Fast Neural Architecture Search of Compact Semantic Segmentation Models via Auxiliary Cells

Vladimir Nekrasov*   Hao Chen*   Chunhua Shen   Ian Reid
The University of Adelaide, Australia
E-mail: {vladimir.nekrasov, hao.chen01, chunhua.shen, ian.reid}@adelaide.edu.au

Abstract

Automated design of neural network architectures tailored for a specific task is an extremely promising, albeit inherently difficult, avenue to explore. While most results in this domain have been achieved on image classification and language modelling problems, here we concentrate on dense per-pixel tasks, in particular, semantic image segmentation using fully convolutional networks. In contrast to the aforementioned areas, the design choices of a fully convolutional network require several changes, ranging from the sort of operations that need to be used—e.g., dilated convolutions—to a solving of a more difficult optimisation problem. In this work, we are particularly interested in searching for high-performance compact segmentation architectures, able to run in real-time using limited resources. To achieve that, we intentionally over-parameterise the architecture during the training time via a set of auxiliary cells that provide an intermediate supervisory signal and can be omitted during the evaluation phase. The design of the auxiliary cell is emitted by a controller, a neural network with the fixed structure trained using reinforcement learning. More crucially, we demonstrate how to efficiently search for these architectures within limited time and computational budgets. In particular, we rely on a progressive strategy that terminates non-promising architectures from being further trained, and on Polyak averaging coupled with knowledge distillation to speed-up the convergence. Quantitatively, in 8 GPU-days our approach discovers a set of architectures performing on-par with state-of-the-art among compact models on the semantic segmentation, pose estimation and depth prediction tasks. Code will be made available here: `https://github.com/drsleep/nas-segm-pytorch`

1. Introduction

For years, the design of neural network architectures was thought to be solely a duty of a human expert - it was her responsibility to specify which type of architecture to use, how many layers should there be, how many channels should convolutional layers have and etc. This is no longer the case as the automated neural architecture search - a way of predicting the neural network structure via a non-human expert (an algorithm) - is fast-growing. Potentially, this may well mean that instead of manually adapting a single state-of-the-art architecture for a new task at hand, the algorithm would discover a set of best-suited and high-performing architectures on given data.

Few decades ago, such an algorithm was based on evolutionary programming strategies where best seen so far architectures underwent mutations and their most promising off-springs were bound to continue evolving [2]. Now, we have reached the stage where a secondary neural network, oftentimes called *controller*, replaces a human in the loop, by iteratively searching among possible architecture candidates and maximising the expected score on the held-out set [50]. While there is a lack of theoretical work behind this latter approach, several promising empirical breakthroughs have already been achieved [3, 51].

At this point, it is important to emphasise the fact that such accomplishments required an excessive amount of computational resources—more than $20,000$ GPU-days for the work of Zoph and Le [50] and $2,000$ for Zoph *et al.* [51]. Although a few works have reduced those to single digit numbers on image classification and language processing tasks [23, 30], we consider more challenging dense per-pixel tasks that produce an output for each pixel in the input image and for which no efficient training regimes have been previously presented. Although here we concentrate only on semantic image segmentation, our proposed methodology can immediately be applied to other per-pixel prediction tasks, such as depth estimation and pose estimation. In our experiments, we demonstrate the transferability of the discovered segmentation architecture to the latter problems. Notably, all of them play an important role in computer vision and robotic applications and so far have been relying on manually designed accurate low-latency models for real-world scenarios.

---

*Equal contribution.

1

The focus of our work is to automatically discover compact high-performing fully convolutional architectures, able to run in real-time on a low-computational budget, for example, on the Jetson platform. To this end, we are explicitly looking for structures that not only improve the performance on the held-out set, but also facilitate the optimisation during the training stage. Concretely, we consider the encoder-decoder type of a fully-convolutional network [25], where encoder is represented by a pre-trained image classifier, and the decoder structure is emitted by the controller network. The controller generates the connectivity structure between encoder and decoder, as well as the sequence of operations (that form the so-called *cell*) to be applied on each connected path. The same cell structure is used to form an auxiliary classifier, the goal of which is to provide intermediate supervision and to implicitly over-parameterise the model. Over-parameterisation is believed to be the primary reason behind the successes of deep learning models, and a few theoretical works have already addressed it in simplified cases [8, 39]. Along with empirical results, this is the primary motivation behind the described approach.

Last, but not least, we devise a search strategy that permits to find high-performing architectures within a small number of days using only few GPUs. Concretely, we pursue two goals here:

i.) To prevent 'bad' architectures from being trained for long; and

ii.) To achieve a solid performance estimate as soon as possible.

To tackle the first goal, we divide the training process during the search into two stages. During the first stage, we fix the encoder's weights and pre-compute its outputs, while only training the decoder part. For the second stage, we train the whole model end-to-end. We validate the performance after the first stage and terminate the training of non-promising architectures. For the second goal, we employ Polyak averaging [31] and knowledge distillation [13] to speed-up convergence.

To summarise, our contributions in this work are to propose an efficient neural architecture search strategy for dense-per-pixel tasks that (i.) allows to sample compact high-performing architectures, and (ii.) can be used in real-time on low-computing platforms, such as JetsonTX2. In particular, the above points are made possible by:

- Devising a progressive strategy able to eliminate poor candidates early in the training;
- Developing a training schedule for semantic segmentation able to provide solid results quickly via the means of knowledge distillation and Polyak averaging;
- Searching for an over-parameterised auxiliary cell that provides better training and is obsolete during inference.

2. Related Work

Traditionally, architecture search methods have been relying upon evolutionary strategies [2, 41, 42], where a population of networks (oftentimes together with their weights) is continuously mutated, and less promising networks are being discarded. Modern neuro-evolutionary approaches [24, 32] rely on the same principles and benefit from available computational resources, that allow them to achieve impressive results. Bayesian optimisation methods estimating the probability density of objective function have long been used for hyper-parameter search [4, 38]. Scaling up Bayesian methods for architecture search is an ongoing work, and few kernel-based approaches have already shown solid performance [15, 43].

Most recently, neural architecture search (NAS) strategies based on reinforcement learning (RL) have attained state-of-the-art results on the tasks of image classification and natural language processing [3, 50, 51]. Relying on enormous computational resources, these algorithms comprise a separate neural network, the so-called *'controller'*, that emits an architecture design and receives a scalar reward after the emitted architecture is trained on the task of interest. Notably, thousand of iterations and GPU-days are needed for convergence. Rather than searching for the whole network structure from scratch, these methods tend to look for *cells*—repeatable motifs that can be stacked multiple times in a feedforward fashion.

Several solutions for making NAS methods more efficient have been recently proposed. In particular, Pham *et al*. [30] unroll the computational graph of all possible architectures and allow sharing the weights among different architectures. This dramatically reduces the number of resources needed for convergence. In a similar vein of research, Liu *et al*. [23] exploit a progressive strategy where the network complexity is gradually increased, while the ranking network is trained in parallel to predict the performance of a new architecture. A few methods have been built around continuous relaxation of the search problem. Particularly Luo *et al*. [26] use an encoder to embed the architecture description into a latent space, and estimator to predict the performance of an architecture given its embedding. While these methods make the search process more efficient, they achieve so by sacrificing the expressiveness of the search space, and hence, may arrive to a sub-optimal solution.

In semantic segmentation [19, 20, 21], up to now all the architectures have been manually designed, closely following the winner entries of image classification challenges. Two prominent directions have emerged over the last few years: the encoder-decoder type [19, 25, 29], where better features are learned at the expense of having a spatially coarse output mask; whereas other popular approach discards several down-sampling layers and relies on di-

lated convolutions for keeping the receptive field size intact [6, 47, 49]. Chen *et al.* [7] have also shown that the combination of those two paradigms lead to even better results across different benchmarks. In terms of NAS in semantic segmentation, independently of us and in parallel to our work, a straightforward adaptation of image classification NAS methods was proposed by Chen *et al.* [5]. In it they randomly search for a single segmentation cell design and achieve expressive results by using *almost* 400 *GPUs over the range of* 7 *days*. In contrast to that, our method first and foremost is able to find compact segmentation models only in a fraction of that time. Secondly, it differs significantly in terms of the search design and search methodology.

For the purposes of a clearer presentation of our ideas, we briefly review knowledge distillation, an approach proposed by Hinton *et al.* [13] to successfully train a compact model using the outputs of a single (or an ensemble of) large network(s) pre-trained on the current task. In it, the logits of the pre-trained network are being used as an additional regulariser for the small network. In other words, the latter has to mimic the outputs of the former. Such a method was shown to provide a better learning signal for the small network. As a result of that, it has already found its way across multiple domains: computer vision [48], reinforcement learning [33], continuous learning [18] – to name a few.

3. Methodology

We start with the problem formulation, proceed with the definitions of an auxiliary cell and knowledge distillation loss, and conclude with the overall search strategy.

We primarily focus on two research questions: (i.) how to acquire a reliable estimate of the segmentation model performance as quickly as possible; and (ii.) how to improve the training process of the segmentation architecture through over-parameterisation, obsolete during inference.

3.1. Problem Formulation

We consider dense prediction task T, for which we have multiple training tuples $\{(X_i, y_i)\}$, where both X_i and y_i are 3-dimensional tensors with equal spatial and arbitrary third dimensions. In this work, X_i is a 3-channel RGB image, while y_i is a C-channel one-hot segmentation mask with C being equal to the number of classes, which corresponds to semantic image segmentation. Furthermore, we rely on a mapping $f : X \rightarrow y$ with parameters θ, that is represented by a fully convolutional neural network. We assume that the network f can further be decomposed into two parts: e - representing encoder, and d - for decoder. We initialise encoder e with weights from a pre-trained classification network consisting of multiple down-sampling operations that reduce the spatial dimensions of the input. The

decoder part, on the other hand, has access to several outputs of encoder with varying spatial and channel dimensions. The search goal is to choose which feature maps to use and what operations to apply on them. We next describe the decoder search space in full detail.

3.1.1 Search Space

We restrict our attention to the decoder part, as it is currently infeasible to perform a full segmentation network search from scratch.

As mentioned above, the decoder has access to multiple layers from the pre-trained encoder with varying dimensions. To keep sampled architectures compact and of approximately equal size, each encoder output undergoes a single 1×1 convolution with the same number of output channels. We rely on a recurrent neural network, the controller, to sequentially produce pairs of indices of which layers to use, and what operations to apply on them. In particular, this sequence of operations is combined to form a cell (see example in Fig. 1). The same cell but with different weights is applied to each layer inside the sampled pair, and the outputs of two cells are summed up. The resultant layer is added to the sampling pool. The number of times pairs of layers are sampled is controlled by a hyper-parameter, which we set to 3 in our experiments, allowing the controller to recover such encoder-decoder architectures as FCN [25], or RefineNet [19]. All non-sampled summation outputs are concatenated, before being fed into a single 1×1 convolution to reduce the number of channels followed by the final classification layer.

Each cell takes a single input with the controller first deciding which operation to use on that input. The controller then proceeds by sampling with replacement two locations out of two, i.e., of input and the result of the first operation, and two corresponding operations. The outputs of each operation are summed up, and all three layers (from each operation and the result of their summation) together with the initial two can be sampled on the next step. The number of times the locations are sampled inside the cell is controlled by another hyper-parameter, which we also set to 3 in our experiments in order to keep the number of all possible architectures to a feasible amount[1]. All existing non-sampled summation outputs inside the cell are summed up, and used as the cell output. In this case, we resort to sum as concatenation may lead to variable-sized outputs between different architectures.

Based on existing research in semantic segmentation, we consider 11 operations:

---

[1]Taking into account symmetrical – thus, identical – architectures, we estimate the number of unique connections in the decoder part to be 120, and the number of unique cells $\sim 10^{10}$, leading to $\sim 10^{12}$, which is on-par with concurrent works.
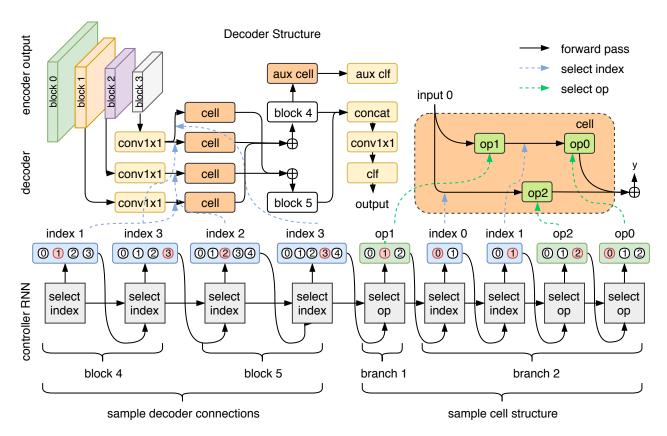
Figure 1 – Example of the encoder-decoder auxiliary search layout. Controller RNN (*bottom*) first generates connections between encoder and decoder (*top left*), and then samples locations and operations to use inside the cell (*top right*). All the cells (including auxiliary cell) share the emitted design. In this example, the controller first samples two indices (*block1* and *block3*), both of which pass through the corresponding cells, before being summed up to create *block4*. The controller then samples *block2* and *block3* that are merged into *block5*. Since *block4* was not sampled, it is concatenated with *block5* and fed into 1×1 convolution followed by the final classifier. The output of *block4* is also passed through an auxiliary cell for intermediate supervision. To emit the cell design, the controller starts by sampling the first operation applied on the cell input (*op1*), followed by sampling of two indices – *index0*, corresponding to the cell input, and *index1* of the output layer after the first operation. Two operations – *op2* and *op0* – are applied on each index, respectively, and their summation serves as the cell output.

- conv 1×1,
- conv 3×3,
- separable conv 3×3,
- separable conv 5×5,
- global average pooling followed by upsampling and conv 1×1,
- conv 3×3 with dilation rate 3,
- conv 3×3 with dilation rate 12,
- separable conv 3×3 with dilation rate 3,
- separable conv 5×5 with dilation rate 6,
- skip-connection,
- zero-operation that effectively nullifies the path.

An example of the search layout with 2 decoder blocks and 2 cell branches is depicted on Fig. 1.[2]

---

[2]Please refer to *Appendix A* for more details on the search space and the sampling procedure.

3.2. Search Strategy

We randomly divide the training set into two disjoint sets - meta-train and meta-val. The meta-train subset is used to train the sampled architecture on the given task (i.e., semantic segmentation), whereas meta-val, on the other hand, is used to evaluate the trained architecture and provide the controller with a scalar, oftentimes called *reward* in the reinforcement learning literature. Given the sampled sequence, its logarithmic probabilities and the reward signal, the controller is optimised via proximal policy optimisation (PPO) [36]. Hence, there are two training processes present: inner - optimisation of the sampled architecture on the given task, and outer - optimisation of the controller. We next concentrate on the inner loop.

3.2.1 Progressive Stages

We divide the inner training process into two stages. During the first stage, the encoder weights are fixed and its out-

4

puts are pre-computed, while only decoder is being trained. This leads to a quick adaptation of the decoder weights and a reasonable estimate of the performance of the sampled architecture. We exploit a simple heuristic to decide whether to continue training the sampled architecture for the second stage, or not. Concretely, the current reward value is being compared with the running mean of rewards seen so far, and if it is higher, we continue training. Otherwise, with probability $1 - p$ we terminate the training process. The probability p is annealed throughout our search (starting from 0.9).

The motivation behind this is straightforward: the results of the first stage, while noisy, can still provide a reasonable estimate of the potential of the sampled architecture. At the very least, they would present a reliable signal that the sampled architecture is non-promising, while spending only few seconds on it. Such a simple approach encourages exploration during early stages of search akin to the ϵ-greedy strategy often used in the multi-armed bandit problem [44].

3.2.2 Fast Training via Knowledge Distillation and Weights' Averaging

Semantic segmentation models are notable for requiring many iterations to converge. Partially, this is addressed by initialising the encoder part from a pre-trained classification network. Unfortunately, no such thing exists for decoder.

Fortunately, though, we can explore several alternatives that provide faster convergence. Besides tailoring our optimisation hyper-parameters, we rely on two more tricks: firstly, we keep track of the running average of the parameters during each stage and apply them before the final validation [31]. Secondly, we append an additional l_2-loss term between the logits of the current architecture and a pre-trained teacher network. We can either pre-compute the teacher's outputs beforehand, or acquire them on-the-fly in case the teacher's computations are negligible.

The combination of both of these approaches allows us to receive a very reliable estimate of the performance of the semantic segmentation model as quickly as possible without a significant overhead.

3.2.3 Intermediate Supervision via Auxiliary Cells

We further look for ways of easing optimisation during fast search, as well as during a longer training of semantic segmentation models. Thus, still aligning with the goal of having a compact but accurate model, we explicitly aim to find ways of performing steps that are beneficial during training and obsolete during evaluation.

One approach that we consider here is to append an auxiliary cell after each summation between pairs of main cells - the auxiliary cell is identical to the main cell and can either be conditioned to output ground truth directly, or to mimic the teacher's network predictions (or the combination of the above two). At the same time, it does not influence the output of the main classifier either during the training or testing and merely provides better gradients for the rest of the network. In the end, the reward per the sampled architecture will still be decided by the output of the main classifier. For simplicity, we only apply the segmentation loss on all auxiliary outputs.

The notion of intermediate supervision is not novel in neural networks, but to the best of our knowledge, prior works have merely been relying on a simple auxiliary classifier, and we are the first to tie up the design of decoder with the design of the auxiliary cell. We demonstrate the quantitative benefits of doing so in our ablation studies (Sect. 4.2).
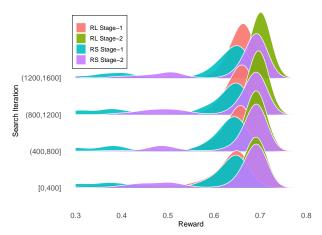
Furthermore, our motivation behind searching for cells that may also serve as intermediate supervisors stems from ever-growing empirical (and theoretical under certain assumptions) evidence that deep networks benefit from over-parameterisation during training [8, 39]. While auxiliary cells provide an implicit notion of over-parameterisation, we could have explicitly increased the number of channels and then resorted to pruning. Nonetheless, pruning methods tend to result in unstructured networks often carrying no tangible benefits in terms of the runtime speed, whereas our solution simply permits omitting unused layers during inference.

4. Experiments

We conduct extensive experiments on PASCAL VOC which is an established semantic segmentation benchmark that comprises 20 semantic classes (and background) and provides 1464 training images [9]. For the search process, we extend those to more than 10000 by exploiting annotations from BSD [11]. As commonly done, during search, we keep 10% of those images for validation of the sampled architectures that provides the controller with the reward signal. For the first stage, we pre-compute the encoder outputs on 4000 images and store them for faster processing.

The controller is a two-layer recurrent LSTM [14] neural network with 100 hidden units. All the units are randomly initialised from a uniform distribution. We use PPO [36] for optimisation with the learning rate of 0.0001.

The encoder part of our network is MobileNet-v2 [34], pretrained on MS COCO [22] for semantic segmentation using the Light-Weight RefineNet decoder [28]. We omit the last layers and consider four outputs from layers $2, 3, 6, 8$ as inputs to decoder; 1×1 convolutional layers used for adaptation of the encoder outputs have 48 output channels during search and 64 during training. Decoder weights are randomly initialised using the Xavier scheme [10]. To perform knowledge distillation, we use Light-Weight RefineNet-152 [28], and apply ℓ_2-loss with the coefficient of 0.3 which was set using the grid search.
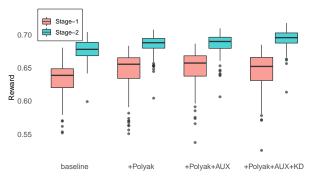
5

Figure 2 – Distribution of rewards per each training stage for reinforcement learning (*RL*) and random search (*RS*) strategies. Higher peaks correspond to higher density.

The knowledge distillation outputs are pre-computed for the first stage and omitted during the second one in the interests of time. Polyak averaging is applied with the decay rates of 0.9 and 0.99, correspondingly. Batch normalisation statistics are updated during both stages.

All our search experiments are being conducted on two 1080Ti GPU cards, with the search process being terminated after 4 days. All runtime measurements are carried out on a single 1080Ti card, or on JetsonTX2, if mentioned otherwise. In particular, we perform the forward pass 100 times and report the mean result together with standard deviation.

4.1. Search Results

For the inner training of the sampled architectures, we devise a fast and stable training strategy: we exploit the Adam learning rule [16] for the decoder part of the network, and SGD with momentum - for encoder. In particular, we use learning rates of $3e$-3 and $1e$-3, respectively. We pre-train each sampled architecture for 5 epochs on the

Figure 3 – Distribution of rewards during each training stage of the search process across setups with Polyak averaging (*Polyak*), intermediate supervision through auxiliary cells (*AUX*) and knowledge distillation (*KD*).

first stage, and for 1 on the second (in case the stopping criterion is not triggered). As the reward signal, we consider the geometric mean of three quantities: namely,

i.) mean intersection-over-union (IoU), or Jaccard Index [9], primarily used across semantic segmentation benchmarks;

ii.) frequency-weighted IoU, that scales each class IoU by the number of pixels present in that class, and

iii.) mean-pixel accuracy, that averages the number of correct pixels per each class. When computing, we do not include background class as it tends to skew the results due to a large number of pixels belonging to background. As mentioned above, we keep the running mean of rewards after the first stage to decide whether to continue training a sampled architecture.

We visualise the reward progress during both stages on Figure 2. As evident from it, the quality of the emitted architectures grows with time - it is even possible that more iterations would lead to better results, although we do not explore that to save the time spent. On the other hand, while random search has the potential of occasionally sampling decent architectures, it finds only a fraction of them in comparison to the RL-based controller.
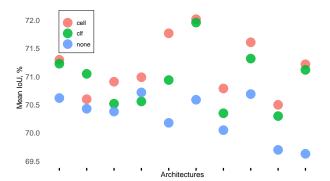
Moreover, we evaluate the impact of the inclusion of Polyak averaging, auxiliary cells and knowledge distillation on each training stage. To this end, we randomly sample and train 140 architectures. We visualise the distributions of rewards on Fig. 3. All the tested settings significantly outperform baseline on both stages, and the highest rewards on the second stage are attained when using all of the components above.

4.2. Effect of Intermediate Supervision via Auxiliary Cells

After the search process is finished, we select 10 architectures discovered by the RL controller with highest rewards and proceed by carrying out additional ablation studies aimed to estimate the benefit of the proposed auxiliary scheme in case the architectures are allowed to train for longer.

In particular, we train each architecture for 20 epochs on BSD together with PASCAL VOC and 30 epochs on PASCAL VOC only. For simplicity, we omit Polyak averaging and knowledge distillation. Three distinct setups are being tested: concretely, we estimate whether intermediate supervision helps at all, and whether auxiliary cell is superior to a plain auxiliar classifier
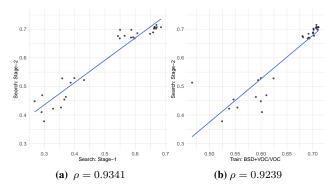
The results of these ablation studies are given in Fig. 4. Auxiliary supervised architectures achieve significantly higher mean IoU, and, in particular, architectures with auxiliary cells attain best results in 8 out of 10 cases, reaching 3 absolute best values across all the setups and architectures.

6

Figure 4 – Ablation studies on the value of intermediate supervision (*none*), and the type of supervision (*cell* or *clf*). Each tick on the x-axis corresponds to a different architecture.

4.3. Relation between search rewards and training performance

We further measure the correlation effect between rewards acquired during the search and mean IoU attained by same architectures trained for longer. To this end, we randomly sample 30 architectures out of those explored by the controller: for fair comparison, we sample 10 architectures with poor search performance (with rewards being less than 0.4), 10 with medium rewards (between 0.4 and 0.6), and 10 with high rewards (> 0.6). We train each architecture on BSD+VOC and VOC as in Sect. 4.2, rank each according to its rewards, and mean IoU, and measure the Spearman's rank correlation coefficient. As visible in Fig. 5, there is a strong correlation between rewards after each stage, as well as between the final reward and mean IoU. This signals that our search process is able to reliably differentiate between poor-performing and well-performing architectures.
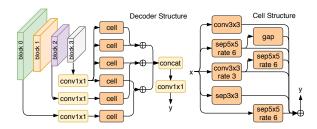

Figure 5 – Correlation between rewards acquired during search stages *(a)* and mean IoU after full training *(b)* of 30 architectures on BSD+VOC/VOC.

4.4. Full Training Results

Finally, we choose 3 best performing architectures from Sect. 4.2 and train each on the full training set, augmented

with annotations from MS COCO [22]. The training setup is analogous to the aforementioned one with the first stage being trained for 30 epochs (on COCO+BSD+VOC), the second stage - for 50 (BSD+VOC), and the last one - for 100 (VOC only). After each stage, the learning rates are halved. Additionally, halfway through the last stage we freeze the batch norm statistics and divide the learning rate in half. We exploit intermediate supervision via auxiliary cells with coefficients of $0.3, 0.25, 0.2, 0.15$ across the stages.

Figure 6 – Automatically discovered decoder architecture (*arch0*). We visualise the connectivity structure between encoder and decoder (*top*), and the cell design (*bottom*). $\oplus$ represents an element-wise summation operation applied to each branch scaled to the highest spatial resolution among them (via bilinear interpolation), while *'gap'* stands for global average pooling.

Quantitative results are given in Table 1.[3] The architectures discovered by our method achieve competitive performance in comparison to state-of-the-art compact models and even do so with a significantly lower number of floating point operations for same output resolution. At the same time, the found architectures can be run in real-time both on a generic GPU card and JetsonTX2.[4] Qualitatively (Fig. 7), our model is able to better recognise similar and easily confused classes (*e.g.* horse – dog in row 3, and cat – dog in row 5), better segment foreground from background and avoid spurious predictions (rows 1,2,4,5).

We visualise[5] the structure of the highest performing architecture (*arch0*) on Fig. 6. With multiple branches encoding information of different scales, it resembles several prominent blocks in semantic segmentation, notably the ASPP module [7]. Importantly, the cell found by our method differs in the way the receptive field size is controlled. Whereas ASPP solely relies on various dilation rates, here convolutions with different kernel sizes arranged in a cascaded manner allow more flexibility. Furthermore, this design is more computationally efficient and has higher expressiveness as intermediate features are easily re-used.

4.5. Transferability to other Dense Output Tasks

4.5.1 Pose Estimation

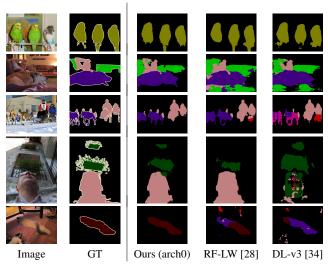We further apply the found architectures on the task of pose estimation. In particular, the MPII [1] and MS COCO Key-

---

[3]Per-class measures are provided in *Appendix B*.

[4]Please refer to *Appendix C* on notes regarding the Jetson's runtime.

[5]Other architectures are visualised in *Appendix A*.

Model	Val mIoU,%	MAdds,B	Params,M	Output Res	Runtime,ms (JetsonTX2/1080Ti)	
DeepLab-v3-ASPP [34]	75.7	5.8	4.5	32×32	69.67±0.53	**8.09±0.53**
DeepLab-v3 [34]	75.9	8.73	**2.1**	64×64	122.07±0.58	11.35±0.43
RefineNet-LW [28]	76.2	9.3	3.3	128×128	144.85 ± 0.49	12.00±0.26
Ours (arch0)	**78.0**	4.47	2.6	128×128	109.36±0.39	14.86±0.31
Ours (arch1)	77.1	**2.95**	2.8	64×64	67.57±0.54	11.04±0.23
Ours (arch2)	77.3	3.47	2.9	64×64	**64.60±0.33**	8.86±0.26

Table 1 – Results on validation set of PASCAL VOC after full training on COCO+BSD+VOC. All networks share the same backbone - MobileNet-v2. FLOPs and runtime are being measured on 512×512 inputs. For DeepLab-v3 we use official models provided by the authors.

Figure 7 – Inference results of our model (*arch0*) on validation set of PASCAL VOC, together with Light-Weight-RefineNet (*RF-LW*) and DeepLab-v3 (*DL-v3*). All the models rely on MobileNet-v2 as the encoder.

point [22] datasets are used as our benchmark. MPII includes 25K images containing 40K people with 16 annotated body joints. The evaluation measure is PCKh [35] with thresholds of 0.5 and 0.1. The COCO dataset comprises 200K images of 250K people with 17 body joints. Based on object keypoint similarity (OKS)[6], we report average precision (AP) and average recall (AR) over 10 different OKS thresholds.

Our quantitative results are in Table 2.[7] We follow the training protocol of Xiao *et al*. [45] and do not tune our architectures. As can be seen from the results, the discovered architectures achieve competitive performance even in comparison to a more powerful ResNet-50-based model.

	MPII		COCO		
Model	Mean@0.5	Mean@0.1	AP	AR	Params,M
DeepLab-v3+ [7]	86.6	31.7	0.668	0.700	5.8
ResNet-50 [45]	**88.5**	**33.9**	**0.704**	**0.763**	34.0
Ours (arch0)	86.5	31.4	0.658	0.691	**2.6**
Ours (arch1)	87.0	32.0	0.659	0.694	2.8
Ours (arch2)	87.1	31.8	0.659	0.693	2.9

Table 2 – Comparisons on MPII validation and COCO val2017. Flip test is used. For COCO, the same detector as in [45] is used for all models. DeepLab-v3+ is our re-implementation based on the official code.

---

[6] http://cocodataset.org/#keypoints-eval

[7] Additional qualitative and quantitative results are in *Appendix B*.

4.5.2 Depth Estimation

Finally, we train the architectures on NYUDv2 [37] for depth prediction. Following previous work [27], we only use 25K training images with depth annotations from the Kinect sensor, and report validation results on 654 images in Table 3. Among other compact real-time networks, we achieve significantly better results across all the metrics without any additional tricks. Note also that the work in [27] trained the depth model jointly with semantic segmentation, thus using extra information.

	Ours				
	arch0	arch1	arch2	RF-LW [27]	CReaM [40]
RMSE (lin)	**0.523**	0.526	0.525	0.565	0.687
RMSE (log)	0.184	**0.183**	0.189	0.205	0.251
abs rel	0.136	**0.131**	0.140	0.149	0.190
sqr rel	0.089	**0.086**	0.093	0.105	–
$\delta < 1.25$	0.830	**0.832**	0.820	0.790	0.704
$\delta < 1.25^2$	0.967	**0.968**	0.966	0.955	0.917
$\delta < 1.25^3$	**0.992**	**0.992**	**0.992**	0.990	0.977
Parameters, M	2.6	2.8	2.9	3.0	**1.5**

Table 3 – Quantitative results on the validation set of NYUDv2. For RMSE, abs rel and sqr rel lower values are better, whereas for accuracy (δ) higher values are better.

5. Discussion and Conclusions

There is little doubt that manual design of neural architectures is a tedious and difficult task to handle. It is even more complicated to come up with a design of compact and high-performing architecture on challenging dense prediction problems, such as semantic segmentation. In this work, we showcased a simple and reliable approach of searching for fully convolutional architectures within a reasonable amount of time and computational resources. Our method is based around over-parameterisation of small networks that allows them to converge to better solutions. We achieved competitive performance to manually designed state-of-the-art compact architectures on PASCAL VOC, while searching only for 4 days on 2 GPU cards. Moreover, best found architectures also attained excellent results on other dense per-pixel tasks — pose estimation and depth prediction.

Our future goals include exploration of alternative ways of over-parameterisation and search space description.

Acknowledgements

References

[1] M. Andriluka, L. Pishchulin, P. Gehler, and B. Schiele. 2d human pose estimation: New benchmark and state of the art analysis. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2014.

[2] P. J. Angeline, G. M. Saunders, and J. B. Pollack. An evolutionary algorithm that constructs recurrent neural networks. *IEEE Trans. Neural Networks*, 1994.

[3] B. Baker, O. Gupta, N. Naik, and R. Raskar. Designing neural network architectures using reinforcement learning. *Proc. Int. Conf. Learn. Representations*, 2017.

[4] J. Bergstra, D. Yamins, and D. D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *Proc. Int. Conf. Mach. Learn.*, 2013.

[5] L. Chen, M. D. Collins, Y. Zhu, G. Papandreou, B. Zoph, F. Schroff, H. Adam, and J. Shlens. Searching for efficient multi-scale architectures for dense image prediction. *arXiv: Comp. Res. Repository*, abs/1809.04184, 2018.

[6] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2018.

[7] L. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *Proc. Eur. Conf. Comp. Vis.*, 2018.

[8] S. Du and J. Lee. On the power of over-parametrization in neural networks with quadratic activation. In *Proc. Int. Conf. Mach. Learn.*, 2018.

[9] M. Everingham, L. J. V. Gool, C. K. I. Williams, J. M. Winn, and A. Zisserman. The pascal visual object classes (VOC) challenge. *Int. J. Comput. Vision*, 2010.

[10] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proc. Int. Conf. Artificial Intell. & Stat.*, 2010.

[11] B. Hariharan, P. Arbelaez, L. D. Bourdev, S. Maji, and J. Malik. Semantic contours from inverse detectors. In *Proc. IEEE Int. Conf. Comp. Vis.*, 2011.

[12] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2016.

[13] G. E. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *Proc. Advances in Neural Inf. Process. Syst.*, 2014.

[14] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 1997.

[15] K. Kandasamy, W. Neiswanger, J. Schneider, B. Póczos, and E. Xing. Neural architecture search with bayesian optimisation and optimal transport. *arXiv: Comp. Res. Repository*, 2018.

[16] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv: Comp. Res. Repository*, abs/1412.6980, 2014.

[17] I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari, and N. Navab. Deeper depth prediction with fully convolutional residual networks. In *Proc. Int. Conf. 3D Vision*, 2016.

[18] Z. Li and D. Hoiem. Learning without forgetting. In *Proc. Eur. Conf. Comp. Vis.*, 2016.
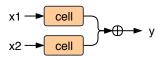
[19] G. Lin, A. Milan, C. Shen, and I. D. Reid. RefineNet: Multi-path refinement networks for high-resolution semantic segmentation. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2017.

[20] G. Lin, C. Shen, I. D. Reid, and A. van den Hengel. Efficient piecewise training of deep structured models for semantic segmentation. *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 3194–3203, 2016.

[21] G. Lin, C. Shen, A. van den Hengel, and I. Reid. Exploring context with deep structured models for semantic segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2017.

[22] T. Lin, M. Maire, S. J. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: common objects in context. In *Proc. Eur. Conf. Comp. Vis.*, 2014.

[23] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L. Li, L. Fei-Fei, A. L. Yuille, J. Huang, and K. Murphy. Progressive neural architecture search. In *Proc. Eur. Conf. Comp. Vis.*, 2018.

[24] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu. Hierarchical representations for efficient architecture search. *Proc. Int. Conf. Learn. Representations*, 2018.

[25] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2015.

[26] R. Luo, F. Tian, T. Qin, and T. Liu. Neural architecture optimization. *Proc. Advances in Neural Inf. Process. Syst.*, 2018.

[27] V. Nekrasov, T. Dharmasiri, A. Spek, T. Drummond, C. Shen, and I. D. Reid. Real-time joint semantic segmentation and depth estimation using asymmetric annotations. *arXiv: Comp. Res. Repository*, abs/1809.04766, 2018.

[28] V. Nekrasov, C. Shen, and I. D. Reid. Light-weight refinenet for real-time semantic segmentation. In *Proc. British Machine Vis. Conf.*, 2018.

[29] H. Noh, S. Hong, and B. Han. Learning deconvolution network for semantic segmentation. In *Proc. IEEE Int. Conf. Comp. Vis.*, 2015.

[30] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean. Efficient neural architecture search via parameter sharing. In *Proc. Int. Conf. Mach. Learn.*, 2018.

[31] B. T. Polyak and A. B. Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 1992.

[32] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin. Large-scale evolution of image classifiers. In *Proc. Int. Conf. Mach. Learn.*, 2017.

[33] A. A. Rusu, S. G. Colmenarejo, Ç. Gülçehre, G. Desjardins, J. Kirkpatrick, R. Pascanu, V. Mnih, K. Kavukcuoglu, and R. Hadsell. Policy distillation. *Proc. Int. Conf. Learn. Representations*, 2016.

[34] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2018.

[35] B. Sapp and B. Taskar. MODEC: multimodal decomposable models for human pose estimation. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 3674–3681, 2013.

[36] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv:*

Comp. Res. Repository, 2017.

[37] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus. Indoor segmentation and support inference from RGBD images. In *Proc. Eur. Conf. Comp. Vis.*, 2012.

[38] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In *Proc. Advances in Neural Inf. Process. Syst.*, 2012.

[39] M. Soltanolkotabi, A. Javanmard, and J. D. Lee. Theoretical insights into the optimization landscape of over-parameterized shallow neural networks. *IEEE Transactions on Information Theory*, 2018.

[40] A. Spek, T. Dharmasiri, and T. Drummond. CReaM: Condensed real-time models for depth prediction using convolutional neural networks. *Proc. IEEE/RSJ Int. Conf. Intelligent Robots & Systems*, 2018.

[41] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial Life*, 2009.

[42] K. O. Stanley and R. Miikkulainen. Evolving neural network through augmenting topologies. *Evolutionary Computation*, 2002.

[43] K. Swersky, D. Duvenaud, J. Snoek, F. Hutter, and M. A. Osborne. Raiders of the lost architecture: Kernels for bayesian optimization in conditional parameter spaces. *arXiv: Comp. Res. Repository*, 2014.

[44] C. J. C. H. Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, 1989.

[45] B. Xiao, H. Wu, and Y. Wei. Simple baselines for human pose estimation and tracking. In *Proc. Eur. Conf. Comp. Vis.*, 2018.

[46] C. Yu, J. Wang, C. Peng, C. Gao, G. Yu, and N. Sang. Bisenet: Bilateral segmentation network for real-time semantic segmentation. In *Proc. Eur. Conf. Comp. Vis.*, 2018.

[47] F. Yu, V. Koltun, and T. A. Funkhouser. Dilated residual networks. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2017.

[48] A. R. Zamir, A. Sax, W. B. Shen, L. J. Guibas, J. Malik, and S. Savarese. Taskonomy: Disentangling task transfer learning. *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2018.

[49] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia. Pyramid scene parsing network. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2017.

[50] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. *Proc. Int. Conf. Learn. Representations*, 2017.

[51] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2018.

Appendix A: Search Space

Decoder connectivity structure

Our fully-convolutional networks follow the encoder-decoder design paradigm. In particular, in place of the encoder we rely on an existing image classifier - here, MobileNet-v2 [34]. The decoder has access to 4 layers from the encoder with varying dimensions. To form connections inside the decoder part, we i.) first sample a pair of indices out of 4 possible choices with replacement, ii.) apply the same set of operations (*cell*) on each sample index, iii.) sum up the outputs (Fig. 8), and iv.) add the resultant layer into the sampling pool. In total, we repeat this process 3 times. Finally, all non-sampled summation outputs are concatenated, before being fed into a single 1×1 convolution to reduce the number of channels followed by the final classification layer.

Figure 8 – Block structure of the decoder. The same cell operation is applied to two different layers specified by the connectivity configuration. If the two features have different size, the smaller one is scaled up via bilinear upsampling to match the larger one.
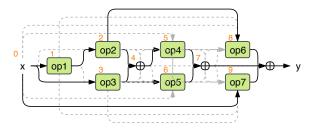
Cell structure

The cell structure is similarly generated via sampling a set of operations and corresponding indices. Nevertheless, there are several notable differences:

1. The operation at each position can vary;

2. A single operation is applied to the input without any aggregation operator;

3. After that, two indices and two operations are being sampled with replacement, with the corresponding outputs being summed up (this is repeated 3 times);

4. The outputs of each operation along with their summation layer are added into the sampling pool.

An example of the cell structure with its complete search space is illustrated in Fig. 9.

Architecture description

We use a list of integers to encode the architecture found by the controller, corresponding to the output sequence of the RNN. Specifically, the list describes the connectivity structure and the cell configuration. For example, the following connectivity structure $[[c_1, c_2], [c_3, c_4], [c_5, c_6]]$ contains three pairs of digits, indicating the input index c_k of a

Figure 9 – Example cell structure of the decoder. The digit at the upper left corner of each operator is the index of the intermediate features. The cell is designed to utilize these features by skip connections. Except the first operator, other operators can be connected from any previous outputs. The solid black lines indicate the used paths and dashed grey lines are other unused possible paths. The cell configuration to generate the above cell is [op1, [1, 0, op2, op3], [4, 3, op4, op5], [2, 0, op6, op7]].

corresponding layer in the sampling pool. The cell configuration, $[o_1, [i_2, i_3, o_2, o_3], [i_4, i_5, o_4, o_5], [i_6, i_7, o_6, o_7]]$, comprises the first operation o_1 followed by three cell branches with the operation o_j applied on the index i_j.

We provide the description of operations in Table 4, and visualise the discovered structures in Fig. 10 (*arch0*), Fig. 11 (*arch1*), and Fig. 12 (*arch2*). Note that inside the cell only the final summation operator is displayed as intermediate summations would lead to identical structures.

Index	Abbreviation	Description
0	conv1x1	conv 1×1
1	conv3x3	conv 3×3
2	sep3x3	separable conv 3×3
3	sep5x5	separable conv 5×5
4	gap	global average pooling followed by upsampling and conv 1×1
5	conv3x3 rate 3	conv 3×3 with dilation rate 3
6	conv3x3 rate 12	conv 3×3 with dilation rate 12
7	sep3x3 rate 3	separable conv 3×3 with dilation rate 3
8	sep5x5 rate 6	separable conv 5×5 with dilation rate 6
9	skip	skip-connection
10	zero	zero-operation that effectively nullifies the path

Table 4 – Operation indices and abbreviations used to describe the cell configuration.

Appendix B: Experimental results

Semantic Segmentation

PASCAL VOC

We start training with the learning rates of $1e$-3 and $3e$-3 – for the encoder and the decoder, respectively. The encoder
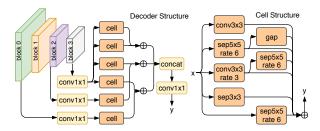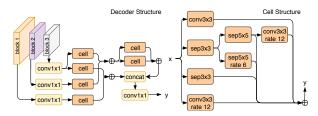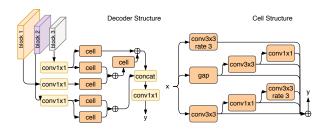
Figure 10 – arch0: $[[[3, 3], [3, 2], [3, 0]], [8, [0, 0, 5, 2], [0, 2, 8, 8], [0, 5, 1, 4]]]$

Figure 11 – arch1: $[[[2, 3], [3, 1], [4, 4]], [2, [1, 0, 3, 6], [0, 1, 2, 8], [2, 0, 6, 1]]]$

Figure 12 – arch2: $[[[1, 3], [4, 3], [2, 2]], [5, [0, 0, 4, 1], [3, 2, 0, 1], [5, 6, 5, 0]]]$

weights are updated using SGD with the momentum value of 0.9, whereas for the decoder part we rely on Adam [16] with default parameters of $\beta_1=0.9$, $\beta_2=0.99$ and $\epsilon=0.001$. We exploit the batch size of 64, evenly divided over two 1080Ti GPU cards. Each image in the batch is randomly scaled in the range of $[0.5, 2.0]$, randomly mirrored, before being randomly cropped and padded to the size of 450×450. During training, in order to calculate the loss term, we upsample the logits to the size of the target mask.

In addition to the results presented in the main text, we provide per-class intersection-over-union values across the models in Table 5.

CityScapes

We also evaluate whether the found decoder designs work well when coupled with other encoders – in particular, we consider ResNet-50 [12] and use the CityScapes dataset with 2975 training and 500 validation images. Our training strategy is as follows: we start with the learning rates of $1e-2$ and $5e-2$ and anneal them using the 'poly' sched-

ule [6] for 500 epochs. For both parts of the network we rely on SGD with the momentum value of 0.9, and train with the batch size of 20. Each image in the batch is randomly scaled in the range of $[0.5, 2.0]$, randomly mirrored, before being randomly cropped and padded to the size of 800×800. As for PASCAL VOC, we upsample the logits to the size of the target mask.

Validation results together with comparison to few other networks are given in Table 6.

Pose estimation

For pose estimation, we crop the human instance with fixed aspect ratios, 1:1 for MPII [1] and 3:4 for COCO [22]. Following Xiao *et al.* [45], the bounding box is further resized such that the longer side is equal to 256. For MPII, $\pm25\%$ scale, ±30 degree rotation and random flip are used for data augmentation. The scale and rotation factors for COCO are $\pm30\%$ and ±40 degrees, respectively. We generate keypoint heatmaps of output stride 4 with Gaussian distribution with $\sigma = 2$. The MobileNet-v2 encoder is initialised from ImageNet. We use the Adam optimiser with the base learning rate of $1e-3$, and reduce it by 10 after epochs 90 and 120. The training terminates at the epoch 140. We use the batch size of 128 evenly split between two 1080Ti GPU cards.

We provide detailed quantitative results on MPII in Table 7 and COCO in Table 8 along with a few qualitative examples on Fig. 13. The discovered architectures are able to infer correctly the location of the majority of keypoints (rows $1, 2, 4, 5$) while failing on a more difficult input image along with other models (row 3).

Depth estimation

For depth estimation, we start training with the learning rates of $1e-3$ and $7e-3$ - for the encoder and the decoder, respectively. For both we use SGD with the momentum value of 0.9, and anneal the learning rates via the *'Poly'* schedule: $lr * (1 - \frac{epoch}{400})^{0.9}$. The training is stopped after 300 epochs. We exploit the batch size of 32, evenly divided over two 1080Ti GPU cards. Each image in the batch is randomly scaled in the range of $[0.5, 2.0]$, randomly mirrored, before being randomly cropped and padded to the size of 500×500. We upsample the logits to the size of the target mask and use the inverse Huber loss [17] for optimisation, ignoring pixels with missing depth measurements.

We visualise qualitative results on the validation set in Fig. 14.

Appendix C: JetsonTX2 runtime

During our experiments we observed a significant difference between models' runtime on JetsonTX2 and 1080Ti. To better understand it, we additionally measured runtime

Model	bg	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mean
DeepLab-v3 [34]	0.94	0.873	0.416	0.849	0.647	**0.753**	0.937	0.86	**0.904**	0.391	**0.893**	0.564	**0.847**	**0.892**	0.831	0.844	0.578	0.859	0.525	0.852	0.677	0.759
RefineNet-LW [28]	0.942	**0.895**	0.594	0.872	0.761	0.669	0.912	0.85	0.876	0.383	0.801	**0.605**	0.804	0.886	0.835	0.854	**0.603**	0.843	0.479	0.834	**0.703**	0.762
Ours (arch0)	**0.947**	0.885	0.558	0.885	0.748	0.74	**0.944**	0.868	0.898	**0.429**	0.863	0.604	0.846	0.842	0.866	0.86	0.592	**0.869**	**0.593**	**0.875**	0.669	**0.780**
Ours (arch1)	0.944	0.888	**0.615**	0.866	**0.781**	0.733	0.933	0.865	0.894	0.394	0.828	0.603	0.833	0.848	0.854	0.855	0.568	0.829	0.555	0.85	0.662	0.771
Ours (arch2)	**0.947**	0.873	0.589	**0.887**	0.753	0.75	0.943	**0.885**	0.895	0.372	0.829	0.635	0.845	0.832	**0.867**	**0.866**	0.555	0.843	0.537	0.851	0.671	0.773

Table 5 – Per-class intersection-over-union on the validation set of PASCAL VOC.

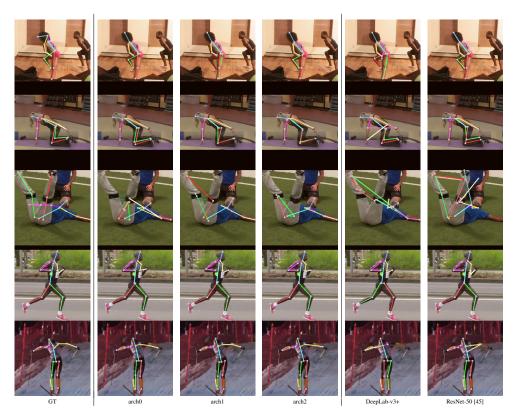GT   arch0   arch1   arch2   DeepLab-v3+   ResNet-50 [45]

Figure 13 – Inference results of our models (*arch0*, *arch1*, *arch2*) on validation set of MPII, along with that of DeepLab-v3+-MobileNet-v2 and ResNet-50 [45].

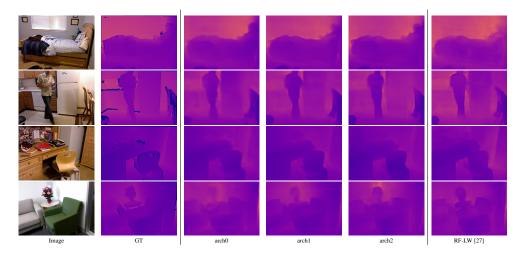Image   GT   arch0   arch1   arch2   RF-LW [27]

Figure 14 – Our depth estimation qualitative results on NYUDv2, along with that of Joint Light-Weight RefineNet [27]. Dark-blue pixels in ground truth are pixels with missing depth measurements.

Model	Backbone	Val mIoU,%,	Params,M
DeepLab-v3+[7]	Xception-65	**78.8**	41
BiSeNet [46]	ResNet-18	78.6	49
Ours (arch0)	ResNet-50	77.1	**24.5**
Ours (arch1)	ResNet-50	77.3	24.7
Ours (arch2)	ResNet-50	76.0	24.8

Table 6 – Results on the validation set of CityScapes.

of each discovered architecture together with Light-Weight RefineNet [28] varying the input resolution.

As evident from Fig. 15, the models with a larger number of floating point operations (i.e., *Arch0* and *RF-LW*) do not scale well with the input resolution. The effect is even more pronounced on JetsonTX2, as been independently confirmed by an NVIDIA employer in a private conversation.

Model	Head	Shoulder	Elbow	Wrist	Hip	Knee	Ankle	Mean	Mean@0.1
DeepLab-v3+ [7]	96.180	94.735	86.859	81.037	87.312	81.281	76.121	86.609	31.735
ResNet-50 [45]	**96.351**	**95.329**	**88.989**	**83.176**	**88.420**	**83.960**	**79.594**	**88.532**	**33.911**
Ours (arch0)	95.873	94.378	86.296	80.195	87.139	81.160	75.885	86.526	31.435
Ours (arch1)	96.317	94.548	86.501	80.932	87.242	81.583	77.374	86.971	31.951
Ours (arch2)	96.146	94.769	87.097	80.574	87.848	81.382	77.586	87.119	31.782

Table 7 – Per-keypoint pose estimation results on the validation set of MPII.

Model	AP	AP_{50}	AP_{75}	AP_m	AP_l	AR
DeepLab-v3+ [7]	0.668	**0.894**	0.740	0.641	0.707	0.700
ResNet-50 [45]	**0.704**	0.886	**0.783**	**0.671**	**0.772**	**0.763**
Ours (arch0)	0.658	**0.894**	0.730	0.631	0.701	0.691
Ours (arch1)	0.659	0.884	0.729	0.633	0.698	0.694
Ours (arch2)	0.659	0.890	0.729	0.631	0.700	0.693

Table 8 – Pose estimation results on the validation set of COCO2017. We report average precision (*AP*) and average recall (*AR*). AP_{50} and AP_{75} stand for average precision computed with the object keypoint similarity (OKS) values of 0.5 and 0.75, respectively, whereas AP_m and AP_l are average precision metrics as measured at medium and large area ranges.
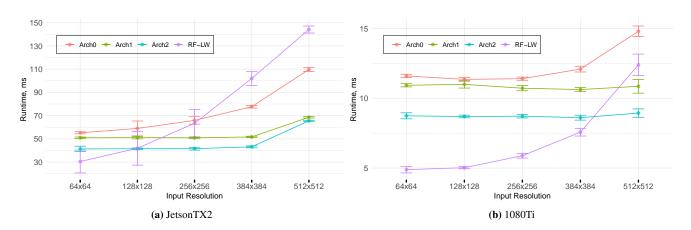

(a) JetsonTX2

(b) 1080Ti

Figure 15 – Models' runtime on JetsonTX2 (**a**) and 1080Ti (**b**). We visualise mean together with standard deviation values over 100 passes of each model.