Efficient Ladder-style DenseNets for Semantic Segmentation of Large Images

Ivan Krešo Josip Krapac Siniša Šegvić
Faculty of Electrical Engineering and Computing
University of Zagreb, Croatia
ivan.kreso@fer.hr josip.krapac@zalando.de sinisa.segvic@fer.hr

Abstract—Recent progress of deep image classification models has provided great potential to improve state-of-the-art performance in related computer vision tasks. However, the transition to semantic segmentation is hampered by strict memory limitations of contemporary GPUs. The extent of feature map caching required by convolutional backprop poses significant challenges even for moderately sized Pascal images, while requiring careful architectural considerations when the source resolution is in the megapixel range. To address these concerns, we propose a novel DenseNet-based ladder-style architecture which features high modelling power and a very lean upsampling datapath. We also propose to substantially reduce the extent of feature map caching by exploiting inherent spatial efficiency of the DenseNet feature extractor. The resulting models deliver high performance with fewer parameters than competitive approaches, and allow training at megapixel resolution on commodity hardware. The presented experimental results outperform the state-of-the-art in terms of prediction accuracy and execution speed on Cityscapes, Pascal VOC 2012, CamVid and ROB 2018 datasets. Source code will be released upon publication.

1 Introduction

Semantic segmentation is a computer vision task in which a trained model classifies pixels into meaningful high-level categories. Due to being complementary to object localization, it represents an important step towards advanced image understanding. Some of the most attractive applications include autonomous control [1], intelligent transportation systems [2], assisted photo editing [3] and medical imaging [4].

Early semantic segmentation approaches optimized a trade-off between multiple local classification cues (texture, color etc) and their global agreement across the image [5]. Later work improved these ideas with non-linear feature embeddings [6], multi-scale analysis [7] and depth [8]. Spatial consistency has been improved by promoting agreement between pixels and semantic labels [9], as well as by learning asymmetric pairwise semantic agreement potentials [10]. However, none of these approaches has been able to match the improvements due to deep convolutional models [7], [11].

Deep convolutional models have caused an unprecedented rate of computer vision development. Model depth has been steadily increasing from 8 levels [12] to 19 [13], 22 [14], 152 [15], 201 [16], and beyond [15]. Much attention has been directed towards residual models (also known as ResNets) [15], [17] in which each processing step is expressed as a sum between a compound non-linear unit and its input. This introduces an auxiliary information path which allows a direct gradient propagation across the layers, similarly to the flow of the state vector across LSTM cells. However, in contrast to the great depth of residual models, Veit et al [18] have empirically determined that most training occurs along relatively short paths. Hence, they have conjectured that a residual model acts as an exponentially large ensemble of moderately deep sub-models. This view is especially convincing in the case of residual connections with identity mappings [17].

Recent approaches [16], [19] replicate and exceed the success of residual models by introducing skip-connections across layers. This encourages feature sharing and discourages overfitting (especially when semantic classes have differing complexities), while also favouring the gradient flow towards early layers. Our work is based on densely connected models (also known as DenseNets) [16] in which the convolutional units operate on concatenations of all previous features at the current resolution. Our DenseNet-based models for semantic segmentation outperform counterparts based on ResNets [17] and more recent dual path networks [20]. Another motivation for using DenseNets is their potential for saving memory due to extensive feature reuse [16]. However, this potential is not easily materialized since straightforward backprop implementations require multiple caching of concatenated features. We show that these issues can be effectively addressed by aggressive gradient checkpointing [21] which leads to five-fold memory reduction with only 20% increase in training time.

Regardless of the particular architecture, deep convolutional models for semantic segmentation must decrease the spatial resolution of deep layers in order to meet strict GPU memory limitations. Subsequently, the deep features have to be carefully upsampled to the image resolution in order to generate correct predictions at semantic borders and small objects. Some approaches deal with this issue by decreasing the extent of subsampling with dilated filtering [22], [23], [24], [25], [26]. Other approaches gradually upsample deep convolutional features by exploiting cached max-pool switches [27], [28] or activations from earlier layers [4], [29], [30], [31], [32]. Our approach is related to the latter group as we also blend the semantics of the deep features with

the location accuracy of the early layers. However, previous approaches feature complex upsampling datapaths which require a lot of computational resources. We show that powerful models can be achieved even with minimalistic upsampling, and that such models are very well suited for fast processing of large images.

In this paper, we present an effective lightweight architecture for semantic segmentation of large images, based on DenseNet features and ladder-style [33] upsampling. We propose several improvements with respect to our previous work [34], which lead to better accuracy and faster execution while using less memory and fewer parameters. Our consolidated contribution is three-fold. First, we present an exhaustive study of using densely connected [16] feature extractors for efficient semantic segmentation. Second, we propose a lean ladder-style upsampling datapath [33] which requires less memory and achieves a better $\overline{\text{IoU}}$/FLOP tradeoff than previous approaches. Third, we further reduce the training memory footprint by aggressive re-computation of intermediate activations during convolutional backprop [21]. The proposed approach strikes an excellent balance between prediction accuracy and model complexity. Experiments on Cityscapes, CamVid, ROB 2018 and Pascal VOC 2012 demonstrate state-of-the-art recognition performance and execution speed with modest training requirements.

2 Related Work

Early convolutional models for semantic segmentation had only a few pooling layers and were trained from scratch [7]. Later work built on image classification models pre-trained on ImageNet [13], [15], [16], which typically perform 5 downsamplings before aggregation. The resulting loss of spatial resolution requires special techniques for upsampling the features back to the resolution of the source image. Early upsampling approaches were based on trained filters [35] and cached switches from strided max-pooling layers [27], [28]. More recent approaches force some strided layers to produce non-strided output while doubling the dilation factor of all subsequent convolutions [23], [36], [37]. This decreases the extent of subsampling while ensuring that the receptive field of the involved features remains the same as in pre-training.

In principle, dilated filtering can completely recover the resolution without compromising pre-trained parameters. However, there are two important shortcomings due to which this technique should be used sparingly [38] or completely avoided [34], [39]. First, dilated filtering substantially increases computational and GPU memory requirements, and thus precludes real-time inference and hinders training on single GPU systems. Practical implementations alleviate this by recovering only up to the last two subsamplings, which allows subsequent inference at $8\times$ subsampled resolution [22], [23]. Second, dilated filtering treats semantic segmentation exactly as if it were ImageNet classification, although the two tasks differ with respect to location sensitivity. Predictions of a semantic segmentation model must change abruptly in pixels at semantic borders. On the other hand, image classification predictions need to be largely insensitive to the location of the object which defines the class. This suggests that optimal semantic segmentation performance might not be attainable with architectures designed for ImageNet classification.

We therefore prefer to keep the downsampling and restore the resolution by blending semantics of deep features with location accuracy of the earlier layers [40]. This encourages the deep layers to discard location information and focus on abstract image properties [33]. Practical realizations avoid high-dimensional features at output resolution [40] by ladder-style upsampling [4], [33], [39]. In symmetric encoder-decoder approaches, [4], [27], [29] the upsampling datapath mirrors the structure of the downsampling datapath. These methods achieve rather low execution speed due to excessive capacity in the upsampling datapath. Ghiasi et al [30] blend predictions (instead of blending features) by prefering the deeper layer in the middle of the object, while favouring the earlier layer near the object boundary. Pohlen et al [41] propose a two-stream residual architecture where one stream is always at the full resolution, while the other stream is first subsampled and subsequently upsampled by blending with the first stream. Lin et al [31] perform the blending by a sub-model called RefineNet comprised of 8 convolutional and several other layers in each upsampling step. Islam et al [42] blend upsampled predictions with two layers from the downsampling datapath. This results in 4 convolutions and one elementwise multiplication in each upsampling step. Peng et al [32] blend predictions produced by convolutions with very large kernels. The blending is performed by one 3×3 deconvolution, two 3×3 convolutions, and one addition in each upsampling step. In this work, we argue for a minimalistic upsampling path consisting of only one 3×3 convolution in each upsampling step [34]. In comparison with symmetric upsampling [4], [27], this substantially reduces the number of 3×3 convolutions in the upsampling datapath. For instance, a VGG 16 feature extractor requires 13 3×3 convolutions in the symmetric case [27], but only 4 3×3 convolutions in the proposed setup. To the best of our knowledge, such solution has not been previously used for semantic segmentation, although there were uses in object detection [39], and instance-level segmentation [43]. Additionally, our lateral connections differ from [30], [32], [42], since they blend predictions, while we blend features. Blending features improves the modelling power, but is also more computationally demanding. We can afford to blend features due to minimalistic upsampling and gradient checkpointing which will be explained later. Similarly to [22], [30], [42], we generate semantic maps at all resolutions and optimize cross entropy loss with respect to all of them.

Coarse spatial resolution is not the only shortcoming of features designed for ImageNet classification. These features may also have insufficient receptive field to support the correct semantic prediction. This issue shows up in pixels situated at smooth image regions which are quite common in urban scenes. Many of these pixels are projected from objects close to the camera, which makes them extremely important for high-level tasks (circumstances of the first fatal incident of a level-2 autopilot are a sad reminder of this observation). Some approaches address this issue by applying 3×3 convolutions with very large dilation factors [23], [37], [38]. However, sparse sampling may trigger undesired effects due to aliasing. The receptive field can

also be enlarged by extending the depth of the model [42]. However, the added capacity may result in overfitting. Correlation between distant parts of the scene can be directly modelled by introducing long-range connections [9], [10], [44]. However, these are often unsuitable due to large capacity and computational complexity. A better ratio between receptive range and complexity is achieved with spatial pyramid pooling (SPP) [45], [46] which augments the features with their spatial pools over rectangular regions of varying size [22]. Our design proposes slight improvements over [22] as detailed in 4.2. Furthermore, we alleviate the inability of SPP to model spatial layout by inserting a strided pooling layer in the middle of the third convolutional block. This increases the receptive range and retains spatial layout without increasing the model capacity.

To the best of our knowledge, there are only two published works on DenseNet-based semantic segmentation [26], [29]. However, these approaches fail to position DenseNet as the backbone with the largest potential for memory-efficient feature extraction[1]. This potential is caused by a specific design which encourages inter-layer sharing [16] instead of forwarding features across the layers. Unfortunately, automatic differentiation is unable to exploit this potential due to concatenation, batchnorm and projection layers. Consequently, straightforward DenseNet implementations actually require a little bit more memory than their residual counterparts [34]. Fortunately, this issue can be alleviated with checkpointing [21], [49]. Previous work on checkpointing segmentation models considered only residual models [25], and therefore achieved only two-fold memory reduction. We show that DenseNet has much more to gain from this technique by achieving up to six-fold memory reduction with respect to the baseline.

3 Comparison between ResNets and DenseNets

Most convolutional architectures [13], [14], [16], [17] are organized as a succession of processing blocks which process image representation on the same subsampling level. Processing blocks are organized in terms of convolutional units [17] which group several convolutions operating as a whole.

We illustrate similarities and differences between ResNet and DenseNet architectures by comparing the respective processing blocks, as shown in Figure 1. We consider the most widely used variants: DenseNet-BC (bottleneck - compression) [16] and pre-activation ResNets with bottleneck [17].

3.1 Organization of the processing blocks

The main trait of a residual model is that the output of each convolutional unit is summed with its input (cf. Figure 1(a)). Hence, all units of a residual block have a constant output dimensionality F_{out}. Typically, the first ResNet unit increases the number of feature maps and decreases the resolution by strided projection. On the other hand, DenseNet units operate on a concatenation of the main input with all preceding units in the current block (cf. Figure 1(b)). Thus, the dimensionality of a DenseNet block increases after each convolutional unit. The number of feature maps produced by each DenseNet unit is called the growth rate and is defined by the hyper-parameter k. Most popular DenseNet variations have k=32, however we also consider DenseNet-161 with k=48.

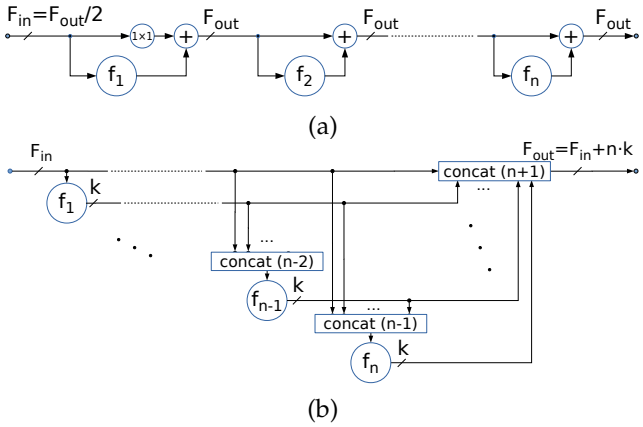

Fig. 1. A pre-activation residual block [17] with n units (a) and the corresponding densely connected block [16] (b). Labeled circles correspond to convolutional units (f_1-f_k), summations (+) and n-way concatenations - concat (n). All connections are 3D tensors D×H×W, where D is designated above the connection line (for simplicity, we assume the batch size is 1). F_{in} and F_{out} denote numbers of feature maps on the processing block input and output, respectively.

In order to reduce the computational complexity, both ResNet and DenseNet units reduce the number of feature maps before 3×3 convolutions. ResNets reduce the dimensionality to $F_{\text{out}}/4$, while DenseNets go to $4k$. DenseNet units have two convolutions (1×1, 3×3), while ResNet units require three convolutions (1×1, 3×3, 1×1) in order to restore the dimensionality of the residual datapath. The shapes of ResNet convolutions are: $1{\times}1{\times}F_{\text{out}} \times F_{\text{out}}/4$, $3{\times}3{\times}F_{\text{out}}/4 \times F_{\text{out}}/4$, and $1{\times}1{\times}F_{\text{out}}/4 \times F_{\text{out}}$. The convolutions in i-th DenseNet unit are $1{\times}1{\times}[F_{\text{in}}$+(i-1)·k]×4k, and 3×3×4k×k. All DenseNet and pre-activation ResNet units [17] apply batchnorm and ReLU activation before convolution (BN-ReLU-conv). On the other hand, the original ResNet design [15] applies convolution at the onset (conv-BN-ReLU), which precludes negative values along the skip connections.

3.2 Time complexity

Both architectures encourage exposure of early layers to the loss signal. However, the distribution of the representation dimensionality differs: ResNet keeps it constant throughout the block, while DenseNet increases it towards the end. DenseNet units have the following advantages: i) producing fewer feature maps (k vs F_{out}), ii) lower average input dimensionality, iii) fewer convolutions per unit: 2 vs 3. Asymptotic per-pixel time complexities of DenseNet and ResNet blocks are respectively: $O(k^2n^2)$=$O(F_{\text{out}}^2)$ and $O(nF_{\text{out}}^2)$. Popular ResNets alleviate asymptotic disadvantage by having only n=3 units in the first processing block which is computationally the most expensive. Hence, the

1. Here we do not consider reversible models [47] due to poor availability of ImageNet-pretrained parameters, and large computational complexity due to increased dimensionality of the deep layers [48].

temporal complexity of DenseNet models is only moderately lower than comparable ResNet models, although the gap increases with capacity [16].

We illustrate this by comparing ResNet-50 ($26{\cdot}10^6$ parameters) and DenseNet-121 ($8{\cdot}10^6$ parameters). Theoretically, ResNet-50 requires around 33% more floating operations than DenseNet-121 [16]. In practice the two models achieve roughly the same speed on GPU hardware due to popular software frameworks being unable to implement concatenations without costly copying across GPU memory. Note however that we have not considered improvements based on learned grouped convolutions [50] and early-exit classifiers [51].

3.3 Extent of caching required by backprop

Although DenseNet blocks do not achieve decisive speed improvement for small models, they do promise substantial gains with respect to training flexibility. Recall that backprop requires caching of input tensors in order to be able to calculate gradients with respect to convolution weights. This caching may easily overwhelm the GPU memory, especially in the case of dense prediction in large images. The ResNet design implies a large number of feature maps at input of each convolutional unit (cf. Figure 1). The first convolutional unit incurs a double cost despite receiving half feature maps due to 2^2 times larger spatial dimensions. This results in c_{RN} per-pixel activations which is clearly $O(n \cdot F_{\mathrm{out}})$:

$$c_{\mathrm{RN}} = (n+1) \cdot F_{\mathrm{out}} \ . \tag{1}$$

The DenseNet design alleviates this since many convolutional inputs are shared. An optimized implementation would need to cache only c_{DN} per-pixel activations:

$$c_{\mathrm{DN}} = F_{\mathrm{in}} + (n-1) \cdot k \approx F_{\mathrm{out}} \ . \tag{2}$$

Equations (1) and (2) suggest that a DenseNet block has a potential for n-fold reduction of the backprop memory footprint with respect to a ResNet block with the same F_{out}. We note that ResNet-50 and DenseNet-121 have equal F_{out} in the first three processing blocks, while ResNet-50 is twice as large in the fourth processing block.

Unfortunately, exploiting the described potential is not straightforward, since existing autograd implementations perform duplicate caching of activations which pass through multiple concatenations. Hence, a straightforward implementation has to cache the output of each unit in all subsequent units at least two times: i) as a part of the input to the first batchnorm ii) as a part of the input to the first convolution (1×1). Due to this, memory requirements of unoptimized DenseNet implementations grow as $O(kn^2)$ [21] instead of $O(kn) = O(F_{\mathrm{out}})$ as suggested by (2). Fortunately, this situation can be substantially improved with gradient checkpointing. The idea is to instruct autograd to cache only the outputs of convolutional units as suggested by (2), and to recompute the rest during the backprop. This can also be viewed as a custom backprop step for DenseNet convolutional units. We postpone the details for Section 5.

3.4 Number of parameters

It is easy to see that the number of parameters within a processing block corresponds to the number of per-pixel multiplications considered in 3.2: each parameter is multiplied exactly once for each pixel. Hence, the number of DenseNet parameters is $O(F_{\mathrm{out}}^2)$, while the number of ResNet parameters is $O(nF_{\mathrm{out}}^2)$. However, the correspondence between time complexity and the number of parameters does not translate to the model level since time complexity is linear in the number of pixels. Hence, per-pixel multiplications make a greater contribution to time-complexity in early blocks than in later blocks. On the other hand, the corresponding contribution to the number of parameters is constant across blocks. Therefore, the largest influence to the model capacity comes from the later blocks due to more convolutional units and larger output dimensionalities.

Table 1 compares the counts of convolutional weights in ResNet-50 and DenseNet-121. We see that DenseNet-121 has twice as much parameters in block 1, while the relation is opposite in block 3. The last residual block has more capacity than the whole DenseNet-121. In the end, DenseNet-121 has three times less parameters than ResNet-50.

TABLE 1
Count of convolutional weights across blocks (in millions). ResNet-50 has n=$[3, 4, 6, 3]$ and F_{out}=$[256, 512, 1024, 2048]$ while DenseNet-121 has n=$[6, 12, 24, 16]$ and F_{out}=$[256, 512, 1024, 1024]$.

block@subsampling	B1@/4	B2@/8	B3@/16	B4@/32
Resnet-50	0.2	1.2	7.1	14.9
DenseNet-121	0.4	1.0	3.3	2.1

3.5 DenseNets as regularized ResNets

We will show that each DenseNet block D can be realized with a suitably engineered ResNet block R such that each R_i produces $F_{\mathrm{out}} = F_{\mathrm{in}} + nk$ maps. Assume that R_i is engineered so that the maps at indices $F_{\mathrm{in}} + ik$ through $F_{\mathrm{in}} + (i+1)k$ are determined by non-linear mapping r_i, while all remaining maps are set to zero. Then, the effect of residual connections becomes very similar to the concatenation within the DenseNet block. Each non-linear mapping r_i observes the same input as the corresponding DenseNet unit D_i: there are F_{in} maps corresponding to the block input, and $(i-1)k$ maps produced by previous units. Hence, r_i can be implemented by re-using weights from D_i, which makes R equivalent to D.

We see that the space of DenseNets can be viewed as a sub-space of ResNet models in which the feature reuse is heavily enforced. A ResNet block capable to learn a given DenseNet block requires an n-fold increase in time complexity since each R_i has to produce $O(kn)$ instead of $O(k)$ feature maps. The ResNet block would also require an n-fold increase of backprop caching and an n-fold increase in capacity following the arguments in 3.3 and 3.4. We conclude that DenseNets can be viewed as strongly regularized ResNets. Hence, DenseNet models may generalize better when the training data is scarce.

4 The Proposed Architecture

We propose a light-weight semantic segmentation architecture featuring high accuracy, low memory footprint and high execution speed. The architecture consists of two datapaths which are designated by two horizontal rails in

Figure 2. The downsampling datapath is composed of a modified DenseNet feature extractor [16], and a lightweight spatial pyramid pooling module (SPP) [22]. The feature extractor transforms the input image into the feature tensor **F** by gradually reducing the spatial resolution and increasing the number of feature maps (top rail in Figure 2). The SPP module enriches the DenseNet features with context information and creates the context-aware features **C**. The upsampling datapath transforms the low-resolution features **C** to high-resolution semantic predictions (bottom rail in Figure 2). This transformation is achieved by efficient blending of semantics from deeper layers with fine details from early layers.

4.1 Feature extraction

The DenseNet feature extractor [16] consists of dense blocks (DB) and transition layers (TD) (cf. Figure 2). Each dense block is a concatenation of convolutional units, while each convolutional unit operates on a concatenation of all preceding units and the block input, as detailed in Section 3. Different from the original DenseNet design, we split the dense block DB3 into two fragments (DB3a and DB3b) and place a strided average-pooling layer (D) in-between them. This enlarges the receptive field of all convolutions after DB3a, while decreasing their computational complexity. In comparison with dilated filtering [23], this approach trades-off spatial resolution (which we later restore with ladder-style blending) for improved execution speed and reduced memory footprint. We initialize the DB3b filters with ImageNet-pretrained weights of the original DenseNet model, although the novel pooling layer alters the features in a way that has not been seen during ImageNet pretraining. Despite this discrepancy, fine-tuning succeeds to recover and achieve competitive generalization. The feature extractor concludes by concatenating all DB4 units into the 64× subsampled representation **F**.

4.2 Spatial pyramid pooling

The spatial pyramid pooling module (SPP) captures wide context information [22], [45], [46] by augmenting **F** with average pools over several spatial grids. Our SPP module first projects **F** to D/2 maps, where D denotes the dimensionality of DenseNet features. The resulting tensor is then average-pooled over four grids with 1, 2, 4, and 8 rows. The number of grid columns is set in accordance with the image size so that all cells have a square shape. We project each pooled tensor to D/8 maps and then upsample with bilinear upsampling. We concatenate all results with the projected **F**, and finally blend with a 1×1×D/4 convolution. The shape of the resulting context-aware feature tensor **C** is H/64×W/64×D/4. The dimensionality of **C** is 48 times less than the dimensionality of the input image (we assume DenseNet-121, D=1024).

There are two differences between our SPP module and the one proposed in [22]. First, we adapt the grid to the aspect ratio of input features: each grid cell always averages a square area, regardless of the shape of the input image. Second, we reduce the dimensionality of input features before the pooling in order to avoid increasing the output dimensionality.

4.3 Upsampling datapath

The role of the upsampling path is to recover fine details lost due to the downsampling. The proposed design is based on minimalistic transition-up (TU) blocks. The goal of TU blocks is to blend two representations whose spatial resolutions differ by a factor of 2. The smaller representation comes from the upsampling datapath while the larger representation comes from the downsampling datapath via skip connection. We first upsample the smaller representation with bilinear interpolation so that the two representations have the same resolution. Subsequently, we project the larger representation to a lower-dimensional space so that the two representations have the same number of feature maps. This balances the relative influence of the two datapaths and allows to blend the two representations by simple summation. Subsequently, we apply a 1×1 convolution to reduce the dimensionality (if needed), and conclude with 3×3 convolution to prepare the feature tensor for subsequent blending. The blending procedure is

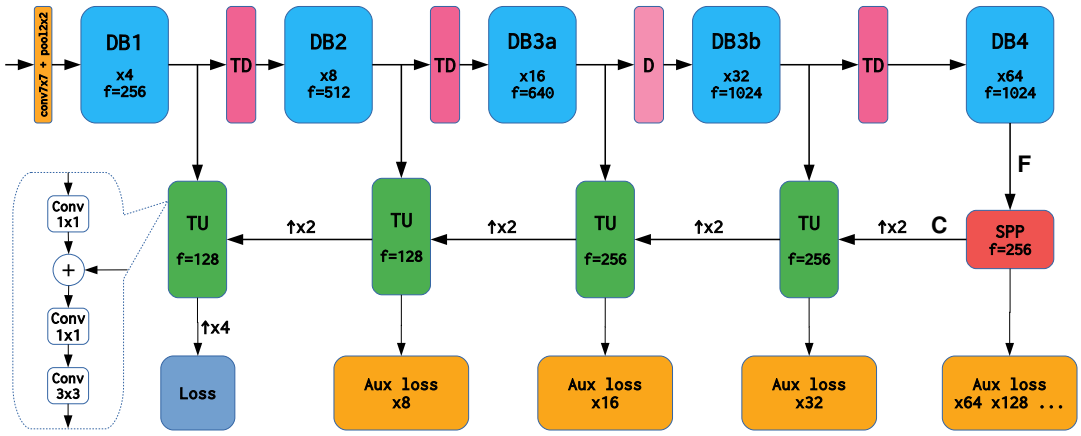

Fig. 2. Architecture of the proposed segmentation model with the DenseNet-121 downsampling datapath. Each dense block (DBx) is annotated with the subsampling factor of the input tensor. The number of output feature maps is denoted with f. The transition-up (TU) blocks blend low-resolution high-level features with high-resolution low-level features.

recursively repeated along the upsampling datapath with skip-connections arriving from outputs of each dense block. The final transition-up block produces logits at the resolution of the DenseNet stem. The dense predictions on input resolution are finally obtained by $4\times$ bilinear upsampling.

The presented minimalistic design ensures fast execution due to only one 3×3 convolution per upsampling step, and low memory footprint due to few convolutions, and low dimensionality of feature tensors. The memory footprint can be further reduced as detailed in Section 5. The proposed upsampling datapath has much fewer parameters than the downsampling datapath and therefore discourages overfitting to low-level texture as we illustrate in the experiments.

5 Gradient checkpointing

Semantic segmentation requires extraordinary amounts of memory during training, especially on large input resolutions. These requirements may lead to difficulties due to strict limitations of GPU RAM. For example, it is well known that small training batches may lead to unstable batchnorm statistics and poor learning performance. This problem can not be overcome by accumulating backward passes between updates, and therefore represents a serious obstacle towards achieving competitive performance.

The extent of backprop-related caching can be reduced with gradient checkpointing [49]. The main idea is to instruct forward pass to cache only a carefully selected subset of all activations. These activations are subsequently used for re-computing non-cached activations during backprop. We refer to explicitly cached nodes of the computational graph as gradient checkpoints. The subgraph between two gradient checkpoints is called a checkpointing segment. The backward pass iterates over all checkpointing segments and processes them as follows. First, forward pass activations are recomputed starting from the stored checkpoint. Second, the gradients are computed via the standard backward pass. The local cache is released as soon as the corresponding segment is processed i.e. before continuing to the next segment.

We note that segment granularity affects space and time efficiency. Enlarging the checkpoint segments always reduces the memory footprint of the forward pass. However, the influence to backward pass memory requirements is non-trivial. Larger segments require more memory individually as they need to re-compute all required activations and store them in the local cache. At some point, we start to lose the gains obtained during forward pass. Our best heuristic was to checkpoint only outputs from 3×3 convolutions as they are the most compute-heavy operations. In other words, we propose to re-compute the stem, all projections, all batchnorms and all concatenations during the backward pass. Experiments show that this approach strikes a very good balance between maximum memory allocation in forward and backward passes.

The proposed checkpointing strategy is related to the previous approach [21] which puts a lot of effort into explicit management of shared storage. However, here we show that similar results can be obtained by relying on the standard PyTorch memory manager. We also show that custom backprop operations can be completely avoided by leveraging the standard PyTorch module `torch.utils.checkpoint`. Finally, we propose to achieve further memory gains by caching only outputs of 3×3 convolutions and the input image. We achieve that by checkpointing the stem, transition-down and transition-up blocks, as well as DenseNet units as a whole. To the best of our knowledge, this is the first account of applying aggressive checkpointing for semantic segmentation.

6 Experiments

Most of our experiments target road-driving images, since the corresponding applications require a very large image resolution (subsections 6.2 and 6.3). Cross-dataset generalization experiments are presented in 6.4. Subsection 6.5 addresses Pascal VOC 2012 [52] as the most popular image segmentation dataset. We show ablation experiments in 6.6 and finally explore advantages of checkpointing in 6.7.

6.1 Training details and notation

We train our models using AMSGrad [53], [54] with the initial learning rate $4 \cdot 10^{-4}$. The learning rate is decreased after each epoch according to cosine learning rate policy. We divide the learning rate by 4 for all pre-trained weights. Batch size is an important hyper-parameter of the optimization procedure. If we train with batch size 1, then the batchnorm statistics fit exactly the image we are training on. This hinders learning due to large covariate shift across different images [14]. We combat this by training on random crops with batch size 16. Before cropping, we apply a random flip and rescale the image with a random factor between 0.5 and 2. The crop size is set to 448, 512 or 768 depending on the resolution of the dataset. If a crop happens to be larger than the rescaled image, then the undefined pixels are filled with the mean pixel. We train for 300 epochs unless otherwise stated. We employ multiple cross entropy losses along the upsampling path as shown in Figure 2. Auxiliary losses use soft targets determined as the label distribution in the corresponding N$\times$N window where N denotes the downsampling factor. We apply loss after each upsampling step, and to each of the four pooled tensors within the SPP module. The loss of the final predictions is weighted by 0.6, while the mean auxiliary loss contributes with factor 0.4. After the training, we recompute the batchnorm statistics as exact averages over the training set instead of decayed moving averages used during training. This practice slightly improves the model generalization.

We employ the following notation to describe our experiments throughout the section. LDN stands for Ladder DenseNet, the architecture proposed in Section 4. The symbol $d{\rightarrow}u$ denotes a downsampling path which reduces the image resolution d times, and a ladder-style upsampling path which produces predictions subsampled u times with respect to the input resolution. For example, LDN121 $64{\rightarrow}4$ denotes the model shown in Figure 2. Similarly, DDN and LDDN denote a dilated DenseNet, and a dilated DenseNet with ladder-style upsampling. The symbol $d\downarrow$ denotes a model which reduces the image resolution d times and has no upsampling path. MS denotes multi-scale evaluation on

5 scales (0.5, 0.75, 1, 1.5 and 2), and respective horizontal flips. $\overline{\text{IoU}}$ and Cat. $\overline{\text{IoU}}$ denote the standard mean IoU metric over classes and categories. The instance-level mean IoU ($\overline{\text{iIoU}}$) metric [55] emphasizes contribution of pixels at small instances, and is therefore evaluated only on 8 object classes. The model size is expressed as the total number of parameters in millions (M). FLOP denotes the number of fused multiply-add operations required for inference on a single 1024×1024 (1MPx) image.

6.2 Cityscapes

The Cityscapes dataset contains road driving images recorded in 50 cities during spring, summer and autumn. The dataset features 19 classes, good and medium weather, large number of dynamic objects, varying scene layout and varying background. We perform experiments on 5000 finely annotated images divided into 2975 training, 500 validation, and 1525 test images. The resolution of all images is 1024×2048.

Table 2 validates several popular backbones coupled with the same SPP and upsampling modules. Due to hardware constraints, here we train and evaluate on half resolution images, and use 448×448 crops. The first section of the table presents the DN121 32↓ baseline. The second section presents our models with ladder-style upsampling. The LDN121 64→4 model outperforms the baseline for 10 percentage points (pp) of $\overline{\text{IoU}}$ improvement. Some improvements occur on small instances as a result of finer output resolution due to blending with low-level features. Other improvements occur on large instances since increased subsampling (64 vs 32) enlarges the spatial context. Note that LDN121 32→4 slightly outperforms LDN121 64→4 at this resolution due to better accuracy at semantic borders. However, the situation will be opposite in full resolution images due to larger objects (which require a larger receptive field) and off-by-one pixel annotation errors. The LDN169 32→4 model features a stronger backbone, but obtains a slight deterioration (0.8pp) with respect to LDN121 32→4. We conclude that half resolution images do not contain enough training pixels to support the capacity of DenseNet-169. The third section demonstrates that residual and DPN backbones achieve worse generalization than their DenseNet counterparts. The bottom section shows that further upsampling (LDN121 32→2) doubles the computational complexity while bringing only a slight accuracy improvement.

Table 3 addresses models which recover the resolution loss with dilated convolutions. The DDN-121 8↓ model removes the strided pooling layers before the DenseNet blocks DB3 and DB4, and introduces dilation in DB3 (rate=2) and DB4 (rate=4). The SPP output is now 8× downsampled. From there we produce logits and finally restore the input resolution with bilinear upsampling. The LDDN-121 8→4 model continues with one step of ladder-style upsampling to obtain 4× downsampled predictions as in previous LDN experiments. We observe a 3pp $\overline{\text{IoU}}$ improvement due to ladder-style upsampling. The LDDN-121 16→4 model dilates only the last dense block and performs two steps of ladder-style upsampling. We observe a marginal improvement which, however, still comes short of LDN121 32→4 from Table 2. Training the DDN-121 4↓ model was infeasible due to huge computational requirements when the last three blocks operate on 4× subsampled resolution. A comparison of computational complexity reveals that the dilated LDDN-121 8→4 model has almost 3× more FLOPs than LDN models with similar $\overline{\text{IoU}}$ performance. Finally, our memory consumption measurements show that LDDN-121 8→4 consumes around 2× more GPU memory than LDN121 32→4. We conclude that dilated models achieve a worse generalization than their LDN counterparts while requiring more computational power.

TABLE 2
Validation of backbone architectures on Cityscapes val. Both training and evaluation images were resized to 1024×512.

Method	Class $\overline{\text{IoU}}$	Class $\overline{\text{iIoU}}$	Cat. $\overline{\text{IoU}}$	Model size	FLOP 1MPx
DN121 32↓	66.2	46.7	78.3	8.2M	56.1G
LDN121 64→4	75.3	54.8	88.1	9.5M	66.5G
LDN121 32→4	76.6	57.5	88.6	9.0M	75.4G
LDN169 32→4	75.8	55.5	88.4	15.6M	88.8G
ResNet18 32→4	70.9	49.7	86.7	13.3M	55.7G
ResNet101 32→4	73.7	54.3	87.8	45.9M	186.7G
ResNet50 32→4	73.9	54.2	87.8	26.9M	109.0G
DPN68 32→4	74.0	53.0	87.8	13.7M	59.0G
LDN121 32→2	**77.5**	**58.9**	89.3	9.4M	154.5G

TABLE 3
Validation of dilated models on Cityscapes val. Both training and evaluation images were resized to 1024×512.

Method	Class $\overline{\text{IoU}}$	Class $\overline{\text{iIoU}}$	Cat. $\overline{\text{IoU}}$	Model size	FLOP 1MPx
DDN-121 8↓	72.5	52.5	85.5	8.2M	147.8B
LDDN-121 8→4	75.5	55.3	88.3	8.6M	174.8B
LDDN-121 16→4	75.8	55.9	88.4	8.9M	87.0B

Table 4 shows experiments on full Cityscapes val images where we train on 768×768 crops. We obtain the most interesting results with the LDN121 64→4 model presented in Figure 2: 79% $\overline{\text{IoU}}$ with a single forward pass and 80.3% with multi-scale (MS) inference. Models with stronger backbones (DenseNet-169, DenseNet-161) validate only slightly better. We explain that by insufficient training data and we expect that successful models need less capacity for Cityscapes than for a harder task of discriminating ImageNet classes.

TABLE 4
Validation of various design options on full-resolution Cityscapes val.

Method	Class $\overline{\text{IoU}}$	Cat. $\overline{\text{IoU}}$	Model size	FLOP 1MPx
LDN121 32→4	78.4	90.0	9.0M	75.4G
LDN121 64→4	79.0	90.3	9.5M	66.5G
LDN121 128→4	78.4	90.1	9.9M	66.2G
LDN161 64→4	79.1	90.2	30.0M	138.7G
LDN121 64→4 MS	80.3	90.6	9.5M	536.2G

Table 5 compares the results of two of our best models with the state-of-the-art on validation and test sets. All mod-

els from the table have been trained only on finely annotated images. The label DWS denotes depthwise separable convolutions in the upsampling path (cf. Table 10). Our models generalize better than or equal to all previous approaches, while being much more efficient. In particular, we are the first to achieve 80% IoU on Cityscapes test fine with only 66.5 GFLOP per MPx. Figure 3 plots the best performing models from Table 5 in (IoU,TFLOP) coordinates. The figure

TABLE 5
Comparison of our two best models with the state-of-the-art on Cityscapes val and test. All models have been trained only on finely annotated images. For models marked with '†' we estimate a lower FLOP-bound by measuring the complexity of the backbone. Most of the IoU results use multi-scale inference while we always show required FLOPs for single-scale inference.

Method	Backbone	Val IoU	Test IoU	FLOP 1MPx
ERFNet [56]	Custom 8×	71.5	69.7	55.4G
SwiftNet [57]	RN18 32×	75.4	75.5	52.0G
LinkNet [58]	RN18 32×	76.4	n/a	201G
LKM [32]	RN50 32×	77.4	76.9	106G†
TuSimple [59]	RN101 D8×	76.4	77.6	722G†
SAC-multiple [60]	RN101 D8×	78.7	78.1	722G†
WideResNet38 [61]	WRN38 D8×	77.9	78.4	2106G†
PSPNet [22]	RN101 D8×	n/a	78.4	722G†
Multi Task [62]	RN101 D8×	n/a	78.5	722G†
TKCN [63]	RN101 D8×	n/a	79.5	722G†
DFN [64]	RN101 32×	n/a	79.3	445G†
Mapillary [25]	WRN38 D8×	78.3	n/a	2106G†
DeepLab v3 [24]	RN101 D8×	79.3	n/a	722G†
DeepLab v3+ [38]	X-65 D8×	79.1	n/a	708G
DeepLab v3+ [38]	X-71 D8×	79.5	n/a	n/a
DRN [65]	WRN38 D8×	79.7	79.9	2106G†
DenseASPP [26]	DN161 D8×	78.9	**80.6**	498G†
LDN121 DWS	DN121 64×	80.2	n/a	**54.2**G
LDN121 64→4	DN121 64×	80.3	80.0	**66.5**G
LDN161 64→4	DN161 64×	**80.7**	**80.6**	139G

clearly shows that our models achieve the best trade-off between accuracy and computational complexity.

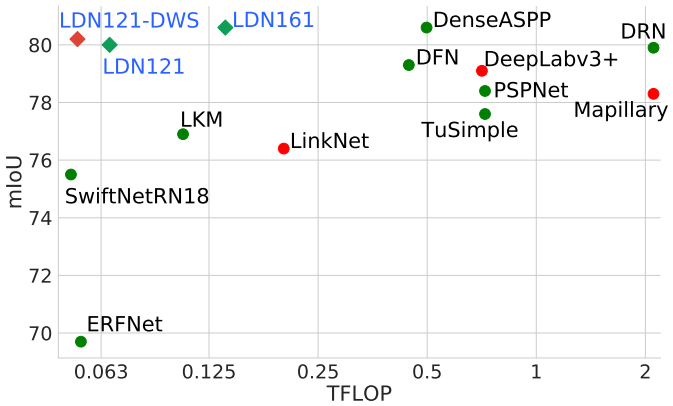

Fig. 3. Accuracy vs forward pass complexity on Cityscapes test (green) and val (red) for approaches from Table 5. All models have been trained only on finely annotated images. LDN121 is the first method to achieve 80% IoU while being applicable in real-time.

6.3 CamVid

The CamVid dataset contains images of urban road driving scenes. We use the 11-class split from [27], which consists of 367 training, 101 validation, and 233 testing images. The resolution of all images is 720×960. Following the common practice, we incorporate the val subset into train because it is too small and too easy to be useful for validation. We train all models from random initialization (RI), and by fine-tuning the parameters pre-trained on ImageNet (PT). We train on 512×512 crops for 400 epochs with pre-training, and 800 epochs with random init. All other hyperparameters are the same as in Cityscapes experiments.

Table 6 shows our results on full-resolution CamVid test. The conclusions are similar as on half-resolution Cityscapes val (cf. Table 2), which does not surprise us due to similar image resolutions. LDN121 32→4 wins both in the pre-trained and in the random init case, with LDN121 64→4 being the runner-up. Table 7 compares our best results with

TABLE 6
Single-scale inference on full-resolution CamVid test with ImageNet pre-training (PT) and random initialization (RI).

Method	PT IoU	RI IoU	Model size	FLOP 1MPx
LDN121 32→4	**77.3**	**70.9**	9.0M	75.4G
LDN121 64→4	76.9	68.7	9.5M	66.5G
ResNet18 32→4	73.2	70.0	13.3M	55.7G
ResNet50 32→4	76.1	69.9	26.9M	109.0G
ResNet101 32→4	76.7	69.4	45.9M	186.7G

the related work on CamVid test where, to the best of our knowledge, we obtain state-of-the-art results.

TABLE 7
Comparison of our models with the state-of-the-art on CamVid test. We use multi-scale inference in experiments on full resolution.

Method	Backbone	ImgNet	Resolution	IoU
Tiramisu [29]	DenseNet		half	66.9
FC-DRN [66]	DenseResNet		half	69.4
G-FRNet [67]	VGG-16	✓	half	68.8
BiSeNet [68]	Xception39	✓	full	65.6
ICNet [69]	ResNet-50	✓	full	67.1
BiSeNet [68]	ResNet-18	✓	full	68.7
LDN121 16→2	DenseNet		half	**69.5**
LDN121 32→4	DenseNet		full	**71.9**
LDN121 16→2	DenseNet	✓	half	**75.8**
LDN121 32→4	DenseNet	✓	full	**78.1**

6.4 Cross-dataset generalization

We explore the capability of our models to generalize across related datasets. We train on Cityscapes and Vistas, and evaluate on Cityscapes, Vistas and KITTI. Mapillary Vistas [70] is a large road driving dataset featuring five continents and diverse lighting, seasonal and weather conditions. It contains 18000 training, 2000 validation, and 5000 test images. In our experiments, we remap annotations to 19 Cityscapes classes, and resize all images to width 2048.

The KITTI dataset contains road driving images recorded in Karlsruhe [71]. It features the Cityscapes labeling convention and depth reconstruction groundtruth. There are 200 training and 200 test images. All images are 370×1226.

Table 8 shows that training only on Cityscapes results in poor generalization due to urban bias and constant acquisition setup. On the other hand, Vistas is much more diverse. Hence, the model trained on Vistas performs very well on Cityscapes, while training on both datasets achieves the best results.

TABLE 8
Cross-dataset evaluation on half-resolution images. We train separate models on Cityscapes, Vistas and their union, and show results on validation sets of the three driving datasets. Note that this model differs from Figure 2 since it splits DB4 instead of DB3.

Method	Training dataset	Cityscapes $\overline{\text{IoU}}$	Vistas $\overline{\text{IoU}}$	KITTI $\overline{\text{IoU}}$
LDN121 64→4 s4	Cityscapes	76.0	44.0	59.5
LDN121 64→4 s4	Vistas	68.7	73.0	64.7
LDN121 64→4 s4	Cit. + Vis.	**76.2**	**73.9**	**68.7**

We briefly describe our submission [72] to the Robust vision workshop held in conjunction with CVPR 2018. The ROB 2018 semantic segmentation challenge features 1 indoor (ScanNet), and 3 road-driving (Cityscapes, KITTI, WildDash) datasets. A qualifying model has to be trained and evaluated on all four benchmarks, and it has to predict at least 39 classes: 19 from Cityscapes and 20 from ScanNet. We have configured the LDN169 64→4 model accordingly, and trained it on Cityscapes train+val, KITTI train, WildDash val and ScanNet train+val. Our submission has received the runnner-up prize. Unlike the winning submission [25], our model was trained only on the four datasets provided by the challenge, without using any additional datasets like Vistas.

6.5 Pascal VOC 2012

PASCAL VOC 2012 contains photos from private collections. There are 6 indoor classes, 7 vehicles, 7 living beings, and one background class. The dataset contains 1464 train, 1449 validation and 1456 test images of variable size. Following related work, we augment the train set with extra annotations from [73], resulting in 10582 training images in augmented set (AUG). Due to annotation errors in the AUG labels, we first train for 100 epochs on the AUG set, and then fine-tune for another 100 epochs on train (or train+val). We use 512×512 crops and divide the learning rate of pretrained weights by 8. All other hyperparameters are the same as in Cityscapes experiments. Table 9 shows that our models set the new state-of-the-art on VOC 2012 without pre-training on COCO.

6.6 Ablation experiments on Cityscapes

Table 10 evaluates the impact of auxiliary loss, SPP, and depthwise separable convolutions on generalization accuracy. The experiment labeled NoSPP replaces the SPP module with a single 3×3 convolution. The resulting 1.5pp performance drop suggests that SPP brings improvement

TABLE 9
Experimental evaluation on Pascal VOC 2012 validation and test.

Method	AUG	MS	Val $\overline{\text{IoU}}$	Test $\overline{\text{IoU}}$
DeepLabv3+ Res101	✓	✓	80.6	n/a
DeepLabv3+ Xcept	✓	✓	81.6	n/a
DDSC [74]	✓		n/a	81.2
AAF [75]	✓	✓	n/a	82.2
PSPNet [22]	✓	✓	n/a	82.6
DFN [64]	✓	✓	80.6	82.7
EncNet [76]	✓	✓	n/a	82.9
LDN121 32→4			76.4	n/a
LDN169 32→4	✓	✓	80.5	81.6
LDN161 32→4			78.6	n/a
LDN161 32→4	✓		80.4	n/a
LDN161 32→4	✓	✓	**81.9**	**83.6**

even with 64 times subsampled features. The subsequent experiment shows that the SPP module proposed in [22] does not work well with our training on Cityscapes. We believe that the 1.4pp performance drop is due to inadequate pooling grids and larger feature dimensionality which encourages overfitting. The NoAux model applies the loss only on final predictions. The resulting 1.2pp performance hit suggests that auxiliary loss succeeds to reduce overfitting on low-level features within the upsampling path. The DWS model reduces the computational complexity by replacing all 3×3 convolutions in the upsampling path with depthwise separable convolutions. This improves efficiency while only marginally decreasing accuracy.

TABLE 10
Impact of auxiliary loss, SPP, and depthwise separable convolutions on generalization accuracy on full-resolution Cityscapes val.

Method	$\overline{\text{IoU}}$	Model size	FLOP 1MPx
LDN121 64→4 NoSPP	77.5	10.2M	66.7G
LDN121 64→4 SPP [22]	77.6	10.6M	66.9G
LDN121 64→4 NoAux	77.8	9.5M	66.5G
LDN121 64→4 DWS	78.6	8.7M	54.2G
LDN121 64→4	**79.0**	9.5M	66.5G

Table 11 shows ablation experiments which evaluate the impact of data augmentations on generalization. We observe that random image flip, crop, and scale jitter improve the results by almost 5pp $\overline{\text{IoU}}$ and conclude that data augmentation is of great importance for semantic segmentation.

TABLE 11
Impact of data augmentation to the segmentation accuracy ($\overline{\text{IoU}}$) on Cityscapes val while training LDN121 64→4 on full images.

augmentation:	none	flip	flip/crop	flip/crop/scale
accuracy ($\overline{\text{IoU}}$):	74.0	75.7	76.7	79.0

6.7 Gradient checkpointing

Table 12 explores effects of several checkpointing strategies to the memory footprint and the execution speed during

training of the default LDN model (cf. Figure 2) on 768×768 images. We first show the maximum memory allocation at any point in the computational graph while training with batch size 6. Subsequently, we present the maximum batch size we could fit into GPU memory and the corresponding training speed in frames per second (FPS). We start from the straightforward baseline and gradually introduce more and more aggressive checkpointing. The checkpointing approaches are designated as follows. The label *cat* refers to the concatenation before a DenseNet unit (cf. Figure 1). The labels 1×1 and 3×3 refer to the first and the second BN-ReLu-conv group within a DenseNet unit. The label *stem* denotes the 7×7 convolution at the very beginning of DenseNet [16], including the following batchnorm, ReLU and max-pool operations. Labels TD and TU correspond to the transition-down and the transition-up blocks. The label *block* refers to the entire processing block (this approach is applicable to most backbones) Parentheses indicate the checkpoint segment. For example, (cat 1×1 3×3) means that only inputs to the concatenations in front of the convolutional unit are cached, while the first batchnorm, the 1×1 convolution and the second batchnorm are re-computed during backprop. On the other hand, (cat 1×1) (3×3) means that each convolution is in a separate segment. Here we cache the input to the concatenations and the input to the second batchnorm, while the two batchnorms are recomputed. Consequently, training with (cat 1×1 3×3) requires less caching and is therefore able to accommodate larger batches.

Now we present the most important results from the table. The fourth row of the table shows that checkpointing the (cat 1×1) subgraph brings the greatest savings with respect to the baseline (4.5GB), since it has the largest number of feature maps on input. Nevertheless, checkpointing the whole DenseNet unit (cat 1×1 3×3) brings further 3GB. Finally, checkpointing stem, transition-down and transition-up blocks relieves additional 1.8GB. This results in more than a five-fold reduction of memory requirements, from 11.3GB to 2.1B.

Experiments with the label (block) treat each dense block as a checkpoint segment. This requires more memory than (cat 1×1 3×3) because additional memory needs to be allocated during recomputation. The approach (cat 1×1) (3×3) is similar to the related previous work [21]. We conclude that the smallest memory footprint is achieved by checkpointing the stem, transition-down and transition-up blocks, as well as DenseNet units as a whole. Table 13 shows that our checkpointing approach allows training the LDN161 model with a six-fold increase of batch size with respect to the baseline implementation. On the other hand, previous checkpointing techniques [25] yield only a two-fold increase of batch size.

TABLE 12
Impact of checkpointing to memory footprint and training speed. We train LDN 32→4 on 768×768 images on a Titan Xp with 12 GB RAM. Column 2 (Memory) assumes batch size 6.

Checkpointing variant	Memory bs=6 (MB)	Max BS	Train FPS
baseline - no ckpt	11265	6	11.3
(3x3)	10107	6	10.5
(cat 1x1)	6620	10	10.4
(cat 1x1) (3x3)	5552	12	9.7
(block) (stem) (TD) (UP)	3902	16	8.4
(cat 1x1 3x3)	3620	19	10.1
(cat 1x1 3x3) (stem) (TD) (UP)	2106	27	9.2

TABLE 13
Comparison of memory footprint and training speed across various model variants. We process 768×768 images on a Titan Xp with 12 GB RAM. The column 3 (Memory) assumes batch size 6.

Variant	Uses Ckpt	Memory (MB)	Max BS	Train FPS
LDN161 32→4		20032	3	5.6
ResNet101 32→4		15002	4	7.8
LDN121 32→4		11265	6	11.3
ResNet50 32→4		10070	6	11.6
ResNet18 32→4		3949	17	24.4
LDN161 32→4	✓	3241	19	4.4
LDN121 32→4	✓	2106	27	9.2

6.8 Qualitative Cityscapes results

Finally, we present some qualitative results on Cityscapes. Figure 4 shows predictions obtained from different layers along the upsampling path. Predictions from early layers miss most small objects, however ladder-style upsampling succeeds to recover them due to blending with high-resolution features.

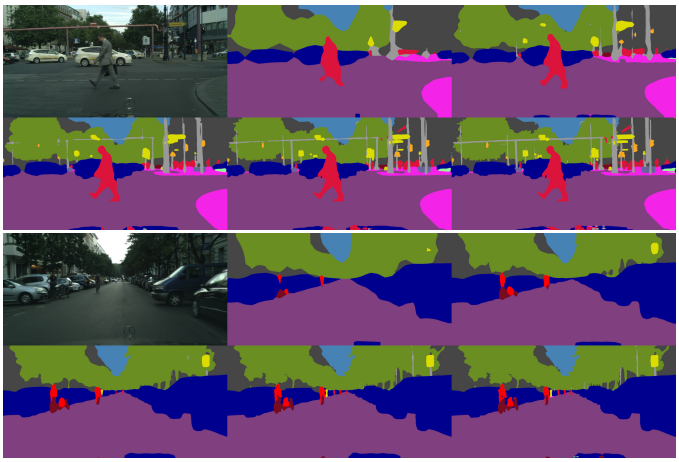

Fig. 4. Impact of ladder-style upsampling to the precision at semantic borders and detection of small objects. For each of the two input images we show predictions at the following levels of subsampling along the upsampling path: 64×, 32×, 16×, 8×, and 4×.

Figure 5 shows images from Cityscapes test in which our best model commits the largest mistakes. It is our impression that most of these errors are due to insufficient context, despite our efforts to enlarge the receptive field. Overall, we achieve the worst IoU on fences (60%) and walls (61%).

Finally, Figure 6 shows some images from Cityscapes test where our best model performs well in spite of occlusions and large objects. We note that small objects are very well recognized which confirms the merit of ladder-style upsampling.

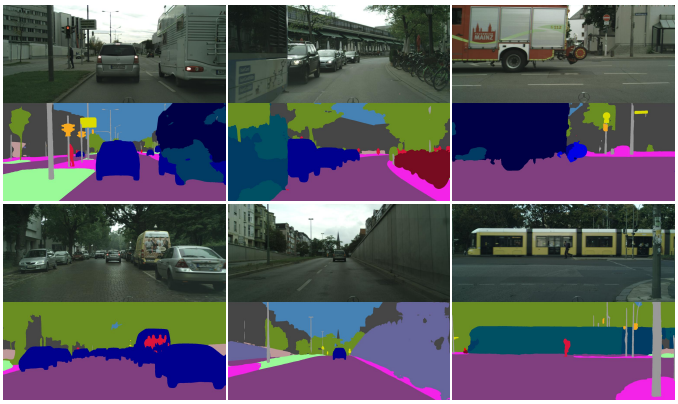

Fig. 5. Images from Cityscapes test where our model misclassified some parts of the scene. These predictions are produced by the LDN161 64→4 model which achieves 80.6% IoU on the test set.

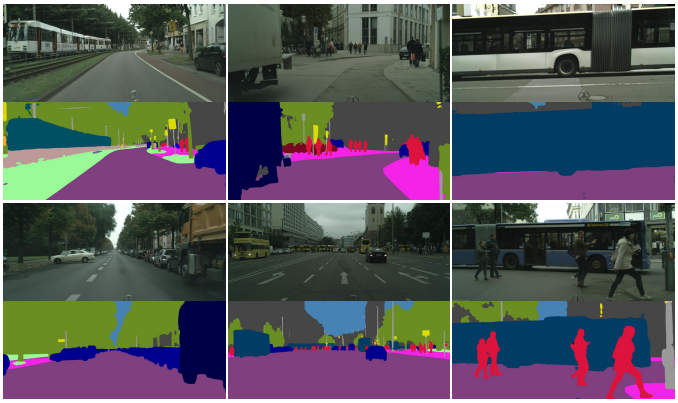

Fig. 6. Images from Cityscapes test where our best model (LDN161 64→4, 80.6% IoU test) makes no significant errors in spite of large objects, occlusions, small objects and difficult classes.

7 Conclusion

We have presented a ladder-style adaptation of the DenseNet architecture for accurate and fast semantic segmentation of large images. The proposed design uses lateral skip connections to blend the semantics of deep features with the location accuracy of the early layers. These connections relieve the deep features from the necessity to preserve low-level details and small objects, and allow them to focus on abstract invariant and context features. In comparison with various dilated architectures, our design substantially decreases the memory requirements while achieving more accurate results.

We have further reduced the memory requirements by aggressive gradient checkpointing where all batchnorm and projection layers are recomputed during backprop. This approach decreases the memory requirements for more than 5 times while increasing the training time for only around 20%. Consequently, we have been able to train on 768×768 crops with batch size 16 while requiring only 5.3 GB RAM.

We achieve state-of-the-art results on Cityscapes test with only fine annotations (LDN161: 80.6% IoU) as well as on Pascal VOC 2012 test without COCO pretraining (LDN161: 83.6% IoU). The model based on DenseNet-121 achieves 80.6% IoU on Cityscapes test while being able to perform the forward-pass on 512×1024 images in real-time (59 Hz) on a single Titan Xp GPU. To the best of our knowledge, this is the first account of applying a DenseNet-based architecture for real-time dense prediction at Cityscapes resolution.

We have performed extensive experiments on Cityscapes, CamVid, ROB 2018 and Pascal VOC 2012 datasets. In all cases our best results have exceeded or matched the state-of-the-art. Ablation experiments confirm the utility of the DenseNet design, ladder-style upsampling and aggressive checkpointing. Future work will exploit the reclaimed memory resources for end-to-end training of dense prediction in video.

References

[1] R. Hadsell, P. Sermanet, J. Ben, A. Erkan, M. Scoffier, K. Kavukcuoglu, U. Muller, and Y. LeCun, "Learning long-range vision for autonomous off-road driving," *J. Field Robotics*, vol. 26, no. 2, pp. 120–144, 2009.

[2] S. Segvic, K. Brkic, Z. Kalafatic, and A. Pinz, "Exploiting temporal and spatial constraints in traffic sign detection from a moving vehicle," *Mach. Vis. Appl.*, vol. 25, no. 3, pp. 649–665, 2014.

[3] Y. Aksoy, T. Oh, S. Paris, M. Pollefeys, and W. Matusik, "Semantic soft segmentation," *ACM Trans. Graph.*, vol. 37, no. 4, pp. 72:1–72:13, 2018.

[4] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *MICCAI*, 2015, pp. 234–241.

[5] J. Shotton, J. M. 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*, vol. 81, no. 1, pp. 2–23, 2009.

[6] G. Csurka and F. Perronnin, "An efficient approach to semantic segmentation," *International Journal of Computer Vision*, vol. 95, no. 2, pp. 198–212, 2011.

[7] C. Farabet, C. Couprie, L. Najman, and Y. LeCun, "Learning hierarchical features for scene labeling," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 8, pp. 1915–1929, 2013.

[8] I. Kreso, D. Causevic, J. Krapac, and S. Segvic, "Convolutional scale invariance for semantic segmentation," in *GCPR*, 2016, pp. 64–75.

[9] P. Krähenbühl and V. Koltun, "Efficient inference in fully connected crfs with gaussian edge potentials," in *NIPS*, 2011, pp. 109–117.

[10] G. Lin, C. Shen, A. van den Hengel, and I. D. Reid, "Efficient piecewise training of deep structured models for semantic segmentation," in *CVPR*, 2016, pp. 3194–3203.

[11] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, 1998.

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012, pp. 1106–1114.

[13] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *ICLR*, 2014, pp. 1–16.

[14] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *CVPR*, 2015.

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.

[16] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *CVPR*. IEEE Computer Society, 2017, pp. 2261–2269.

[17] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *ECCV*, 2016, pp. 630–645.

[18] A. Veit, M. J. Wilber, and S. J. Belongie, "Residual networks behave like ensembles of relatively shallow networks," in *NIPS*, 2016, pp. 550–558.

[19] G. Larsson, M. Maire, and G. Shakhnarovich, "Fractalnet: Ultra-deep neural networks without residuals," in *ICLR*, 2017.

[20] Y. Chen, J. Li, H. Xiao, X. Jin, S. Yan, and J. Feng, "Dual path networks," in *NIPS*, 2017, pp. 4470–4478.

[21] G. Pleiss, D. Chen, G. Huang, T. Li, L. van der Maaten, and K. Q. Weinberger, "Memory-efficient implementation of densenets," *CoRR*, vol. abs/1707.06990, 2017.

[22] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," in *ICCV*, 2017.

[23] 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.*, vol. 40, no. 4, pp. 834–848, 2018.

[24] L. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation," *CoRR*, vol. abs/1706.05587, 2017.

[25] S. Rota Bulò, L. Porzi, and P. Kontschieder, "In-place activated batchnorm for memory-optimized training of DNNs," in *CVPR*, June 2018.

[26] M. Yang, K. Yu, C. Zhang, Z. Li, and K. Yang, "Denseaspp for semantic segmentation in street scenes," in *CVPR*, 2018, pp. 3684–3692.

[27] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 12, pp. 2481–2495, 2017.

[28] H. Noh, S. Hong, and B. Han, "Learning deconvolution network for semantic segmentation," in *ICCV*, 2015.

[29] S. Jégou, M. Drozdzal, D. Vázquez, A. Romero, and Y. Bengio, "The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation," *CoRR*, vol. abs/1611.09326, 2016.

[30] G. Ghiasi and C. C. Fowlkes, "Laplacian pyramid reconstruction and refinement for semantic segmentation," in *ECCV*, 2016, pp. 519–534.

[31] G. Lin, A. Milan, C. Shen, and I. D. Reid, "Refinenet: Multi-path refinement networks for high-resolution semantic segmentation," in *CVPR*, 2017.

[32] C. Peng, X. Zhang, G. Yu, G. Luo, and J. Sun, "Large kernel matters - improve semantic segmentation by global convolutional network," in *CVPR*, July 2017.

[33] H. Valpola, "From neural PCA to deep unsupervised learning," *CoRR*, vol. abs/1411.7783, 2014.

[34] I. Kreso, J. Krapac, and S. Segvic, "Ladder-style densenets for semantic segmentation of large natural images," in *ICCV CVRSUAD*, 2017, pp. 238–245.

[35] E. Shelhamer, J. Long, and T. Darrell, "Fully convolutional networks for semantic segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 4, pp. 640–651, 2017.

[36] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "Overfeat: Integrated recognition, localization and detection using convolutional networks," in *ICLR*, 2014, pp. 1–16.

[37] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," in *ICLR*, 2016, pp. 1–9.

[38] L. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," in *ECCV*, 2018, pp. 833–851.

[39] T. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie, "Feature pyramid networks for object detection," in *CVPR*, 2017, pp. 936–944.

[40] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *CVPR*, 2015, pp. 3431–3440.

[41] T. Pohlen, A. Hermans, M. Mathias, and B. Leibe, "Full-resolution residual networks for semantic segmentation in street scenes," in *CVPR*, July 2017.

[42] M. A. Islam, M. Rochan, B. Neil D. B. and Y. Wang, "Gated feedback refinement network for dense image labeling," in *CVPR*, July 2017.

[43] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, "Mask R-CNN," in *ICCV*, 2017, pp. 2980–2988.

[44] H. Zhao, Y. Zhang, S. Liu, J. Shi, C. C. Loy, D. Lin, and J. Jia, "Psanet: Point-wise spatial attention network for scene parsing," in *ECCV*, 2018, pp. 270–286.

[45] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories," in *CVPR*, 2006, pp. 2169–2178.

[46] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 9, pp. 1904–1916, 2015.

[47] A. N. Gomez, M. Ren, R. Urtasun, and R. B. Grosse, "The reversible residual network: Backpropagation without storing activations," in *NIPS*, 2017, pp. 2211–2221.

[48] J. Jacobsen, A. W. M. Smeulders, and E. Oyallon, "i-revnet: Deep invertible networks," in *ICLR*, 2018.

[49] T. Chen, B. Xu, C. Zhang, and C. Guestrin, "Training deep nets with sublinear memory cost," *CoRR*, vol. abs/1604.06174, 2016.

[50] G. Huang, S. Liu, L. van der Maaten, and K. Q. Weinberger, "Condensenet: An efficient densenet using learned group convolutions," in *CVPR*, 2018.

[51] G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, and K. Q. Weinberger, "Multi-scale dense networks for resource efficient image classification," in *ICLR*, 2018.

[52] M. Everingham, S. M. A. Eslami, L. V. Gool, C. K. I. Williams, J. M. Winn, and A. Zisserman, "The pascal visual object classes challenge: A retrospective," *International Journal of Computer Vision*, vol. 111, no. 1, pp. 98–136, 2015.

[53] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.

[54] S. J. Reddi, S. Kale, and S. Kumar, "On the convergence of adam and beyond," in *ICLR*, 2018.

[55] M. Cordts, M. Omran, S. Ramos, T. Scharwächter, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset," in *CVPRW*, 2015.

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

[57] M. Orsic, I. Kreso, P. Bevandic, and S. Segvic, "In defense of pre-trained imagenet architectures for real-time semantic segmentation of road-driving images," in *CVPR*, 2019.

[58] A. Chaurasia and E. Culurciello, "Linknet: Exploiting encoder representations for efficient semantic segmentation," in *VCIP*. IEEE, 2017, pp. 1–4.

[59] P. Wang, P. Chen, Y. Yuan, D. Liu, Z. Huang, X. Hou, and G. Cottrell, "Understanding Convolution for Semantic Segmentation," *CoRR*, vol. abs/1702.08502, 2017.

[60] R. Zhang, S. Tang, Y. Zhang, J. Li, and S. Yan, "Scale-adaptive convolutions for scene parsing," in *ICCV*, Oct 2017.

[61] Z. Wu, C. Shen, and A. van den Hengel, "Wider or deeper: Revisiting the resnet model for visual recognition," *CoRR*, vol. abs/1611.10080, 2016.

[62] A. Kendall, Y. Gal, and R. Cipolla, "Multi-task learning using uncertainty to weigh losses for scene geometry and semantics," in *CVPR*, 2018, pp. 7482–7491.

[63] T. Wu, S. Tang, R. Zhang, J. Cao, and J. Li, "Tree-structured kronecker convolutional network for semantic segmentation," *CoRR*, vol. abs/1812.04945, 2018.

[64] C. Yu, J. Wang, C. Peng, C. Gao, G. Yu, and N. Sang, "Learning a discriminative feature network for semantic segmentation," in *CVPR*, 2018, pp. 1857–1866.

[65] Y. Zhuang, F. Yang, L. Tao, C. Ma, Z. Zhang, Y. Li, H. Jia, X. Xie, and W. Gao, "Dense relation network: Learning consistent and context-aware representation for semantic image segmentation," in *ICIP*, 2018, pp. 3698–3702.

[66] A. Casanova, G. Cucurull, M. Drozdzal, A. Romero, and Y. Bengio, "On the iterative refinement of densely connected representation levels for semantic segmentation," *CoRR*, vol. abs/1804.11332, 2018.

[67] M. A. Islam, M. Rochan, S. Naha, N. D. B. Bruce, and Y. Wang, "Gated feedback refinement network for coarse-to-fine dense semantic image labeling," *CoRR*, vol. abs/1806.11266, 2018.

[68] C. Yu, J. Wang, C. Peng, C. Gao, G. Yu, and N. Sang, "Bisenet: Bilateral segmentation network for real-time semantic segmentation," in *ECCV*, 2018, pp. 334–349.

[69] H. Zhao, X. Qi, X. Shen, J. Shi, and J. Jia, "Icnet for real-time semantic segmentation on high-resolution images," in *ECCV*, vol. 11207, 2018, pp. 418–434.

[70] G. Neuhold, T. Ollmann, S. Rota Bulò, and P. Kontschieder, "The mapillary vistas dataset for semantic understanding of street scenes," in *ICCV*, 2017.

[71] H. Alhaija, S. Mustikovela, L. Mescheder, A. Geiger, and C. Rother, "Augmented reality meets computer vision: Efficient data generation for urban driving scenes," *International Journal of Computer Vision (IJCV)*, 2018.

[72] I. Kreso, M. Orsic, P. Bevandic, and S. Segvic, "Robust semantic segmentation with ladder-densenet models," *CoRR*, vol. abs/1806.03465, 2018.

[73] B. Hariharan, P. Arbelaez, L. D. Bourdev, S. Maji, and J. Malik, "Semantic contours from inverse detectors," in *ICCV*, 2011, pp. 991–998.
[74] P. Bilinski and V. Prisacariu, "Dense decoder shortcut connections for single-pass semantic segmentation," in *CVPR*. IEEE Computer Society, 2018, pp. 6596–6605.
[75] T. Ke, J. Hwang, Z. Liu, and S. X. Yu, "Adaptive affinity fields for semantic segmentation," in *ECCV*, 2018, pp. 605–621.
[76] H. Zhang, K. J. Dana, J. Shi, Z. Zhang, X. Wang, A. Tyagi, and A. Agrawal, "Context encoding for semantic segmentation," in *CVPR*, 2018, pp. 7151–7160.