

Module: backbone.py

YOLO belongs to the family of One-Stage Detectors (You only look once-one-stage detection). One-stage detection (also referred to as one-shot detection) is that you only look at the image once. If we would need to answer what does it mean in fewer sentences it would sound the following:

1. It is a sliding window and classification approach, where you look at the image and classify it for every window
2. In a region proposal network, you look at the image in two steps-the first to identify regions where there might be objects, and the next to specify it.

Backbone :

Backbone here refers to the feature-extraction architecture. YOLO by different names, such as YOLO Tiny or Darknet53. The difference between these is the backbone.

YOLO-Tiny has only 9 convolutional layers, so it's less accurate but faster, less resource hungry and better suited for mobile and embedded projects, Darknet53 has 53 convolutional layers, so it's more accurate but slower.

The YOLOv4 paper mentioned that the backbone used isn't Darknet53 but CSPDarknet53.

Backbone is one of the ways where we can improve accuracy, we can design a deeper network to extend the receptive field and to increase model complexity. And to ease the training difficulty, skip-connections can be applied. We can expand this concept further with highly interconnected layers.

A Dense Block in YOLOv4 backbone contains multiple convolution layers with each layer H_i composed of batch normalization, ReLU, and followed by convolution.

Instead of using the output of the last layer only, H_i takes the output of all previous layers as well as the original as its input.

i.e. x_0, x_1, \dots , and x_{i-1} . Each H_i below outputs four feature maps. Therefore, at each layer, the number of feature maps is increased by four-the growth rate.

Then a DenseNet can be formed by composing multiple Dense Blocks with a transition layer in between that is composed of convolution and pooling.

CSP stands for Cross-Stage-Partial connections. The idea here is to separate the input feature maps of the DenseBlock into two parts, one that will go through a block of convolutions, and one that won't.

The Cross Stage Partial architecture is derived from the DenseNet architecture which uses the previous input and concatenates it with the current input before moving into the dense layer.

CSPNet separates the feature map of the base layer into two-parts, one part will go through a dense block and a transition layer, the other part is then combined with a transmitted feature map to the next stage.

```

39 def cspdarknet53(input_data):
40
41     input_data = common.convolutional(input_data, (3, 3, 3, 32), activate_type="mish")
42     input_data = common.convolutional(input_data, (3, 3, 32, 64), downsample=True, activate_type="mish")
43
44     route = input_data
45     route = common.convolutional(route, (1, 1, 64, 64), activate_type="mish")
46     input_data = common.convolutional(input_data, (1, 1, 64, 64), activate_type="mish")
47     for i in range(1):
48         input_data = common.residual_block(input_data, 64, 32, 64, activate_type="mish")
49     input_data = common.convolutional(input_data, (1, 1, 64, 64), activate_type="mish")
50
51     input_data = tf.concat([input_data, route], axis=-1)
52     input_data = common.convolutional(input_data, (1, 1, 128, 64), activate_type="mish")
53     input_data = common.convolutional(input_data, (3, 3, 64, 128), downsample=True, activate_type="mish")
54     route = input_data
55     route = common.convolutional(route, (1, 1, 128, 64), activate_type="mish")
56     input_data = common.convolutional(input_data, (1, 1, 128, 64), activate_type="mish")
57     for i in range(2):
58         input_data = common.residual_block(input_data, 64, 64, 64, activate_type="mish")
59     input_data = common.convolutional(input_data, (1, 1, 64, 64), activate_type="mish")
60     input_data = tf.concat([input_data, route], axis=-1)
61
62     input_data = common.convolutional(input_data, (1, 1, 128, 128), activate_type="mish")
63     input_data = common.convolutional(input_data, (3, 3, 128, 256), downsample=True, activate_type="mish")
64     route = input_data
65     route = common.convolutional(route, (1, 1, 256, 128), activate_type="mish")
66     input_data = common.convolutional(input_data, (1, 1, 256, 128), activate_type="mish")
67     for i in range(8):
68         input_data = common.residual_block(input_data, 128, 128, 128, activate_type="mish")
69     input_data = common.convolutional(input_data, (1, 1, 128, 128), activate_type="mish")
70     input_data = tf.concat([input_data, route], axis=-1)
71
72     input_data = common.convolutional(input_data, (1, 1, 256, 256), activate_type="mish")
73     route_1 = input_data
74     input_data = common.convolutional(input_data, (3, 3, 256, 512), downsample=True, activate_type="mish")

```

```

71
72 input_data = common.convolutional(input_data, (1, 1, 256, 256), activate_type="mish")
73 route_1 = input_data
74 input_data = common.convolutional(input_data, (3, 3, 256, 512), downsample=True, activate_type="mish")
75 route = input_data
76 route = common.convolutional(route, (1, 1, 512, 256), activate_type="mish")
77 input_data = common.convolutional(input_data, (1, 1, 512, 256), activate_type="mish")
78 for i in range(8):
79     input_data = common.residual_block(input_data, 256, 256, 256, activate_type="mish")
80 input_data = common.convolutional(input_data, (1, 1, 256, 256), activate_type="mish")
81 input_data = tf.concat([input_data, route], axis=-1)
82
83 input_data = common.convolutional(input_data, (1, 1, 512, 512), activate_type="mish")
84 route_2 = input_data
85 input_data = common.convolutional(input_data, (3, 3, 512, 1024), downsample=True, activate_type="mish")
86 route = input_data
87 route = common.convolutional(route, (1, 1, 1024, 512), activate_type="mish")
88 input_data = common.convolutional(input_data, (1, 1, 1024, 512), activate_type="mish")
89 for i in range(4):
90     input_data = common.residual_block(input_data, 512, 512, 512, activate_type="mish")
91 input_data = common.convolutional(input_data, (1, 1, 512, 512), activate_type="mish")
92 input_data = tf.concat([input_data, route], axis=-1)
93
94 input_data = common.convolutional(input_data, (1, 1, 1024, 1024), activate_type="mish")
95 input_data = common.convolutional(input_data, (1, 1, 1024, 512))
96 input_data = common.convolutional(input_data, (3, 3, 512, 1024))
97 input_data = common.convolutional(input_data, (1, 1, 1024, 512))
98
99 input_data = tf.concat([tf.nn.max_pool(input_data, ksize=13, padding='SAME', strides=1), tf.nn.max_pool(input_data, ksize=9, padding='SAME',
100     , tf.nn.max_pool(input_data, ksize=5, padding='SAME', strides=1), input_data], axis=-1)
101 input_data = common.convolutional(input_data, (1, 1, 2048, 512))
102 input_data = common.convolutional(input_data, (3, 3, 512, 1024))
103 input_data = common.convolutional(input_data, (1, 1, 1024, 512))
104
105 return route_1, route_2, input_data
106

```

```

107 def cspdarknet53_tiny(input_data):
108     input_data = common.convolutional(input_data, (3, 3, 3, 32), downsample=True)
109     input_data = common.convolutional(input_data, (3, 3, 32, 64), downsample=True)
110     input_data = common.convolutional(input_data, (3, 3, 64, 64))
111
112     route = input_data
113     input_data = common.route_group(input_data, 2, 1)
114     input_data = common.convolutional(input_data, (3, 3, 32, 32))
115     route_1 = input_data
116     input_data = common.convolutional(input_data, (3, 3, 32, 32))
117     input_data = tf.concat([input_data, route_1], axis=-1)
118     input_data = common.convolutional(input_data, (1, 1, 32, 64))
119     input_data = tf.concat([route, input_data], axis=-1)
120     input_data = tf.keras.layers.MaxPool2D(2, 2, 'same')(input_data)
121
122     input_data = common.convolutional(input_data, (3, 3, 64, 128))
123     route = input_data
124     input_data = common.route_group(input_data, 2, 1)
125     input_data = common.convolutional(input_data, (3, 3, 64, 64))
126     route_1 = input_data
127     input_data = common.convolutional(input_data, (3, 3, 64, 64))
128     input_data = tf.concat([input_data, route_1], axis=-1)
129     input_data = common.convolutional(input_data, (1, 1, 64, 128))
130     input_data = tf.concat([route, input_data], axis=-1)
131     input_data = tf.keras.layers.MaxPool2D(2, 2, 'same')(input_data)
132
133     input_data = common.convolutional(input_data, (3, 3, 128, 256))
134     route = input_data
135     input_data = common.route_group(input_data, 2, 1)
136     input_data = common.convolutional(input_data, (3, 3, 128, 128))
137     route_1 = input_data
138     input_data = common.convolutional(input_data, (3, 3, 128, 128))
139     input_data = tf.concat([input_data, route_1], axis=-1)
140     input_data = common.convolutional(input_data, (1, 1, 128, 256))
141     route_1 = input_data
142     input_data = tf.concat([route, input_data], axis=-1)

```

```

140     input_data = common.convolutional(input_data, (1, 1, 480, 480))
141     route_1 = input_data
142     input_data = tf.concat([route, input_data], axis=-1)
143     input_data = tf.keras.layers.MaxPool2D(2, 2, 'same')(input_data)
144
145     input_data = common.convolutional(input_data, (3, 3, 512, 512))
146
147     return route_1, input_data
148
149 def darknet53_tiny(input_data):
150     input_data = common.convolutional(input_data, (3, 3, 3, 16))
151     input_data = tf.keras.layers.MaxPool2D(2, 2, 'same')(input_data)
152     input_data = common.convolutional(input_data, (3, 3, 16, 32))
153     input_data = tf.keras.layers.MaxPool2D(2, 2, 'same')(input_data)
154     input_data = common.convolutional(input_data, (3, 3, 32, 64))
155     input_data = tf.keras.layers.MaxPool2D(2, 2, 'same')(input_data)
156     input_data = common.convolutional(input_data, (3, 3, 64, 128))
157     input_data = tf.keras.layers.MaxPool2D(2, 2, 'same')(input_data)
158     input_data = common.convolutional(input_data, (3, 3, 128, 256))
159     route_1 = input_data
160     input_data = tf.keras.layers.MaxPool2D(2, 2, 'same')(input_data)
161     input_data = common.convolutional(input_data, (3, 3, 256, 512))
162     input_data = tf.keras.layers.MaxPool2D(2, 1, 'same')(input_data)
163     input_data = common.convolutional(input_data, (3, 3, 512, 1024))
164
165     return route_1, input_data
166

```

Mish activation:

Mish is a novel self regulated non-monotonic activation function which can be defined by,

$$f(x) = x \tanh(\text{softplus}(x)).$$

Mish properties help in better expressivity and information flow. Mish avoids saturation, which generally causes training to slow down due to near-zero gradients drastically. Being bounded below is also advantageous since it results in strong regularization effects.