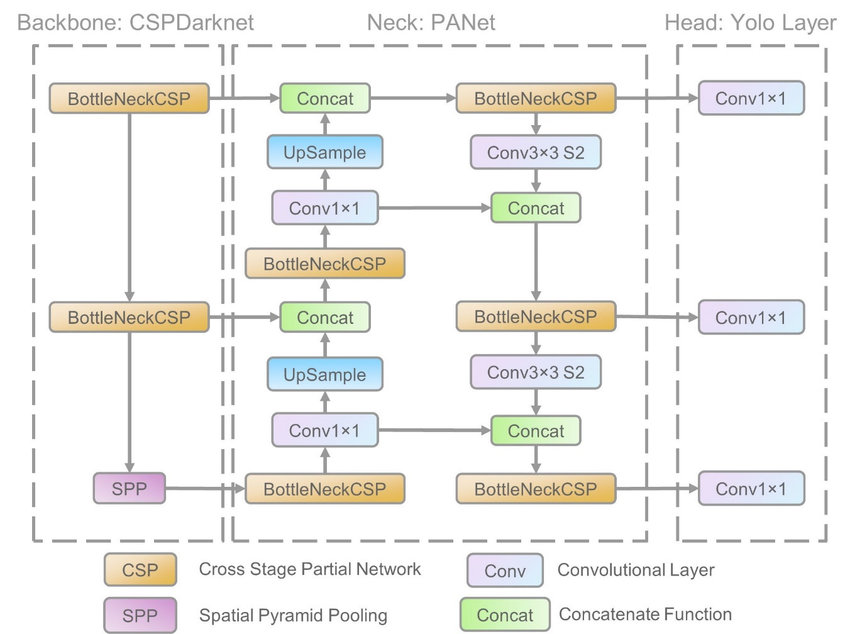
**STUDY ON YOLOv5 ARCHITECTURE:**

**YOLOv5:**

1. Glenn Jocher released YOLOv5 with a number of differences and improvements.
2. The release of YOLOv5 includes five different models sizes: YOLOv5s (smallest), YOLOv5m, YOLOv5l, YOLOv5x (largest).
3. Because YOLOv5 is implemented in PyTorch initially, it benefits from the established PyTorch ecosystem: support is simpler, and deployment is easier.
4. Also as a more widely known research framework, iterating on YOLOv5 may be easier for the broader research community. This also makes deploying to mobile devices simpler as the model can be compiled to ONNX and CoreML with ease.
5. YOLOv5 is fast – blazingly fast. In a YOLOv5 Colab notebook, running a Tesla P100, the inference times up to 0.007 seconds per image, meaning 140 frames per second (FPS)! By contrast, YOLOv4 achieved 50 FPS after having been converted to the same Ultralytics PyTorch library.
6. YOLOv5 achieves 140 frames per second in batch, which the YOLOv5 implementation tested utilizes by default. When batch size is set to 1, YOLOv4 achieves 30FPS while YOLOv5 outputs 10 FPS. Please see our detailed methodology update.
7. YOLOv5 is accurate. In tests on the blood cell count and detection (BCCD) dataset, they have achieved roughly 0.895 mean average precision (mAP) after training for just 100 epochs.
8. YOLOv5 is small. Specifically, a weights file for YOLOv5 is 27 megabytes. The weights file for YOLOv4 (with Darknet architecture) is 244 megabytes. YOLOv5 is nearly 90 percent smaller than YOLOv4. This means YOLOv5 can be deployed to embedded devices much more easily.
9. YOLOv5 is still under development.

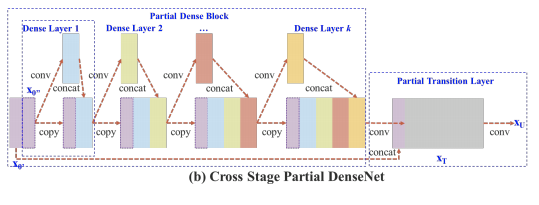
**Network Architecture:**



1. The network architecture consists of three parts:
   1. Backbone: CSPDarknet
   2. Neck: PANet
   3. Head: YOLOLayer
2. The data are first input to CSPDarknet for feature extraction.
3. Features extracted are fed to PANet for feature fusion.
4. YOLO layer outputs detection results ( class, score, location, size.)
5. Yolov5 is based on Yolov1- Yolov4. Continuous improvements have made it achieve top performances on two official object detection datasets: Pascal VOC (visual object classes) [32] and Microsoft COCO (common objects in context) [33].
6. Yolov5 incorporated cross stage partial network (CSPNet) into Darknet, creating CSPDarknet as its backbone.
7. CSPNet solves the problems of repeated gradient information in large-scale backbones, and integrates the gradient changes into the feature map, thereby decreasing the parameters and FLOPS (floating-point operations per second) of model, which not only ensures the inference speed and accuracy, but also reduces the model size.
8. The Yolov5 applied path aggregation network (PANet) as its neck to boost information flow.
9. PANet adopts a new feature pyramid network (FPN) structure with enhanced bottom-up path, which improves the propagation of low-level features.
10. At the same time, adaptive feature pooling, which links feature grid and all feature levels, is used to make useful information in each feature level propagate directly to following subnetwork.
11. PANet improves the utilization of accurate localization signals in lower layers, which can obviously enhance the location accuracy of the object.
12. The head of Yolov5, namely the Yolo layer, generates 3 different sizes (18 × 18, 36 × 36, 72 × 72) of feature maps to achieve multi-scale prediction, enabling the model to handle small, medium, and big objects. Multi-scale detection ensures that the model can follow size changes.

**CSP Backbone:**

1. Both YOLOv4 and YOLOv5 implement the CSP Bottleneck to formulate image features.
2. The CSP addresses duplicate gradient problems in other larger ConvNet backbones resulting in less parameters and less FLOPS for comparable importance. This is extremely important to the YOLO family, where inference speed and small model size are of utmost importance.
3. The CSP models are based on DenseNet.
4. DenseNet was designed to connect layers in convolutional neural networks with the following motivations: to alleviate the vanishing gradient problem (it is hard to backprop loss signals through a very deep network), to bolster feature propagation, encourage the network to reuse features, and reduce the number of network parameters.
5. In CSPResNext50 and CSPDarknet53, the DenseNet has been edited to separate the feature map of the base layer by copying it and sending one copy through the dense block and sending another straight on to the next stage.
6. The idea with the CSPResNext50 and CSPDarknet53 is to remove computational bottlenecks in the DenseNet and improve learning by passing on an unedited version of the feature map.

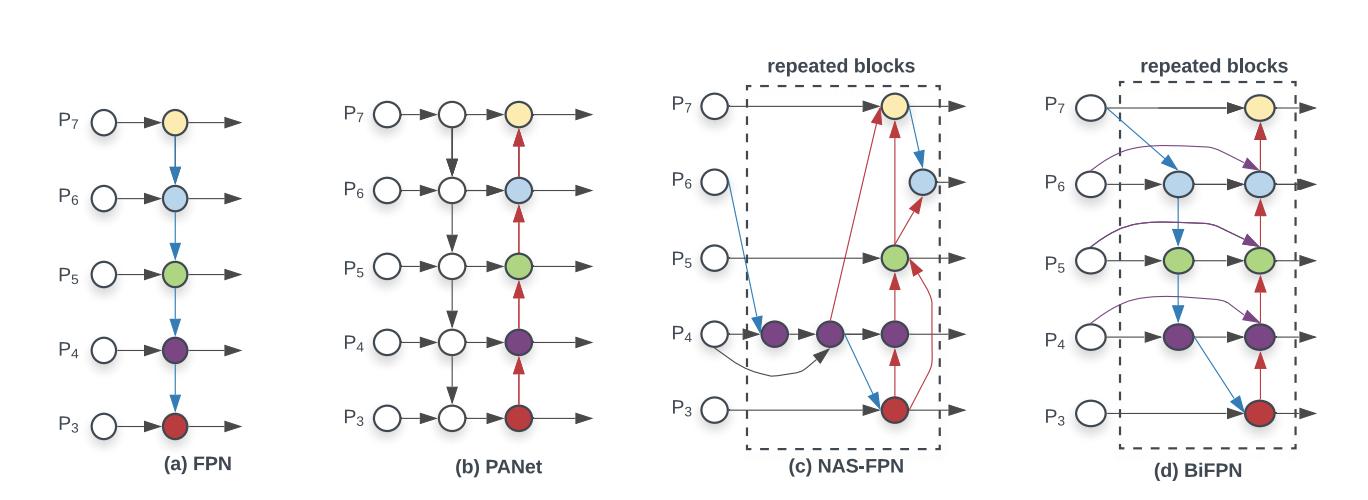


1. A stage of CSPDenseNet is composed of a partial dense block and a partial transition layer. In a partial dense block, the feature maps of the base layer in a stage are split into two parts through channel x0. The former is directly linked to the end of the stage, and the latter will go through a dense block. All steps involved in a partial transition layer are as follows: First, the output of dense layers, [x 00 0 , x1, ..., xk], will undergo a transition layer. Second, the output of this transition layer, xT , will be concatenated with x 00 0 and undergo another transition layer, and then generate output xU .

Ref.: <https://arxiv.org/pdf/1911.11929.pdf>

**PANet Neck:**

1. Both YOLOv4 and YOLOv5 implement the PA-NET neck for feature aggregation.



1. Each one of the P\_i above represents a feature layer in the CSP backbone.
2. The above picture comes from research done by Google Brain on the EfficientDet object detection architecture.
3. The EfficientDet authors found BiFPN to be the best choice for the detection neck, and it is may be an area of further steady for YOLOv4 and YOLOv5 to explore with other implementations here.
4. It is certainly worth noting here that YOLOv5 borrows research inquiry from YOLOv4 to decide on the best neck for their architecture. YOLOv4 investigated various possibilities for the best YOLO neck including:

FPN

PAN

NAS-FPN

BiFPN

ASFF

SFAM

1. EfficientDet is built on top of EfficientNet, a convolutional neural network that is pretrained on the ImageNet image database for classification. EfficientDet pools and mixes portions of the image at given granularities and forms features which are passed through a NAS-FPN feature fusion layer. The NAS-FPN combines various features at varying granularities and passes them forward to the detection head, where bounding boxes and class labels are predicted.
2. EfficientDet is a family of models expressing the same architecture at different model size scales.

Ref.: <https://blog.roboflow.com/breaking-down-efficientdet/>

(Refer Yolov4 part2 for Backbone/Neck/Yolo layer explanations)

Implementation Ref.:

<https://michaelohanu.medium.com/yolov5-tutorial-75207a19a3aa>

<https://www.programmersought.com/article/20545272716/>

<https://lionbridge.ai/articles/create-an-end-to-end-object-detection-pipeline-using-yolov5/>

