TRIDENT PYRAMID NETWORKS: THE IMPORTANCE OF PROCESSING AT THE FEATURE PYRAMID LEVEL FOR BETTER OBJECT DETECTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Feature pyramids have become ubiquitous in multi-scale computer vision tasks such as object detection. Based on their importance, we divide a computer vision network into three parts: a backbone (generating a feature pyramid), a core (refining the feature pyramid) and a head (generating the final output). Most existing networks operating on feature pyramids, named cores, are shallow and mostly focus on communication-based processing in the form of top-down and bottom-up operations. We present a new core architecture called Trident Pyramid Network (TPN), that allows for a deeper design and for a better balance between communication-based processing and self-processing. We show consistent improvements when using our TPN core on the COCO object detection benchmark, outperforming the popular BiFPN baseline by 1.5 AP. Additionally, we show that it is more beneficial to put additional computation into the TPN core, rather than into the backbone, by outperforming a ResNet-101+FPN baseline with our ResNet-50+TPN network by 1.7 AP, while operating under similar computation budgets. This emphasizes the importance of performing computation at the feature pyramid level in modern-day object detection systems. Code will be released.

1 INTRODUCTION

Many computer vision tasks such as object detection and instance segmentation require strong features both at low and high resolution to detect both large and small objects respectively. This is in contrast to the image classification task where low resolution features are sufficient as usually only a single object is present in the center of the image. Networks developed specifically for the image classification task (*e.g.* Simonyan & Zisserman (2014); He et al. (2016a); Xie et al. (2017)), further denoted by *backbones*, are therefore insufficient for multi-scale vision tasks. Especially poor performance is to be expected on small objects, as shown in Lin et al. (2017a).

In order to alleviate this problem, named the *feature fusion problem*, top-down mechanisms are added Lin et al. (2017a) to propagate semantically strong information from the low resolution to the high resolution feature maps, with improved performance on small objects as a result. Additionally, bottom-up mechanisms can also be appended Liu et al. (2018) such that the lower resolution maps can benefit from the freshly updated higher resolution maps. These top-down and bottom-up mechanisms can now be grouped into a layer, after which multiple of these layers can be concatenated, as done in Tan et al. (2020). We call this part of a computer vision network the *core*, laying in between the *backbone* and the task-specific *head* (see Figure 1). In general, we define a core module to be any module taking as input a feature pyramid and outputting an updated feature pyramid.

These top-down and bottom-up operations can be regarded as *communication*-based processing operating on two feature maps, as opposed to *content*-based self-processing operating on a single feature map. Existing cores such as FPN Lin et al. (2017a), PANet Liu et al. (2018) and BiFPN Tan et al. (2020) mostly focus on communication-based processing, as this nicely supplements the backbone merely consisting of self-processing. However, too much communication can lead to the core talking in circles, with diminished efficiency as a result. We hence argue that within the core *communication*-based processing should be alternated with some healthy portion of self-processing.

Figure 1: High-level view of a computer vision network. The backbone (*left*) processes the image to output a set of feature maps (*i.e.* a feature pyramid). The core (*middle*) takes in a feature pyramid (denoted by FP) and returns an updated feature pyramid. Finally, the head (*right*) produces the loss during training and makes predictions during inference from the final feature pyramid. In this work, we focus on improving the core.

Based on this observation, we design the Trident Pyramid Network (TPN) core consisting of sequential top-down and bottom-up operations alternated with parallel self-processing mechanisms. The TPN core is equipped with hyperparameters controlling the amount of communication-based processing and self-processing. During the experiments, we empirically investigate what the optimal balance is between communication-based processing and self-processing (see Section 4).

The TPN core is compared to various baselines on the COCO object detection benchmark Lin et al. (2014). Specific care is taken to ensure the baselines have similar computational characteristics, such that a fair comparison can be made. Using a ResNet-50 backbone and a simple one-stage detector head, our TPN core peaks at 41.8 AP on the COCO validation set when using the 3x training schedule (see Section 4). Additionally, we empirically show that it is more beneficial to put additional computation into the TPN core, rather than into the backbone or the detection head. This shows the importance of core modules within object detection systems.

2 Related work

In order to obtain multi-scale features, early detectors performed predictions on feature maps directly coming from the backbone, such as MS-CNN Cai et al. (2016) and SSD Liu et al. (2016). As the higher resolution maps from the backbone contain relatively weak semantic information, top-down mechanisms were added to propagate semantically strong information from lower resolution maps back to the higher resolution maps as in FPN Lin et al. (2017a) and TDM Shrivastava et al. (2016). Since, many variants and additions have been proposed: PANet Liu et al. (2018) appends bottom-up connections, M2det Zhao et al. (2019) uses a U-shape feature interaction architecture, ZigZagNet Lin et al. (2019) adds additional pathways between different levels of the top-down and bottom-up hierarchies, NAS-FPN Ghiasi et al. (2019) uses Neural Architecture Search (NAS) to automatically design a feature interaction topology, and BiFPN Tan et al. (2020) modifies PANet by removing some connections, adding skip connections and using weighted feature map aggregation. All of the above variants focus on improving the communication between the different feature maps. We argue however that to be effective, extra content-based self-processing is needed in between the communication flow.

3 Method

3.1 TPN core architecture

Generally speaking, the core receives a feature pyramid as input, and outputs an updated feature pyramid. Here, a feature pyramid is defined as a collection of feature maps, with feature maps defined as a collection of feature vectors (called features) organized in a two-dimensional map. More specifically, feature map P_l denotes a feature map of level l which is 2^l times smaller in width and height compared to the initial image resolution. A popular choice for the feature pyramid Lin et al. (2017b) is to consider feature maps $\{P_3, P_4, P_5, P_6, P_7\}$, which we will use as the default setting throughout our discussions and experiments.

The core is constructed from three building blocks: top-down operations, self-processing operations and bottom-up operations (see Figure 2). In this subsection, we focus on how these operations are best combined, independently of their precise implementations. We call this configuration of operations making up a core, the *core architecture*. The specific implementations corresponding to the top-down, self-processing and bottom-up operations will be discussed in following two subsections.

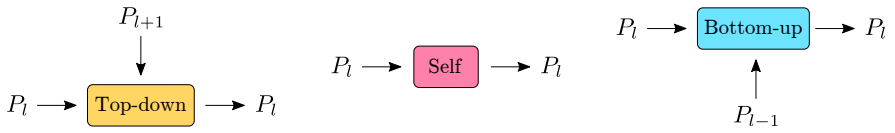

Figure 2: Collection of building blocks for core architecture design. Here P_l denotes feature map of level l which is 2^l times smaller compared to the initial image resolution. (*Left*) General top-down operation updating feature map P_l with information from lower resolution map P_{l+1}. (*Middle*) General self-processing operation updating feature map P_l with information from itself, *i.e.* from feature map P_l. (*Right*) General bottom-up operation updating feature map P_l with information from higher resolution map P_{l-1}.

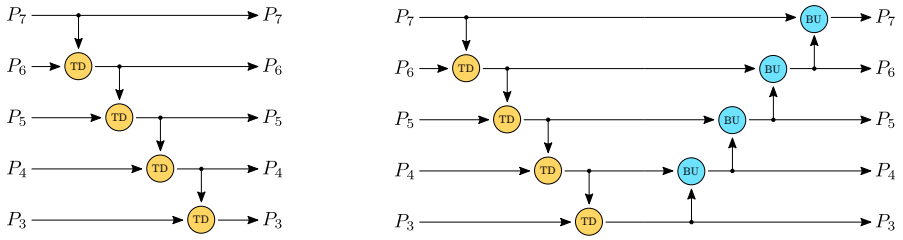

Figure 3: (*Left*) A layer from the FPN core architecture. (*Right*) A layer from the PANet core architecture.

Using these general building blocks, we can recreate the popular FPN Lin et al. (2017a) and PANet Liu et al. (2018) core architectures in Figure 3. Note that the architectures slightly differ from those found in the original works Lin et al. (2017a) and Liu et al. (2018), as the input and output projection layers are missing. These input and output layers are meant to respectively transition from backbone feature sizes to core feature sizes, and from core feature sizes to head feature sizes. We decided to move these transition layers from the core to the backbone and head instead, such that multiple core layers can easily be concatenated. Note moreover that Figure 3 only defines the architecture of the core, without specifying the implementation of the top-down and bottom-up operations. These implementations could hence differ from those found in the original works Lin et al. (2017a) and Liu et al. (2018).

From the FPN and PANet core architectures from Figure 3, we make following two observations. First, we can see that the top-down and bottom-up operations are sequential. Secondly, we observe the absence of any self-processing operations in both core architectures. In what follows, we discuss both aspects in more depth.

First, we discuss the trade-off between sequential and parallel operations in greater detail. By *sequential* operations we mean that P_l is updated with the new $P_{l\pm1}$ instead of with the old one, forcing the operations to be performed sequentially as the new feature maps must be available. Alternatively, one could instead opt for *parallel* operations by solely relying on the old feature maps. The choice between parallel and sequential could be regarded as a trade-off between speed and accuracy, while maintaining a similar memory consumption. Given that the top-down and bottom-up operations can be quite expensive, especially on high resolution maps, it is important to get the most out of every single operation. We hence believe that the sequential variant should be preferred here for the top-down and bottom-up operations, as found in the FPN and PANet core architectures.

Secondly, we discuss the lack of self-processing operations in the FPN and PANet core architectures. When looking at the PANet architecture, we see that bottom-up operations immediately follow the top-down operations. We argue that this is sub-optimal. Take for example a look at the P_4-P_3 top-down operation, followed immediately by the P_3-P_4 bottom-up operation. The P_3 map was just updated with information from P_4 and now P_3 must immediately communicate its content back to P_4 before having the possibility to digest and work on this new information. We hence argue that the top-down and bottom-up operations should be separated with self-processing operations. This gives the feature maps the opportunity to work on themselves before communicating back to their peers.

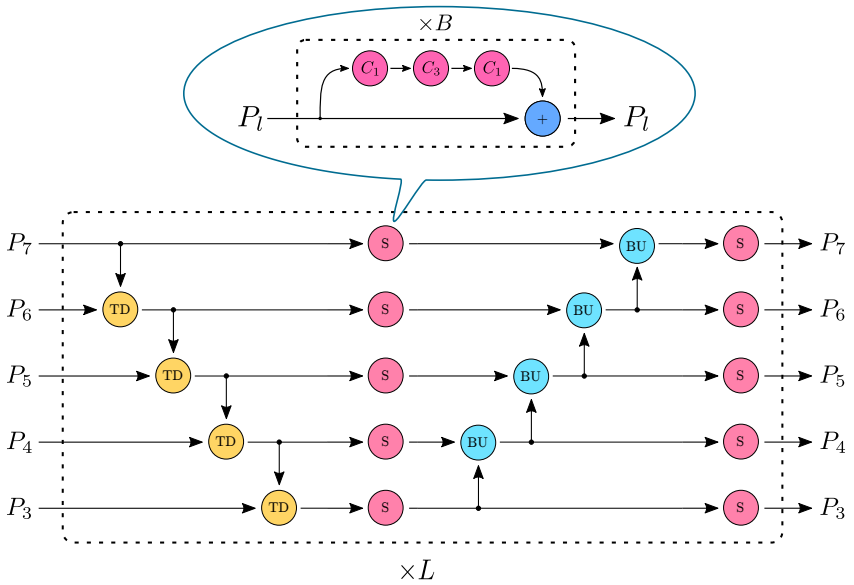

Figure 4: Our TPN core architecture consisting of L consecutive TPN core layers (*bottom*), with each self-processing operation consisting of B consecutive bottleneck layers (*top*).

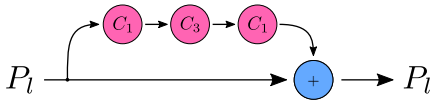

Figure 5: Bottleneck layer used as base self-processing operation. It is a skip-connection operation with a residual branch consisting of three convolution operations: a convolution operation of kernel size 1 reducing the original feature size to the hidden feature size, a content convolution operation of kernel size 3 applied on the hidden feature size, and finally a convolution operation of kernel size 1 expanding the hidden feature size back to the original feature size. Note that each convolution operation (*i.e.* pink convolution node) consists of the actual convolution preceded He et al. (2016b) by group normalization Wu & He (2018) and a ReLU activation function.

By combining the insights from previous two discussions, we arrive at the Trident Pyramid Network (TPN) core architecture consisting of sequential top-down and bottom-up operations alternated with self-processing operations (see lower part of Figure 4). The name is inspired by the top-down, first self-processing and bottom-up operations resembling a trident. Note that the TPN self-processing operations happen in parallel. This is not necessary, but we believe this to be the most natural choice.

3.2 Communication vs. content

When looking at the TPN core in Figure 4, we have on the one hand communication-based top-down and bottom-up operations, and on the other hand content-based self-processing operations. One might now wonder what the optimal balance between communication and content is. In order to investigate this matter, we introduce hyperparameters enabling us to control the amount of content-based processing on the one hand, and the amount of communication-based processing on the other hand.

First, we take a closer look at the self-processing operation. In general, we consider this self-processing operation to be a sequence of layers, with each layer containing the base self-processing operation. In this paper, we chose the bottleneck architecture from He et al. (2016a) (see Figure 5) as the base self-processing operation. The number of bottleneck layers B per self-processing operation then determines the amount of content-based processing within the TPN core.

Secondly, with each TPN layer consisting of one top-down and one bottom-up sequence, the number of TPN core layers L determines the amount of communication-processing within the TPN core. In

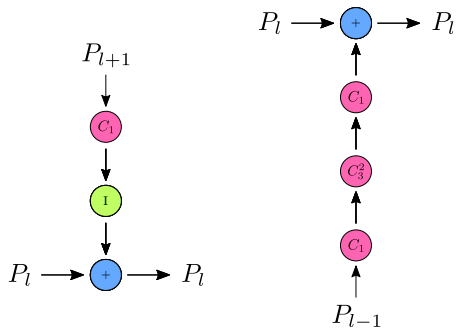

Figure 6: Implementation of the top-down (*left*) and bottom-up (*right*) operations. The pink convolution nodes are defined as in Figure 5, with the subscript denoting the kernel size and the superscript denoting the stride (stride 1 when omitted). The green node is an interpolation node resizing the input feature map to the required resolution by using bilinear interpolation.

Figure 4, an overview of our TPN core architecture is shown, displaying the number of bottleneck layers B and the number of TPN core layers L. By varying these hyperparameters B and L, we can hence control the balance between content-based processing and communication-based processing respectively. In Subsection 4.2, we empirically find out which combination works best.

3.3 Top-down and bottom-up operations

Let us now take a closer look at the top-down and bottom-up operations. Generally speaking, these operations update a feature map based on a second feature map, either having a lower resolution (top-down case) or a higher resolution (bottom-up case). Our implementation of the top-down and bottom-up operations are shown in Figure 6. The operations consist of adding a modified version of $P_{l\pm1}$ to P_l. This is similar to traditional skip-connection operations, with the exception that the *residual features* originate from a different feature map. The residual branch of the top-down operation consists of a linear projection followed by bilinear interpolation. The presence of the linear projection is important here, as it makes the expectation of the residual features zero at initialization. Failing to do so can be detrimental, especially when building deeper core modules, as correlated features add up without constraints. An alternative consists in replacing the blue addition nodes with averaging nodes. This however fails to keep the skip connection computation free (due to the 0.5 factor), which is undesired He et al. (2016b). The residual branch of the bottom-up operation is similar to the bottleneck residual branch in Figure 5. Only the middle 3×3 convolution has stride 2 instead of stride 1, avoiding the need for an interpolation step later in the residual branch.

4 Experiments

4.1 Setup

Dataset. We perform our experiments on the 2017 COCO dataset Lin et al. (2014), where we train on the 118k training images and evaluate on the remaining 5k validation images.

Implementation details. Throughout our experiments, we use an ImageNet Deng et al. (2009) pretrained ResNet-50 backbone He et al. (2016a), with frozen stem, stage 1 and batchnorm layers (see Radosavovic et al. (2020) for used terminology).

Our feature pyramid consists of five feature maps, ranging from P_3 to P_7, each having feature size 256. Our initial feature pyramid is constructed based on the backbone output feature maps C_3 to C_5 from stages 2, 3 and 4 respectively. Remember that the subscript denotes how many times the feature map was downsampled with factor 2 compared to the input image. The initial P_3 to P_5 maps are obtained by applying simple linear projections on C_3 to C_5, whereas the initial P_6 and P_7 maps are obtained by applying a simple network on C_5 consisting of 2 convolutions with stride 2 with a ReLU activation in between (similar to Lin et al. (2017b)). Throughout our core modules, we use

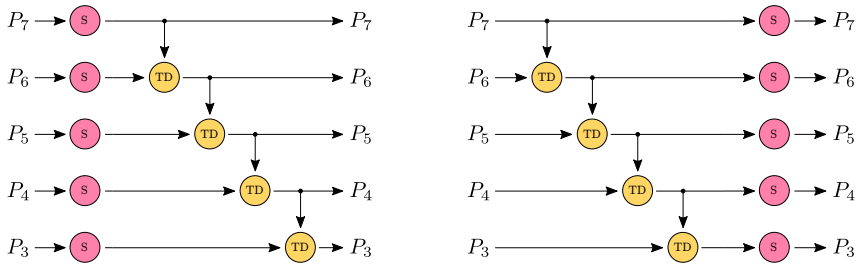

Figure 7: (*Left*) The baseline bFPN core architecture simulating a heavier backbone followed by a single FPN layer. (*Right*) The baseline hFPN core architecture simulating a single FPN layer followed by a heavier head.

group normalization Wu & He (2018) with 8 groups. For the bottleneck layers (see Figure 5), we use a hidden feature size of 64.

As detection head, we use the one-stage detector head from RetinaNet Lin et al. (2017b), with 1 hidden layer in both classification and bounding box subnets. This makes the head considerably lighter compared to the default RetinaNet head, which contains 4 hidden layers in both classification and bounding box subnets. We follow the implementation and settings from Wu et al. (2019), except that the last layer of the subnets has kernel size 1 (instead of 3) and that we normalize the losses per feature map (instead of over the whole feature pyramid).

We train our models with the AdamW optimizer Loshchilov & Hutter (2017) with weight decay 10^{-4} using an initial learning rate of 10^{-5} for the backbone parameters and an initial learning rate of 10^{-4} for the remaining model parameters. Our results are obtained by using the 3x training schedule, consisting of 36 epochs with learning rate drops after the 27th and 33rd epoch with a factor 0.1. Our models are trained on 2 GPUs with batch size 2, while using the same data augmentation scheme as in Carion et al. (2020).

4.2 TPN EXPERIMENTS

Baselines. In this subsection, we perform experiments to evaluate the TPN core. As baselines, we consider the FPN and PANet core architectures, as presented in Figure 3, equipped with top-down and bottom-up operations from Figure 6. Multiple of these FPN and PANet layers will be concatenated such that they share similar computational characteristics as the tested TPN modules. As the FPN layer was not designed to be concatenated many times, we provide two additional baselines (see Figure 7) simulating a heavier backbone and a heavier head, while using a single FPN layer. As such, we will not only be able to evaluate whether the TPN core outperforms other cores, but also whether it outperforms detection networks with a simple core but with heavier backbones or heads while operating under similar computation budgets.

Results. The experiment results evaluating four different TPN configurations against the four core baselines, are found in Table 1.

First, notice how the B and L hyperparameters (defining the TPN configuration) are chosen in order to obtain models with similar computational characteristics. Here hyperparameter B denotes the number of bottleneck layers per self-processing node, while hyperparameter L denotes the number of consecutive core layers (see Figure 4). These similar computational characteristics ensure us that a fair comparison can be made between the different core models.

Secondly, notice the relatively poor performance of the FPN baseline, especially on large objects. This can be understood from its architecture (Figure 3) by observing that the P_7 feature map does not receive any update. Moreover, we observe from its architecture that information is only flowing in a top-down fashion, making it less suitable as building block for concatenation. This again emphasizes the importance of the additional bFPN and hFPN baselines, which better use the FPN architecture as can be seen from the results in Table 1. We can even see that the bFPN and hFPN baselines outperform the PANet baseline under a similar computational budget. Note that the 33.3 AP of the FPN baseline might seem low compared to the values found in the literature (*e.g.* see RetinaNet entry

Table 1: Experiment results on the 2017 COCO validation set of different TPN cores (top four rows) and its baselines (bottom four rows). The three leftmost columns specify the core model, the middle six columns show its performance and the five rightmost columns show its computational characteristics. These characteristics were obtained on a GeForce GTX 1660 Ti GPU by computing the core output starting from backbone feature maps C_3 to C_5 with shapes $[2, 512, 128, 128]$, $[2, 1024, 64, 64]$ and $[2, 2048, 32, 32]$ respectively. The training characteristics are found under the columns 'Params', 'tFPS' and 'tMem', whereas the inference characteristics are found under the 'iFPS' and 'iMem' columns. Here the FPS metric should be interpreted as the number of times the GPU can process above input when using the specified core. Note that both forward and backward passes are used to obtain the training characteristics.

Arch	B	L	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	Params	tFPS	tMem	iFPS	iMem
TPN	7	1	41.3	60.5	44.2	26.3	45.9	52.5	11.7 M	5.0	2.13 GB	14.2	0.41 GB
TPN	3	2	41.6	60.9	44.6	**26.4**	45.8	53.2	11.6 M	5.3	1.97 GB	14.7	0.41 GB
TPN	2	3	**41.8**	61.1	44.4	26.2	46.1	**53.7**	12.1 M	5.0	2.06 GB	13.8	0.42 GB
TPN	1	5	**41.8**	**61.2**	**45.0**	26.0	**46.3**	53.4	12.5 M	4.9	1.99 GB	13.8	0.42 GB
bFPN	14	–	39.6	60.3	42.4	24.2	43.5	51.3	11.4 M	5.2	2.09 GB	14.8	0.33 GB
hFPN	14	–	40.0	60.2	43.0	25.6	43.9	51.1	11.4 M	5.2	2.08 GB	14.6	0.37 GB
FPN	–	52	33.3	56.2	34.2	20.6	38.7	41.2	20.0 M	3.8	1.97 GB	9.1	0.32 GB
PANet	–	13	38.8	59.4	40.9	24.2	43.1	50.3	13.4 M	5.2	2.06 GB	14.5	0.31 GB

Table 2: Comparison of our best performing model with other prominent methods on the 2017 COCO validation set. All models use a ResNet-50 He et al. (2016a) backbone. Here the 'Aug.' column denotes the used data augmentation scheme, either following Detectron2 (d2) Wu et al. (2019) or DETR Carion et al. (2020). As in Carion et al. (2020), the number of FLOPS are computed with the `flop_count_operators` tool from Detectron2 Wu et al. (2019), applied on the first 100 images of the validation set.

Model	Aug.	Epochs	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	Params	GFLOPS
Faster RCNN-FPN Wu et al. (2019)	d2	37	40.2	61.0	43.8	24.2	43.5	52.0	42 M	180
RetinaNet Wu et al. (2019)	d2	37	38.7	58.0	41.5	23.3	42.3	50.3	38 M	208
DETR Carion et al. (2020)	DETR	500	42.0	62.4	44.2	20.5	45.8	61.1	41 M	86
Deformable DETR Zhu et al. (2020)	DETR	50	43.8	62.6	47.7	26.4	47.1	58.0	40 M	173
TPN (**ours**)	DETR	36	41.8	61.1	44.4	26.2	46.1	53.7	37 M	121

from Table 2). This is however due to the fact that our RetinaNet head only uses a single hidden layer in both subnets, as opposed to the 4 hidden layers from the default RetinaNet implementation.

Thirdly, when comparing our TPN cores (top four rows) with the baseline cores (bottom four rows), we clearly see the superiority of the TPN core. All four TPN configurations outperform the best performing baseline (hFPN) with 1.3 up to 1.8 AP. These TPN improvements are moreover found across all object sizes. This hence shows that the TPN core not only outperforms other cores (*i.e.* FPN and PANet), but also other detection networks using heavier backbones or heads while operating under similar overall computation budgets. This highlights the importance of core modules operating on feature pyramids in general object detection networks (see Figure 1).

Fourthly, we observe that the results between the four different TPN configurations are very similar, all four obtaining between 41.3 and 41.8 AP. It appears that having more TPN core layers L is slightly more beneficial than having more bottleneck layers B under similar computational budgets. Especially on large objects we observe noticeable differences ranging from 52.5 to 53.7 AP_L. We can hence conclude that the balance between communication and content processing, should be favored towards communication-based processing. However, when taking this principle too far and only keeping communication-based processing as in PANet, worse results are to be expected, as evidenced by the inferior PANet AP of 38.8.

4.3 Comparison with other methods

In Table 2, we compare the performance of our best performing model against other prominent object detection networks. We make following two sets of observations.

First, we compare our model with the popular two-stage detector Faster RCNN-FPN Ren et al. (2015); Lin et al. (2017a) and the popular one-stage detector RetinaNet Lin et al. (2017b) (top two rows). Under similar training schedules, we observe that our method performs better across all object sizes, while having a similar amount of parameters and requiring fewer FLOPS. Note that the RetinaNet model closely resembles our model, except that it uses a FPN core and a heavier head using 4 hidden layers Wu et al. (2019) instead of 1. The results again show that it is more beneficial to put additional computation into the core (TPN) rather than into the head (RetinaNet).

Secondly, we compare our model with the DETR Carion et al. (2020) and Deformable DETR Zhu et al. (2020) models (middle two rows). These are two recent object detectors based on transformers with set prediction. On small objects, we observe similar results for our model and Deformable DETR, while DETR performs significantly worse. This can be understood from the fact that DETR only operates on a single low resolution feature map instead of on a feature pyramid, once again proving the importance of feature pyramids. On large objects on the other hand, we see a clear superiority of the transformer-based detectors with set prediction compared to our model using a simple RetinaNet-like head. Given that our TPN model only specifies the core module, we could combine it with this new type of detection head that uses a transformer decoder combined with set prediction Carion et al. (2020). We believe this should improve the performance of our TPN model for large objects in a similar way as for the DETR and Deformable DETR models. We will further investigate this in future work.

5 Conclusion

In this paper, we introduced a new type of core architecture, called the Trident Pyramid Network (TPN). We show consistent improvements when using our TPN core on the COCO object detection benchmark compared to various baselines. From our TPN experiments, we see that both communication-based processing and self-processing are crucial within core modules in order to obtain good results. We additionally observe that our TPN performance could not be matched by heavier backbones or heads under similar overall computation budgets, showing the importance and effectiveness of the core within object detection systems.

References

Zhaowei Cai, Quanfu Fan, Rogerio S Feris, and Nuno Vasconcelos. A unified multi-scale deep convolutional neural network for fast object detection. In *European conference on computer vision*, pp. 354–370. Springer, 2016.

Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European Conference on Computer Vision*, pp. 213–229. Springer, 2020.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. Nas-fpn: Learning scalable feature pyramid architecture for object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7036–7045, 2019.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016a.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pp. 630–645. Springer, 2016b.

Di Lin, Dingguo Shen, Siting Shen, Yuanfeng Ji, Dani Lischinski, Daniel Cohen-Or, and Hui Huang. Zigzagnet: Fusing top-down and bottom-up context for object segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7490–7499, 2019.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pp. 740–755. Springer, 2014.

Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2117–2125, 2017a.

Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pp. 2980–2988, 2017b.

Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8759–8768, 2018.

Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pp. 21–37. Springer, 2016.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10428–10436, 2020.

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *arXiv preprint arXiv:1506.01497*, 2015.

Abhinav Shrivastava, Rahul Sukthankar, Jitendra Malik, and Abhinav Gupta. Beyond skip connections: Top-down modulation for object detection. *arXiv preprint arXiv:1612.06851*, 2016.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Mingxing Tan, Ruoming Pang, and Quoc V Le. Efficientdet: Scalable and efficient object detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10781–10790, 2020.

Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 3–19, 2018.

Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. `https://github.com/facebookresearch/detectron2`, 2019.

Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1492–1500, 2017.

Qijie Zhao, Tao Sheng, Yongtao Wang, Zhi Tang, Ying Chen, Ling Cai, and Haibin Ling. M2det: A single-shot object detector based on multi-level feature pyramid network. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pp. 9259–9266, 2019.

Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable detr: Deformable transformers for end-to-end object detection. *arXiv preprint arXiv:2010.04159*, 2020.