Unlocking Early-Exiting Semantic Segmentation with Branched Networks

Mateus S. Gilbert[1], Roberto G. Pacheco[3], Rodrigo S. Couto[1],
Marcello L. R. de Campos[2], Miguel Elias M. Campista[1]

[1]GTA/DEL-POLI/PEE-COPPE - Universidade Federal do Rio de Janeiro (UFRJ) – Rio de Janeiro, Brazil
[2]SMT/DEL-POLI/PEE-COPPE - Universidade Federal do Rio de Janeiro (UFRJ) – Rio de Janeiro, Brazil
[3]Universidade Federal de Mato Grosso (UFMT) – Cuiabá, Brazil

***Abstract*—Early-exit Deep Neural Networks (DNNs) and DNNs partitioning are valuable options to ease DNN implementation for image classification in resource-constrained devices and latency-sensitive applications. By inserting side branches into the usual DNN architecture, the resulting model can perform early classification attempts, using features extracted in layers that come before the model's output. DNN partitioning spreads DNN layers subsets across devices, the edge, and the cloud, bringing branches closer to the device. Offloading to the cloud is limited to hard-to-classify data. Given the success in image classification, it is natural to try to replicate these results with semantic segmentation. In this direction, we propose Branched-DeepLabV3 (BranDeepLabV3) as an early attempt to bridge the gap between early-exit DNNs and semantic segmentation. Our results show that, by adding side branches to a semantic segmentation DNN, we can deliver adequate outputs without traversing the whole network. Moreover, our qualitative results show the usefulness of early-exits in semantic segmentation, as even coarse segmentations can be good enough to supply time-sensitive applications.**

***Index Terms*—semantic segmentation, early-exiting, DNN partitioning**

I. Introduction

Recently, many problems in computer vision have been tackled with the support of Deep Learning (DL). They take advantage of Deep Neural Networks (DNNs) capacity to interactively learn the appropriate data transformations needed to extract relevant features to accomplish the intended task [1]. When working with images, identifying relevant features can vary depending on the application, and DNNs can dismiss the need for manual selection of feature extractors or expert assistance [2]. In computer vision, areas such as autonomous driving, smart health, and environmental monitoring are a few of many that resort to DNN to build applications that aim to identify objects in an image [3]. Because DL-based computer vision is a thriving research area, and the recent abundance of cameras in urban environments, there is an insurgence of papers proposing computer vision assisted wireless communication applications [4, 5]. However, computer vision can become particularly challenging in applications that require the identification of multiple objects at low latency.

979-8-3503-2687-1/23/$31.00 ©2023 IEEE

Semantic segmentation is concerned with identifying objects in a picture, showing their contour and the area they occupy. Hence, it goes beyond simple image classification, aiming to segment the image into multiple classes, presenting the user with a detailed description of which objects are in the picture [3]. Given its relative complexity, which involves pixel-level classification together with clustering [6], many researchers are leaning towards DL to implement semantic-segmentation-based applications. Models that are variations of ResNet, SegNet, and other DNNs are now state-of-the-art for many semantic segmentation datasets [3, 7, 8]. However, many state-of-the-art models are built with multiple layers that require significant computational resources and energy consumption, hampering DNN adoption in resource-constrained devices. Moreover, even if these devices are capable of storing the DNN and running it locally, using non-state-of-the-art equipment can introduce processing latency, compromising the intended application.

Researchers are actively exploring alternatives to enable general DNN implementation in resource-constrained devices. These alternatives typically target hardware improvements and modifications to the DNN's architecture and deployment. Hardware improvements are focused on constructing more affordable devices that can run DNNs locally, whereas changes to the DNN model focuses on adding or replacing features in its architecture, or exploring deployment alternatives, such as edge computing and model quantization [9]. Among these models' changes, early exiting and DNN partitioning show promising results in addressing resource-constrained implementation issues. Early exiting consists of adding output layers (i.e., exit branches) at the early stages of the network architecture to anticipate classification without traversing the whole network [10]. DNN partitioning, on the other hand, consists of splitting the network layers so that the early stages are located at the network edge while the following ones are located at the cloud [11, 12]. Recent works have shown that combining both approaches is also viable, yielding positive results for image classification, as it simultaneously minimizes latency and resource consumption [12].

This paper proposes Branched-DeepLabV3 (BranDeep-

LabV3) to perform faster and distributed semantic segmentation. The idea is to enable the utilization of semantic segmentation in resource-constrained devices and time-sensitive applications, based on the positive results found in image classification with early exiting and DNN partitioning. We use a pre-trained DeepLabV3 [13], capable of semantic segmentation, and add side branches to perform early classification. We propose a DDN that takes advantage of the features extracted at the layers near the model's input (to be placed near the device) to complete the segmentation task, decreasing the interaction with the cloud server. Moreover, the results show that even when early classification is not an option, the proposed DNN is capable of delivering coarse segmentation that can be beneficial to latency-sensitive applications. Hence, the main contribution of this paper is to attest that semantic segmentation with an early-exit DNN is possible, and is a valuable alternative to deliver semantic segmentation to time-sensitive applications running in resource-constrained devices.

II. Related Work

Our work focuses manly in extending early-exiting DNNs for the semantic segmentation paradigm, and proposing a DNN architecture capable of performing the intended task in an edge-cloud co-inference environment. Teerapittayanon et al. showed in image classification, through their proposed BranchyNets, that features extracted in layers closer to the network's input are rich enough to enable the classification of the simpler samples of a given task, reducing the processing latency associated with traversing the whole network [10]. Pacheco et al. deployed an early-exit DNN in an edge-cloud co-inference, where layers connected to side branches are stored at the network edge, whereas the remaining layers (connected to the last exit) are stored in the cloud [12]. They showed the latency advantages of early-exit DNNs, which can make DL-enabled solutions more accessible for applications in resource-constrained devices and latency-sensitive applications. Our work builds upon both by proposing a model capable of performing early-exit semantic segmentation. Furthermore, we extend early exit utilization beyond halting image classification, showing that we can assist latency-sensitive applications by delivering crude but useful segmentation. As far as we know, this is the first paper that proposes a branched-DNN architecture for early-exiting semantic segmentation, suited for an edge-cloud co-inference.

III. Semantic Segmentation and DNNs

When compared with regular image classification, semantic segmentation involves localizing objects in an image, instead of simply determining if the object is present in it [3]. Hence, it requires the classification of each individual pixel as either belonging to a class or not. Consequently, we move from identifying if the picture contains an object from a class (e.g., a dog) to outlining the object's contour and accurately determining the region it occupies (e.g., where the dog is and which pixels belong to it).

(a) Original image. (b) Segmentation

Figure 1. Example taken from Pascal VOC 2012 [15].

By shifting from traditional image classification to pixel-level classification, using DNNs for semantic segmentation becomes more challenging than regular image classification for at least two reasons. First, it requires that feature maps (i.e., outputs of the convolutional layer) carry localized information [14]. Hence, each layer must learn, besides the features to extract during each stage of the network, where the said feature appears in the image. Additionally, the objects of interest can appear in distinct resolutions [14], meaning that the whole network should be capable of performing multiresolution analysis.

Figure 1 shows an example of semantic segmentation that has to identify and determine the area of the objects of interest within an image. For instance, if a model is trained to identify people and vehicles in the foreground, using the image seen in Figure 1(a), it must recognize and differentiate cyclists and their bicycles. Moreover, it is desirable that each instance be assigned to a class of interest, individually classifying the objects found. Hence, if designed correctly, the model produces the image seen in Figure 1(b), classifying each object even when occlusions (i.e., when one object overlaps another) occur.

A. Convolutional Layers and Computer Vision

Before introducing the DeepLabV3 network, we quickly review basic DNNs features and their relation to computer vision. The convolutional layer is a building block of many DNNs that deal with image-like data. Convolutional layers consist of a set of kernels, which are small sets of weights, that operate on a localized neighborhood within a datum [16]. Considering the input data as an image, these kernels function as filters that analyze small data portions in a sliding-window fashion to extract relevant features. Moreover, stacking convolutional layers that promote dimensionality reduction enriches the quality of the features extracted, allowing extraction at different image resolutions.

The success of convolutional layers in computer vision can also be attributed to two architectural conventions. The first is to employ multiple kernels in the same layer, which allows it to extract multiple features at the same resolution. The other convention is to stack multiple convolutional layers by connecting a layer's output to another layer's input to compose convolutional blocks. These blocks combine the

information extracted from previous layers to enable the extraction of more complex features. Additionally, it is common to use downsampling layers, such as convolutional layers with non-unitary strides or pooling layers, to increase the field of view (FoV) of the layers close to the model's output. This strategy enhances the DNN's capacity to extract features in low resolution. Even though this is proven helpful in image classification, since models become more robust to image variations, it can lead to loss of localized information, hampering the semantic segmentation task [14].

B. DeepLabV3

DeepLabV3 is a Residual Network (ResNet) [17] variation for semantic segmentation [13]. The DeepLabV3 model is obtained by removing the downsampling layers closer to the network's output in favor of layers that perform atrous (or dilated) convolution [13]. The atrous convolution enables resolution reduction by skipping neighboring samples. Thus, for each location i in an output y,

$$y[i] = \sum_k x[i + rk]w[k],$$

where x and w are the input and the convolutional filter, respectively, and r is the input sampling rate [13]. In essence, the atrous convolution allows extracting a feature using a greater FoV, acting analogously to a filter with $r - 1$ zeros between two consecutive filter values [13]. By adopting atrous convolution, the DeepLabV3 does not require consecutive dimensionality reductions to extract low-resolution features in large feature maps, thus preserving the localized information.

The atrous convolution is inserted in DeepLabV3 through a structure called Atrous Spatial Pyramid Pooling (ASPP). In it, four atrous convolutions with distinct rates are applied in parallel to an input feature map [13]. This enables a single DNN stage to extract information at multiple resolutions, which can be complementary. Moreover, it is known that employing a single atrous rate can be harmful to the overall DNN, as this can lead to loss of information [8].

IV. Early Exit and DNN Partitioning

Many state-of-the-art models in DL are composed of numerous DNN layers. These layers may be composed of a great number of neurons that connect to one another or, in the case of convolutional layers, may have many kernels to extract and fuse distinct features. This poses a challenge when deploying these models in resource-constrained devices or latency-sensitive applications. Hence, extensive research is being made into alternatives to mitigate these obstacles. Among them, our work leans on two recent proposals: early exiting and DNN partitioning.

Usually, a DNN with a substantial number of layers and other resources is needed to address complex tasks. However, in some cases, this depth is unnecessary to deal with the simpler samples of a given task. For instance, when dealing with images with elements that are easy to recognize, features extracted at early stages can be enough to conduct the intended task [10]. In these cases, data can exit the DNN before reaching the model's original output, making the additional processing of the subsequent layers unnecessary. The early-exit DNN is constructed over the architecture of a regular neural network (which we will call base DNN) by inserting additional exits (side branches) into its intermediate layers [10]. These exits perform early attempts on classification, using information extracted in the layers that precede it. If the side branch is capable of delivering a classification with a satisfactory level of confidence (e.g., above a certain threshold) the task can be finalized without using the whole base DNN.

Figure 2 shows an example of an early-exit DNN, stored at the edge. The side branches (in dark cyan) sticking out of the base DNN (grayish blocks) are responsible for the early-exiting attempts. In latency sensitive applications, these side branches can deliver an early guess of the correct answer that can be aggregated with further information, or used as the actual DNN's answer. Additionally, we can trace a smaller DNN following the data flow from the model's input layer to each early exit, if we ignore the layers that come after it. This is a feature that can be exploited by DNN partitioning.

The idea behind DNN partitioning is splitting the DNN among the cloud and instances close to the device, relying on low-cost local computations and edge computing. As can be seen in Figure 2, the early-exit DNN is split among the edge and the cloud, allowing for the early-exit estimates to be performed closer to the device, whilst the remaining layers are stored in the cloud (represented by the computational block v_{n+1}). With this approach, it is possible to minimize the classification latency by exploiting not only the early-exit DNN architecture but also the first layers' proximity to the device. As pointed out by Pacheco et al., the data transfer to the server, which is time-demanding, will happen only when the branches fail to perform an early classification [12]. In this approach, the device resorts to the complex resource-demanding layers stored in the cloud only when necessary. Hence, the iteration with the cloud can be reduced, minimizing energy expenses and communication latency. Moreover, it no longer transfers the raw data to the cloud, which is also time and energetic demanding, shifting to sending the feature maps of the layer that precedes the split (in Figure 2, this would be node v_n).

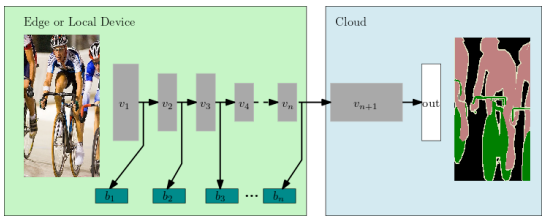


Figure 2. Sketch of the proposed BranDeepLabV3. The convolutional blocks of the base DNN (in gray) width reflects the number of layers.

V. BRANCHED-DEEPLABV3

We propose an early-exit DNN implementation that is suited for semantic segmentation. Much like in the case with the models constructed for image classification, the idea is to exploit the features extracted in early layers to classify pixels as belonging to a class, allowing simpler data to exit the DNN earlier. Additionally, our proposed DNN can deliver coarse segmentation that can be helpful to time-sensitive applications. For instance, even if an early exit is unable to outline an object in the image, it can deliver a "blob" that indicates that there is something and where in the picture it is. Thus, it can enable time-sensitive applications to identify objects faster than if it waited for the whole DNN processing, like obstacles in the line of sight of antennas [5], or vulnerable road users in autonomous driving.

The Branched-DeepLabV3 (BranDeepLabV3) is constructed on top of a pretrained DeepLabV3, available with *torchvision* module, which will function as the backbone of our proposed model. Specifically, it consists of a modified ResNet-101, in which we insert side branches that are equally spaced from one another in terms of floating-point operations (FLOPs). By measuring FLOPs we can have a rough estimate of both energy consumption and inference time [18]. So, if we can reduce FLOPs, we can expect these two requirements to decrease. The inserted layers are similar in structure to the output of the regular DeepLabV3 network, meaning that we also employ ASPP layers in the side branches to generate the early attempts on semantic segmentation. Hence, even at higher resolutions, we can take advantage of the atrous convolution to extract the needed low-resolution features. However, layer positioning is not as straightforward as in the typical image classification because the side branches impact the intermediate feature maps. For instance, in early experiments, it was observed a significant loss in performance at DeepLabV3's output layer when we made its parameters to be equal to the pretrained model. To this, we can attribute the fact that the branches modify the layers that preceded it to ease the features' localization, thus impacting the feature maps of the remainder of the backbone DNN.

Once trained, the resulting model can be split among local device, edge and cloud. Figure 2 brings a sketch of the proposed model, showing layer disposition. Branches can be inserted after any layer in the base DNN, even at higher resolutions (after blocks v_1 to v_3). Here, we take advantage of the ASPP layer that is suitable to extract low-resolution features at high resolutions. Considering that no branch is stored in the cloud, which will store the remaining layers leading to the model's output (see Figure 2), branches can be split between the local device and the edge based on the requirements of applications and devices. For example, when dealing with resource-constrained devices we can add more branches in the edge, whereas we can opt to place more branches in local devices when dealing with time sensitive applications. Additionally, in order to reduce the number of FLOPs of the overall model, some layers can be discarded after the training is complete. As the results will show, layers that are close together tend to have similar performance, hence can be considered redundant in our proposed implementation.

VI. EXPERIMENTAL SETUP

Our experiments are performed using the *Pytorch* framework and are available at `GitHub`[1] . The models were trained using the PASCAL VOC 2012 semantic segmentation dataset [15]. This dataset was compiled for a computer vision competition, in which many DNN-based solutions are state-of-the-art. It consists of $2,913$ colored images with $6,929$ identifiable objects. These images contain 20 distinct labeled classes, such as people, some groups of animals, vehicles, and indoor objects. The dataset administrators split all training images into two sets containing half the images. We used one of these sets as the training set, whereas the other was split into 60% and 40% for testing and validation sets, respectively. This gives a $50:20:30$ division of samples for training, validation, and test, respectively.

All images are scaled to have a size equal to 256×256. To increase the training sample diversity, each training image can be modified in a manner that its brightness, saturation, contrast, and hue can be altered. This modification is applied solely to the original image, and the model has to learn how to deal with this corruption in order to generate the correct segmentation. This is advantageous because we generate new data that wasn't present in the original dataset, and preserve the main images' features that we want the model to learn. During each training iteration, the samples are grouped in batches of 10 image samples that are fed into the model's input layer.

We use the Lovász-Softmax loss function to train our models, which is an extension of the regular mean-insertion-over-union (mIoU) [19]. mIoU is the metric of choice of most semantic segmentation papers and is computed by

$$mIoU(Y, \hat{Y}) = \frac{1}{|C|} \sum_{c \in C} \frac{Y_c \cap \hat{Y}_c}{Y_c \cup \hat{Y}_c} \tag{1}$$

$$= \frac{1}{|C|} \sum_{c \in C} \frac{TP_c}{TP_c + FP_c + FN_c}, \tag{2}$$

where Y_c and $\hat{Y}_c$ are the images containing the ground-truth and guesses of class c (TP and FP are the true and false positives, FN is the false negative). Essentially, it averages out the IoU of each class, meaning that it penalizes models that are "specialists" in a few classes. Additionally, we employ Stochastic Gradient Descent and follow the same learning rate scheduling defined in Berman et al. [19]. After preliminary analysis, we've opted to set $lr_{base} = 2.5 \cdot 10^{-4}$ for the branches, $lr_{base} = 2.5 \cdot 10^{-6}$ for the base DNN, and $lr_{base} = 2.75 \cdot 10^{-4}$ for the original output layer. After each epoch, the trained model is evaluated with the validation set, and the best weight configuration, i.e., the one that

[1]https://github.com/MateusGilbert/brandeeplabv3

obtained the best results in the validation set, is saved as the final model. In our experiments, we evaluate the DNNs by computing the average of the mIoU for each exit, which includes all side branches and the DeepV3Lab original output layer.

VII. Experimental Results

In our experiments, we start with a pre-trained DeepLabV3 with $mIoU = 0.707$ in the test set, and analyse configurations with 3, 5, 7 and 9 side branches. We explore how this insertion impacts the quality of the trained branches and how they impact the DeepLabV3 original output performance, given the problems exposed in Section V. Tables I to IV show the mIoU (Equation 2) of each early exit and the number of FLOPs needed to reach each output. FLOPs were computed using a 256×256 RGB picture, and we ignore the FLOP computation of the branches that preceded it. We have opted to do so because, as discussed in Section V, we assume that some branches can be discarded after training.

As anticipated, it is observable a progressive improvement of mIoU as we move toward the BranDeepLabV3 output. Furthermore, two additional conclusions can be drawn from these results. The first, regardless of the model's number of branches, exits whose positions are close in terms of FLOPs tend to have the same mIoU. For instance, $b1$ in Table I and $b2$ in Table III (implementations with 3 and 7 side branches, respectively) have nearly identical mIoU with the same FLOPs. This suggests that the branch position influences the maximum mIoU achievable. However, having more branches means that a designer has more freedom to choose which exits to discard when deploying the trained model. Additionally, mIoU averages IoU across all classes, meaning that it is possible to train these branches to specialize in easier or task-sensitive classes (e.g., identifying people on autonomous driving). The last result is that the more branches we insert to the base model, the better the performance of the original output and of the branches closer to it. For instance, the original output (*out* in the tables) has a progressive improvement as the number of early exits increases. Moreover, this performance improvement is more pronounced with the branch inserted at around 51.44 FLOPs, which starts with a mIoU score of 0.662 in the model with 3 early exits, but can reach an mIoU of 0.674 in a model with 9 inserted branches. As was discussed in Section III, each early exit needs the feature maps that enter it to carry positional information. Hence, during the training process, modifications it promotes on the layers that precede it can impact the next branches (and the base model's output) negatively. The results suggest that adjacent exits can help one another during training. However, further investigation is needed to reach a definitive conclusion.

Table I
PERFORMANCE OF BRANDEEPLABV3 WITH 3 SIDE BRANCHES

	b1	b2	b3	out
mIoU	0.255	0.527	0.662	0.701
FLOPs (G)	20.49	30.83	51.44	60.60

Table II
PERFORMANCE OF BRANDEEPLABV3 WITH 5 SIDE BRANCHES

	b1	b2	b3	b4	b5	out
mIoU	0.201	0.340	0.528	0.666	0.686	0.703
FLOPs (G)	17.04	23.94	30.83	51.44	56.02	60.60

Table III
PERFORMANCE OF BRANDEEPLABV3 WITH 7 SIDE BRANCHES

	b1	b2	b3	b4
mIoU	0.193	0.259	0.420	0.528
FLOPs (G)	14.74	20.49	26.24	30.83

	b5	b6	b7	out
mIoU	0.631	0.674	0.689	0.705
FLOPs (G)	36.58	51.44	56.02	60.60

Table IV
PERFORMANCE OF BRANDEEPLABV3 WITH 9 SIDE BRANCHES

	b1	b2	b3	b4	b5
mIoU	0.176	0.206	0.290	0.398	0.514
FLOPs (G)	13.59	17.04	21.64	25.09	29.68

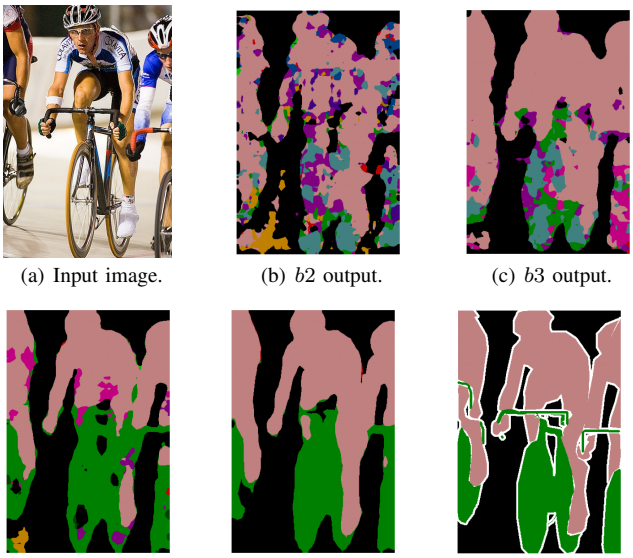
	b6	b7	b8	b9	out
mIoU	0.562	0.655	0.674	0.688	0.707
FLOPs (G)	33.13	37.73	51.44	56.02	60.60

Figure 3 shows some output images from BranDeepLabV3 with 7 side branches, along with the ground truth (Figure 3(f)). Even though the segmentations in the early stages are crude, for some applications they can be detailed enough. For instance, in autonomous driving, $b2$ or $b3$ outputs (Figures3(b) and 3(c)) might be good enough to identify a cyclist, enabling the agent to take a quick response. Hence, even if it would be beneficial to have a finer segmentation (produced at a cloud, for example), the cruder early exit can be used in latency-sensitive applications to initiate an action. Additionally, it is possible to fine-tune the early exit to identify sensitive classes. Again, returning to the autonomous driving example, to minimize collisions with humans, we can perform training that encourages the side branches to identify vulnerable road users. Differently, in other applications, early exists can be trained to differentiate end users from possible obstacles, assisting time-sensitive proactive applications in computer-vision assisted communication [5].

Figure 4 brings the relationship between image size and the number of FLOPs needed to reach each exit of the 7-branched BranDeepLabV3, showing in black the number of FLOPs needed to produce each exit in Figure 3. Images can appear in many shapes, meaning the number of computations and expended energy can vary a lot from image to image. It is noticeable that the size of the image is an issue that can worsen both latency response and energy consumption, as the later layers are the ones where the increase of FLOPs is more pronounced. Thus, it is evident that early exits are a valuable tool for time-sensitive and resource-constrained implementations.

VIII. Conclusion

Given the success of early-exit DNNs in image classification, it is only natural to try to replicate it in semantic



(a) Input image. (b) $b2$ output. (c) $b3$ output.

(d) $b5$ output. (e) *out* output. (f) Ground truth.

Figure 3. Segmented images from BranDeepLabV3 with 7 side branches.

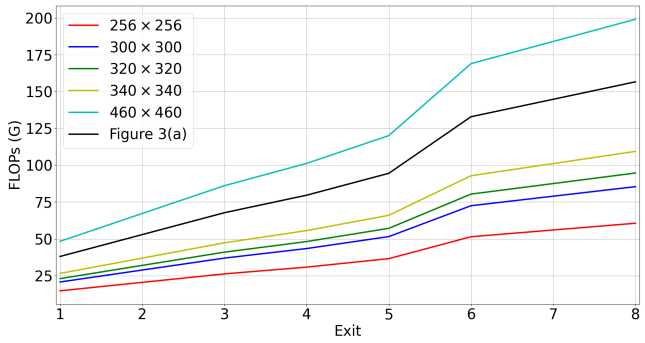


Figure 4. Number of FLOPs of each exit of the BranDeepLabV3 with 7 early exits.

segmentation. Additionally, many time-sensitive applications that require semantic segmentation can benefit from early attempts of segmentation. In this paper, we present a new implementation suited for early-exit semantic segmentation, showing the usefulness of the approach. The results show that the earlier exits' FLOPs, which allows us to estimate latency and energy consumption, increase at a slower pace than the later stages, and that they are capable of delivering coarse segmentations that can outline relevant features. For instance, we show that in earlier stages, where mIoU is below 0.4, the early exits are capable of delivering a segmentation displaying relevant features. This result, in particular, indicates an opportunity of training the early exits to distinguish sensitive classes, to be investigated in future research.

Acknowledgment

This work was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. It was also supported by CNPq, FAPERJ Grants E-26/211.144/2019 and E-26/202.689/2018, and FAPESP Grant 15/24494-8.

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[2] N. O'Mahony, S. Campbell, A. Carvalho, S. Harapanahalli, G. V. Hernandez, L. Krpalkova, D. Riordan, and J. Walsh, "Deep learning vs. traditional computer vision," in *Advances in Computer Vision: Proceedings of the 2019 Computer Vision Conference (CVC), Volume 1 1*. Springer, 2020, pp. 128–144.

[3] A. Garcia-Garcia, S. Orts-Escolano, S. Oprea, V. Villena-Martinez, P. Martinez-Gonzalez, and J. Garcia-Rodriguez, "A survey on deep learning techniques for image and video semantic segmentation," *Applied Soft Computing*, vol. 70, pp. 41–65, 2018.

[4] V. M. De Pinho, M. L. R. De Campos, L. U. Garcia, and D. Popescu, "Vision-aided radio: User identity match in radio and video domains using machine learning," *IEEE Access*, vol. 8, pp. 209 619–209 629, 2020.

[5] T. Nishio, Y. Koda, J. Park, M. Bennis, and K. Doppler, "When wireless communications meet computer vision in beyond 5g," *IEEE Communications Standards Magazine*, vol. 5, no. 2, pp. 76–83, 2021.

[6] M. Thoma, "A survey of semantic segmentation," *arXiv preprint arXiv:1602.06541*, 2016.

[7] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 801–818.

[8] P. Wang, P. Chen, Y. Yuan, D. Liu, Z. Huang, X. Hou, and G. Cottrell, "Understanding convolution for semantic segmentation," in *2018 IEEE winter conference on applications of computer vision (WACV)*. Ieee, 2018, pp. 1451–1460.

[9] M. Mohammadi, A. Al-Fuqaha, S. Sorour, and M. Guizani, "Deep learning for iot big data and streaming analytics: A survey," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 2923–2960, 2018.

[10] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," in *2016 23rd international conference on pattern recognition (ICPR)*. IEEE, 2016, pp. 2464–2469.

[11] R. G. Pacheco and R. S. Couto, "Inference time optimization using branchynet partitioning," in *2020 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2020, pp. 1–6.

[12] R. G. Pacheco, K. Bochie, M. S. Gilbert, R. S. Couto, and M. E. M. Campista, "Towards edge computing using early-exit convolutional neural networks," *Information*, vol. 12, no. 10, p. 431, 2021.

[13] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation," 2017.

[14] L.-C. 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 transactions on pattern analysis and machine intelligence*, vol. 40, no. 4, pp. 834–848, 2017.

[15] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results," http://host.robots.ox.ac.uk/pascal/VOC/voc2012/.

[16] Y. LeCun, Y. Bengio *et al.*, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.

[17] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[18] S. Laskaridis, S. I. Venieris, M. Almeida, I. Leontiadis, and N. D. Lane, "Spinn: synergistic progressive inference of neural networks over device and cloud," in *Proceedings of the 26th annual international conference on mobile computing and networking*, 2020, pp. 1–15.

[19] M. Berman, A. R. Triki, and M. B. Blaschko, "The lovász-softmax loss: A tractable surrogate for the optimization of the intersection-over-union measure in neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4413–4421.