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 *Hi* composed of batch normalization, ReLU, and followed by convolution.

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

i.e. x_0, x_1, ..., and x_{i-1}. Each *Hi* 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.

```python
def cspdarknet53(input_data):

    input_data = common.convolutional(input_data, (3, 3,  3,  32), activate_type="mish")
    input_data = common.convolutional(input_data, (3, 3, 32,  64), downsample=True, activate_type="mish")

    route = input_data
    route = common.convolutional(route, (1, 1, 64, 64), activate_type="mish")
    input_data = common.convolutional(input_data, (1, 1, 64, 64), activate_type="mish")
    for i in range(1):
        input_data = common.residual_block(input_data,  64,  32, 64, activate_type="mish")
    input_data = common.convolutional(input_data, (1, 1, 64, 64), activate_type="mish")

    input_data = tf.concat([input_data, route], axis=-1)
    input_data = common.convolutional(input_data, (1, 1, 128, 64), activate_type="mish")
    input_data = common.convolutional(input_data, (3, 3, 64, 128), downsample=True, activate_type="mish")
    route = input_data
    route = common.convolutional(route, (1, 1, 128, 64), activate_type="mish")
    input_data = common.convolutional(input_data, (1, 1, 128, 64), activate_type="mish")
    for i in range(2):
        input_data = common.residual_block(input_data, 64,  64, 64, activate_type="mish")
    input_data = common.convolutional(input_data, (1, 1, 64, 64), activate_type="mish")
    input_data = tf.concat([input_data, route], axis=-1)

    input_data = common.convolutional(input_data, (1, 1, 128, 128), activate_type="mish")
    input_data = common.convolutional(input_data, (3, 3, 128, 256), downsample=True, activate_type="mish")
    route = input_data
    route = common.convolutional(route, (1, 1, 256, 128), activate_type="mish")
    input_data = common.convolutional(input_data, (1, 1, 256, 128), activate_type="mish")
    for i in range(8):
        input_data = common.residual_block(input_data, 128, 128, 128, activate_type="mish")
    input_data = common.convolutional(input_data, (1, 1, 128, 128), activate_type="mish")
    input_data = tf.concat([input_data, route], axis=-1)

    input_data = common.convolutional(input_data, (1, 1, 256, 256), activate_type="mish")
    route_1 = input_data
    input_data = common.convolutional(input_data, (3, 3, 256, 512), downsample=True, activate_type="mish")
```

```python

    input_data = common.convolutional(input_data, (1, 1, 256, 256), activate_type="mish")
    route_1 = input_data
    input_data = common.convolutional(input_data, (3, 3, 256, 512), downsample=True, activate_type="mish")
    route = input_data
    route = common.convolutional(route, (1, 1, 512, 256), activate_type="mish")
    input_data = common.convolutional(input_data, (1, 1, 512, 256), activate_type="mish")
    for i in range(8):
        input_data = common.residual_block(input_data, 256, 256, 256, activate_type="mish")
    input_data = common.convolutional(input_data, (1, 1, 256, 256), activate_type="mish")
    input_data = tf.concat([input_data, route], axis=-1)

    input_data = common.convolutional(input_data, (1, 1, 512, 512), activate_type="mish")
    route_2 = input_data
    input_data = common.convolutional(input_data, (3, 3, 512, 1024), downsample=True, activate_type="mish")
    route = input_data
    route = common.convolutional(route, (1, 1, 1024, 512), activate_type="mish")
    input_data = common.convolutional(input_data, (1, 1, 1024, 512), activate_type="mish")
    for i in range(4):
        input_data = common.residual_block(input_data, 512, 512, 512, activate_type="mish")
    input_data = common.convolutional(input_data, (1, 1, 512, 512), activate_type="mish")
    input_data = tf.concat([input_data, route], axis=-1)

    input_data = common.convolutional(input_data, (1, 1, 1024, 1024), activate_type="mish")
    input_data = common.convolutional(input_data, (1, 1, 1024, 512))
    input_data = common.convolutional(input_data, (3, 3, 512, 1024))
    input_data = common.convolutional(input_data, (1, 1, 1024, 512))

    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'
                            , tf.nn.max_pool(input_data, ksize=5, padding='SAME', strides=1), input_data], axis=-1)
    input_data = common.convolutional(input_data, (1, 1, 2048, 512))
    input_data = common.convolutional(input_data, (3, 3, 512, 1024))
    input_data = common.convolutional(input_data, (1, 1, 1024, 512))

    return route_1, route_2, input_data

```

```python
def cspdarknet53_tiny(input_data):
    input_data = common.convolutional(input_data, (3, 3, 3, 32), downsample=True)
    input_data = common.convolutional(input_data, (3, 3, 32, 64), downsample=True)
    input_data = common.convolutional(input_data, (3, 3, 64, 64))

    route = input_data
    input_data = common.route_group(input_data, 2, 1)
    input_data = common.convolutional(input_data, (3, 3, 32, 32))
    route_1 = input_data
    input_data = common.convolutional(input_data, (3, 3, 32, 32))
    input_data = tf.concat([input_data, route_1], axis=-1)
    input_data = common.convolutional(input_data, (1, 1, 32, 64))
    input_data = tf.concat([route, input_data], axis=-1)
    input_data = tf.keras.layers.MaxPool2D(2, 2, 'same')(input_data)

    input_data = common.convolutional(input_data, (3, 3, 64, 128))
    route = input_data
    input_data = common.route_group(input_data, 2, 1)
    input_data = common.convolutional(input_data, (3, 3, 64, 64))
    route_1 = input_data
    input_data = common.convolutional(input_data, (3, 3, 64, 64))
    input_data = tf.concat([input_data, route_1], axis=-1)
    input_data = common.convolutional(input_data, (1, 1, 64, 128))
    input_data = tf.concat([route, input_data], axis=-1)
    input_data = tf.keras.layers.MaxPool2D(2, 2, 'same')(input_data)

    input_data = common.convolutional(input_data, (3, 3, 128, 256))
    route = input_data
    input_data = common.route_group(input_data, 2, 1)
    input_data = common.convolutional(input_data, (3, 3, 128, 128))
    route_1 = input_data
    input_data = common.convolutional(input_data, (3, 3, 128, 128))
    input_data = tf.concat([input_data, route_1], axis=-1)
    input_data = common.convolutional(input_data, (1, 1, 128, 256))
    route_1 = input_data
    input_data = tf.concat([route, input_data], axis=-1)
```

```
    input_data = common.convolutional(input_data, (1, 1, 128, 256))
    route_1 = input_data
    input_data = tf.concat([route, input_data], axis=-1)
    input_data = tf.keras.layers.MaxPool2D(2, 2, 'same')(input_data)

    input_data = common.convolutional(input_data, (3, 3, 512, 512))

    return route_1, input_data

def darknet53_tiny(input_data):
    input_data = common.convolutional(input_data, (3, 3, 3, 16))
    input_data = tf.keras.layers.MaxPool2D(2, 2, 'same')(input_data)
    input_data = common.convolutional(input_data, (3, 3, 16, 32))
    input_data = tf.keras.layers.MaxPool2D(2, 2, 'same')(input_data)
    input_data = common.convolutional(input_data, (3, 3, 32, 64))
    input_data = tf.keras.layers.MaxPool2D(2, 2, 'same')(input_data)
    input_data = common.convolutional(input_data, (3, 3, 64, 128))
    input_data = tf.keras.layers.MaxPool2D(2, 2, 'same')(input_data)
    input_data = common.convolutional(input_data, (3, 3, 128, 256))
    route_1 = input_data
    input_data = tf.keras.layers.MaxPool2D(2, 2, 'same')(input_data)
    input_data = common.convolutional(input_data, (3, 3, 256, 512))
    input_data = tf.keras.layers.MaxPool2D(2, 1, 'same')(input_data)
    input_data = common.convolutional(input_data, (3, 3, 512, 1024))

    return route_1, input_data

```

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.