

SUPPLEMENTARY MATERIAL TO THE PAPER:

"PYRAMIDAL CONVOLUTION: RETHINKING CONVOLUTIONAL NEURAL NETWORKS FOR VISUAL RECOGNITION"

Anonymous authors

Paper under double-blind review

This supplementary material just presents additional architectures details and/or analysis on the main results already introduced in the paper (we could not fit more details into the main paper as we propose architectures and run experiments on four core recognition tasks). It contains five main sections: Section 1 presents the details of our architecture for object detection; Section 2 presents the details for the video classification pipeline; Section 3 contains the results of PyConvResNet using additional data augmentation. Section 4 additional comparisons with the existing works. Section 5 presets a direct comparison with the inception module and also the integration of our PyConv in the inception architecture. Section 6 presents a head-to-head comparison with other strong image segmentation methods. Finally, Section 7 shows some visual examples on image segmentation.

1 PYCONV ON OBJECT DETECTION

As we already presented in the the main paper the final result on object detection, that we outperform the baseline by a significant margin (see main contribution (4) in the main paper), this section provides the details of our architecture on object detection and the exact numbers of the results.

As our proposed PyConv uses different levels of kernel sizes in parallel, it can provide significant benefits for object detection task, where the objects can appear in the image at different scales. For object detection, we integrate our PyConv in a powerful approach, Single Shot Detector (SSD) [1]. SSD is a very efficient single stage framework for object detection, which performs the detection at multiple feature maps resolutions. Our proposed framework for object detection, PyConvSSD, is illustrated in Fig. 1. The framework contains two main parts:

(1) **PyConvResNet Backbone.** In our framework we use the proposed PyConvResNet as backbone, which was previously pre-trained on ImageNet dataset [2]. To maintain a high efficiency of the framework, and also to have a similar number of output feature maps as in the backbone used in [1], we remove from our PyConvResNet backbone all layers after the third stage. We also set all strides in the stage 3 of the backbone network to 1. With this, PyConvResNet provides (as output of the stage 3) 1024 output feature maps ($S3_{FM}$) with the spatial resolution 38×38 (for an input image size of 300×300).

(2) **PyConvSSD Head.** Our PyConvSSD head illustrated in Fig. 1 uses the proposed PyConv to further extract different features using different kernel sizes in parallel. Over the resulted feature maps for the third stage of the backbone we apply a PyConv with four levels (kernel sizes: 9×9 , 7×7 , 5×5 , 3×3). Also PyConv performs the downsampling (stride $s=2$) of the feature maps using these multiple kernel sizes in parallel. As the feature maps resolution decreases we also decrease the levels of the pyramid for PyConv. The last two PyConv contains only one level (which is basically the standard 3×3) as the spatial resolution of the feature maps is very small. Note that the last two PyConvs use a stride $s=1$ and the spatial resolution is decreased just by not using padding. Thus, the head decreases the spatial resolution of the feature maps from 38×38 to 1×1 . All the output feature maps from the PyConvs in the head are used for detections.

For each of the six output feature maps selected for detection $\{S3_{FM}, H_{FM1}, H_{FM2}, H_{FM3}, H_{FM4}, H_{FM5}\}$ the framework performs the detection using a corresponding number of default boxes (anchor boxes) $\{4, 6, 6, 6, 4, 4\}$ for each spatial location. For instance, for ($S3_{FM}$) output feature maps with the spatial resolution 38×38 , using the four default boxes on each location results in 5776 detections. For localizing each bounding box, there are four values that network should predict (loc: $\Delta(cx, cy, w, h)$, where cx and cy represent the center point of the bounding box, w and h the width

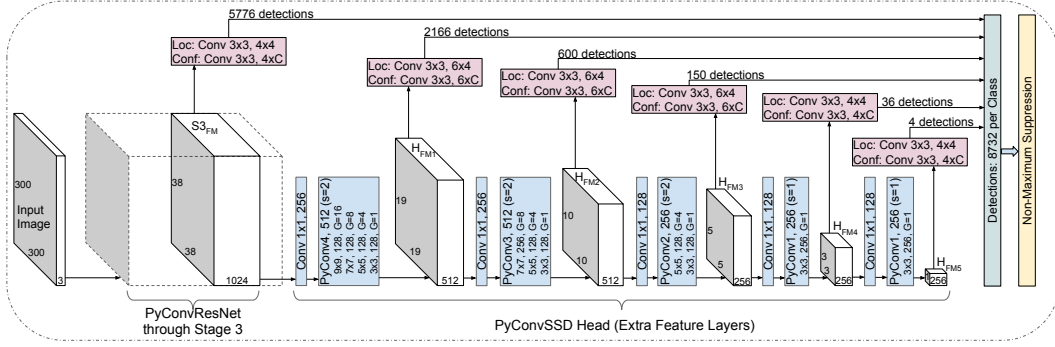


Figure 1: PyConvSSD framework for object detection.

Table 1: PyConvSSD with 300×300 input image size (results on COCO val2017).

Architecture	Avg. Precision, IoU:			Avg. Precision, Area:			Avg. Recall, #Dets:			Avg. Recall, Area:			params	GFLOPs
	0.5:0.95	0.5	0.75	S	M	L	1	10	100	S	M	L		
baseline SSD-50	26.20	43.97	26.96	8.12	28.22	42.64	24.50	35.41	37.07	12.61	40.76	57.25	22.89	20.92
PyConvSSD-50	29.16	47.26	30.24	9.31	31.21	47.79	26.14	37.81	39.61	13.79	43.87	60.98	21.55	19.71
baseline SSD-101	29.58	47.69	30.80	9.38	31.96	47.64	26.47	38.00	39.64	14.09	43.54	61.03	41.89	48.45
PyConvSSD-101	31.27	50.00	32.67	10.65	33.76	51.75	27.33	39.33	41.07	15.48	45.53	63.44	39.01	45.02

and height of the bounding box). This bounding box offset output values are measured relative to a default box position, relative to each feature maps location. Also, for each bounding box, the network should output the confidences for each class category (in total C class categories). For providing the detections the framework uses a classifier which is represented by a 3×3 convolution, that outputs for each bounding box the confidences for all class categories (C). For localization the framework uses also a 3×3 convolution to output the four localization values for each regressed default bounding box. In total, the framework outputs 8732 detections (for 300×300 input image size), which pass through a non-maximum suppression to provide the final detections.

Different from the original SSD framework [1], for a fair and direct comparison, in the baseline SSD, we replaced the VGG backbone [3] with ResNet [4], as ResNet is far superior to VGG in terms of recognition performance and computational costs as shown in [4]. Therefore, as main differences from our PyConvSSD, the baseline SSD in this work uses ResNet [4] as backbone and the SSD head uses standard 3×3 conv (instead of PyConv) as in the original framework [1]. For showing the exact numbers to compare our PyConvSSD with the baseline on object detection, we use COCO dataset [5], which contains 81 categories. We use for training COCO train2017 (118K images) and for testing COCO val2017 (5K images). We train for 130 epochs using 8 GPUs with 32 batch size each, resulting in 60K training iterations. We use for training SGD optimiser with momentum 0.9, weight decay 0.0005, with the learning rate 0.02 (reduced by 1/10 before 86-th and 108-th epoch). We also use a linear warmup in the first epoch [6]. For data augmentation, we perform random crop as in [1], color jitter and horizontal flip. We use an input image size of 300×300 and report the metrics as in [1].

Table 1 shows the comparison results of PyConvSSD with the baseline, over 50- and 101-layers backbones. While being more efficient in terms of number of parameters and FLOPs, the proposed PyConvSSD reports significant improvements over the baseline over all metrics. Notably, PyConvSSD with 50 layers backbone is even competitive with the baseline using 101 layers as backbone. This results show a grasp of the benefits for PyConv on object detection task.

2 PYCONV ON VIDEO CLASSIFICATION

In the main paper we introduced the main result for video classification, that we report significant results over the baseline (see main contribution (4)). This section presents the details of the architecture and the exact numbers. PyConv can show significant benefits on video related tasks as it can enlarge the receptive field and process the input using multiple kernels scales in parallel not only spatially but

Table 2: ResNet3D architecture for video recognition.

stage	output	ResNet3D-50	PyConvResNet3D-50
	$16 \times 112 \times 112$	$5 \times 7 \times 7$, 64 stride (1,2,2)	$5 \times 7 \times 7$, 64 stride (1,2,2)
		$1 \times 3 \times 3$ max pool stride (1,2,2)	
1	$16 \times 56 \times 56$	$\begin{bmatrix} 1 \times 1 \times 1, 64 \\ 3 \times 3 \times 3, 64 \\ 1 \times 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1 \times 1, 64 \\ \text{PyConv4, 64:} \\ 7 \times 9 \times 9, 16, G=16 \\ 5 \times 7 \times 7, 16, G=8 \\ 3 \times 5 \times 5, 16, G=4 \\ 3 \times 3 \times 3, 16, G=1 \\ 1 \times 1 \times 1, 256 \end{bmatrix} \times 3$
2	$16 \times 28 \times 28$	$\begin{bmatrix} 1 \times 1 \times 1, 128 \\ 3 \times 3 \times 3, 128 \\ 1 \times 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1 \times 1, 128 \\ \text{PyConv3, 128:} \\ 5 \times 7 \times 7, 64, G=8 \\ 3 \times 5 \times 5, 32, G=4 \\ 3 \times 3 \times 3, 32, G=1 \\ 1 \times 1 \times 1, 512 \end{bmatrix} \times 4$
3	$8 \times 14 \times 14$	$\begin{bmatrix} 1 \times 1 \times 1, 256 \\ 3 \times 3 \times 3, 256 \\ 1 \times 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \times 1, 256 \\ \text{PyConv2, 256:} \\ 3 \times 5 \times 5, 128, G=4 \\ 3 \times 3 \times 3, 128, G=1 \\ 1 \times 1 \times 1, 1024 \end{bmatrix} \times 6$
4	$4 \times 7 \times 7$	$\begin{bmatrix} 1 \times 1 \times 1, 512 \\ 3 \times 3 \times 3, 512 \\ 1 \times 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1 \times 1, 512 \\ \text{PyConv1, 512:} \\ 3 \times 3 \times 3, 512, G=1 \\ 1 \times 1 \times 1, 2048 \end{bmatrix} \times 3$
	$1 \times 1 \times 1$	global avg pool 400-d fc	global avg pool 400-d fc
# params		47.00 $\times 10^6$	44.91 $\times 10^6$
FLOPs		93.26 $\times 10^9$	91.81 $\times 10^9$

also in the temporal dimension. Extending the networks from image recognition to video involves extending the 2D spatial convolution to 3D spatio-temporal convolution. Table 2 presents the baseline network ResNet3D and our proposed network PyConvResNet3D, which are the initial 2D networks extended to work with video input. The input for the network is represented by 16-frame input clips, with spatial size is 224×224 . As the temporal size is smaller than spatial dimensions, for our PyConv we do not need to use equally large size on the upper layers of the pyramid. In the first stage of the network, our PyConv with four layers contains kernel sizes of: $7 \times 9 \times 9$, $5 \times 7 \times 7$, $3 \times 5 \times 5$ and $3 \times 3 \times 3$ (the temporal dimension comes first).

For video classification, we perform the experiments on Kinetics-400 [7], which is a large-scale video recognition dataset that contains $\sim 246k$ training videos and 20k validation videos, with 400 action classes. Similar to image recognition, use the SGD optimizer with a standard momentum of 0.9 and weight decay of 0.0001, we train the model for 90 epochs, starting with a learning rate of 0.1 and reducing it by 1/10 at the 30-th, 60-th and 80-th epochs, similar to [4], [6]. The models are trained from scratch, using the weights initialization of [8] for all convolutional layers; for training we use a minibatch of 64 clips over 8 GPUs. Data augmentation is similar to [3], [9]. For training, we randomly select 16-frame input clips from the video. We also skip four frames to cover a longer video period within a clip. The spatial size is 224×224 , randomly cropped from a scaled video, where the shorter side is randomly selected from the interval [256, 320], similar to [3], [9]. As the networks on video data are prone to overfitting due to the increase in number of parameters, we use dropout [10] after the global average pooling layer, with a 0.5 dropout ratio. For the final validation, following common practice, we uniformly select a maximum of 10 clips per video. Each clip is scaled to 256 pixels for the shorter spatial side. We take 3 spatial crops to cover the spatial dimensions. In total, this results in a maximum of 30 clips per video, for each of which we obtain a prediction. To get the final prediction for a video, we average the softmax scores. We report both, top-1 and top-5 error rates.

Table 3 presents the result comparing our network, PyConvResNet3D, with the baseline over 50-layers depth. PyConvResNet3D improves significantly the results over baseline, for top-1 error, from 37.01% to 34.56%. In the same time our network requires less number of parameters and FLOPs than the baseline. Fig. 2 shows the training and validation curves where we can see that our network

Table 3: Video recognition error rates.

Architecture	top-1(%)	top-5(%)	params M	GFLOPs
ResNet3D-50 [4]	37.01	15.41	47.00	93.26
PyConvResNet3D-50	34.56	13.34	44.91	91.81

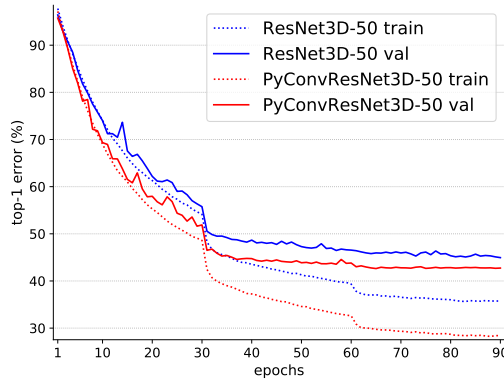


Figure 2: Training and validation curves on Kinetics-400 dataset (these results are computed during training over independent clips).

improves significantly the training convergence. This results show the potential of PyConv on video related tasks.

3 PYCONV ON IMAGENET WITH MORE COMPLEX TRAINING SETTINGS

The ImageNet results presented in the paper mainly aim to show the advantages of our PyConv over the standard convolution by running all the networks with the same standard training settings for a fair comparison. Note that there are other works which report better results on ImageNet, such as [11]–[13]. However, the improvements are mainly due to the training settings. For instance, [12] uses very complex training settings, such as, complex data augmentation (autoAugment [14]) with different regularization techniques (dropout [15]), stochastic depth [16], the training is performed on a powerful Google TPU computational architecture over 350 epochs with a large batch of 2048. The works [11], [13], besides using a strong computational architecture with many GPUs, take advantage of a large dataset of 3.5B images collected from Instagram (this dataset is not publicly available). Therefore, these resources are not handy to everyone. However, the results show that PyConv is superior to standard convolution and combining it with [11]–[13] can bring further improvements. While on ImageNet we do not have access to such scale of computational and data resources to directly compete with state-of-the-art, we do push further and show that our proposed framework obtains state-of-the-art results on challenging task of image segmentation.

To support our claim, that our networks can be easily improved using more complex training settings, we integrate an additional data augmentation, CutMix [17]. As CutMix requires more epochs to converge, we increase the training epochs to 300 and use a cosine scheduler [18] for learning rate decaying. To speed-up the training, we increase the batch size to 1024 and use mixed precision [19]. Table 4 presents the comparison results of PyConvResNet for the baseline training settings and with the CutMix data augmentation. For both depths, 50- and 101-layers, just adding these simple additional training settings improve significantly the results. For the same trained models, in addition to the standard test crop size of 224×224 we also run the testing on 320×320 crop size. This results show that there is still room for improvement if more complex training settings are included (as the training settings from [12]) and/or additional data used for training (as in [11], [13]), however, this requires significantly more computational and data resources, which are not easily available.

Table 4: Validation error rates comparison results of PyConvResNet on ImageNet with different training settings, for network depth 50 and 101.

Network	test crop: 224×224			test crop: 320×320			params
	top-1	top-5	GFLOPs	top-1	top-5	GFLOPs	
PyConvResNet-50	22.12	6.20	3.88	21.10	5.55	7.91	24.85
PyConvResNet-50 + augment	20.56	5.31	3.88	19.41	4.75	7.91	24.85
PyConvResNet-101	20.99	5.53	7.31	20.03	4.82	14.92	42.31
PyConvResNet-101 + augment	19.42	4.87	7.31	18.51	4.28	14.92	42.31

4 EXTENDED RELATED WORK

(Due to limited number of pages, we could not fit more related works in the main paper. As it will be provided one additional page in the next step, we plan integrating the below comparisons in the main paper)

The inception module used in various CNN architectures [20]–[22] in another approach to integrate kernels at multiple scales. However, our PyConv is different from inception module in both key components: **a) spatial kernel sizes:** our PyConv is capable of using multiple scales of the kernels while maintaining a similar number of parameters and computational costs as the standard convolution. Inception module is not capable of building real high spatial sizes for the kernels without affecting the computational costs compared to standard 3x3 convolution as it simulates higher kernels sizes by factorizing it in successive convolutions. We provide more details in the next section.

b) kernel depths: inception module uses only the full connectivity (depth) of the kernels, while our PyConv uses a pyramid of various kernels connectivity (depths). The depths of our PyConv kernels range from full to very low connectivity. This wide range of kernel depths is not only important for controlling the computational costs, it is also important for network learning and convergence. As important to have a varying spatial sizes of the kernels for learning different spatial dependencies/details, similarly, it is important to have a varying kernel depths to learn different feature maps dependencies/entanglements. The full connectivity of the kernels is important for learning very complex feature maps entanglements, but the price to pay is the fact that the learning convergence is very difficult (see ResNet in Fig.7 in the main paper) due to high complexity. On the other hand, having lower depth of the kernels reduces the complexity entanglement, helping significantly the starting of the convergence. Thus, the lower connectivity is very critical at the beginning of the training, for starting the convergence, while higher connectivity becomes more critical towards the end, where more complex relationships can be finally achieved through learning. This is one of the reasons that our networks converge very fast (see Fig.7 in the main paper), in fact we are able to outperform ResNet (in the first interval) by just using half of the epochs. These two interconnected pyramids (for the kernels spatial size and connectivity) for our PyConv provide a very diverse pool of variations of different kernel types that the network can explore during learning, leading to improved convergence and learning capabilities for visual patterns.

The work [23] introduces mixed depthwise convolution (MixConv). Besides a slight variation of above point b), there is also another fundamental difference that separates our work from MixConv: as we can see from Fig.3 of MixConv work [23], it splits the input feature maps into separate parts and applies independently a different kernel size for each part. Thus, each spatial kernel size has only access to a limited portion of the input feature maps. For instance, the kernels with 3x3 size run only on a limited number of input feature maps, without having access to the remaining ones. Similarly, for the next kernel sizes. This is an important shortcoming of MixConv, as there is no explainable reason for giving, let’s say, only the first 16 input feature maps to 3x3 kernels and only the next 16 maps to 5x5 kernels. While in our PyConv, each kernel level gets access to all input feature maps, thus, each level is able to extract a more complete representation of the input.

5 PYCONV ON INCEPTION ARCHITECTURE

The Inception family [20]–[22] is another powerful type of CNN, which uses the inception block to construct the network. To further show the advantages of our PyConv we replaced the inception

Table 5: Validation error rates comparison results on ImageNet for Inception architecture.

Architecture	top-1 (%)	top-5 (%)	params ($\times 10^6$)	FLOPs ($\times 10^9$)	Latency (sec./batch)	Throughput (img./sec.)
Inception-ResNet-v2 [22]	20.49	5.29	55.84	16.75	0.965	265
PyConvInception	20.31	5.21	<u>43.48</u>	<u>11.92</u>	<u>0.738</u>	<u>347</u>

Table 6: Head-to-Head comparison on image segmentation (using ResNet with 50 layers as backbone) on ADE20K.

Head	output stride backbone: 8					output stride backbone: 16				
	mean IoU	pixel Acc.	params	GFLOPs		mean IoU	pixel Acc.	params	GFLOPs	
baseline [24]: 3×3 conv	37.87	78.17	35.42	131.37		36.84	77.84	35.42	39.52	
DeepLabv3 [25]: ASPP	40.91	79.92	41.48	151.17		40.34	79.44	41.48	44.47	
PSPNet [24]: PPM	41.24	80.01	49.06	165.42		39.75	79.17	49.06	48.08	
PyConvSegNet: PyConvPH	41.54	80.18	<u>34.40</u>	<u>116.84</u>		40.43	79.45	<u>34.40</u>	<u>36.08</u>	

blocks in Inception-ResNet-v2 [22] with our PyConv building blocks. The comparison results are presented in Table 5, where we can see that our PyConv provides significant improvements. We follow [22] and use for these experiments an image crop size of 299×299 . Note that [22] reports 19.9% top-1 error rate, the difference from our experiments is due to different training settings, but more importantly, the single crop - single model experimental results (Table 2. in [22]) are reported on the non-blacklisted subset of the validation set of ImageNet (refer to [22] for more details).

For comparing the running time we report also the latency (measured in terms of seconds per batch) and throughput (images per second), the results are the average over an entire training epoch. We can see that our network is significantly faster than Inception-ResNet-v2 [22]. The reason for this discrepancy in running time is due to the fact that Inception blocks [22] factorizes a convolution in several successive convolutions to avoid increasing significantly the number of parameters and FLOPs. For instance, Inception simulates a spatial 7×7 kernel by factorizing it into two successive convolutions of 1×7 and 7×1 . This is just a rough approximation of 7×7 kernel as it can learn only horizontal and vertical patterns but not at different other angles as 7×7 . Furthermore, this is not scalable, because if we want increase further the size of the kernel it will still bring additional costs, so they did not go more than that. Importantly, factorizing a convolution in several successive convolutions as in Inception blocks (even for two chained 3×3 conv for simulating 5×5), reduces the degree of parallelism and can affect negatively the running time, while in our case, all levels in our PyConv run independently in parallel (thus, there is no theoretical limitation regarding the running time) and we can scale the kernel size at very high spatial resolutions without increasing the computational costs.

6 HEAD-TO-HEAD COMPARISON ON IMAGE SEGMENTATION

We compare our proposed framework, PyConvSegNet, with two of the most powerful architectures for semantic segmentation [24] and [25]. Table 6 presents head-to-head comparison of our method with state-of-the-art heads on image segmentation: PSPNet with Pyramid Pooling Module (PPM) head, and DeepLabv3 with Atrous Spatial Pyramid Pooling (ASPP). The baseline is constructed as in [24], which as head, it basically applies a 3×3 conv over the output feature maps provided by the backbone. For a fair and direct comparison, all methods use the same auxiliary loss (deep supervision) exactly as in [24]. For a comprehensive view, the reports in terms of number of parameters and FLOPs include the auxiliary loss components. As [24] uses an output stride for the backbone of 8 and [25] uses 16, we report the experiments for both cases. We run these experiments using the ResNet with 50 layers as backbone. Table 6 shows that our proposed head is not only more accurate than the other methods, but it is also more efficient, requiring significantly smaller number of parameters and FLOPs than [24] and [25]. We can also see that without a strong head on top of the backbone, the baseline reports significantly worse results.

7 QUALITATIVE EXAMPLES ON IMAGE SEGMENTATION

Fig. 3 shows some qualitative examples for visually comparing our proposed approach for image segmentation, PyConSegNet, with state-of-the-art approaches PSPNet [24] and DeepLabv3 [25]. For the numeric results, refer to Table 4 in the main paper (for the output stride backbone 8). This examples show the visual comparison results between our proposed head, PyConvPH (PyConv parsing head), with ASPP (Atrous Spatial Pyramid Pooling) of [25] and PPM head (Pyramid Pooling Module) of [24].

Very suggestive is the last row example of Fig. 3, where we can clearly notice the difference in segmentation details. It is remarkable that our proposed head can compete at a high level with other state-of-the art approaches for image segmentation while having significantly less requirements in terms of number of parameters and computational complexity. For instance, in comparison with our PyConSegNet, PSPNet [24] requires over 40% more parameters and FLOPs, while DeepLabv3 [25] requires over 20% more parameters and close to 30% more FLOPs.

In the second row example of Fig. 3 we can also notice a failure case of our approach, which confuses the door with a window. However, this case is quite difficult and confusing even for a human eye. Fig. 4 shows some visual results of our approach, PyConSegNet, using 50-, 101-, 152-layers for the PyConvResNet backbone. For the exact number, refer to Table 5 in the main paper (multi-scale inference). Note in the second row of Fig. 4 how the quality of the segmentation for the fan (ceiling mount air fan) is improving while increasing the depth of our PyConvResNet backbone.

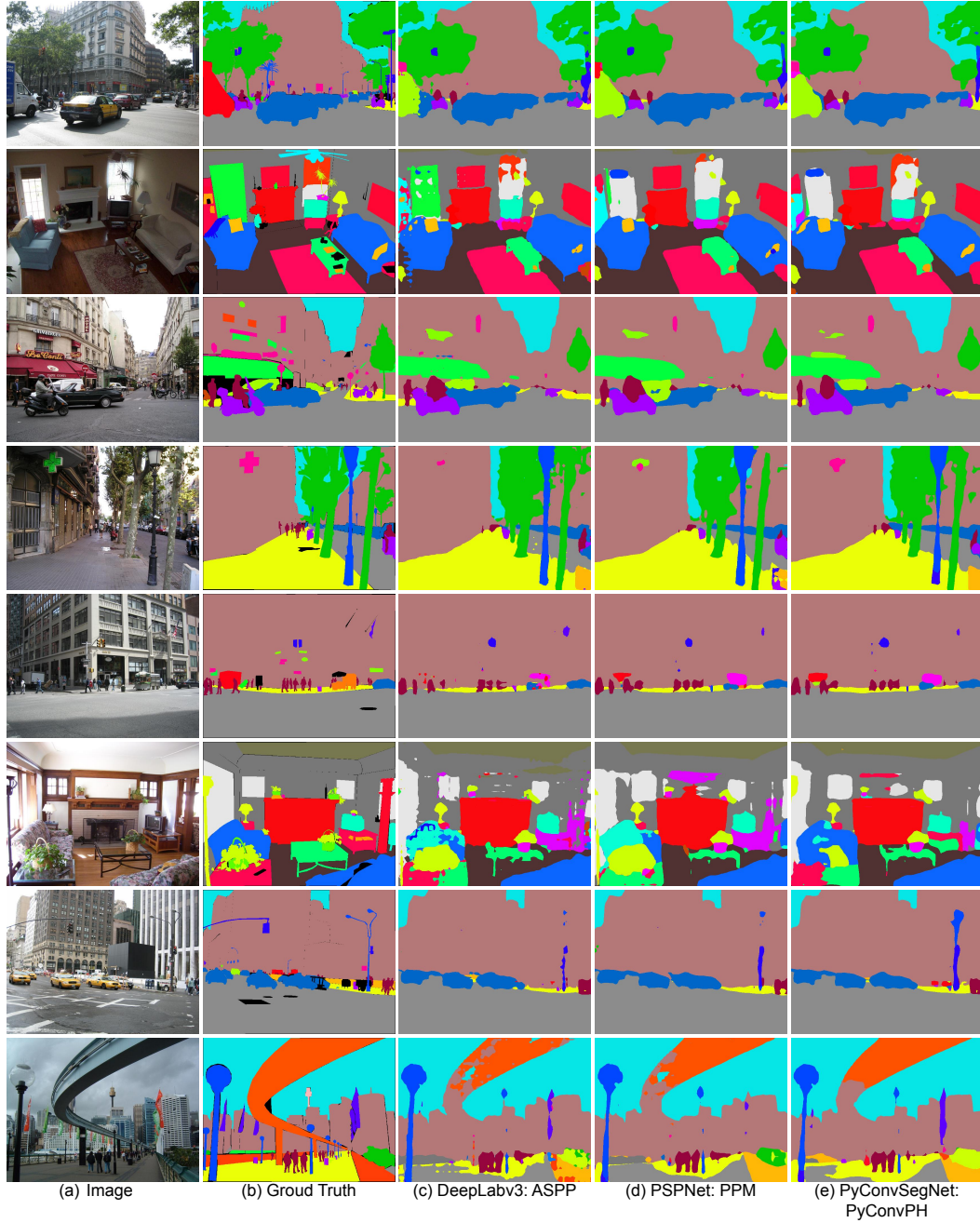


Figure 3: Visual comparison results of our approach PyConvSegNet (with PyConvPH head) with state-of-the-art approaches: PSPNet [24] (with PPM head) and DeepLabv3 [25] (with ASPP head). The images are from ADE20K dataset [26] validation.

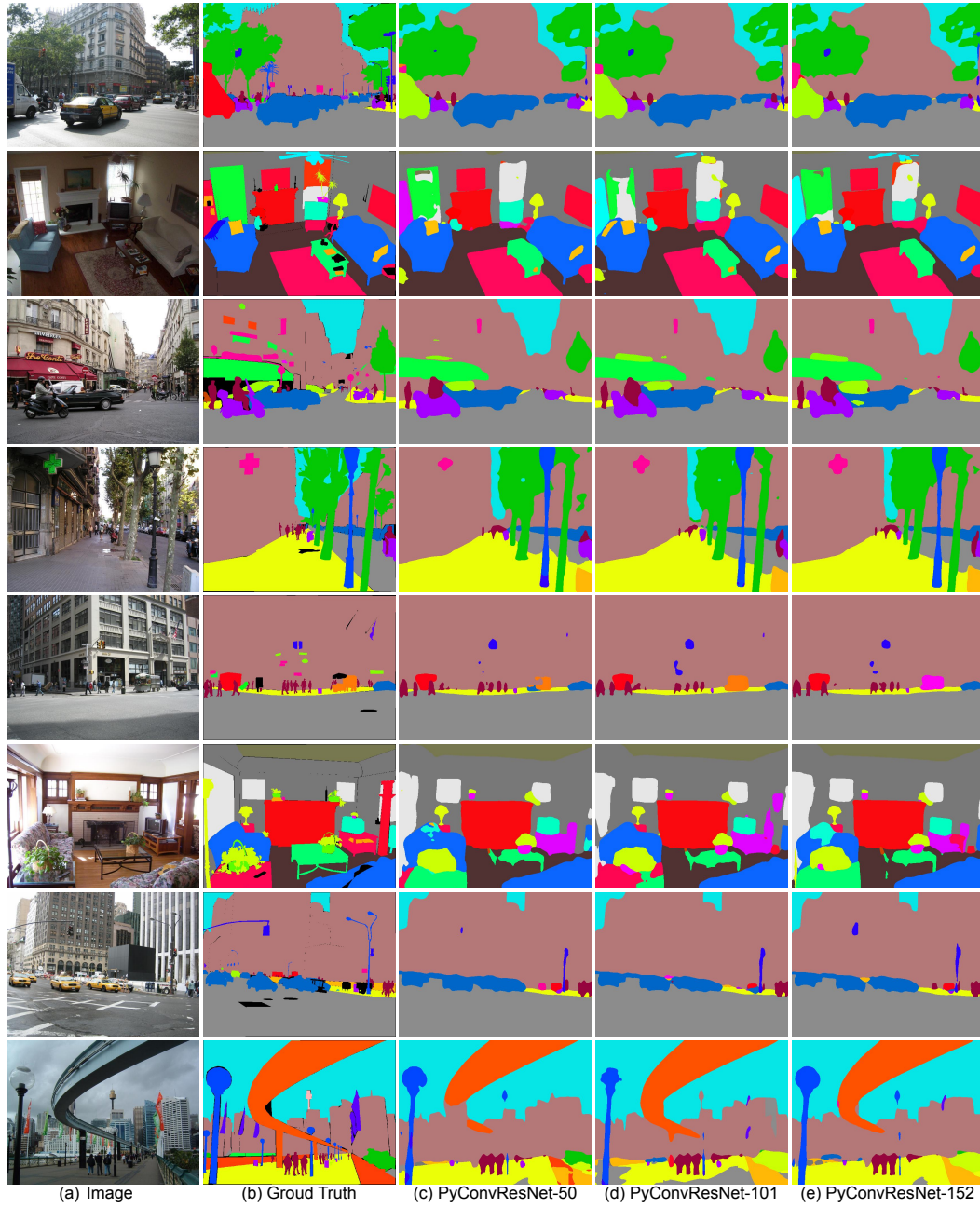


Figure 4: Visual results of our approach, PyConvSegNet, on 50-, 101-, 152-layers deep backbone PyConvResNet. The images are from ADE20K dataset [26] validation set.

REFERENCES

- [1] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, “Ssd: Single shot multibox detector,” in *ECCV*, 2016.
- [2] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, “Imagenet large scale visual recognition challenge,” *IJCV*, vol. 115, no. 3, pp. 211–252, 2015.
- [3] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv:1409.1556*, 2014.
- [4] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *CVPR*, 2016.
- [5] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, “Microsoft coco: Common objects in context,” in *ECCV*, 2014.
- [6] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, “Accurate, large minibatch sgd: Training imagenet in 1 hour,” *arXiv:1706.02677*, 2017.
- [7] W. Kay, J. Carreira, K. Simonyan, B. Zhang, C. Hillier, S. Vijayanarasimhan, F. Viola, T. Green, T. Back, P. Natsev, *et al.*, “The kinetics human action video dataset,” *arXiv:1705.06950*, 2017.
- [8] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,” in *ICCV*, 2015.
- [9] X. Wang, R. Girshick, A. Gupta, and K. He, “Non-local neural networks,” in *CVPR*, 2018.
- [10] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, “Improving neural networks by preventing co-adaptation of feature detectors,” *arXiv:1207.0580*, 2012.
- [11] D. Mahajan, R. Girshick, V. Ramanathan, K. He, M. Paluri, Y. Li, A. Bharambe, and L. van der Maaten, “Exploring the limits of weakly supervised pretraining,” in *ECCV*, 2018.
- [12] M. Tan and Q. V. Le, “Efficientnet: Rethinking model scaling for convolutional neural networks,” in *ICML*, 2019.
- [13] H. Touvron, A. Vedaldi, M. Douze, and H. Jégou, “Fixing the train-test resolution discrepancy,” in *NeurIPS*, 2019.
- [14] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le, “Autoaugment: Learning augmentation strategies from data,” in *CVPR*, 2019.
- [15] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *JMLR*, vol. 15, no. 1, pp. 1929–1958, 2014.
- [16] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, “Deep networks with stochastic depth,” in *ECCV*, 2016.
- [17] S. Yun, D. Han, S. J. Oh, S. Chun, J. Choe, and Y. Yoo, “Cutmix: Regularization strategy to train strong classifiers with localizable features,” in *ICCV*, 2019.
- [18] I. Loshchilov and F. Hutter, “Sgdr: Stochastic gradient descent with warm restarts,” in *ICLR*, 2017.
- [19] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, *et al.*, “Mixed precision training,” *arXiv preprint arXiv:1710.03740*, 2017.
- [20] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in *CVPR*, 2015.
- [21] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision,” in *CVPR*, 2016.
- [22] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, “Inception-v4, inception-resnet and the impact of residual connections on learning,” in *AAAI*, 2017.
- [23] M. Tan and Q. V. Le, “Mixconv: Mixed depthwise convolutional kernels,” *arXiv preprint arXiv:1907.09595*, 2019.
- [24] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, “Pyramid scene parsing network,” in *CVPR*, 2017.
- [25] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, “Rethinking atrous convolution for semantic image segmentation,” *arXiv:1706.05587*, 2017.
- [26] B. Zhou, H. Zhao, X. Puig, T. Xiao, S. Fidler, A. Barriuso, and A. Torralba, “Semantic understanding of scenes through the ade20k dataset,” *IJCV*, vol. 127, no. 3, pp. 302–321, 2019.