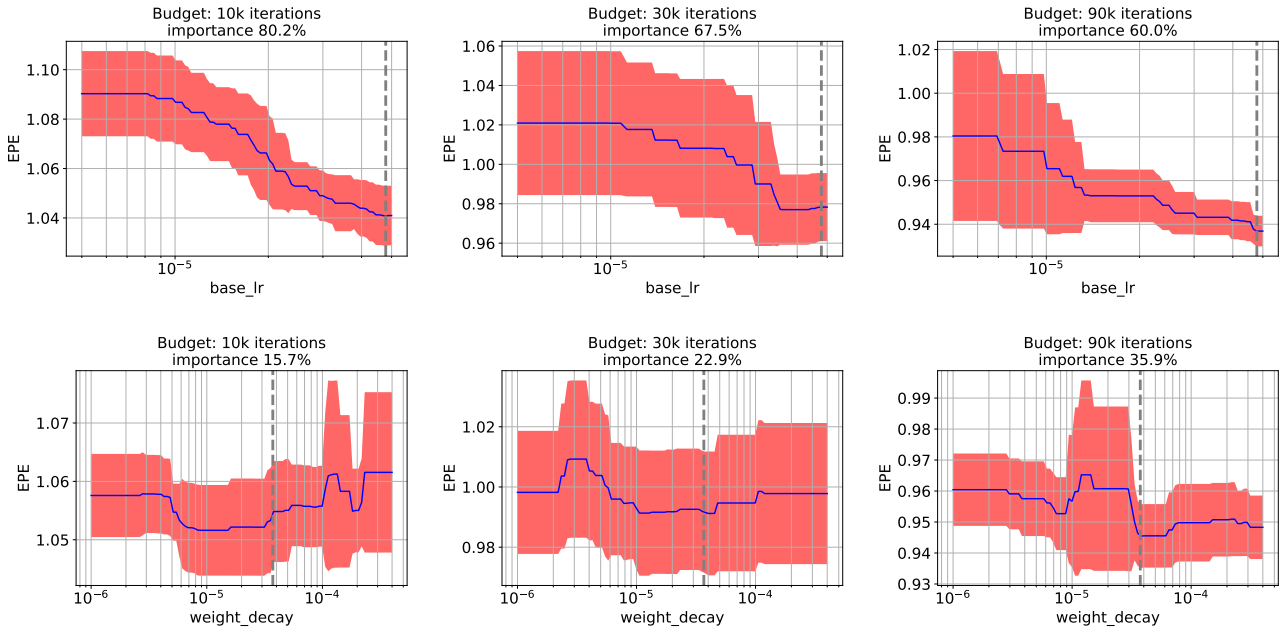


Figure 6: fANOVA plots for all the budgets we run BOHB on the KITTI dataset. The solid blue line represents the estimated mean EPE (+/- 1std shown by red shaded areas) as a function of hyperparameters as modelled by the random forest we fit to the observations. The importance on top of each plot indicates the fraction of the total variance explained by the individual choice, while the dashed gray line the optimal value as determined by BOHB.

ing rate and weight decay for AutoDispNet-C on the KITTI dataset. From the plots in Figure 6 we can see that the learning rate has a higher contribution to the total performance variance throughout all budgets compared to weight decay. However, the optimal values for these two hyperparameters, as determined by BOHB, remain unchanged across these budgets.

References

[1] Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1437–1446, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. 3

[2] Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. An efficient approach for assessing hyperparameter importance. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 754–762, Bejing, China, 22–24 Jun 2014. PMLR. 3

[3] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 1

[4] Kevin Jamieson and Ameet Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In Arthur Gretton and Christian C. Robert, editors, *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, volume 51 of *Proceedings of Machine Learning Research*, pages 240–248, Cadiz, Spain, 09–11 May 2016. PMLR. 3

[5] Iro Laina, Christian Rupprecht, Vasileios Belagiannis, Federico Tombari, and Nassir Navab. Deeper depth prediction with fully convolutional residual networks. In *3D Vision (3DV)*, pages 239–248, 2016. 2

[6] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18:1–52, 04 2018. 3

[7] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019. 1