IDC410

A course on Image Processing and Machine Learning

(Lecture 17 and 18)

Shashikant Dugad,

IISER Mohali

Feature Pyramid Network (FPN)

Slides made using

1. **Lectures given by Kapil Sachdeva on YouTube**

2. **https://jonathan-hui.medium.com/understanding-feature-pyramid-networks-for-object-detection-fpn-45b227b9106c**

3. **arXiv:1612.03144v2 [cs.CV] 19 Apr 2017**

Feature Pyramid Network (FPN)

What is to be determined?

- **Type of Class (I)**
- **Size of Object (L, W)**
- **Position of Object (X_0, Y_0)**
- **Objectiveness (Probability)**

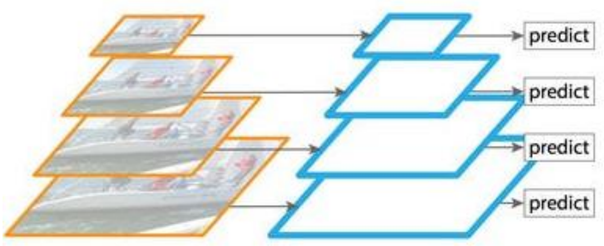
- **Full description of an identified object typically requires 5 integers and 1 Real number**

Feature Pyramid Network (FPN)

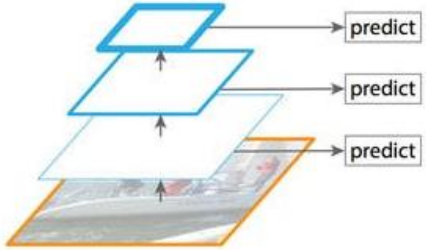
- **Detecting objects at different scales (sizes) is challenging**
  - **Same object may look smaller or bigger depending on its distance from the camera.**
  - **Object of one type may be of smaller size as compared to other type of object**

- **Pyramid of the image at different scale may be an option to detect objects at different scales. However it is computational taxing and not so accurate**

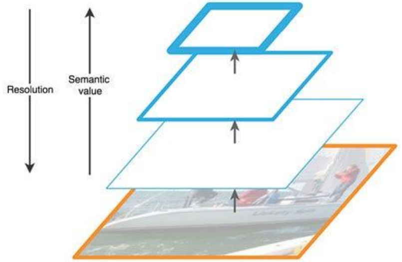
- **Pyramid of feature maps may be more appropriate**
  - **Feature maps closer to the image layer composed of low-level structures with better spatial resolution. However, they lack feature (semantic) strength and hence, may not be effective for accurate object detection.**
  - **Feature map at the top may have higher semantic content required for accurate object detection but deficient in spatial resolution**



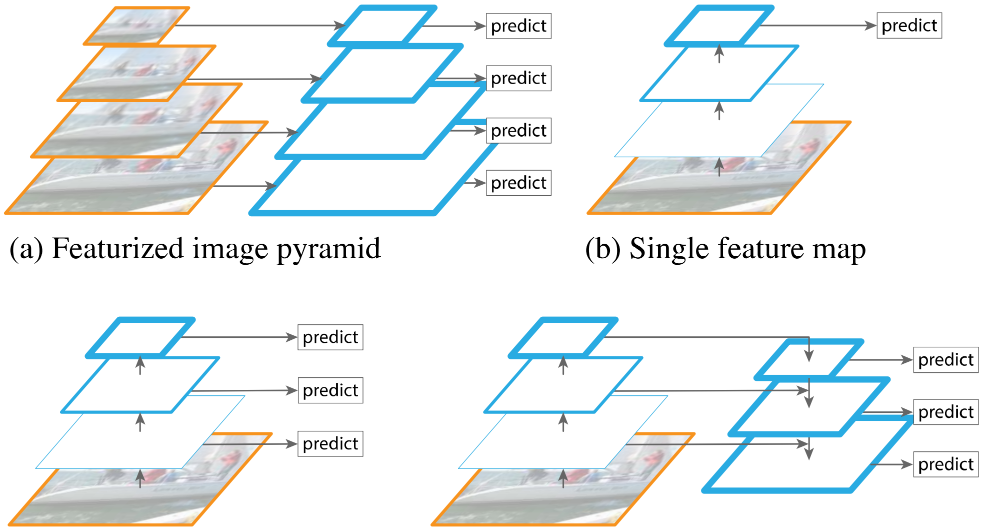
Pyramid of images



Pyramid of feature maps



Feature Pyramid Network (FPN)



(a) Featurized image pyramid

(b) Single feature map

(c) Pyramidal feature hierarchy

(d) Feature Pyramid Network

a) Using an image pyramid to build a feature pyramid. Features are computed on each of the image scales independently, which is slow.

b) Recent detection systems have opted to use only single scale features for faster detection.

c) An alternative is to reuse the pyramidal feature hierarchy computed by a ConvNet as if it were a featurized image pyramid.

d) Our proposed Feature Pyramid Network (FPN) is as fast as (a) or (b), but more accurate. In this figure, feature maps are indicated by blue outlines and thicker outlines denote semantically stronger features

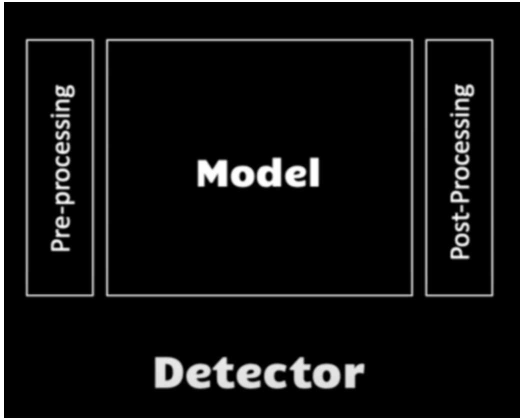
Feature Pyramid Network (FPN)

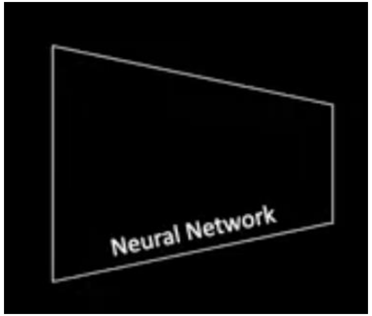
- The FPN combines high-resolution features with lower-resolution features, enabling the network to capture both fine details and broader context. The architecture involves two pathways: a bottom-up pathway that generates the feature pyramid, and a top-down pathway that fuses multi-scale information.

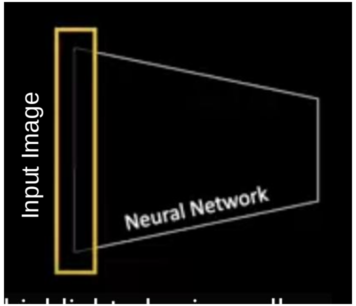
- The top-down pathway involves upscaling features from higher levels and merging them with features from lower levels through latera connections

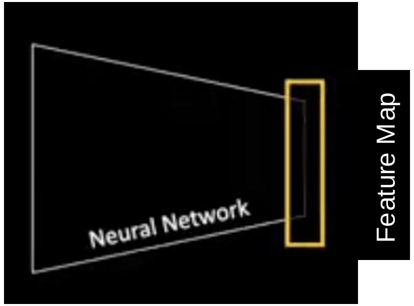
- This fusion of multi-scale features by FPN, allows it to effectively detect objects of different sizes or scales

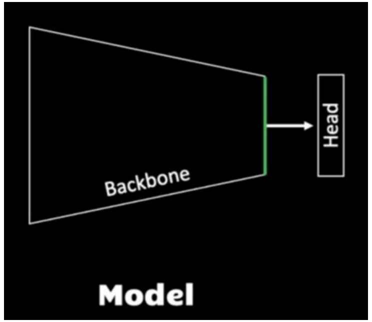
Feature Pyramid Network (FPN)



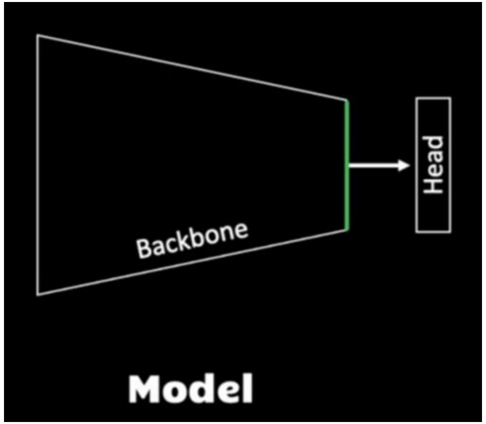


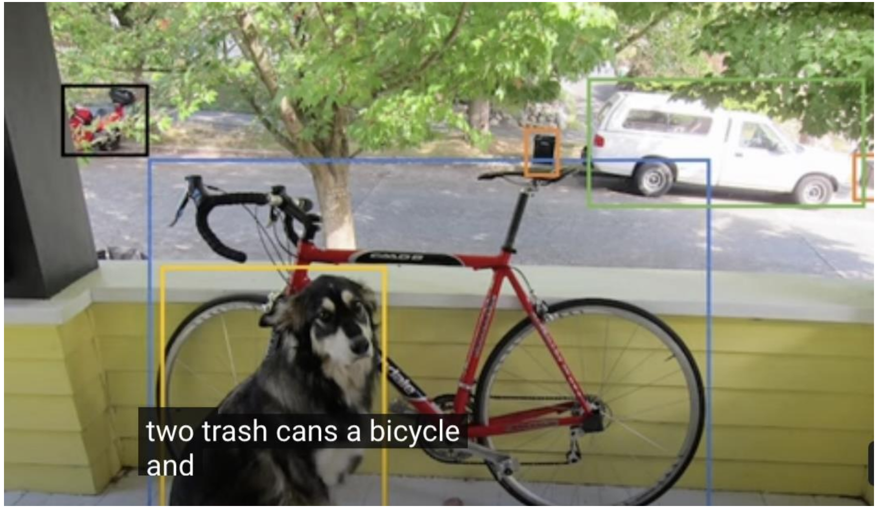


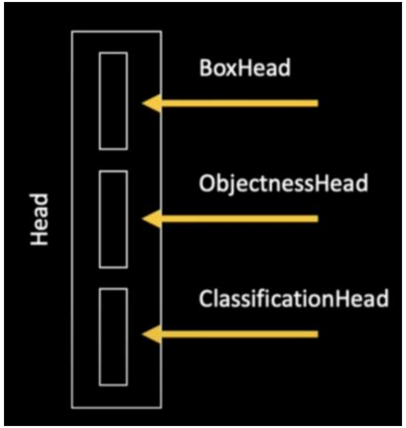




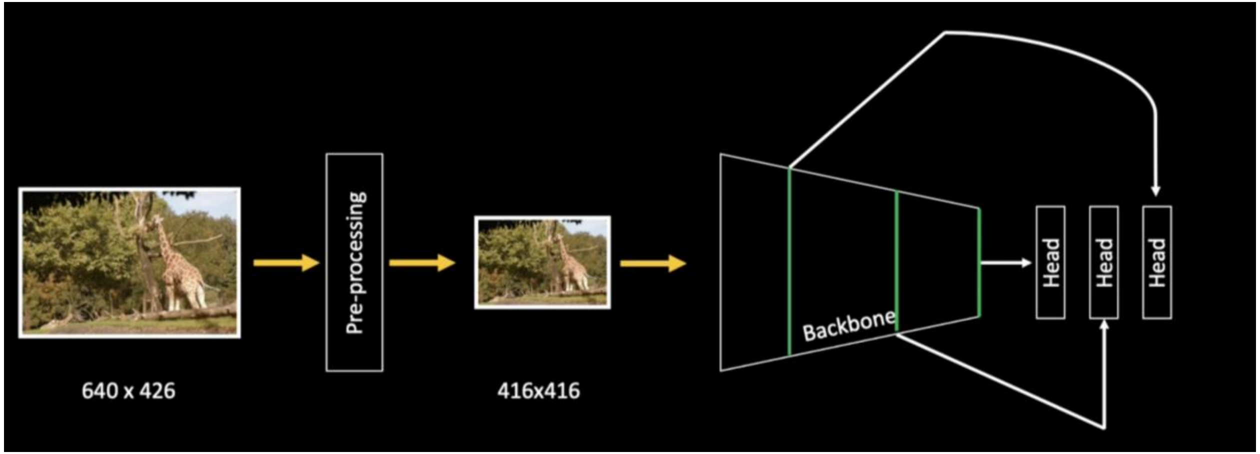
Feature Pyramid Network (FPN)

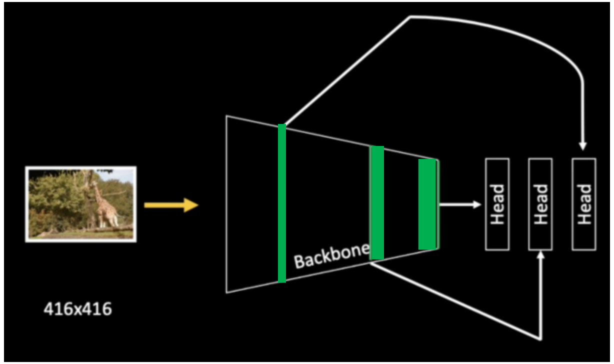


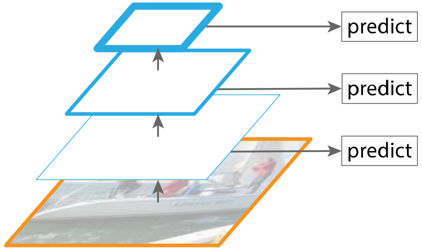




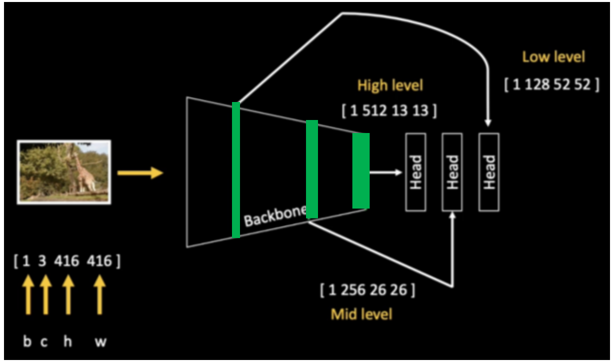
Feature Pyramid Network (FPN)



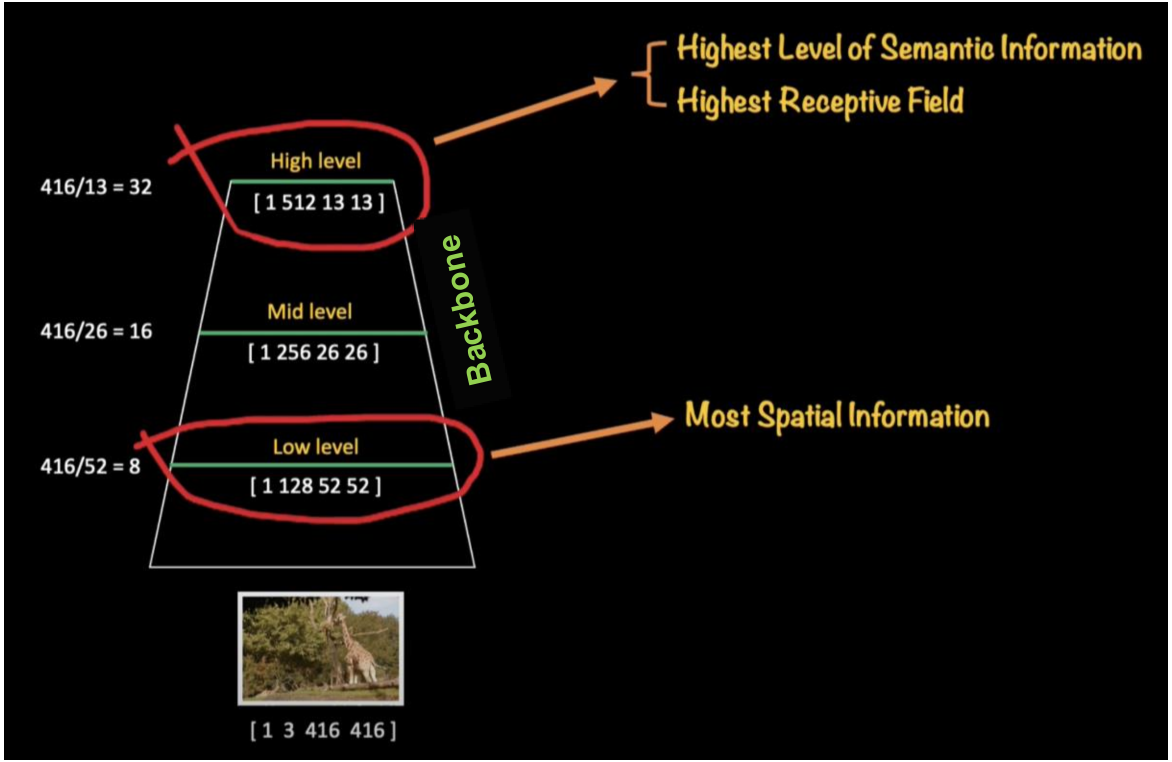


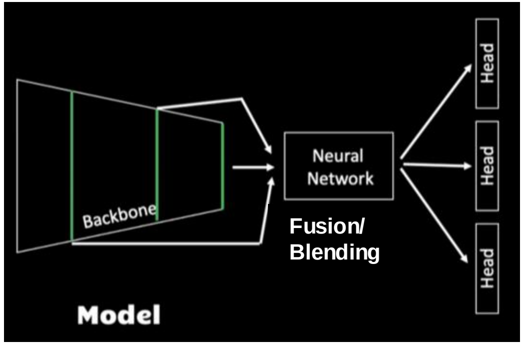


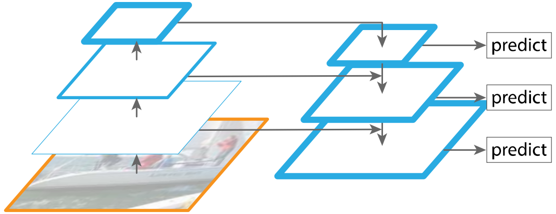
(c) Pyramidal feature hierarchy



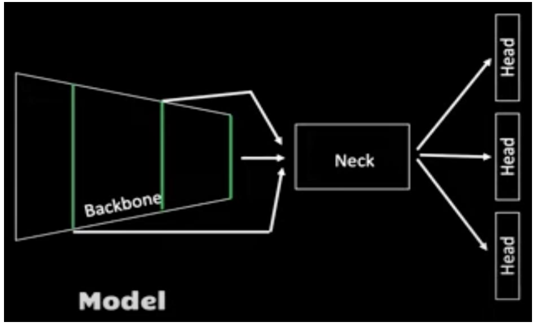
Feature Pyramid Network (FPN)



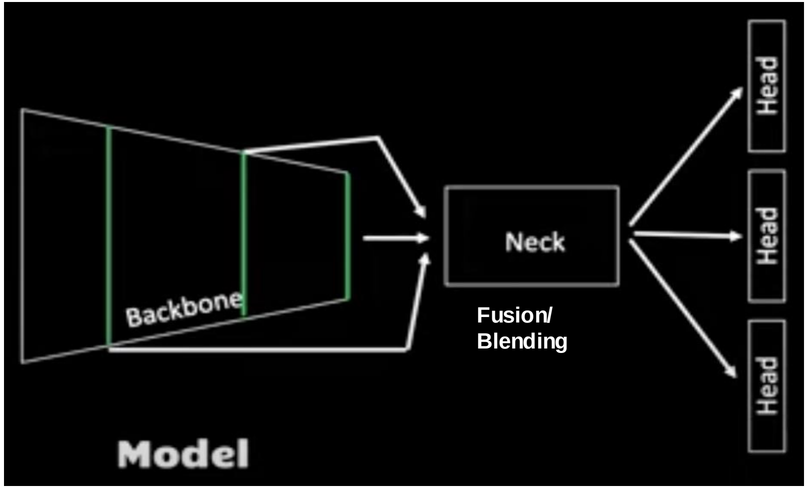


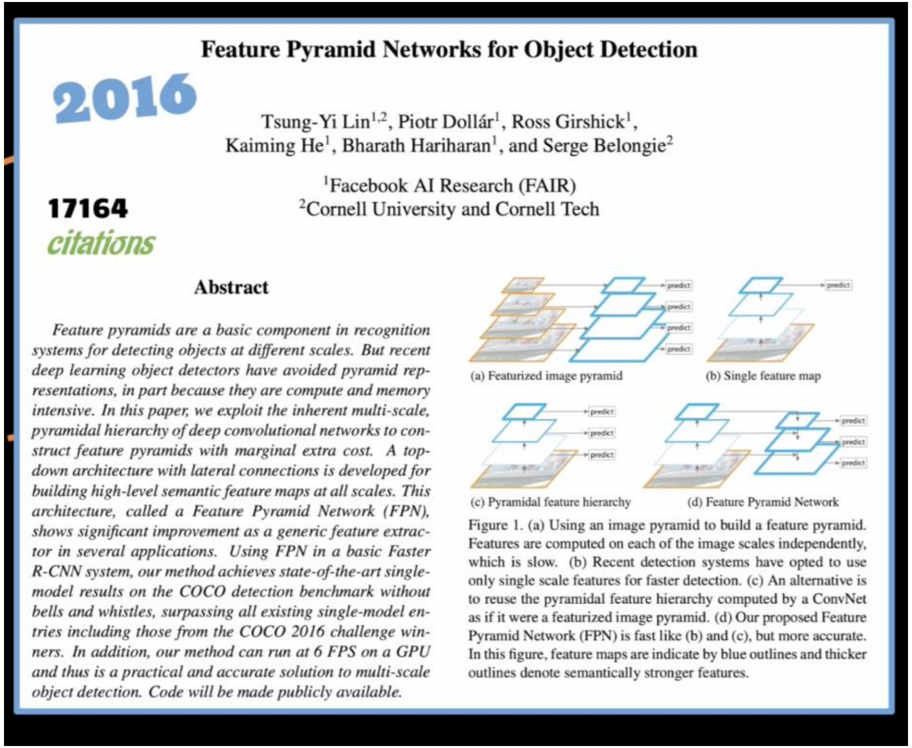


(d) Feature Pyramid Network



Feature Pyramid Network (FPN)





Feature Pyramid Networks for Object Detection

Tsung-Yi Lin[1,2], Piotr Dollár[1], Ross Girshick[1],
Kaiming He[1], Bharath Hariharan[1], and Serge Belongie[2]

[1]Facebook AI Research (FAIR)
[2]Cornell University and Cornell Tech

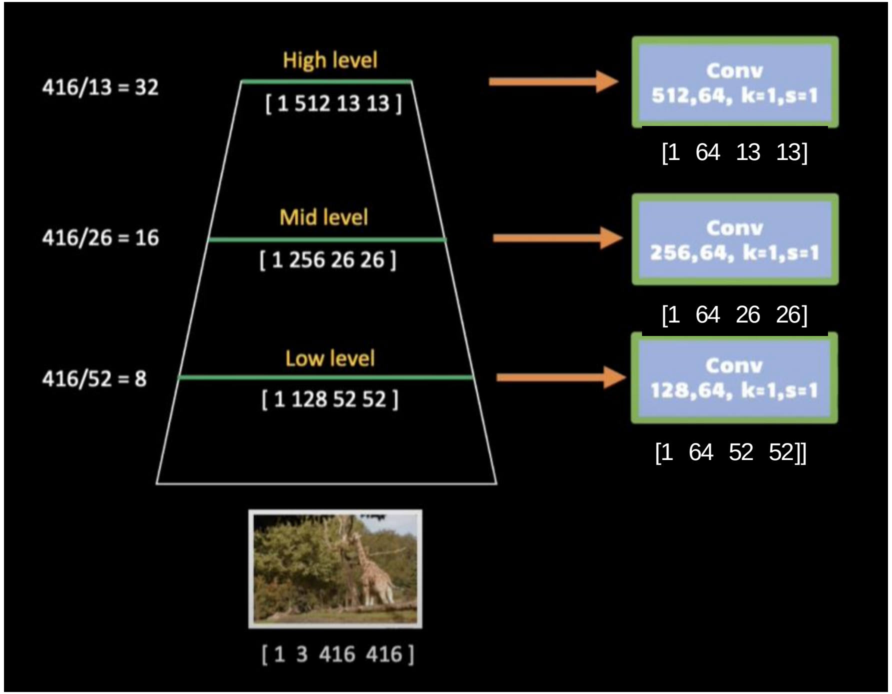
Abstract

Feature pyramids are a basic component in recognition systems for detecting objects at different scales. But recent deep learning object detectors have avoided pyramid representations, in part because they are compute and memory intensive. In this paper, we exploit the inherent multi-scale, pyramidal hierarchy of deep convolutional networks to construct feature pyramids with marginal extra cost. A top-down architecture with lateral connections is developed for building high-level semantic feature maps at all scales. This architecture, called a Feature Pyramid Network (FPN), shows significant improvement as a generic feature extractor in several applications. Using FPN in a basic Faster R-CNN system, our method achieves state-of-the-art single-model results on the COCO detection benchmark without bells and whistles, surpassing all existing single-model entries including those from the COCO 2016 challenge winners. In addition, our method can run at 6 FPS on a GPU and thus is a practical and accurate solution to multi-scale object detection. Code will be made publicly available.

(a) Featurized image pyramid (b) Single feature map (c) Pyramidal feature hierarchy (d) Feature Pyramid Network

Figure 1. (a) Using an image pyramid to build a feature pyramid. Features are computed on each of the image scales independently, which is slow. (b) Recent detection systems have opted to use only single scale features for faster detection. (c) An alternative is to reuse the pyramidal feature hierarchy computed by a ConvNet as if it were a featurized image pyramid. (d) Our proposed Feature Pyramid Network (FPN) is fast like (b) and (c), but more accurate. In this figure, feature maps are indicate by blue outlines and thicker outlines denote semantically stronger features.

Feature Pyramid Network (FPN)




```
hl_fm = torch.randn(size=(1, 512, 13, 13))
ml_fm = torch.randn(size=(1, 256, 26, 26))
ll_fm = torch.randn(size=(1, 128, 52, 52))
```

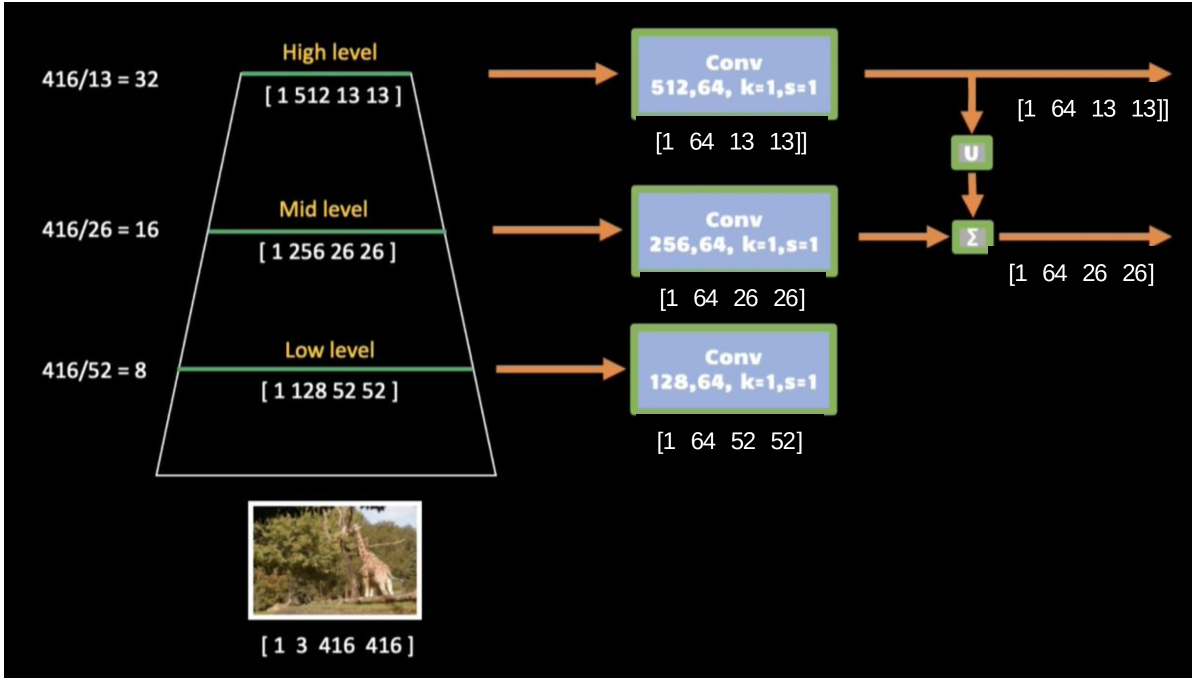
```
conv_hl_reduce = ConvBlockReduceChannels(in_channels=512, out_channels=64)
conv_ml_reduce = ConvBlockReduceChannels(in_channels=256, out_channels=64)
conv_ll_reduce = ConvBlockReduceChannels(in_channels=128, out_channels=64)
```

```
hl_fm_r = conv_hl_reduce(hl_fm)
ml_fm_r = conv_ml_reduce(ml_fm)
ll_fm_r = conv_ll_reduce(ll_fm)
```

```
print(f"New HL shape - {hl_fm_r.shape}")
print(f"New ML shape - {ml_fm_r.shape}")
print(f"New LL shape - {ll_fm_r.shape}")
```

```
New HL shape - torch.Size([1, 64, 13, 13])
New ML shape - torch.Size([1, 64, 26, 26])
New LL shape - torch.Size([1, 64, 52, 52])
```

Feature Pyramid Network (FPN)




```
print(f"New HL shape - {hl_fm_r.shape}")
print(f"New ML shape - {ml_fm_r.shape}")
print(f"New LL shape - {ll_fm_r.shape}")
```

```
New HL shape - torch.Size([1, 64, 13, 13])
New ML shape - torch.Size([1, 64, 26, 26])
New LL shape - torch.Size([1, 64, 52, 52])
```

```
hl_upsampler = nn.Upsample(scale_factor=2, mode="nearest")
```

```
hl_fm_r_upsampled = hl_upsampler(hl_fm_r)

hl_fm_r_upsampled.shape
```

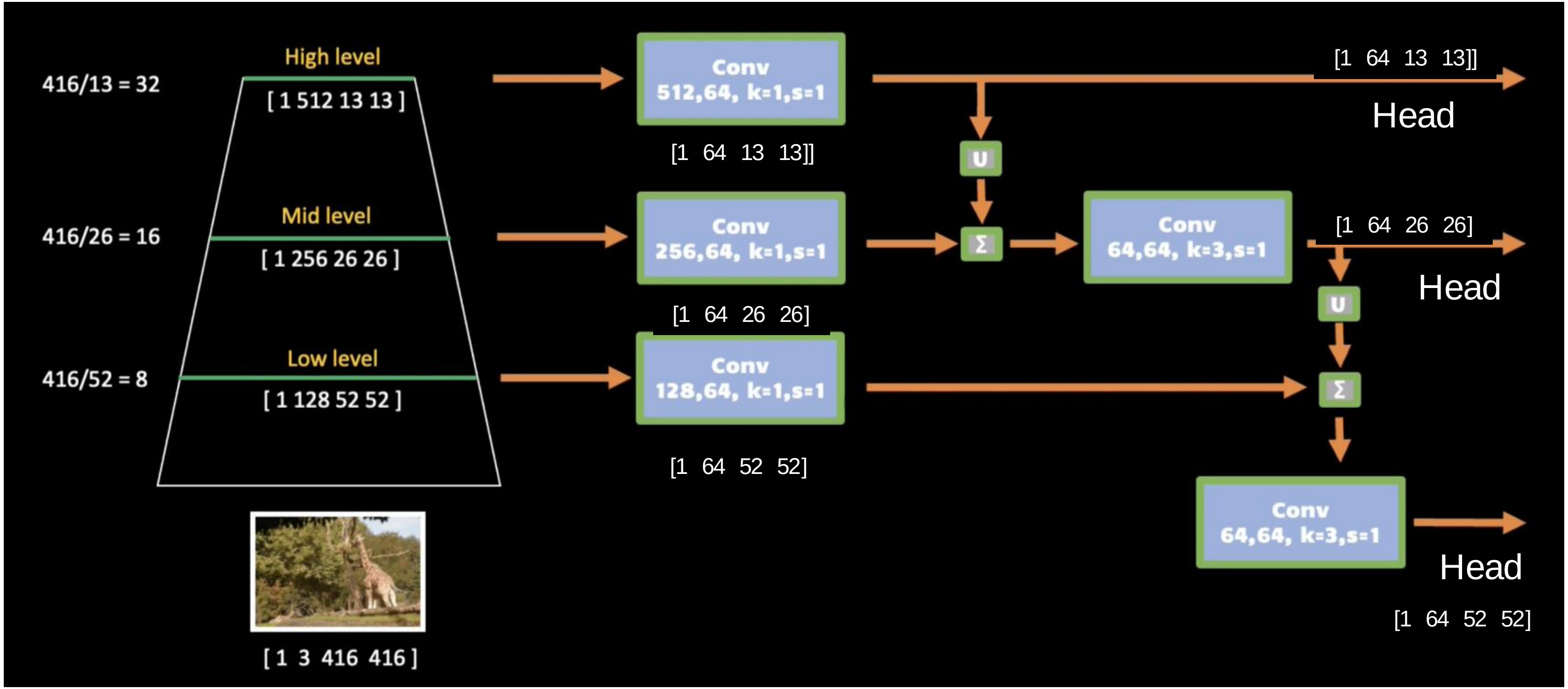
```
torch.Size([1, 64, 26, 26])
```

```
hl_ml_fused = torch.add(hl_fm_r_upsampled, ml_fm_r)

hl_ml_fused.shape
```

```
torch.Size([1, 64, 26, 26])
```

Feature Pyramid Network (FPN)



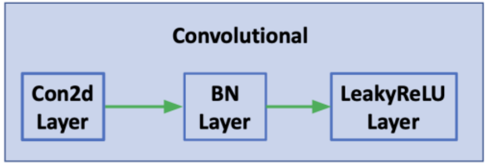
Backbone: Darknet - 19

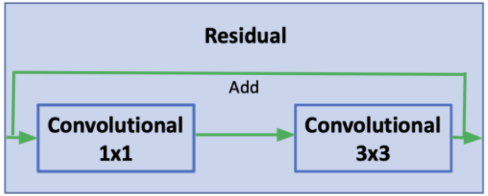
Type	Filters	Size/Stride	Output
Convolutional	32	3×3	224×224
Maxpool		$2 \times 2/2$	112×112
Convolutional	64	3×3	112×112
Maxpool		$2 \times 2/2$	56×56
Convolutional	128	3×3	56×56
Convolutional	64	1×1	56×56
Convolutional	128	3×3	56×56
Maxpool		$2 \times 2/2$	28×28
Convolutional	256	3×3	28×28
Convolutional	128	1×1	28×28
Convolutional	256	3×3	28×28
Maxpool		$2 \times 2/2$	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Maxpool		$2 \times 2/2$	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	1000	1×1	7×7
Avgpool		Global	1000
Softmax			

- In ***Darknet-19***, there are no skip connections, therefore, cannot have large number of convolutional layers
- ***Maxpool*** is used extensively resulting in loss of information
- Final output is used for prediction
- Not so useful for employing ***FPN*** like architecture

Backbone: Darknet - 53

Repeat Factor	Type	Filters	Size/Stride	Output
	Input	-	-	736x736x3
	Convolutional	32	3x3x3/1	736x736x32
	Convolutional	64	3x3x32/2	368x368x64
	Convolutional	32	1x1x64/1	368x368x32
1x	Convolutional	64	3x3x32/1	368x368x64
	Residual			368x368x64
	Convolutional	128	3x3x64/2	184x184x128
	Convolutional	64	1x1x128/1	184x184x64
2x	Convolutional	128	3x3x64/1	184x184x128
	Residual			184x184x128
	Convolutional	256	3x3x128/2	92x92x256
	Convolutional	128	1x1x256/1	92x92x128
8x	Convolutional	256	3x3x128/1	92x92x256
	Residual			92x92x256
	Convolutional	512	3x3x256/2	46x46x512
	Convolutional	256	1x1x512/1	46x46x256
8x	Convolutional	512	3x3x256/1	46x46x512
	Residual			46x46x512
	Convolutional	1024	3x3x512/2	23x23x1024
	Convolutional	512	1x1x1024/1	23x23x512
4x	Convolutional	1024	3x3x512/1	23x23x1024
	Residual			23x23x1024
	Avgpool		Global	
	Connected		1000	
	Softmax			





- The *Darkent-53* has *53* convolutional layers with each convolutional layer being subjected to back-normalization and then followed by the activation function (mostly *LeakyReLU*)
- The # of filters, their size and stride for *Darknet-53* have been mentioned in *3rd* and *4th* column of the table.
- *Darknet-53* also uses the *Residual layers*, which adds the input of the block (shown in yellow) to the output of the same block
- The count *(Repeat factor)* describes the number of times that specific block is repeated in the network

Backbone: Darknet - 53

Repeat Factor	Type	Filters	Size/Stride	Output
	Input	-	-	736x736x3
	Convolutional: C1	32	3x3x3/1	736x736x32
	Convolutional: C2	64	3x3x32/2	368x368x64
1x	Convolutional: C3	32	1x1x64/1	368x368x32
	Convolutional: C4	64	3x3x32/1	368x368x64
	Residual: R1			368x368x64
	Convolutional: C5	128	3x3x64/2	184x184x128
2x	Convolutional: C6	64	1x1x128/1	184x184x64
	Convolutional: C7	128	3x3x64/1	184x184x128
	Residual: R2			184x184x128
	Convolutional: C8	256	3x3x128/2	92x92x256
8x	Convolutional: C9	128	1x1x256/1	92x92x128
	Convolutional: C10	256	3x3x128/1	92x92x256
	Residual: R3			92x92x256
	Convolutional: C11	512	3x3x256/2	46x46x512
8x	Convolutional: C12	256	1x1x512/1	46x46x256
	Convolutional: C13	512	3x3x256/1	46x46x512
	Residual: R4			46x46x512
	Convolutional: C14	1024	3x3x512/2	23x23x1024
4x	Convolutional: C15	512	1x1x1024/1	23x23x512
	Convolutional: C16	1024	3x3x512/1	23x23x1024
	Residual: R5			23x23x1024
	Avgpool		Global	
	Connected		1000	
	Softmax			

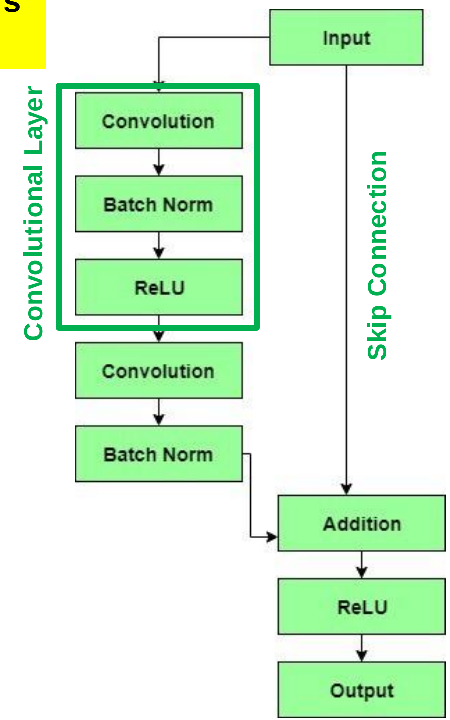
Last THREE layers; Avgpool, Connected and Softmax are NOT relevant for YOLO

These layers are relevant when Darknet53 is used in standalone manner

Cycle #1	Cycle #2
C6_1(C5)	C6_2(R2_1)
C7_1(C6_1)	C7_2(C6_2)
R2_1 = C5 + C7_1	R2_2 = R2_1 + C7_2

C8 is applied on R2_2 →C8(R2_2)

Cycle #1	Cycle #2	Cycle #3	Cycle #4
C15_1(C14)	C15_2(R5_1)	C15_3(R5_2)	C15_4(R5_3)
C16_1(C15_1)	C16_2(C15_2)	C16_3(C15_3)	C16_4(C15_4)
R5_1 = C14 + C16_1	R5_2 = R5_1 + C16_2	R5_3 = R5_2 + C16_3	R5_4 = R5_3 + C16_4

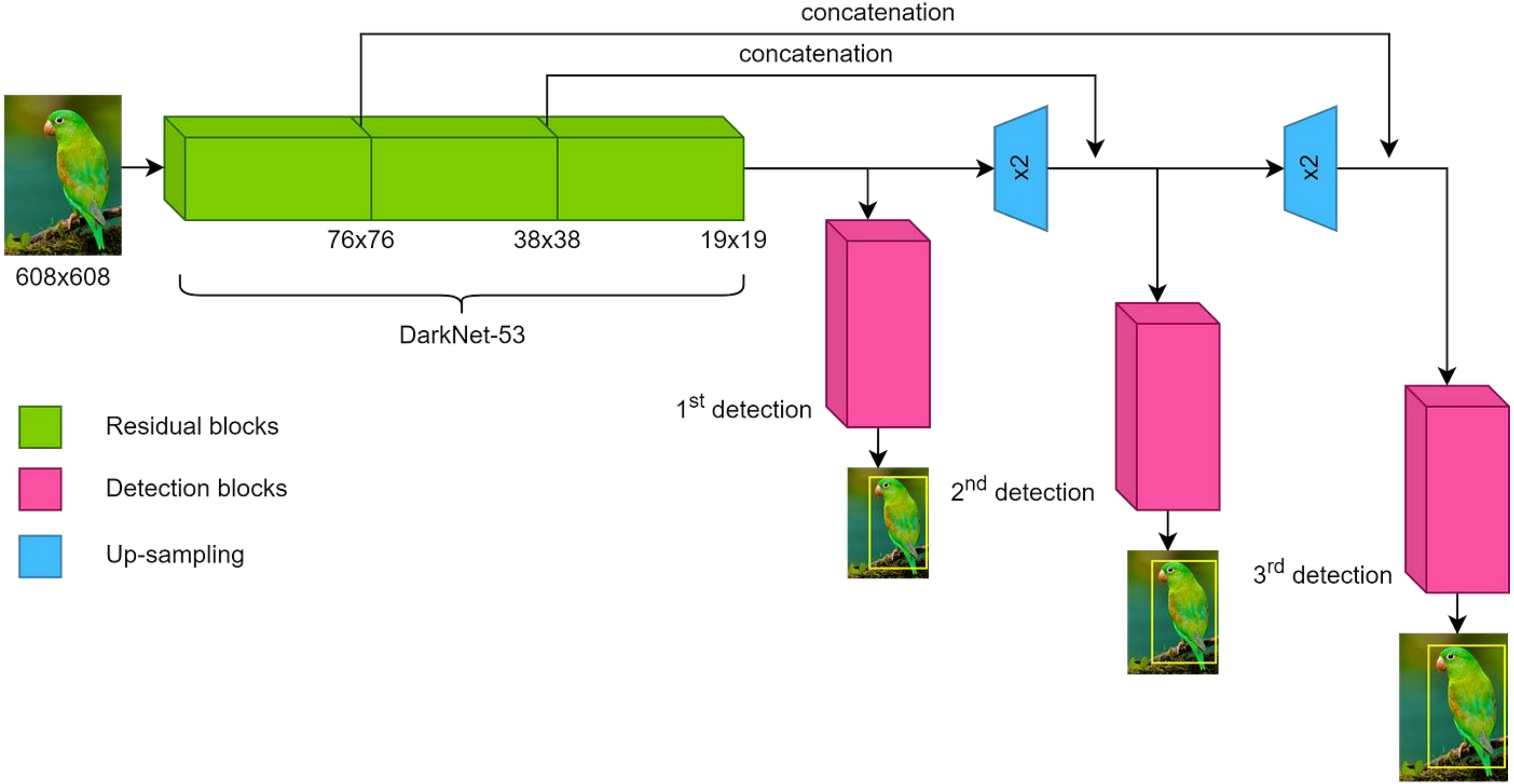


Darknet Standalone Performance

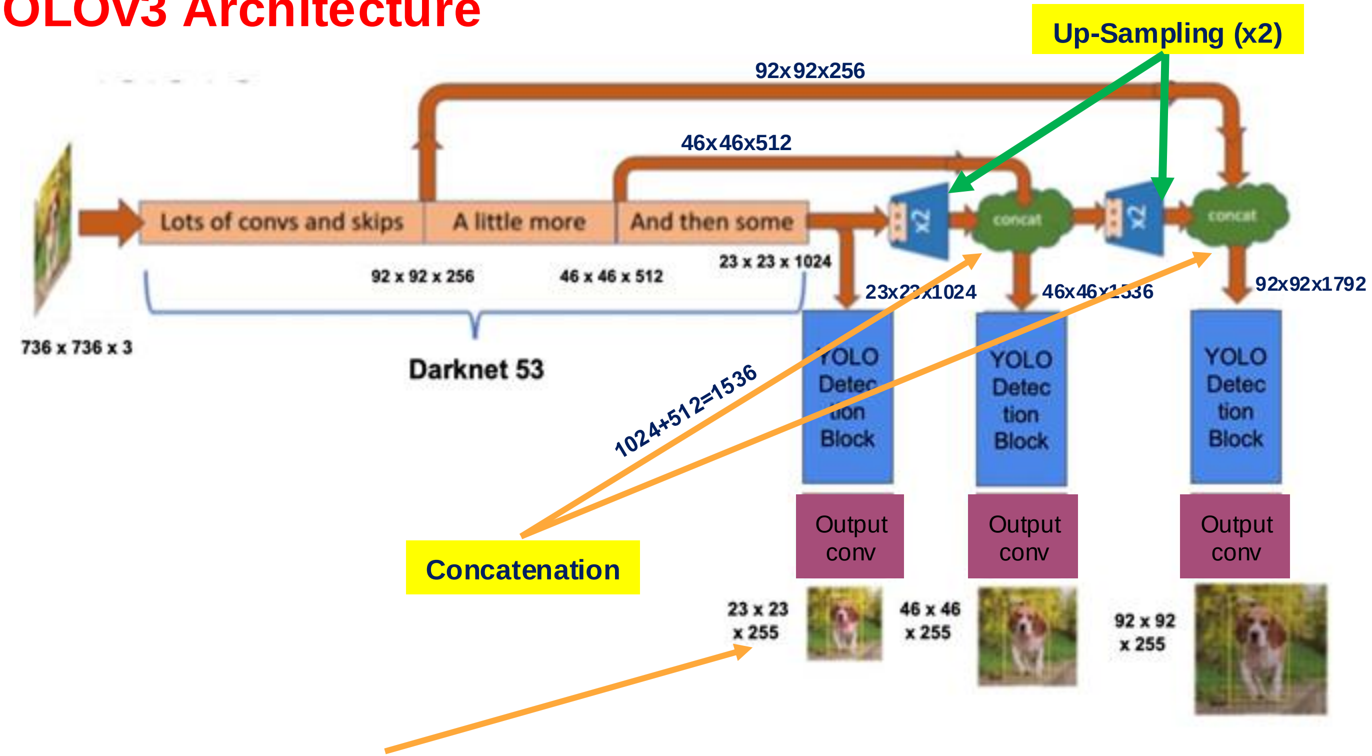
This new network is much more powerful than Darknet-19 but still more efficient than ResNet-101 or ResNet-152. Here are some ImageNet results:

Backbone	Top-1	Top-5	Bn Ops	BFLOP/s	FPS
Darknet-19 [15]	74.1	91.8	7.29	1246	**171**
ResNet-101[5]	77.1	93.7	19.7	1039	53
ResNet-152 [5]	**77.6**	**93.8**	29.4	1090	37
Darknet-53	77.2	**93.8**	18.7	**1457**	78

YOLOv3 Architecture



YOLOv3 Architecture



Explanation of number **255** in 23x23x255 OR 46x46x255 OR 92x92x255

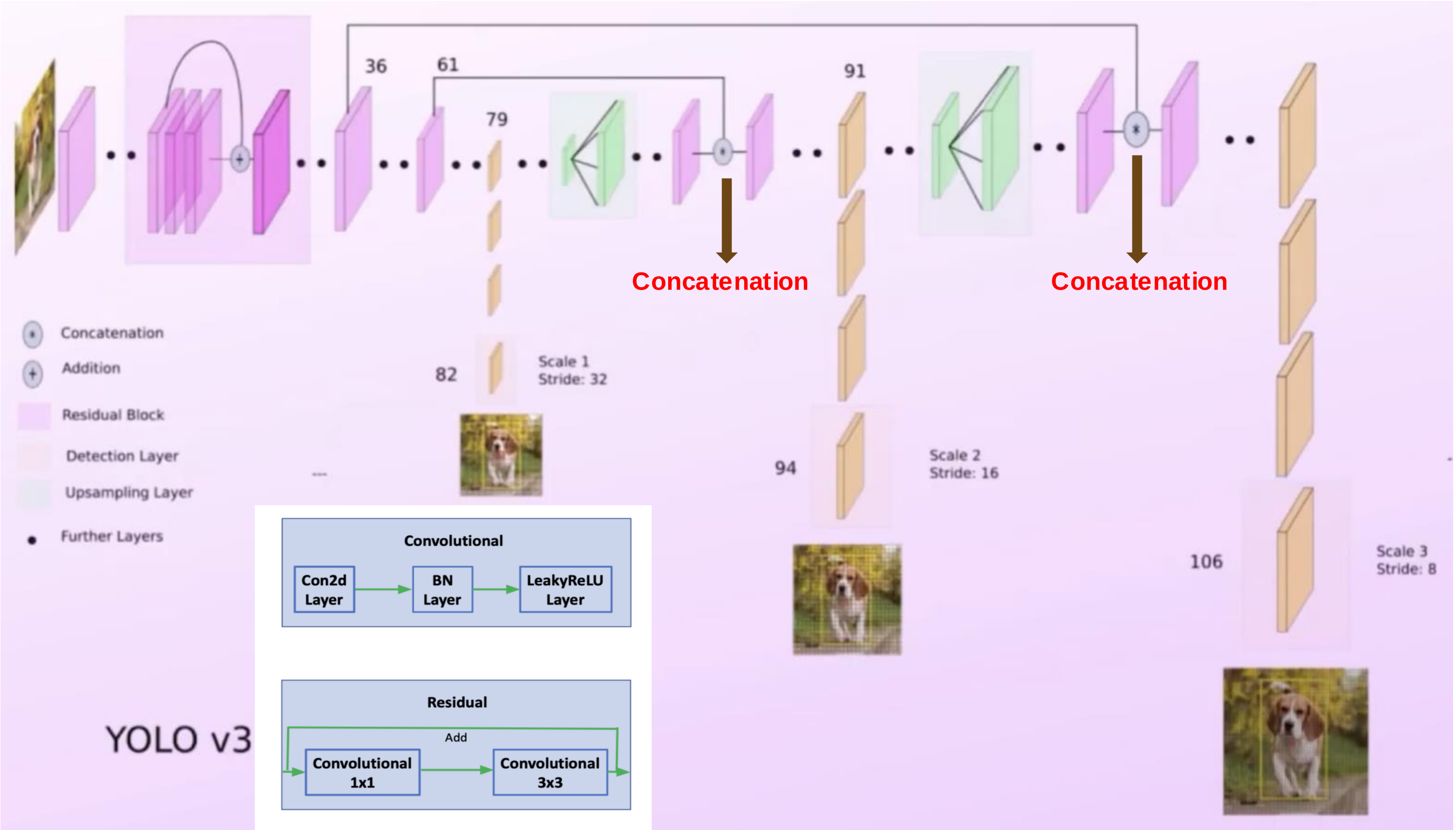
- Number of classes in COCO dataset = 80
- Number Anchor boxes in each grid cell = 3
- Size of each Anchor box = 5 + *# of classes = 5+80 = 85*
- Total data-length of each grid cell is = 3x85 = **255**

YOLO Predictions at Different Scales

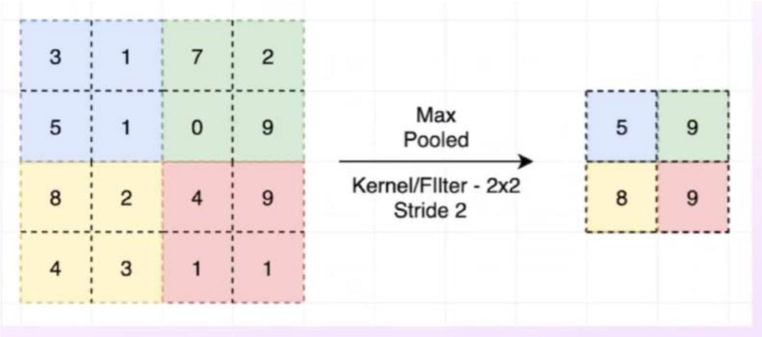
- In early YOLO architectures, the detection occurred only at the final layer
- However, since YOLOv3 the objects are detected from three different stages/layers of the network.
- The output form Darknet-53 is upscaled from (23x23x1024) to (46x46x1024) and is concatenated with the network stage of size (46x46x512) for a output of size (46x46x1536) which gets passed through a separate YOLO detection
- Similarly the concatenated output of size (46x46x1536) is upscaled again to size of (92x92x1536), which gets further concatenated with network stage of size (92x92x256) for a output size of (92x92x1792) gets passed through another separate YOLO detection.
- The detections at (46 x 46 x 1536) and (92 x 92 x 1792), in addition to detection at (23 x 23 x 1024) helps to find smaller objects, because of the larger feature grid across the image.

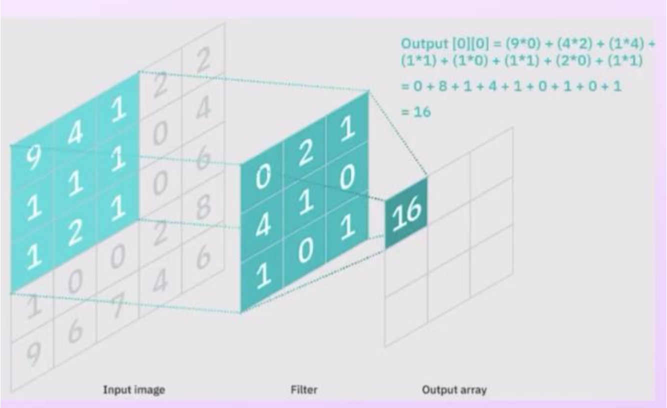
Source: https://pyimagesearch.com/2022/05/09/an-incremental-improvement-with-darknet-53-and-multi-scale-predictions-yolov3/

YOLOv3 Architecture



Max Pooling Vs Filters





Max Pooling: **Though it reduces the size, it also dilutes the features leading to loss of information**

Filter: **It can reduce the size of the output *(say, stride=2)* with relatively better ability to retain the original feature thus, does not lead to heavy loss of information**