Image Panoptic Segmentation Challenges using COCO Dataset
Project Report : CS 7643

Anurag Sen, Jie Song, Zhiling Zhou, Zixiang (David) Zhu
Georgia Institute of Technology
North Ave NW, Atlanta, GA 30332, United States
(asen48,jsong407,zhiling_zhou,zzhu72)@gatech.

Abstract

In this project, we present using various deep learning approaches to perform image panoptic segmentation on MS-COCO dataset. Specifically, we explored both a bottom-up approach (Panoptic DeepLab) and a top-down approach (Panoptic FPN) for image panoptic segmentation tasks. We experimented with various modifications to the existing state-of-the-art models and discovered promising results.

1. Background

Traditionally, there are two main categories of image segmentation tasks that are widely adopted by many applications: image instance segmentation [**?**] and image semantic segmentation [**?**]. Instance segmentation detects and delineates each distinct object of interest appearing in an image, while semantic segmentation assigns each pixel in an image a class label that it represents. In recent years, image panoptic segmentation has emerged as a new category of image segmentation task that combines both instance segmentation and semantic segmentation [**?**]. Specifically, panoptic segmentation generates an output that not only tells which pixels represent each individual object of interest (things) but also assigns a class label to the background pixels (stuff) (Figure 1). Panoptic segmentation has gained greater popularity in the past few years. One application that heavily leverages panoptic segmentation is self-driving cars, where the software needs to distinguish objects (persons, other cars, traffic lights, etc.) from surroundings (road, pavement, sky, etc.) in camera images [**?**]. Another application of panoptic segmentation is in medical imaging of tumor cells, where both the tumor instances and the amorphous regions help shape the context of the disease [**?**, **?**].

Figure 1: Types of image segmentation tasks

Challenges with Panoptic Segmentation It has been widely recognized that there are two main challenges in image panoptic segmentation when using Deep Convolutional Neural Network: 1) reduced feature resolution caused by consecutive pooling operations or convolution striding, and 2) the existence of objects at multiple scales in image.

Panoptic FPN is a panoptic segmentation model that uses feature pyramid network (FPN) as the backbone to perform both instance segmentation and semantic segmentation. Panoptic FPN is a proposal-based architecture, because it performs ROI pooling on top of FPN which are then used by Mask-RCNN for instance segmentation. Panoptic FPN addresses both aforementioned challenges with its FPN architecture, which can be trained to produce high-resolution, rich, multi-scale features for both instance and semantic segmentation branches. The architecture of Panoptic FPN is studied in more detail in Section 4.

Panoptic DeepLab is a proposal-free architecture that directly predicts object center and offsets from feature maps. Panoptic DeepLab addressed the aforementioned two challenges with its Atrous Spatial Pyramid Pooling (ASPP) filters. ASPP and Panoptic DeepLab architecture are discussed in more detail in Section 3.

2. Dataset, Library, and Framework

Dataset We used COCO 2017 image segmentation dataset to train and test our panoptic segmentation models [**?**]. COCO is a large-scale object detection, segmentation, and captioning dataset. The dataset consists of 118k training images and 5k validation images. There are a total of 133 classes that were annotated for panoptic segmentation tasks.

Library and Framework We used Detectron2 as the main library to build, train, and test our panoptic segmentation models. Detectron2 is a Pytorch-based library that provides state-of-the-art detection and segmentation algorithms, including Panoptic FPN and Panoptic DeepLab methods for panoptic segmentation [**?**]. In this project, we made our own modifications to these algorithms and evaluated their performance on the COCO dataset.

3. Panoptic DeepLab

Panoptic DeepLab is a single-shot, bottom-up panoptic segmentation method that adopts dual-ASPP and dual-decoder structures for semantic and instance segmentation. The semantic segmentation branch closely resembles to DeepLabV3 (an architecture dedicated for semantic segmentation), while the instance segmentation branch performs class-agnostic object detection through an object center heatmap head and an object center offset head.

Inspired from DeepLabV3, Panoptic DeepLab uses Atrous Spatial Pyramid Pooling (ASPP) to resample feature maps at multiple rates prior to decoder convolutions. Specifically, ASPP is able to probe the original feature map with multiple filters that have complementary fields of view, through which objects as well as their contexts are captured at multiple scales.

Panoptic DeepLab uses ResNet as its backbone encoder (feature extractor). The output of the last stage of ResNet is fed into two ASPP layers in parallel, which are used to extract denser feature maps at different scales. From there the output of one ASPP is fed into a semantic segmentation decoder, while the output of the other ASPP is fed into an instance segmentation decoder. Feature maps in both decoders undergo an upsampling process, after which the outputs of both decoders are fused together into one segmentation output.

Parts of Panoptic DeepLab with learned parameters are: ResNet encoder (convolutional layers), ASPP heads (layers of ASPP filters), semantic and instance segmentation decoders (which incorporates convolutional layers with upsample pooling). The post-processing stage where semantic and instance segmentation outputs are fused does not have learned parameters.

Panoptic-DeepLab is trained with three loss functions: 1) weighted bootstrapped cross entropy loss for semantic segmentation head; 2) MSE loss for center heatmap head; and 3) L1 loss for center offset head. The final loss L is computed as follows.

$$L = \lambda_{sem} L_{sem} + \lambda_{heatmap} L_{heatmap} + \lambda_{offset} L_{offset}$$

3.1. Approach

A state-of-the-art Panoptic DeepLab model is already made available in the Detectron2 library. However because the default Panoptic DeepLab model was heavy to train, several exploratory modifications were made to the default model with the hope that some of them can make the model be trained reasonably well with fewer iterations.

Freezing Pre-trained ResNet Backbone at Different Stages The default Panoptic DeepLab model uses an ImageNet pre-trained ResNet-50 as its backbone encoder, but the default model does not freeze any ResNet stages during COCO training. In our experiments, we explored freezing different stages of pre-trained ResNet backbone, with the hope that doing so would reduce the number of parameters to train and therefore can result in faster model parameter convergence.

Using Pre-trained ResNet-101 as Backbone We experimented with using a pre-trained ResNet-101 as the backbone encoder instead of the default ResNet-50. We did this exploration with the hope that a richer ResNet as a feature extractor can feed to ASPP with better feature representations that can provide more useful contexts for decoders to perform semantic and instance segmentation.

Using Single ASPP Head The default Panoptic DeepLab model has dual ASPP heads. Because ASPP is responsible for extracting features and contexts from the input feature map with different fields of view, we were hoping that both semantic and instance segmentation decoders can share the same features extracted by a single ASPP head, which in turn could reduce the number of parameters learned by DeepLab model and make the model converge faster.

Using Different ASPP Dilation Rates In the default DeepLab model, the three ASPP layers in each head have

dilation rates of 6, 12, and 18, respectively. Because the dilation rate controls how big ASPP fields of views are, this configuration is critical to the performance of DeepLab. We explored changing ASPP dilation rates to different values in order to evaluate the effect of ASPP fields of view on the performance of Panoptic DeepLab.

3.2. Data Pre-Processing, Training and Evaluation Procedures

Data Pre-Processing Panoptic DeepLab models implemented by Detectron2 expect input as a batch of 3-channel RGB images together with their corresponding 1-channel annotations images, where each pixel in the annotations image represents the class ID that the corresponding pixel in input image represents. All input images are re-sized to a fixed size so large input images can fit into memory. COCO dataset does not provide the 1-channel annotations images, but Detectron2 provides a tool to convert COCO-style JSON annotations into 1-channel annotations images.

Training Procedure Each variant of Panoptic DeepLab model was trained on the COCO 2017 training dataset with panoptic segmentation annotations. All models were trained with a learning rate of 0.0025, using Adam optimizer with momentum of 0.9 to perform gradient descent.

Model Evaluation During inference time, for each input image, the output of Panoptic DeepLab is a 1-channel annotation image of the same size as the input image, where each pixel value in the output image is the class label ID represented by the corresponding input image pixel at that position.

In order to evaluate the performance of segmentation results, a Panoptic Quality (PQ) score of segmentation output with respect to the ground truth annotations was calculated for each test image. We evaluated each model with its average PQ score produced from all testing images. A model that produces output segmentations with higher PQ scores is regarded to have better performance than a model with lower PQ scores.

PQ score is a combination of Segmentation Quality score (SQ) and Recognition Quality score (RQ). SQ evaluates how closely output segments are matched with their ground truths. RQ is a combination of precision and recall, which attempts to identify how effective the model is at getting a prediction right.

$$PQ = \underbrace{\frac{\sum_{(p,g)\in TP} IOU_{(p,g)}}{|TP|}}_{\text{segmentation quality (SQ)}} \times \underbrace{\frac{|TP|}{|TP| + \frac{1}{2}|FP| + \frac{1}{2}|FN|}}_{\text{recognition quality (RQ)}}$$

where p is prediction output, g is ground truth annotations

3.3. Experiments and Results

	PQ	SQ	RQ
All	6.351	44.339	8.523
Things	4.259	43.053	5.773
Stuff	9.510	46.279	12.673

Table 1: PQ, SQ, and RQ scores of Default Panoptic DeepLab (after 10k iterations).

3.3.1 Freezing Pre-trained ResNet at Different Stages

We chose to freeze the first 2, 4, and all 5 stages of pre-trained ResNet. Our experiment shows that with just the first 2 stages of pre-trained ResNet frozen, the Panoptic DeepLab model was able to yield a higher PQ score as compared to the default model where all stages of ResNet were learned (8.06 vs. 6.35). In addition, we also saw that freezing the first 4 stages of pre-trained ResNet resulted in an even better PQ score (16.01). This score is also higher than if all 5 stages of ResNet were frozen (14.85).

The experiment result matches with our earlier hypothesis: freezing some lower-level feature extractors in ResNet and only performing fine-tuning in other parts can make parameters in Panoptic DeepLab model converge faster. This is because freezing lower-level convolutional layers essentially reduces the number of parameters to be learned during training with new data.

Another finding is that not freezing the last stage of ResNet can result in better performance than if all ResNet stages were frozen. This is likely because the last few layers of feature extractor (ResNet) are of greater importance to ASPP because ASPP is built directly on top of feature maps generated by the last stage of ResNet; therefore training the last few layers of ResNet together with ASPP and subsequent decoders would tend to make the gradient of weights w.r.t. loss in ASPP be smoother as opposed to freezing the weights of last stage of ResNet.

Freeze up to	PQ: All	PQ: Things	PQ: Stuff
N/A	6.351	4.259	9.510
2$^{\text{nd}}$ stage	8.060	6.090	11.034
4$^{\text{th}}$ stage	16.016	15.349	17.021
5$^{\text{th}}$ stage	14.856	13.261	17.263

Table 2: PQ scores of Panoptic DeepLab with different stages of ResNet frozen (after 10k iterations).

3.3.2 Using Pre-trained ResNet-101 as Backbone

Our experiment shows that ResNet-101-based Panoptic DeepLab was not able to outperform its ResNet-50-based counterpart if only lower layers (stage ≤ 2) in pre-trained ResNet were frozen during training. However, if the first 4 stages were frozen, or if all 5 stages were frozen, we saw a significant improvement on PQ scores generated by ResNet-101-based Panoptic DeepLab when compared to the default Panoptic DeepLab with ResNet-50 backbone.

This finding matches with our earlier hypothesis. Specifically, ResNet-101 is a richer model that is able to extract more complex features from input images, but it does so with more parameters to train than ResNet-50. Given to the higher model complexity of ResNet-101, when no stage was frozen or only the first 2 stages were frozen, ResNet-101 weights would take much longer time to converge than ResNet-50, which would result in poorer performance when given a constraint on training iterations. However, if we were only doing fine-tuning (or fine-tuning together with the last stage of pre-trained ResNet), the decoders of DeepLab would be able to leverage the already pre-trained ResNet-101 as a richer feature extractor, and with little additional cost because all weights in lower levels of ResNet-101 were frozen. As a result, the Panoptic DeepLab with ResNet-101 backbone would be able to converge faster than its counterpart with ResNet-50 backbone.

Freeze up to	PQ: All	PQ: Things	PQ: Stuff
2^{nd} stage	7.250	5.340	10.132
4^{th} stage	19.484	19.691	19.172
5^{th} stage	17.731	16.865	19.037

Table 3: PQ scores of custom Panoptic DeepLab with pre-trained ResNet-101 as backbone and different ResNet stages frozen (after 10k iterations).

3.3.3 Using Single ASPP Head

We implemented a custom version of Panoptic DeepLab with single ASPP head. Our experiment shows that Panoptic DeepLab with single ASPP head did not outperform the default model with dual ASPP heads (PQ of 18.10 vs. PQ of 19.48). This finding suggests that instance segmentation and semantic segmentation branches likely require different contextual and decoding information from ASPP. Our finding matches with the experiment results published in the Panoptic DeepLab paper, which also shows that using dual ASPP heads is able to produce better segmentation outputs than using single ASPP head.

	PQ: All	PQ: Things	PQ: Stuff
Dual ASPP Heads	19.484	19.691	19.172
Single ASPP Head	18.104	18.329	17.763

Table 4: PQ scores of Panoptic DeepLab with dual ASPP heads and single ASPP Head. Both models have pre-trained ResNet-101 as backbone, and first 4 stages of ResNet freezed (after 10k iterations).

3.3.4 Using Different ASPP Dilation Rates

We experimented with enlarging and shrinking ASPP fields of view by changing ASPP filters' dilation rates. The results showed that there is no significant change of segmentation performance if ASPP fields of views were shrunk by half, whereas there is a significant drop in performance if ASPP field of view is doubled.

ASPP Dilation Rates	PQ: All	PQ: Things	PQ: Stuff
3, 6, 9	19.356	19.513	19.120
6, 12, 18	19.484	19.691	19.172
12, 24, 36	18.677	18.879	18.371

Table 5: PQ scores of custom Panoptic DeepLab with different ASPP dilation rates. All models have pre-trained ResNet-101 as backbone, and first 4 ResNet stages freezed (after 10k iterations).

3.3.5 Hyper-parameter Tuning

From the experiment results shown above, the best Panoptic DeepLab variant when trained with 10k iterations was the one that was based on pre-trained ResNet-101 and had the first 4 ResNet stages frozen. We then performed hyper-parameter tuning on top of this model in order to further improve its performance.

Batch Size	PQ: All	PQ: Things	PQ: Stuff
5	19.484	19.691	19.172
8	22.502	23.011	21.734

Table 6: PQ scores of custom Panoptic DeepLab with ResNet-101 as backbone, first 4 ResNet stages frozen, trained with batch size of 8 (after 10k iterations).

Increasing Batch Size

Adding Weight Decay We explored adding a small weight decay to the optimizer as a regularization factor, however the resulting models did not show promising improvements in their PQ scores. This is likely because 1) there are already dropout layers incorporated in Panoptic DeepLab and 2) our batch size is small which already has a regularization effect (added noise).

3.3.6 Other considerations

As was observed in Figure 2, overall validation loss kept decreasing in an oscillating fashion while training in the first 10k iterations. Therefore we did not observe model overfitting during our training.

Figure 2: Validation loss Ccrve of Panoptic DeepLab training.

4. Panoptic FPN

Panoptic FPN uses FPN (feature pyramid network) as its backbone to generate rich features at different scales. FPN takes a standard network with features at multiple spatial resolutions (i.e., ResNet), and adds a light top-down pathway with lateral connections. Panoptic FPN performs ROI pooling by attaching a Faster R-CNN to FPN. Mask-RCNN is used to generate masks for detected objects from generated ROIs. In order to perform semantic segmentation, Panoptic FPN attaches to FPN with a simple and fast semantic segmentation branch, where each FPN level is upsampled by convolutions and bilinear upsampling until it reaches 1/4 scale, theses outputs are then summed and finally transformed into a pixel-wise output.

Panoptic FPN generates 3 losses in its instance segmentation branch: classification loss, L_c, bounding box loss, L_b, and mask loss, L_m. In addition, Panoptic FPN generates 1 loss in its semantic segmentation branch (L_s). The semantic segmentation loss is a per-pixel cross entropy loss between the predicted and the ground-truth labels. The final loss is calculated as follows:

$$L = \lambda_i(L_c + L_b + L_m) + \lambda_s L_s$$

where λ_i and λ_s are hyper-parameters.

Two major parts of panoptic FPN have learnable parameters. The first part is underlying backbone structures, convolution layers specifically. The second part is the connection layers between feature maps and FPN layers. The model is very complex and due to time constraints, the training loss curves seem to suggest that the model slowly learns even after 270k iterations, so overfitting has yet been identified for now. Major hyperparameters are base learning rate, batch size, and the depth of backbone network, and Adam based optimizers were used. Based the experiment results, batch size and learning rate are two significant hyperparamters affecting performance, but unfortunately due to the limited GPU quota imposed by Google Cloud, we were prevented to apply larger batch sizes (e.g., 16 and plus) because of the memory limits of a single GPU.

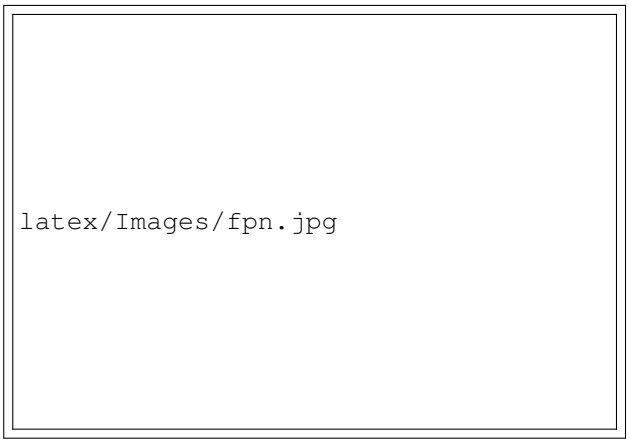

Figure 3: Panoptic FPN architecture

4.1. Approach

The methods were developed based on Detectron2 library as well. One of the backbones used by the panoptic segmentation tasks is ResNet of various depths. ResNet is applied to extract feature maps of different depths, and FPN is applied to combine these feature maps into a "pyramid" style structure. This structure captures feature information of different resolutions, and its outcomes are further fed into regional proposal algorithms and subsequent semantic and instance segmentation tasks. The selection of backbones therefore determines how feature maps are generated, and this task tends to investigate modified versions of the existing backbone architectures and how the modifications may

affect performance, thus deepening our understanding towards panoptic segmentation. Specifically, two modifications were conducted as detailed in the following text (Figure 5 in Appendix B). As the fundamental structure of the library is changed, pre-trained weights are no longer usable. We expect that the resulting performance may be below expectation.

Two bottleneck layers within a ResNet block The purpose of the first modification is to investigate whether a redundant design of the residual block may improve the performance. This minor modification was shown by Figure 5b. The redundant design may provide additional information to a residual block and therefore can enhance the quality of feature maps that will be heavily relied by sebsquent FPN processing and segmentation tasks.

Implementation of a VGG backbone The second modification tends to understand how this very classic CNN architecture performs panoptic segmentation tasks (Figure 5a). If the performance is sufficiently sound, the simpler model is always preferred. Figure shows the implementation details of a VGG architecture into a panoptic FPN framework. This VGG architecture is not implemented in the original detectron2 library and built from scratch by us.

Using a Res2Net backbone The Res2Net backbone architecture was another backbone investigated because it had motivating features like introducing a 'scale' dimension to existing blocks in the ResNet architecture [**?**]. The introduction of grouping features in a single residual block enhances the receptive fields at lower levels of the network. Typically these Res2Net blocks or modules are built in a hierarchical structure that enable it to be easily embedded into existing networks.

4.2. Experiments and Results

Methodology Similar to Section 3, these modified models were trained using the COCO 2017 training and validation data sets. The same learning rate of 0.0025 and a batch size of 2 (due to Memory limits based on a single GPU unit imposed by the Google Cloud policy) were applied to the majority of experiments. Major varying hyperparamter is the number of iterations, but some experiments involve some changes in learning rates and batch sizes.

4.2.1 Modified ResNet backbone

It is satisfactory that the PQ score achieved more than 30 after a training scheme of 270k iterations (Table 9), as this architecture has more weights than the original ResNet-50 one and no pre-trained model is available. The training is also slow, potentially because a significant amount of time was spent on fitting parameter weights of the modified backbone.

	PQ	SQ	RQ
All	31.935	75.797	39.197
Things	36.803	79.470	44.311
Stuff	24.586	70.252	31.478

Table 7: PQ scores of custom Panoptic FPN with ResNet-50 as backbone (two 3X3 bottleneck layers, 270k iterations, learing rate of 0.005, batch size of 4)

4.2.2 Implementation of a VGG based backbone

VGG has a straightforward architecture, but as pointed out by He *et al.* [**?**], it could suffer from vanishing gradient problems with deeper layers. This problem may lead to worse performance that those advanced backbone representations (such as ResNet) presented previously. The model took a long time to learn and resulting evaluation metrics appear to be not so promising (Table 8).

Number of iterations	PQ	SQ	RQ
50k	–	75.797	39.197
100k	–	79.470	44.311
250k	–	70.252	31.478

Table 8: PQ scores (all) of custom Panoptic FPN with a VGG as backbone (learning rate of 0.0025, batch size of 2)

4.2.3 Introduction of a Res2Net backbone

Using the Res2Net backbone, we see a reasonable increase in PQ scores. From simply altering the number of iterations from 7K to 10K and increasing the batch size from 2 to 5, we see a jump in PQ scores (Table 9). Reasonably, given more iterations and some more tuning, embedding a Res2Net backbone to the model can provide dramatic changes to performance of the model.

	PQ [1]	PQ[2]
All	8.254	19.082
Things	7.215	21.148
Stuff	9.824	15.964

Table 9: PQ scores of Panoptic FPN with Res2Net as backbone with slightly different hyper-parameters to illustrate jump in PQ values where [1]7K Iterations + Batch Size:2 and [2] 10K Iterations + Batch Size: 5

5. Conclusions

This project explored state-of-the-art architectures tackling panoptic segmentation tasks, i.e., panoptic FPN with Mask-RCNN and Panoptic Deeplab, and evaluated some

newer architectures recently developed, e.g, DETR. While our experiments generate promising performance, certain future directions could worth investigating. Such topics include different feature extraction mechanisms (e.g., transformer), the expandable capabilities of pre-trained models, and efficient training of deep backbone networks.

6. Work Division

For the contributions of individual team members, please refer to Table 10.

A. Panoptic DeepLab

Figure 4 presents an illustratoin of the Panoptic DeepLab architecture.

B. Panoptic FPN

Figure 5 shows the modifications to the original backbone architectures used by the detectron2 library. Figure 6 shows a demonstration of the trained model based on a modified ResNet50 backbone.

Student Name	Contributed Aspects	Details
Anurag Sen Jie Song Zhiling Zho Zixiang (David) Zhu		

Table 10: Contributions of team members.

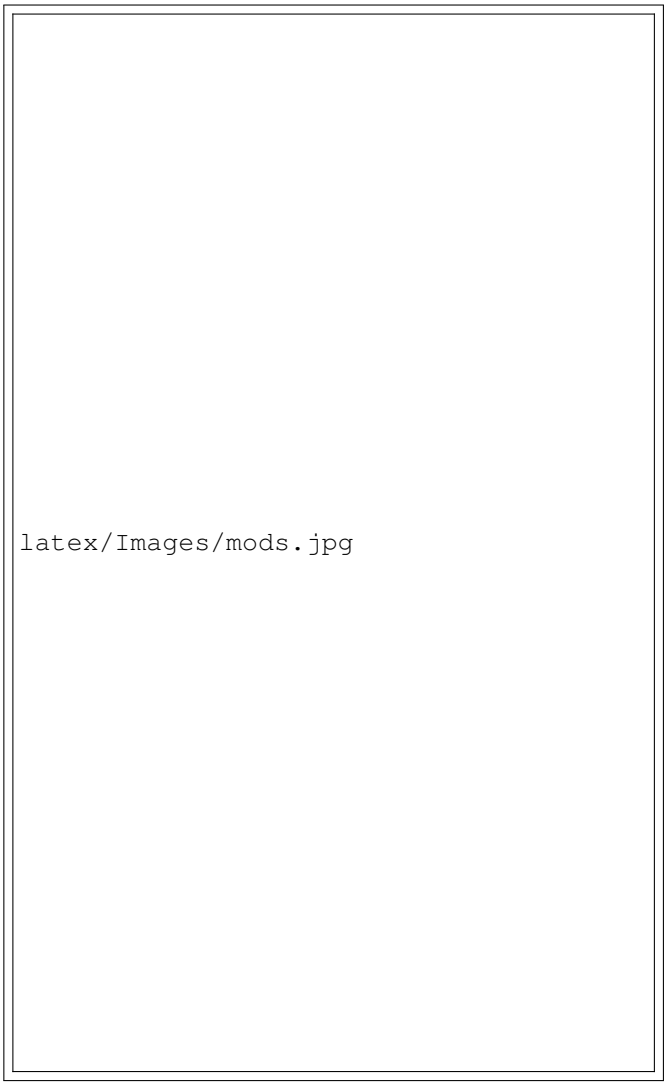

Figure 5: Modifications to the backbones of a Panoptic FPN architecture